Report on Project for MI 2.01

Hong Hanh

04 April 2019

1 Achievement

In this project, we implement a tool allowing the execution of word count application following the map-reduce principle and compare the execution time between centralized and map-reduce (2,3 nodes) version. The 1.1 GB dataset is generated with 1000 common words. The functions are executed successfully mentioned below:

- **Map**: Read file, produce count table, integrated into **Daemon**, and send results to **Launch**.
- **Reduce**: Compute the sum from map and send the results.
- **Daemon**: Bind socket for different ports, receive connection for receiving blocks and requesting **Map**'s execution.
- **Split**: Split 1GB file into blocks and sends blocks to **Daemon**.
- **Launch**: Fork and exec **Split**, receive progress from **Split** using pipe,connect to **Daemon**, and send signal to execute **Map**.
- **Centralized**: Count word in a big text file.
- **Generate**: Generate 1.1 GB text data from 1000 most common words.

The execution time of map-reduce version and centralized version are demonstrated in the table below:

		Execution time (s)
Centralized version		435
Map-reduce verion	**2 Daemons**	235
	3 Daemons	192

Table 1: Execution time of different versions

As we can see, the results illustrates that the the program with multiple Daemons reduces execution time to approximately a half compared to the centralized version.

2 Instructions on local machine

We instruct the implementation with the centralized version and the map-reduce case of 2 daemons (similar execution with 3 and 4 daemons).

- Build program:
  gcc -o generate generate.c
  gcc -o split split.c utils.c
  gcc -o daemon daemon.c
  gcc -o reduce reduce.c
  gcc -o map map.c
  gcc -o launch launch.c utils.c
  gcc -o centralized centralized.c

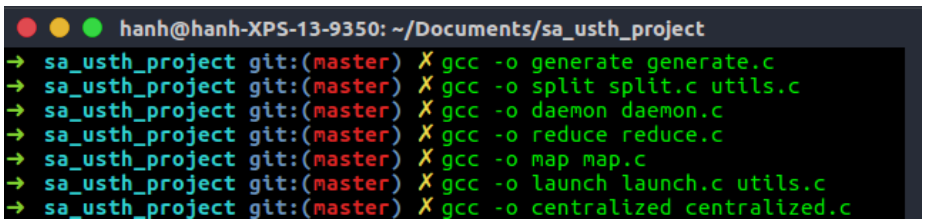

Figure 1: Syntax to build program

- Execute program:

  – Generate 1 GB data:
    ./generate data_1GB.txt 1024

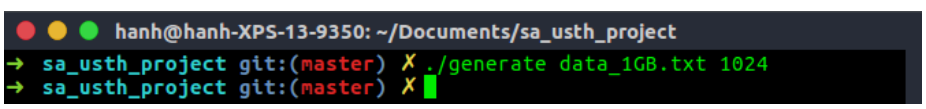

Figure 2: Generate 1GB data

  – Centralized version:
    ./centralized data_1GB.txt "server/centralized_output"

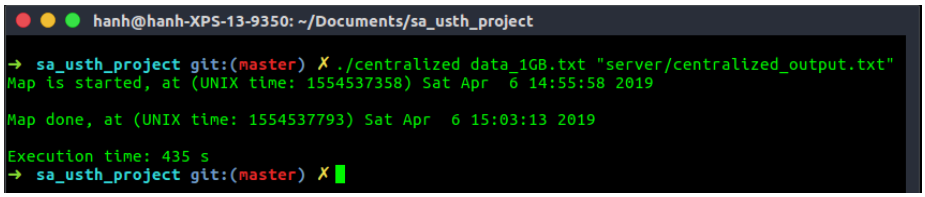

Figure 3: Execute counting with centralized version

– Map-reduce version (2 nodes):
  ./launch 2 "server/reduce_output.txt"
  ./split 2 data_1GB.txt
  ./daemon
  ./daemon

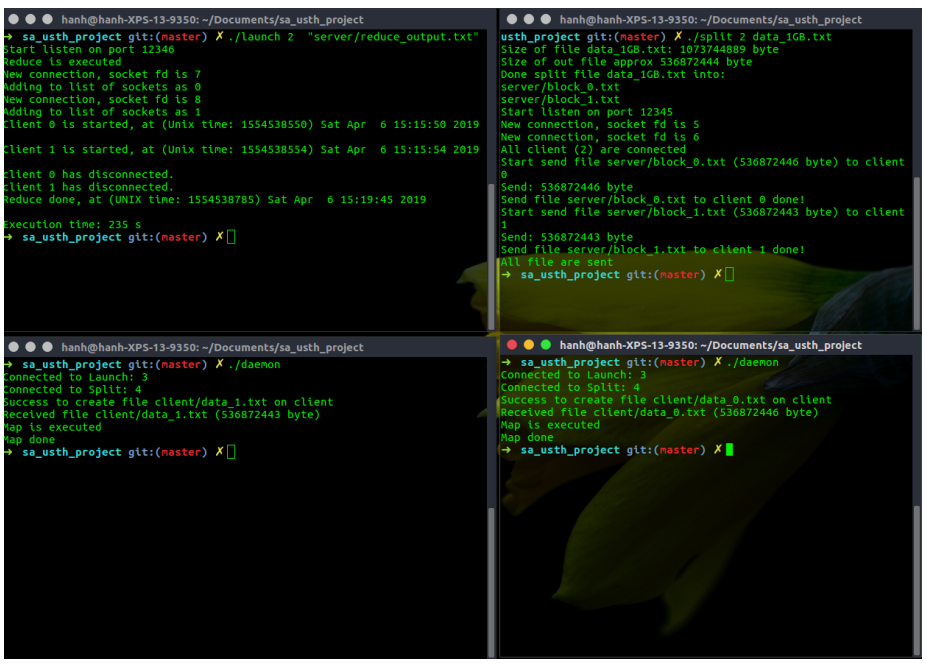

Figure 4: Execute counting with 2 Daemons

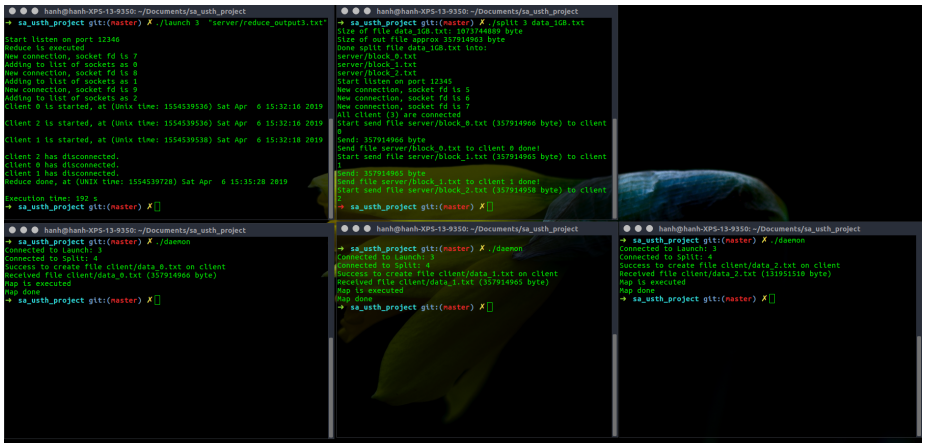

Figure 5: Similarly, execute counting with 3 Daemons

3 Workflow

For the map-reduce version, we executed the following programs:

- Execute **Generate** to generate 1.1 GB data.
- Run **Launch** with max number of clients (in our case, 2) and the file containing the final result ("server/reduce_output.txt").
  - Execute **Reduce**
  - Listen to port 12346.
- Run **Split** with max number of clients (i.e 2) and the 1.1 GB dataset.
  - Split file of into 2 file blocks (number of file blocks equals number of clients).
  - Listen to port 12345.
- Execute 2 **Daemon**s.
- **Split** sends data to **Daemon**s, **Daemon**s saves data to local file.
- **Daemon** runs **Map** with local file.
- **Daemon** sends output to **Launch**.
- **Launch** waits the results of all **Daemon**s and put the result to **Reduce**
- **Launch** saves **Reduce**'s output to file.

The details are commented on source code.