CSC 258 – Assignment 4 MPI & Hadoop

Shuyang Liu
April 28th, 2017

1. Files

```
.
├── Hadoop – The directory containing all of the src code for Hadoop program
│   ├── build
│   │   ├── classes
│   │   │   ├── Histogram.class
│   │   │   ├── WordCount.class
│   │   │   ├── WordCount$IntSumReducer.class
│   │   │   └── WordCount$TokenizerMapper.class
│   │   └── WordCount.jar    - JAR file for execution
│   ├── build.xml – Ant building script
│   ├── run – running script
│   └── src
│       ├── Histogram.java – A Hadoop HDFS Tool for outputting histogram of word length
│       └── WordCount.java – A Hadoop MapReduce program for word length frequency
│
└── MPI – The directory containing all of the src code for MPI program
    └── mpi
        ├── gauss.c – MPI program of Gaussian Elimination
        ├── linux – build directory
        │   ├── gauss – executable program
        │   ├── gauss.o
        │   ├── hosts – a list of all host machines in the cluster
        │   ├── hosts~
        │   ├── hosts.bk
        │   ├── matrix_10.dat
        │   └── matrix_2000.dat – testing data
        ├── Makefile – building script
        └── MPICH-BUILD-NOTES
```

2. Running Instruction

2.1 Hadoop

For compiling the Hadoop program using the build script, simply run:

```
$ ant
```

This would generate a .jar file for execution on the Hadoop VM
For running the .jar file, simply run the running script:

```
$ ./run
```

2.2 MPI

For compiling the MPI program, run the following command:

```
$ make
```

This would build the program in ./linux directory
For running the program,

```
$ mpirun -n <# of processes> -ppn <# of proc per node> ./gauss
./matrix_2000.dat
```

3. Descriptions of Implementation

3.1 Hadoop

Hadoop is a framework tool that provide services such as organizing and managing large amount of data in distributed systems. In this Assignment 4, two major components of Hadoop are used for querying word length frequencies: HDFS and MapReduce. HDFS is a file system that manages big data files in a set of machine nodes. Using the VM interface provided by Hadoop, programmer is able to access the files inside as if in a single file system.

In this assignment, a HDFS Tool is implemented for generating a histogram of word length frequencies. Since I cannot directly access the file system without using either HDFS commands or the supported Java API, having a HDFS tool for accessing output files would be a choice for generating the histogram. The HDFS Tool is written using the Java API provided by Hadoop. Inside class Histogram, there is a run method implemented for outputting histograms. It can access the output files by instantiating a `FileSystem` object and use this object to navigate through the HDFS. This method runs after MapReduce finished execution and generate output file in `/output/` folder. It directly output the results to terminal using `System.out` stream

Another part of this assignment is implementing a MapReduce program consists a mapper and a reducer. The mapper first maps each string token (separated by spaces) with the word length. In this program, the word length (of type `IntWritable`) is used as output key and a value of one (of type `IntWritable`) is used as value for each entry. The reducer then reduces

the entries by summing up the number of values for each key. Eventually, pairs of `<word_length, number of words>` is stored to `/output/part-r-00000`.

3.2 MPI for Gaussian Elimination

The MPI program for Gaussian Elimination was written by modifying the original gauss.c code. In the program, process 0 is used as master process whereas others are worker processes. Only the master process have a full record of matrix, where in other processes matrix is only a NULL pointer. In this way, there would only be one copy of matrix over all of the processes. The master only send necessary information to worker processes, such as `nsize`, and current row of iteration. There are two main parts in `computeGauss` function. One is the scaling part, and another one is the factorization part. In the scaling part, elements of the ith row are divided into sets, which are assigned to different worker processes using `MPI_Scatterv`. Later, after workers finish processing the elements, they are sent back to master process by `MPI_Gatherv`. The master process is in charge of copy back the newly computed values to the original matrix. In the factorization part, blocks of rows are assigned to worker processes in order to factorize the values. Similar MPI functions are used in this part.

4. Results and Discussions

4.1 Hadoop

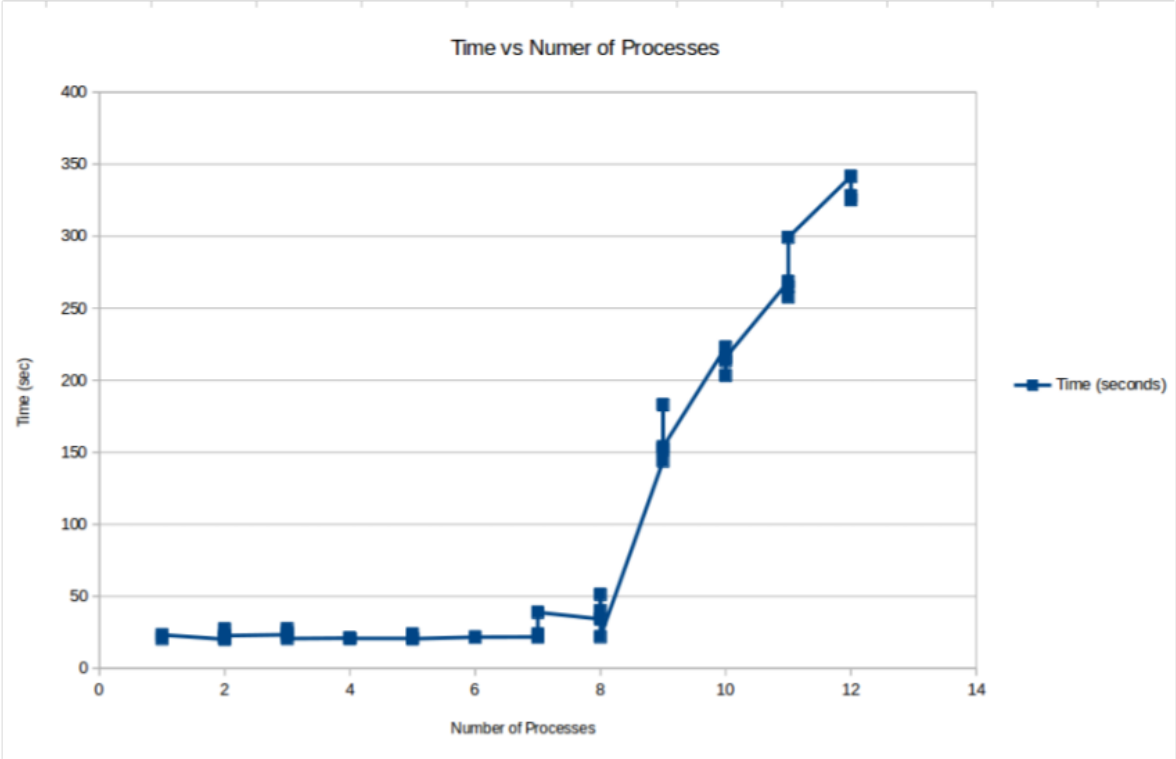
For test file 1-5 in input directory:

```
 1: *******                                            22474
 2: ***********************************                104452
 3: ***************************************************142609
 4: ************************************               107247
 5: ************************                           72616
 6: ******************                                 56784
 7: ****************                                   46933
 8: ***********                                        32628
 9: ********                                           23375
10: *****                                              15299
11: **                                                 8394
12: *                                                  4942
13:                                                    2302
14:                                                    1086
15:                                                    644
16:                                                    247
17:                                                    146
18:                                                    86
19:                                                    54
20+:                                                    26
```

4.2 MPI

master
node01

np	ppn	Time (seconds)
1	1	20.523231
1	2	20.421923
1	3	20.414221
1	4	23.224946
2	1	20.205672
2	2	21.03168
2	3	27.319608
2	4	22.689261
3	1	23.32652
3	2	27.549622
3	3	24.680651
3	4	20.675911
4	1	20.981808
4	2	20.75517
4	3	20.752274
4	4	20.77889
5	1	20.78748
5	2	20.832997
5	3	23.701727
5	4	20.545878
6	1	21.659193
6	2	21.858004
6	3	21.861495
6	4	21.899773
7	1	21.784167
7	2	23.841143
7	3	23.619349
7	4	38.843424
8	1	34.210552
8	2	51.28602
8	3	40.034412
8	4	21.904674
9	1	143.90781
9	2	151.96466
9	3	182.93564
9	4	153.74751
10	1	222.93538
10	2	203.28996
10	3	213.50298
10	4	215.84329
11	1	268.52607
11	2	264.88567
11	3	257.82723
11	4	299.21799
12	1	341.56376
12	2	325.33375
12	3	328.03201



As the number of processes exceed the number of hardware nodes, the time is takes for finish execution grows rapidly. This may due to the unfairness in work splitting and on overhead in synchronizations.

Another interesting result being observed from this program execution is that, adding more processes does not really give any speedup in this program. This can be explained if looking at the actual implementation of the program. In the `computeGauss` function, before every time the master process call `MPI_Scatterv`, it has to copy the matrix content to a send buffer for later distributing work to worker processes. This may take longer time than computing matrix values. Also, when receiving values from worker processes, the same copy routine is executed to copy the values back to original matrix. This copy routine may creates overheads and take longer than the actual computing time. Therefore, the expected speedup may be zeroed with the copying routines.

Another possible factor that can affect the performance, in a distributed system, would be the network speed between each node. The master process has to wait until it receive back

the values that the workers send to it. Slow network speed can cause it to wait too long to finish the works.

5. References

https://hadoop.apache.org/docs/r2.6.2/api/org/apache/hadoop/io/Text.html

https://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/Mapper.html

https://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/Reducer.html

https://hadoop.apache.org/docs/r2.7.3/hadoop-project-dist/hadoop-common/FileSystemShell.html#rmr

https://hadoop.apache.org/docs/r2.6.1/api/org/apache/hadoop/mapred/OutputFormat.html