National University
of computer and emerging sciences

PROJECT REPORT

MAP REDUCE

GROUP MEMBERS:

Ayesha Saif 19k-0271

Haris Aqeel 19k-0249

M.Jahanzeb 19k-1326

Objective:

The objective of the project is to use MAPREDUCE to perform a word count and compare strong scalability with the weak one.

Introduction:

We will be doing a version of map-reduce using MPI to perform word counting over a large number of files. There are 3 steps to this process:

1. is to read in the master file list which will contain the names of all the files that are to be counted. Note that only 1 of your processes should read this file. Then each of the processes should receive their portion of the file from the master process. Once a process has received its list of files to process, it should then read in each of the files and perform a word counting, keeping track of the frequency each word found in the files occurs. We will call the histogram produced the local histogram. This is similar to the map stage or map-reduce.

2. is to then combine frequencies of words across processes. For example the word 'cat' might be counted in multiple processes and we need to add up all these occurrences. This is similar to the reduce stage of map-reduce.

3. is to have each of the processes send their local histograms to the master process. The master process just needs to gather up all this information. Note that there will be duplicate words between processes. The master should then print out the results to the screen.

Methodology:

The program has been implemented to reach the goal of having a perfect division of the problem between the workers.

To reach this goal, instead of splitting the problem by the number of files or by the number of rows, the approach is to read the size in byte of each file, make the sum, and split by the number of workers. In this way each worker reads and performs the word count on the same sized problem. This approach splits the problem in the best way but makes it more difficult to manage. Each file can have a different size and each worker must process a fixed size.

To handle this situations, at worker must first calculate the part processed by the previous workers and then begin to process its own part.

To manage the read and word count, a file reader has been implemented that handles all the conditions including, when it starts, when it's on a space, when it's on a new line, when it's at an EOF, etc.

When a worker end its part inside a word, this is ignored and is processed by the next worker.

Implementation details

MPI features

In this project has been used the following MPI features:

- MPI_Bcast
- MPI_Gather
- MPI_Gatherv
- MPI_Pack
- MPI_Unpack

Custom Structures

Some custom structures have been defined:

File Information

Contains the information of a file to read:

- the path (of dynamic length)
- the size of a file

File Information Container

Is a dynamic list of File Information structure, contains:

- the entries
- the number of entries
- the total size of the files

These are the main functions used to manage the entries plus an additional library to manage the communication of File Information Container structure through MPI.

Counter

Keeps track of the frequency of a single word, contains:

- the word (of dynamic length)
- the counter

Counter Container

Is a dynamic list of Counter structure, contains:

- the entries
- the number of entries

These are the main functions to manage the entries plus an additional library to manage the communication of Counter Container structure through MPI and to make the merge of multiple Counter Container structures coming to the master process from the workers.

Logs

It has been implemented an MPI library for the print of messages and logs in the standard output and in the log files.

Instructions

Compile

Inside the src directory run the following command:

make

Local Execution

Inside the src directory run the following command:

```
mpirun -np NUMBER_OF_PROCESSORS main LIST_OF_FILES
```

- **NUMBER_OF_PROCESSORS**

  Is the number of processors that MPI will use for the execution.

- **LIST_OF_FILES (optional)**

  Is the path of the file that contains the paths of all files to be read and counted.

If omitted, it will be used the file: "**data/00_master.txt**"

Conclusion

In conclusion, the program was able to read multiple files and count the number of words in them, printing this value in a new file it creates. We were also able to successfully draw a comparison between strong scaling as well as weak scaling and conclude that the execution time decrease in case of the former. This can be easily spotted in the diagrams shown below.