Cmpe 300 Fall 2017
Application Project

Barın ÖZMEN
2012400045

Problem Description

The project is about to implement a mapReduce algorithm on MPI to count the word occurrences in a file. MPI is a library for c to make able to run a program parallel. This is the first time i met with parallel codes, and the perception part is a little bit tricky. In sequential code, it is a really is to write with only a simple sorting algorithm. Yet, to complete and understand MPI part, i needed to look from a different perspective.

Difficulties & Problem Solutions

To solve the problem, i spent 2 days to figure out MPI working mechanism. When the program started, the code is shared by other processes. You can split it with getting the rank of the process. Starting from the root_process(0), there are other slave processes which their ranks goes up through 1, 2, 3... Finally, i understood the concept of parallel programming. In the root_process, you can send the input to other slave processes with "MPI_Send" by typing its rank, tag and Byte space which was the most problematic part on that project according to me. I tried to sent a struct array speech with MPI_Send, yet couldn't done it properly for a long time. The problem is fixed when i changed MPI_Byte with MPI_LONG_DOUBLE_INT. On the other hand, to recieve array that sent with MPI_Sent, typing MPI_Recv() with rank, tag and the other parameters is enough.
The other part of the project is a implementing a sorting algorithm which we have done it many times. The easiest part and finished quickly without any trouble.

Inputs & Outputs

To run the program, typing **$lamboot** to terminal is need for starting MPI. Then, by typing **$mpicc mapReduce.c** to compile c source file. And last, **$mpirun -np —num_procs— ./a.out —input_file—** to start the program. num_procs is an integer value to assign how many proccess is going to start run and input_file for the speech file which going to be sorted and reduced.

Output file is as follows:

1) **intermediateMappedFile.txt:** After splited words into slave processes, each process map the words and send them to root process. When all mapped input values are collected, it is written in that file.
2) **intermediateMergedFile.txt:** Split the input again, each process is sort its part and send them to root process again. This file contains each sorted part for each process.
3) **lastMergedFile.txt:** Lastly, root process merges all sorted part into 1 part and prints it in file.
4) **reducedFile.txt:** Last version which words are reduced and the values updated.