Map-Reduce like Program using multi-Threading

Michael Pini - GN020133

February 28, 2019

1 Introduction

In this report I will discuss my MapReduce equivalent system that I developed in Java using multi threading and data manipulation. The system is similar to a MapReduce system due to the fact it breaks the problem up into blocks and would ideally run calculations on the data simultaneously (to gain superior data manipulation speeds) and combine then reduce the data to find the answer. An example of a MapReduce like system would be if a king wished to find their tallest citizen, they could send a representative to each town who would bring back the tallest person they could find. The tallest people can then be compared to see who is the absolute tallest. This method is far more efficient than sending one representative to every single town. This report will describe my development of this prototype system to provide a good understanding of how it was developed. The report will also go into how my system emulates a MapReduce system to show the differences and similarities between the systems. The report will also discuss the outputs generated by the software and discuss how the work can be improved.

2 High-level description of the development of the prototype software.

The prototype software that I developed reads the two CSV(Comma Separated values) tables into two dimensional ArrayLists. The data is then split into four equal parts (to the closest integer) and sent to 4 different threads that perform data manipulation tasks to complete the set objectives. The manipulated data is then retrieved from the threads and combined into a singular dataset. The datasets are then reduced when necessary by deleting/combining duplicate rows and performing more data manipulations. The software then prints the results to the console showing the combined and reduced datasets. The primary dataset (AComp Passenger data) is then saved to a pre-defined file location in CSV format. This format can be opened with many software packages such as Microsoft Excel to view the table. The software utilizes three classes, the first class is used to read in the files, handle the thread initializations and manage results retrieved from the threads. The second class is used to create and manage a new thread. Finally, the third class is used to perform the majority of the data manipulation for the threads.

2.1 Git command line process undertaken

For my GitHub upload I used an imbedded tool within eclipse called egit. I used this instead of using console commands because it would be faster and easier to merge the file in future revisions of the software. Additionally, due to previous experience using the console, egit seemed like a faster and easier method. I decided to use HTTPS to push my git instead of SSH because SSH will prompt the user for an SSH passphrase (password). HTTPS does not require this therefore it is faster and easier.

2.2 MapReduce

MapReduce is a processing technique that is implemented to perform parallel processing on data. MapReduce is useful because it is excellent at scalability therefore can be used to process big data. MapReduce works by taking a problem, breaking it down into its base components, performing parallel manipulations and reducing the data to form a result. As the name implies, MapReduce consists of two main phases, the map and the reduce phase. The map phase is used to create and process the input dataset. The mapper generally takes each line of code individually (potentially using a buffer) and processes the data in a predefined way to get a new dataset. The mapper then passes this new dataset to the combine phase which is used to combine pairs of key/value. the new dataset that has been produced by the mapper/s and combined by the combiner can then be passed to the shuffle and sort phase which can be used to shuffle or sort the output data to allocate which reducer the data should be sent to and perform any sorting necessary. In the reduce phase the data with sorted key/value pairs is processed to make it useful in solving a problem/finding an answer. Using this technique with parallel processing makes the framework extremely scalable therefore It can be easily implemented to work with huge datasets. In summery the input data is split between mappers that produce key/value pairs which are then shuffled and optionally sorted then sent into the reducers that produce the final result.

2.3 How my software is similar to a MapReduce system

One way in which my software is similar to a MapReduce system is that both systems use parallel- processing. My system uses threads which would ideally run in parallel with each other and process their datasets individually. My system also combines the data and reduces it to provide a final result. An example of how my software prototype acted in a similar way to a MapReduce system is when it was used to find the person with the most nautical miles travelled. My program split the data into four sets of input data and passed each set to four separate threads (each thread is like a mapper). The threads then calculate the person with the most nautical miles travelled for their dataset and send the data to be reduced into just one person. The main issue with my software in comparison to a MapReduce system is that mine is not accurate because I did not shuffle the data or use key/value pairs therefore if a person was in two different datasets their nautical miles travelled would be incorrect.

3 The strategy derived to handle input data error detection/correction and run-time recovery

For the input data I removed some rows but also corrected others. I removed the entire row if any columns were missing data, contained only a “0” or had invalid characters such as”

For run-time recovery I used try and catch statements therefore if an error/exception occurs I can redirect the code to do something else. An example of how I used this is when a thread starts it checks to see if the start was successful, if not it calls itself recursively to attempt to start the thread again.

4 The output format of any reports that each job produces

The primary output I used throughout my project is a two-dimensional Java ArrayList although sometimes I used a one/three-dimensional ArrayList. I used this because it could store a dataset with any number of rows or columns and data manipulation can be easily performed upon it. The software also saves a version of the AComp Passenger dataset that has all of the errors corrected or moved to a CSV file called output. After each thread has completed their tasks, they print their outputs to the console to give the user an understanding of the results each thread has discovered. The combined and reduced outputs are then printed to the console to demonstrate the overall findings of the software.

5 Self-appraisal of my (equivalent) MapReduce runtime software

In this section of the report I will discuss how my project went and identify some areas in which I could Improve my software.

5.1 Areas I did well in

I used lots of methods instead of an iterative approach therefore the software should be more maintainable and be easier to expand upon. This also promotes code re-use which is beneficial because it saves development time. One example of this is my printTable method which prints out a table to the console. This was used frequently during testing and is also extremely useful for printing out the results. Another area I did well in is commenting my code. I used both standard and java comments regularly throughout my project. This improves the maintainability of my software and also means I can use eclipse to generate the Javadoc comments into a helpful file. Another aspect of the project I am proud of is the methods used to pre-process the datasets. I am proud of this because I used many techniques that I haven't used before and even data-types such as sets in interesting ways.

5.2 Areas I Could improve upon

The primary aspect of my program that could be improved is the multi-threading. Threads did not operate properly when run in parallel (which is their purpose) and I could not find out how this issue could be fixed or even what caused it. The only way I was able to remedy this is by waiting for each thread to finish and doing it sequentially. This is bad because it completely negates the purpose for using threads or splitting the data. Because of this issue my program is slower than it should be, and many aspects have become redundant. Not implementing a shuffle or sort this would improve the accuracy of my results greatly because currently if there is a key data entry in two or more subsets of data it would not be processed correctly therefore the final result could be inaccurate. Another aspect of my program that could be improved is that it is very messy. This is a negative aspect of the project because it would be difficult for another developer to work on my code because there is too much code that could be shortened. My code could probably be shortened from the ¿1000 lines it currently is to ¡500. This would be a huge improvement because it would be far less confusing and more maintainable.