1 Activity: Distributed Algorithm: Sorting

Task 1: Getting started

1. **Explain the main structure of this code and the advantages and disadvantages of the setup of the distributed algorithm.**

This code contains 6 classes, Branch, MergeSort, NetworkUtils, Node and Sorter. Sorter that is a node and has numerous methods that including initialize, peek, remove and error. We can have multiple sorter nodes. In the main it has a run method that is related to the node class. The sorter gets an array of elements sent over and will sort these in the priority queue and send the first element back or remove one of the elements.The node class contains the run method and can call different methods. It also sets up a server that accepts a connection. Gets a call and closes that connection. The branch class gets a bigger array and splits into 2 arrays and sends to sorter where they will sort them. Branch will get the smallest elements and compare them. Branch will communicate with sorter to sort list. MergeSort calls the Test method which defines an array and will then send this array to a host and port which will be our branch. The advantages of the set of the distributed algorithm is that using the parallel processes can lead to faster results and improve system performance while reducing redundancies but on the flip side, merge sort is nlogn and reading and writing the data has overhead. This may be a situation where it may be better to use one node. Also we can run into issues if nodes fail with sorting.

2. **Run the code with different arrays to sort (different sizes, numbers etc.) and include code to measure the time (you can just enter start and end times). In your Readme describe your experiments and your analyzes of them. E.g. why is the result as it is? Does the distribution help? Why, why not? See this as setting up your own experiment and give me a good description and evaluation.**

Initially when I ran the code as is I received a build measurement of 3 seconds. I decided to first extend the length of the array by doubling it's size to 28 integers. After running the build I got the same time measurement. Next, I tried increasing the number range. The lowest 1, and the highest 10000 and put the highest at the beginning or the unsorted array and the lowest at the end. My speed increased to 722ms. Next, I duplicated the array 4 times with 56 integers repeating. My execution speed lowered slightly to 760 ms. I then reduced the amount of original integers the first original 5 integers and got the fastest execution at 693 ms. The distributed algorithm seems to have some value as you increase the number of integers in the array but it was hard to find concrete trends in the experiments or meaningful differences. I believe this algorithm due to its time complexity may not hold much value as a distributed system as the overhead from the message passing may make this algorithm as a distributed algorithm not worth it.

3. Experiment with the "tree" setup, what happens with more or less nodes when sorting the same array and different arrays? When does the distribution make things better? Does it ever make things faster? As in the previous step experiment and describe your

experiment and your results in detail.

After adding two more nodes I received an execution time of 4s. When doubling and tripling the size of the array I got around the same execution time at 4s. When I reduced the nodes I got a quicker execution time at 3s or less. It appeared from my experiments that the more distribution, the higher the execution time while the size of the array didn't seem to have much effect on the execution time. Distribution does not seem to make things much faster from my experiments.

4. Explain the traffic that you see on Wireshark. How much traffic is generated with this setup and do you see a way to reduce it?

When examining the traffic on Wireshark I see a large amount of TCP traffic for each node running as well as branch and starter. It is a large amount of traffic back and forth between all of the distributed players. One way to reduce the traffic would be to filter and examine the TCP traffic one port at a time.

Task 2: Running it outside of localhost

1. Do you expect changes in runtimes? Why, why not?

I do expect changes in runtimes due to the communication overhead outside the localhost to the AWS server.

2. Do you see a difference how long it takes to sort the arrays? Explain the differences (or why there are not differences)

It took slightly longer I am assuming due to the communication overhead. There was approx 2 second difference from running locally after running.

Task 3: How to improve

1. **Where is the most time lost and could you make this more efficient?**

It seems like the most time lost is due to communication among the various nodes in the system. I believe a way to make this are efficient would be by reducing the nodes to for sorting as merge sort is already Onlogn and likely is not worth it to split up the work.

2. Does it make sense to run the algorithm as a distributed algorithm? Why or why not?

I believe it doesn't make sense to run merge sort as a distributed algorithm as it is Onlogn. In the end, the cost of message passing and communication overhead outweighs the benefits. This would make more sense for a sorting also such as section sort which is On^2.