NWEN303 Project 2 – Distributed Shaker Sort

Introduction

This report documents Project 2 in NWEN303 where the Shaker Sort (aka Cocktail Sort) algorithm was implemented to sort a list of Integers over a distributed system using the Java programming language.

Design

The solution to the shaker sort algorithm works by the master accepting worker node connections before the sort is run. It sets up the worker nodes by passing the workers right neighbours address and port number to the worker so the worker can connect to the right and then the left will connect to them. Each node will now have a connection to their neighbours, depending on if they are a left, centre or right node.

The shaker sort algorithm starts by the left node sending its highest value to its right. The right node will receive this, send its value at position 0 back, compare the received value with position 0 in its list of values and replace that value with the received value if the received value is higher. The sending node receives the position 0 value from the right and replaces its highest position value with the received value if it is lower. This process steps up the chain of worker threads until it gets to the last one and a similar process occurs going back down the chain.

These processes occur until each workers list is sorted. A workers list is sorted when the functions bubbleUp() and bubbleDown() don't move any values in the list. These functions are used when passing maximum values up to the right or minimum values down to the left.

A problem while developing this solution was each worker node had to be ready to communicate to its neighbours before any nodes could start preforming the shaker sort algorithm. Otherwise socket exceptions would occur because the connection might have not been accepted yet by the neighbour. Certain Boolean values were passed between the master and the workers as flags to tell the master that the worker was ready for the next part in setting up. The worker then would block on its inputstream until a Boolean would be passed back from the master when all nodes were ready and the worker could continue to the next setup phase. This process also had to be implemented when the workers list was sorted because the worker still had to run the algorithm until the other nodes had finished sorting.

Testing

Tested with all nodes localhost.

```
Port number: 5000
Amount of nodes connecting?: 5
Listening for node connections...
Node: 1 connected
Node: 2 connected
Node: 3 connected
Node: 4 connected
Node: 5 connected
Size of List?: 35
Maximum value?: 100
Generating List.....
Unsorted list: [85, 18, 29, 69, 94, 63, 41, 39, 22, 53, 33, 97, 13, 77, 16, 11, 45, 90, 47, 79, 73, 44, 63,
38, 19, 35, 17, 47, 75, 0, 68, 34, 45, 44, 79]
Sorted List: [0, 11, 13, 16, 17, 18, 19, 22, 29, 33, 34, 35, 38, 39, 41, 44, 44, 45, 45, 47, 47, 53, 63, 63,
68, 69, 73, 75, 77, 79, 79, 85, 90, 94, 97]
* * * Statistics * * *
Number of elements: 35
Number of worker nodes: 5
Time taken for concurrent shaker sort: 24.0 msec
Average time per element: 0.6857142857142857 msec
```

Tested with nodes on different machines.

```
Port number: 5000
Amount of nodes connecting?: 5
Listening for node connections...
Node: 1 connected
Node: 2 connected
Node: 3 connected
Node: 4 connected
Node: 5 connected
Size of List?: 35
Maximum value?: 100
Generating List.....
Unsorted list: [62, 59, 25, 51, 98, 72, 42, 39, 91, 14, 20, 8, 43, 45, 66, 97, 96, 87, 46, 34, 13, 5, 46, 33,
7, 80, 26, 62, 65, 44, 17, 54, 99, 39, 12]
Sorted List: [5, 7, 8, 12, 13, 14, 17, 20, 25, 26, 33, 34, 39, 39, 42, 43, 44, 45, 46, 46, 51, 54, 59, 62, 62,
65, 66, 72, 80, 87, 91, 96, 97, 98, 99]
* * * Statistics * * *
Number of elements: 35
Number of worker nodes: 5
Time taken for concurrent shaker sort: 845.0 msec
Average time per element: 24.142857142857142 msec
```

How to run

Source code is included, import into eclipse to view or run locally.

Otherwise the Easiest way to run program is run the jar file to start the master then the workers.

To do this:

In a shell run **java –cp sort.jar Connection** to start the master.
Enter port number for workers to connect to, then the number of workers nodes that you will start.
The master will now wait until the nodes are all connected.

Once all nodes are connected the master will ask for a list size which is the total size of a list to be sorted.
Then it will ask for a maximum number, random numbers between 0 and that number will generated and a list will be made.

The algorithm will run and the sorted list and statistics printed.

To start a node from any machine.
In a shell run **java –cp sort.jar Node**
Enter the Master Node Address, eg localhost or 127.0.0.1 if master is on local machine or the ip or domain name eg gotham.ecs.vuw.ac.nz if the master is on a forign machine.
Enter the port the master is on.
Enter the Serversocket other nodes will connect to this node by.
The node is now connected if all the values are right.