DISTRIBUTED COMPUTING
Programming Assignment 1: Implementing Vector Clocks
Submitted by: Malsawmsanga Sailo (CS23MTECH11010)

Objective:

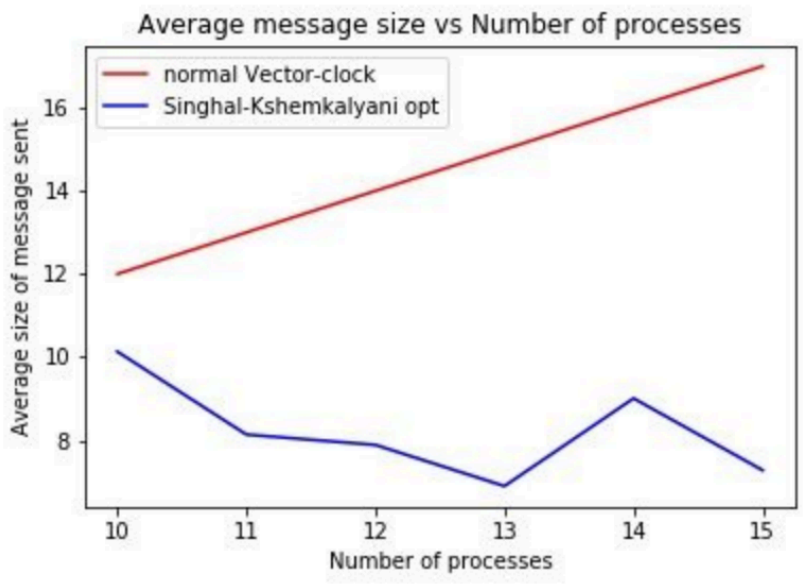
To implement Vector-clocks and Singhal-Kshemkalyani optimization on a Distributed System using C++.

Note: In this assignment, I used sockets instead of ZeroMQ or MPI.

Implementation:

1. The text file provided "inp-params.txt" is parsed and then converted to a matrix.
2. 'n' number of threads are used to simulate 'n' number of different processes.
3. For every thread that simulates a process, another two threads are created. One thread to simulate internal events and message send, another thread to listen for incoming messages from other processes.
4. TCP socket is used to communicate between different processes. Each process has their own socket port assigned to them, depending on their process id.
5. The message number and the sender process is piggybacked in every message sent to other processes.

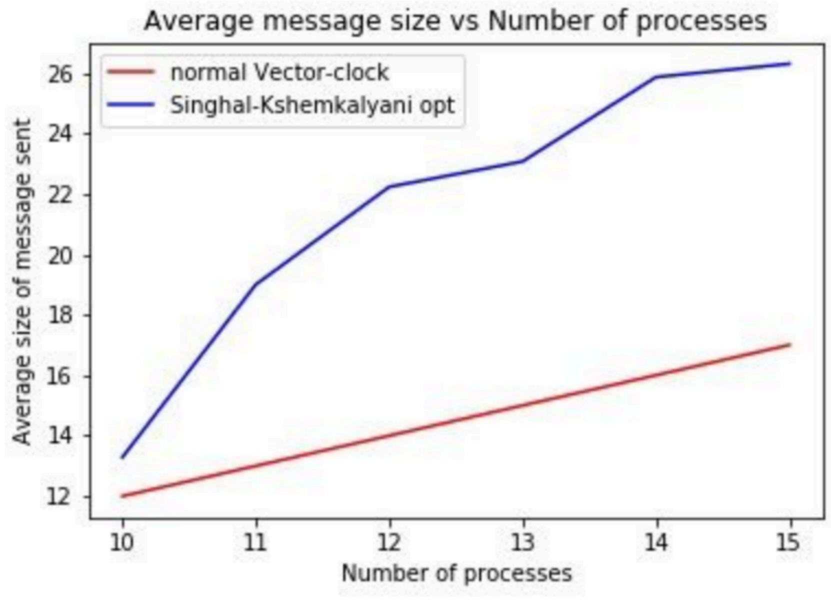
Graphs and Analysis:



For the above graph, a sparse graph is used.

In a graph with lesser nodes, the difference in bandwidth used between Vector-clock and Singhal-Kshemkalyani optimization is low. But, as we increase the number of processes, the bandwidth used to exchange messages between processes is lower in Singhal-Kshemkalyani compared to Vector-clock.

The decrease in bandwidth used from Vector-clock to Singhal-Khsemkalyani also has another impact on the performance of the assignment program. I have observed that for the same input graph, Singhal-Kshemkalyani takes less time to execute the program than Vector-clock. The time difference increases as the number of processes increases, which is likely the effect of the bandwidth being utilized to send only necessary information to other processes, while the Vector-clock algorithm sends ‘n’ number of entries.



The graph shown above uses a fully connected graph topology. This is used to observe the worst case possible.

In Singhal-Kshemkalyani, the optimization works well when there are relatively less updates made to the vector clock since the last sent of the process that it is sending messages from. If all the values in the vector clock change from its last sent, then it needs to send twice the number of entries, since it sends both the tuple index and its value. The bandwidth used in this case would be greater than the Vector-clock algorithm, since Vector-clock sends ‘n’ number of entries regardless of the graph topology.

Error:

- When running the output file (vc/sk) multiple times, we need to give a sufficient amount of time, if not it returns error_no:98.