Distributed Computing: Autumn 2018

Programming Assignment 1: Implementing Vector Clocks

SAKSHAM MITTAL

CS16BTECH11032

25th August, 2018

GOAL

The goal of this assignment is to implement vector-clocks and Singhal-Kshemkalyani optimization on a Distributed System. Implement both these locking algorithms in C++. Then, you have to compare compare the overheads incurred with message stored and exchanged.

CODE DETAILS

The input is read from the file 'inp-params.txt'. After that n threads are created, one for each node of the graph.

Each thread calls the manage() function, in which 2 thread: sender and receiver are created. This is done because each node can send or receive (or do both) messages.

The sender thread makes a list of all threads to which it has to communicate using the adjacency list and tries to establish connection. After establishing connection with all the threads it wants to, it can now send messages as and when required.

The receiver thread on the other hand, keeps on listening on its port number for any possible connection request. If a connection is established, it keeps on reading on that socket till a message is received.

So, in a broader picture, all connections are made before any events(internal or send, receive) happen. Both the sender and receiver threads of all nodes keep a sockets array(for all the socket descriptors from which message can be sent or received).

Also, the it is ensured deadlock will not happen. This is done by using mutex locks, which lock the critical section code(like both sender and receiver trying to update the vector clock) while changing variables which multiple threads may use.

After all events have happened, all the sockets are closed and the threads exit.

SAMPLE INPUT/OUTPUT

The code VC-CS16BTECH11032.cpp was run on the following input:

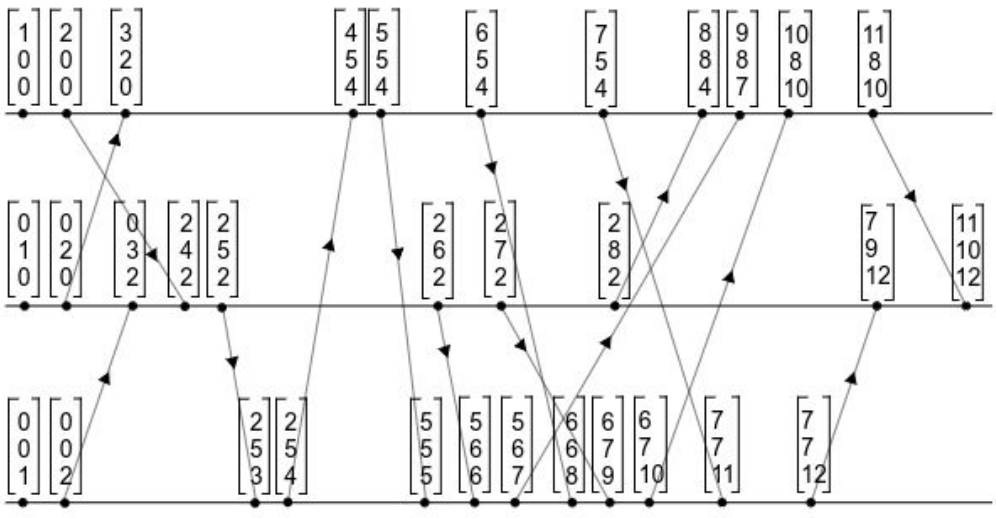
3 500 0.2 5

1 2 3

2 1 3

3 2 1

The following output was found:



```
Process2 excecutes internal event e21 at 5:44, vc: [0 1 0]
Process3 excecutes internal event e31 at 5:44, vc: [0 0 1]
Process1 excecutes internal event e11 at 5:44, vc: [1 0 0]
Process2 sends message m21 to process1 at 5:46, vc: [0 2 0]
Process3 sends message m31 to process2 at 5:46, vc: [0 0 2]
Process1 sends message m11 to process2 at 5:46, vc: [2 0 0]
Process1 receives m21 from process2 at 5:46, vc: [3 2 0]
Process2 receives m31 from process3 at 5:46, vc: [0 3 2]
Process2 receives m11 from process1 at 5:46, vc: [2 4 2]
Process2 sends message m22 to process3 at 5:48, vc: [2 5 2]
Process3 receives m22 from process2 at 5:48, vc: [2 5 3]
Process3 sends message m32 to process1 at 5:48, vc: [2 5 4]
Process1 receives m32 from process3 at 5:48, vc: [4 5 4]
Process1 sends message m12 to process3 at 5:48, vc: [5 5 4]
Process3 receives m12 from process1 at 5:48, vc: [5 5 5]
Process2 sends message m23 to process3 at 5:50, vc: [2 6 2]
Process3 receives m23 from process2 at 5:50, vc: [5 6 6]
Process3 sends message m33 to process1 at 5:50, vc: [5 6 7]
Process1 sends message m13 to process3 at 5:50, vc: [6 5 4]
Process3 receives m13 from process1 at 5:50, vc: [6 6 8]
Process2 sends message m24 to process3 at 5:52, vc: [2 7 2]
Process3 receives m24 from process2 at 5:52, vc: [6 7 9]
Process3 sends message m34 to process1 at 5:52, vc: [6 7 10]
Process1 sends message m14 to process3 at 5:52, vc: [7 5 4]
Process3 receives m14 from process1 at 5:52, vc: [7 7 11]
Process2 sends message m25 to process1 at 5:54, vc: [2 8 2]
Process1 receives m25 from process2 at 5:54, vc: [8 8 4]
Process1 receives m33 from process3 at 5:54, vc: [9 8 7]
Process1 receives m34 from process3 at 5:54, vc: [10 8 10]
Process3 sends message m35 to process2 at 5:54, vc: [7 7 12]
Process2 receives m35 from process3 at 5:54, vc: [7 9 12]
Process1 sends message m15 to process2 at 5:54, vc: [11 8 10]
Process2 receives m15 from process1 at 5:54, vc: [11 10 12]
```

GRAPH

The following graph was plotted using the following variables :

N = 10 to 15 with increment of 1

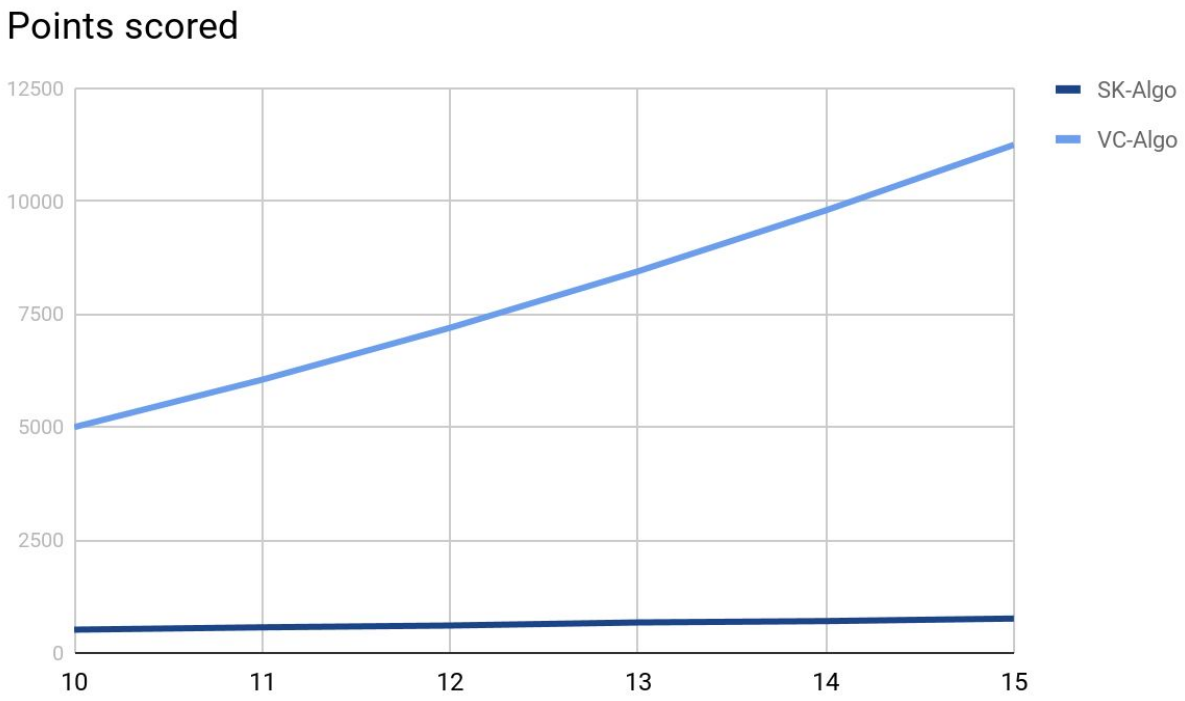
λ = 20

α = 1.5

m = 50

n	SK-Algo	VC-Algo
10	516	5000
10	568	6050
10	608	7200
13	676	8450
14	706	9800
15	762	11250

Exact data points



ANALYSIS OF GRAPH

The graph clearly shows that Singhal–Kshemkalyani's differential technique is more efficient in terms of usage of space while sending messages between processes.

The difference is more evident when the number of processes becomes large(and hence the number of threads) as between successive message sends to the same process, only a

few entries of the vector clock at the sender process are likely to change. This becomes more likely when the number of processes is large because only a few of them will interact frequently by passing messages.