

Distributed Computing

Assignment 1: Implementation of Vector Clock

Implementation details

The inputs are read from the file 'inp-params.txt'. The topology of graph is created.

For each nodes in the graph, separate thread is created which will call fun() function, in which 2 thread: sender and receiver are created. This is done because each node can send or receive messages.

The receiver thread for each node, keeps on listening on its port no. for any incoming connection request from the nodes it is connected to in the graph.

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

Both the sender and receiver threads of all nodes keep a socket array (for all the socket descriptors from which message can be sent or received).

To ensure deadlock not to happen, mutex lock are used which will lock the critical sections like when we are updating vector clocks and writing on log file to avoid any preemption.

In Singhal-Kshemkalyani optimization code, LS and LU vectors are used in addition with vector clock which guide us to send only the tuples, so that whole vector clock will not be sent and message will be smaller in size.

In both code basically the sender thread by using random function decides that it want to execute internal event or message sent event, if it's an internal event it updates its logical clock and if it is a sent event, it piggybacks it's vector clock in the message and in In Singhal-Kshemkalyani code ,it sends only those entries that differs since last sent. Receiver thread collects the message, updates it's vector clock.

After all events happens, all sockets are closed and thread exits.

Space Used:

Space utilized for storing the vector clock by each process is $N * \text{sizeof}(\text{int})$ bytes in normal vector clock implementation whereas Singhal-Kshemkalyani is using $3 * N * (\text{sizeof})$ bytes ie N for vector clock and LS and LU vectors taking each $N * \text{sizeof}(\text{int})$ bytes.

Space utilized for sending message for normal vector clock always is constant ie. We are sending all the N entries each time but Singhal-Kshemkalyani space used is much less as only updated entries are to be sent which can be seen in graph later in this report.

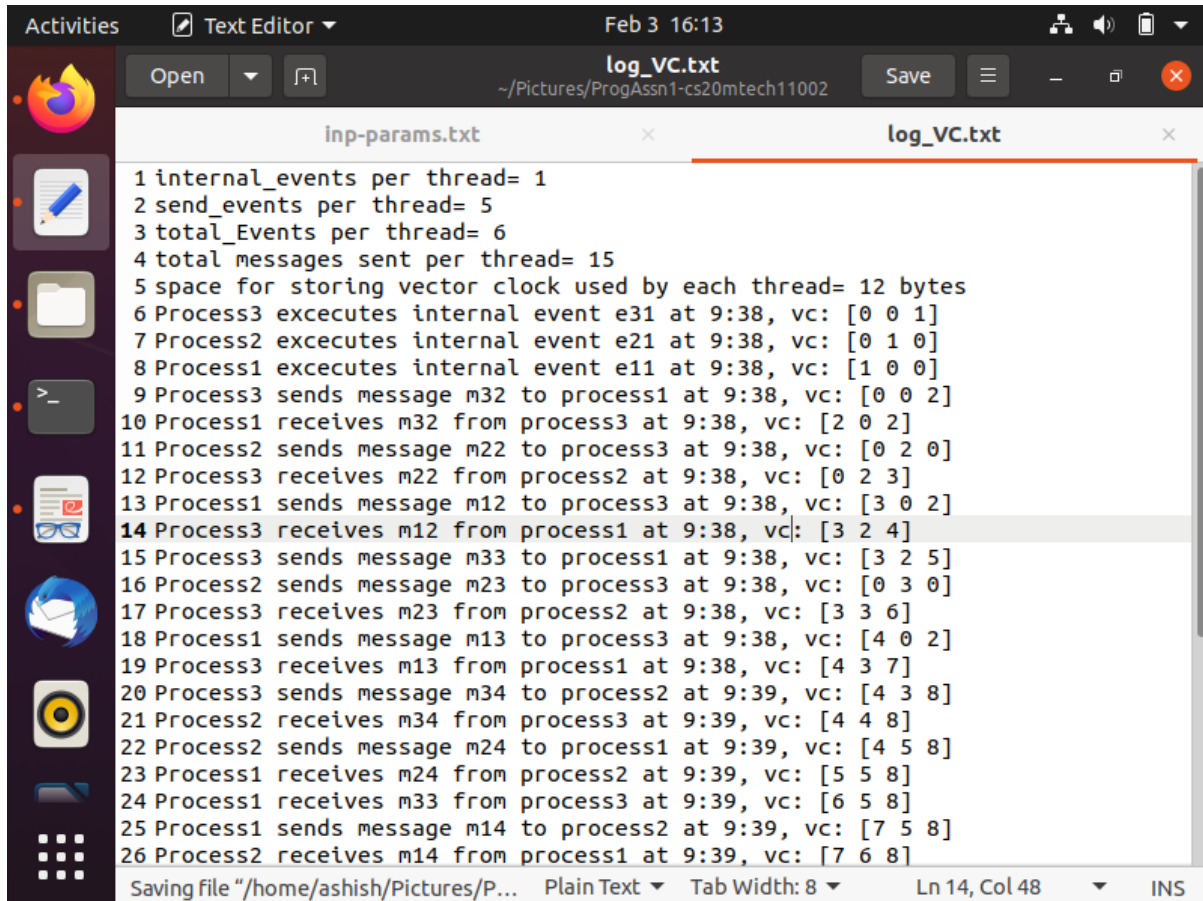
Sample Input/OutPut

3 400 0.2 5

1 2 3

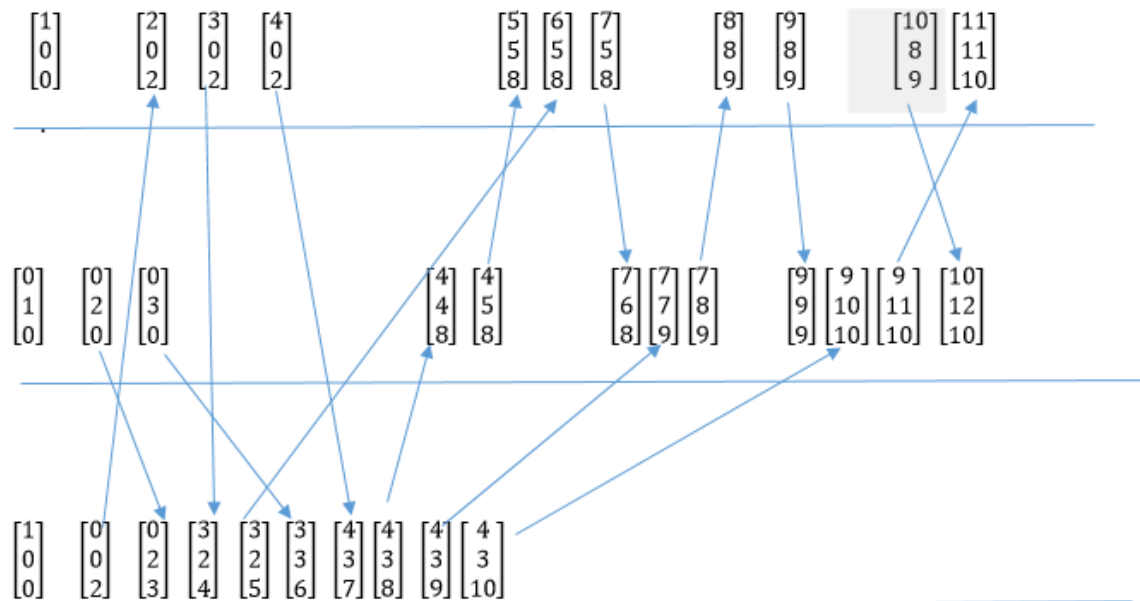
2 1 3

3 2 1



```
Activities Text Editor Feb 3 16:13
log_VC.txt
~/Pictures/ProgAssn1-cs20mtech11002 Save
inp-params.txt log_VC.txt
1 internal_events per thread= 1
2 send_events per thread= 5
3 total_Events per thread= 6
4 total messages sent per thread= 15
5 space for storing vector clock used by each thread= 12 bytes
6 Process3 executes internal event e31 at 9:38, vc: [0 0 1]
7 Process2 executes internal event e21 at 9:38, vc: [0 1 0]
8 Process1 executes internal event e11 at 9:38, vc: [1 0 0]
9 Process3 sends message m32 to process1 at 9:38, vc: [0 0 2]
10 Process1 receives m32 from process3 at 9:38, vc: [2 0 2]
11 Process2 sends message m22 to process3 at 9:38, vc: [0 2 0]
12 Process3 receives m22 from process2 at 9:38, vc: [0 2 3]
13 Process1 sends message m12 to process3 at 9:38, vc: [3 0 2]
14 Process3 receives m12 from process1 at 9:38, vc: [3 2 4]
15 Process3 sends message m33 to process1 at 9:38, vc: [3 2 5]
16 Process2 sends message m23 to process3 at 9:38, vc: [0 3 0]
17 Process3 receives m23 from process2 at 9:38, vc: [3 3 6]
18 Process1 sends message m13 to process3 at 9:38, vc: [4 0 2]
19 Process3 receives m13 from process1 at 9:38, vc: [4 3 7]
20 Process3 sends message m34 to process2 at 9:39, vc: [4 3 8]
21 Process2 receives m34 from process3 at 9:39, vc: [4 4 8]
22 Process2 sends message m24 to process1 at 9:39, vc: [4 5 8]
23 Process1 receives m24 from process2 at 9:39, vc: [5 5 8]
24 Process1 receives m33 from process3 at 9:39, vc: [6 5 8]
25 Process1 sends message m14 to process2 at 9:39, vc: [7 5 8]
26 Process2 receives m14 from process1 at 9:39, vc: [7 6 8]
Saving file "/home/ashish/Pictures/P... Plain Text Tab Width: 8 Ln 14, Col 48 INS
```

All execution events in all nodes—



Graph

The following graph was plotted using the following variables :

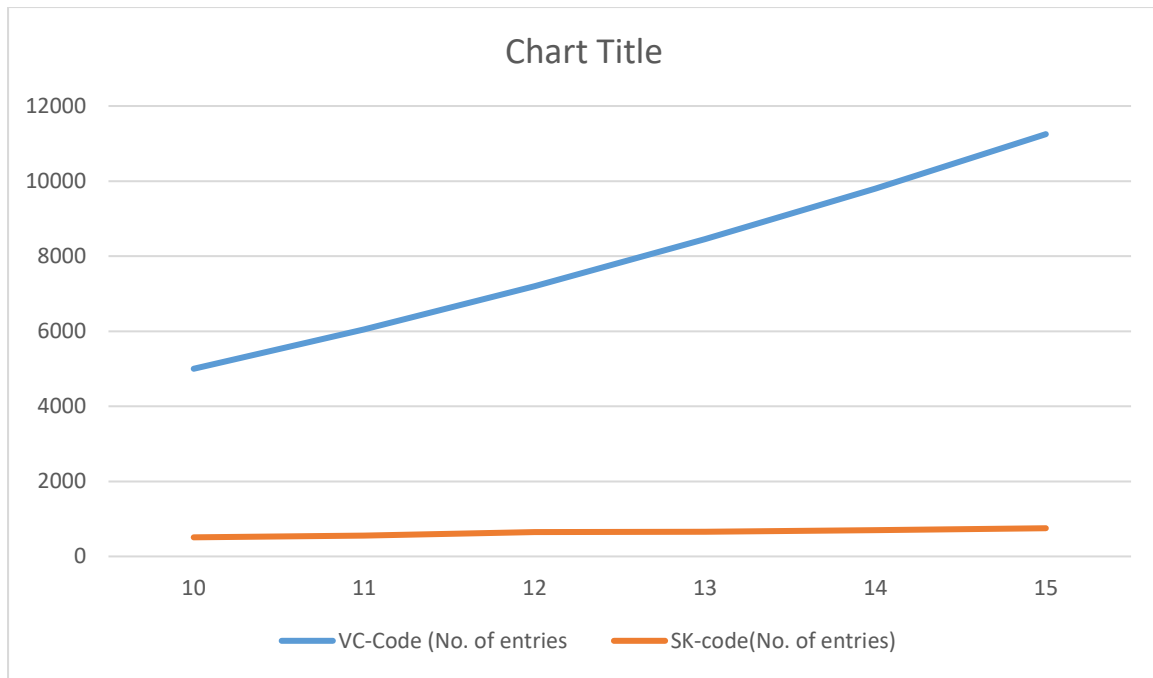
$N = 10$ to 15 with increment of 1

$\lambda = 20$

$\alpha = 1.5$

$m = 50$

No.	VC-Code (No. of entries)	SK-code(No. of entries)
10	5000	509
11	6050	551
12	7200	648
13	8450	656
14	9800	703
15	11250	751



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

As we can see in the graph that as we are incrementing the nodes, the difference between the VC and SK technique is increasing because in case of large nodes, very few nodes will be interacting frequently which results in very few entries are likely to be changed in successive message exchange between the 2 processes.