Distributed Computing: Autumn 2018

Programming Assignment 1: Implementing Vector Clocks

Submission Date: ~~23rd August 2018~~ 25th August, 9:00 pm

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.

Details. You are given as input a system of n nodes (processes) connected to each other in the form of a graph topology. These nodes communicate with their neighbors (in the graph) through messages. You can implement these nodes as processes/threads communicating through sockets.

As studied in the class, each process executes three events: internal, message send & message receive. To simulate these events you can assume the following: each process creates and executes internal and message send events with a delay that is exponentially distributed with inter-event time λ ms. Assume that the ratio of internal to message send events on each process is α (which for instance can be 1.5). The application terminates when each processes has sent a total of m messages.

In this setting, implement the normal vector-clocks and Singhal-Kshemkalyani's optimization of vector-clocks. Then demonstrate the savings in message communication obtained by using the optimization.

Input: The input to the program will be a file, named inp-params.txt, consisting of all the parameters described above and the graph topology. The first line of the input file will contain the parameters :n, λ, α, m. For example, $n = 15, \lambda = 5, \alpha = 1.5, m = 40$

The second line onwards, will contain the graph topology in the form of an adjacency list. For instance if n is 3, a sample topology showing a complete graph is as follows:

1 2 3

2 1 3

3 1 2

Output: Your program output should demonstrate the strong consistency property of vector-clocks: If two events x and y have timestamps vh and vk, respectively, then

$$x \rightarrow y \Leftrightarrow vh < vk \tag{1}$$

$$x \parallel y \Leftrightarrow vh \parallel vk \tag{2}$$

To demonstrate this, you have to output the contents of all the events onto a common logfile with two time-stamps: real time-stamps and vector time-stamps. Since you will be executing all these programs on the same machine or ina LAN environment like IITH, you can assume that the clocks of the machines are synchronized.

The contents of the logfile should be as follows for both the algorithms:
Process1 executes internal event e11 at 10:00, vc: [1 0 0 0]
Process2 executes internal event e21 at 10:01, vc: [0 1 0 0]
Process3 sends message m31 to process2 at 10:02, vc: [0 0 1 0]
Process3 executes internal event e32 at 10:03, vc: [0 0 2 0]
Process2 executes internal event e22 at 10:04, vc: [0 2 0 0]
Process2 receives m31 from process3 at 10:05, vc: [0 3 1 0]

.
.
.

The output should demonstrate Eqn(1) and Eqn(2). **In addition to this, you have to output space utilized by each process for storing the vector clocks and sending messages by both the alorithms.**

Report: You have to submit a report for this assignment. As mentioned earlier, this report should contain a comparison of the performance of vector-clocks and Singhal-Kshemkalyani optimization. You must run both these algorithms multiple times to compare the performances and display the result in form of a graph.

You run both these algorithms varying the number of processes from 10 to 15 in the increments of 1 while keeping other parameters same. You can assume any topology for creating the graph with each of these processes. Assume that m is 50 for all the executions.

You measure the average number of vector-clock entries sent in each message. Clearly this number is going to be fixed for the normal vector-clock. But for Singhal-Kshemkalyani optimization, this number should be lesser.

The graph in the report will be as follows: the x-axis will vary the number of threads from 10 to 15 in the increments of 1 (as explained above) while the y-axis will show the average number of entries in each message sent. Finally, you must also give an analysis of the results while explaining any anomalies observed.

Deliverables: You have to submit the following:

- The source files containing the actual program to execute. The normal vector clock implementation should be named as VC-<rollno>.cpp and Singhal Kshemkalyani's optimization as SK-<rollno>.cpp.
- A readme.txt that explains how to execute the program.
- The report as explained above.

Zip all the files and name it as ProgAssn1-<rollno>.zip. Please follow the naming convention strictly. Otherwise, your submission will not be evaluated. Then upload the zip on the google classroom page of this course. Submit it by ~~**23rd August 2018**~~ **25th August, 9:00 pm**.