CS5320: Distributed Computing, Spring 2020

Programming Assignment 2: Termination Detection

Submission Deadline: 7th March 2020, 21:00 hrs

1 Problem Statement

Consider the following hypothetical scenario: Let there be a collection of a certain number of healthy cells (processes) in a cell culture. The healthy cells are all initially white in colour. At a particular instant, a non-empty subset of cells are chosen at random and infected with a certain infectious virus. The cells infected with virus become red in colour. The red cells are capable of infecting other healthy cells, and a single red cell can infect all its healthy neighbours at a single instance. Another non-empty subset of cells are chosen at random after a certain time gap, which is administered with an antidote to the virus. The antidote is assumed to be very weak and hence it does not remain effective for a long period of time. The cells containing the antidote are blue in colour. Upon activation, a red cell can turn white or blue cells to red, while a blue cell can turn both red and white to blue. Assuming the rate of activation of red cells to be less than blue, we have to determine when the cell culture is completely free of the virus, i.e. all cells turn blue.

This assignment asks you to model this as a distributed graph problem and detect termination in the system. Implement the following algorithms (1) A&M section 7.5, A spanning-tree-based termination detection algorithm and (2) The Message-optimal termination detection explained in A&M section 7.6 or the one discussed in the class based on Dijkstra & Scholten.

The above problem needs to be implemented in C++, using **sockets** or **MPI** for communication between the cells/processes.

2 Model Specifications

You are given as input a system of N nodes (processes) i.e cells, connected to each other in the form of a connected graph topology. Each node communicates with its neighbour nodes (in the graph) through messages. You can implement these nodes as processes/threads communicating through sockets or MPI. The following rules are to be observed while the game is played out:

1. All cells are initially white. A white cell remains idle in the system and has no power to infect or cure anyone else.
2. As the computation proceeds a subset, S_r, of the white cells will be chosen at random to be changed to red after a time that is exponentially distributed with an average W_r.
3. The size of the subset, $|S_r| = I_r$, chosen cannot be more than 10% of the available pool of cells, i.e. $I_r \leq (0.1 * N)$.
4. A red cell infects its neighbors by sending messages. It sends (red) messages to its neighbors at regular intervals. Each time it sends messages, it sends them to a p% percentage of cells of its neighbors and changes their to color to red.

5. The red cells send successive red messages to its neighbors with a exponential delay λ_{red}.

6. Sometime after red cells have been introduced in the system a subset, S_b of the cells (white/red) are chosen at random to be changed to blue. This waiting time is exponential in nature with an average W_b.

7. A blue cell has the ability to cures its neighboring red cells by sending them blue messages. A red cell on receiving blue messages turns into blue. Similarly a white cell on a receiving blue message turn into blue as well.

8. The size of the subset, $|S_b| = I_b$, chosen cannot be more than 10% of the total number of available cells, i.e. $I_b \leq (0.1 * N)$. And $I_r \leq I_b$.

9. Like red cells, blue cells too send messages to their neighbors at regular intervals. Each time it sends messages, it sends them to a q% percentage of cells of its neighbors and change their color to blue. Here $p < q$.

10. The blue cells send blue messages to its neighbors with a exponential delay λ_{blue}. Note that $\lambda_{red} > \lambda_{blue}$

11. As mentioned above, a blue process on receiving a red message becomes red in color. This captures the concept idea that the antidote is assumed to be weak. Hence it does not remain effective for a long period of time.

12. We say that the system has terminated when all the cells are blue in colour. You can assume that the TD algorithm was executing from the start of the computation.

13. In this scenario, a red cell is an active process and a blue cell is a passive process. It can be seen that a passive process can repeatedly become active and vice versa.

14. Also note that in the traditional termination detection scenario, once a process has become passive, it does not send any more message. But here, a blue process continues to send red messages. So detecting termination here implies detecting absence of red processes and messages.

The above points are illustrated in the Figure 1.

3 Distributed System Model

Due to practical constraints, you have to simulate distributed system on a single machine. You can consider the distributed system consisting of multiple processes (running on the same machine). Each process will in turn consist of two threads simultaneously. The first thread will take care of internal computations and sending events. The second thread will handle the receiving of messages. Since receives can occur asynchronously, we need a separate thread to handle them. The send event signifies the sending of coloured messages to other cells and the receive event is the change of state of a cell upon receiving a certain coloured message. The send events will be at the control of the user, whereas the receive will execute in any concurrent order. Please refer to the send and receive specifications in the previous assignment for more details.
The following is an outline for the send and receive events:

```
1
2
3  /* Processes use send and receive functions described below to send and receive messages
       respectively. */
4  void send(destination_id)
5  {
6
7   /* Sleep with exponential average of λ_snd
8   This sleep simulates some work done by the sender process.
9   */
```

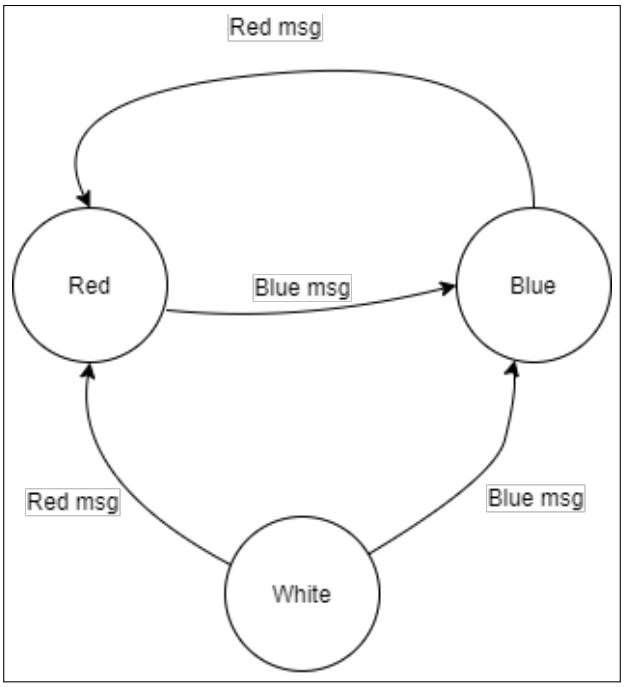

Figure 1: Activation Rules of the game

```
10   sndSleepTime = exp_rand(λ_snd);
11   sleep(sndSleepTime);
12
13   network_send(destination_id);
14  }
15
16  void receive(sender_id)
17  {
18   network_receive(sender_id);
19   ...
20   // computations
21
22  }
```

Pseudocode 1: Outline for send/receive events

4 Input

The input to the program will be a file, named inp-params.txt, consisting of the following parameters and the graph topology. The first line of the input file will contain: $N\ W_r\ I_r\ W_b\ I_b\ \lambda_{blue}\ \lambda_{red}\ \lambda_{snd}\ p\ q$ where

N - Total number of processes or cells.
W_r - average of exponential wait time before introduction of red cells into the system.
I_r - number of red cells introduced into the system.
W_b - average of exponential wait time before introduction of blue cells into the system after the introduction of red cells.
I_b - number of red cells introduced into the system.
λ_{blue} - exponential delay between successive blue messages that a blue cell sends to its neighbours upon activation.

λ_{red} - exponential delay between successive blue messages that a blue cell sends to its neighbours upon activation.
λ_{snd} - parameter used to simulate exponential delay in message send.
p - percentage of neighbors a red cell can affect
q - percentage of neighbors a blue cell can affect

Please note that you have to choose all the values in such way that the distributed associated with it will have values in milliseconds.
Suppose $N = 100,\ W_r = 2,\ I_r = 3,\ W_b = 2,\ I_b = 5,\ \lambda_{red} = 2,\ \lambda_{blue} = 0.5,\ p = 0.2,\ q = 0.8$. Then first line of the input will be as follows:
100 2 3 2 5 0.5 1.5 0.2 0.8

From the second line onwards, the input should contain details of a graph in adjacency list format. Sample input graph is as follows:

```
1 2 3
2 1 4
3 1
4 2
```

After the adjacency graph representation, the next line can be the description of a spanning tree on the above graph. Both the algorithm described above assume a spanning tree. So you can assume that the spanning tree is given to you as input. Further, you can assume that the root of is the coordinator that detects termination.

5 Output

Your program output should demonstrate the spreading of the virus and the ultimate termination. To demonstrate this, you have to output the contents of all the events onto a common output log file, out-log.txt.

```
Cell 2 turns red at 13:00
Cell 7 turns red at 13:01
Cell 5 turns red at 13:01
Cell 7 sends red msg to Cell 1 at 13:02
Cell 1 receives red msg at 13:02
Cell 5 sends red msg to Cell 3 at 13:02
Cell 4 turns blue at 13:03
Cell 1 turns red at 13:03
Cell 3 receives red msg at 13:04
.
.
.
Cell 5 initiates Termination Process at 13:10
.
.
```

Here, all the times mentioned are system time. To show the differences in time accurately, please record them in nanoseconds.

6 Report

You have to submit a report for this assignment explaining the design and the implementation details. You must run both the algorithms, spanning tree and optimal (any variant) multiple times to compute the following plots. Each plot will have two curves.

Plot 1: X-axis will vary the number of cells playing the game, i.e. the value of N. Take N to be from 10 to 30, with increment of 5. Y-axis will be the number of messages sent and received for detecting termination of the entire system. Here please measure only the control messages, i.e. the messages sent the termination detection algorithm

Plot 2: X-axis will have the same specifications as in the previous graph. Y-axis will be the time in nanoseconds for detecting termination of the entire system, measured from the instance when the first blue cell was introduced.

7 Deliverables

You have to submit the following:

1. The source file containing the actual program to execute i.e Virus-⟨RollNumber⟩.cpp
2. A readme.txt that explains how to execute the program.
3. The report as explained above.

Zip all these files and name them as: Virus-Assgn2-⟨RollNumber⟩.zip. If you don't follow the naming convention, then your assignment will not be evaluated.

8 Evaluation

The TAs will use the following evaluation policy:

1. Design: 40%
2. Execution: 50%
3. Indentation and Documentation (with comments): 10%

Late Submission Penalty:
For each day after the deadline, your submission will be penalized by 10% of the total marks.

9 Plagiarism Policy

Please go through the following link to get acquainted with the plagiarism policy of the CSE Department:
https://cse.iith.ac.in/academics/plagiarism-policy.html