CS5320: Distributed Computing, Spring 2020

Programming Assignment 1: External Clock Synchronization

Submission Deadline: ~~09 February, 11th February 2020~~
15th February 2020, 21:00 hrs

1 Problem Statement

The goal of this assignment is to implement clock synchronization using P2P communication on a Distributed System. Implement this in **C++ using sockets**. Our aim is to synchronize the clocks, despite communication delays.

2 Distributed System with Drifting Clocks Details

You are given as input a system of N nodes (processes) i.e time servers connected to each other in the form of a fully connected graph topology. Each node communicates with every other node (in the graph) through messages. You can implement these nodes as processes/threads communicating through sockets.

As studied in the class, two servers exchange a pair of messages between each other and calculate a pair of (x, y). A good approximation of the real offset δ can be obtained from that pair in which the round-trip delay y is the smallest.

To simulate these events you can assume the following: Each server p has a local time t_p , which it can read initially from the system clock. But to make the situation realistic, assume that there is an error factor of ϵ (small randomly generated value which can be negative as well). So the time of the server p is: $t_p + \epsilon$. Our objective is to synchronize these local times as much as possible.

Two servers calculate K pairs of (x, y). Between each successive synchronization round, the servers sleep for a delay that is exponentially distributed with inter-event time λ_p ms. This sleep simulate the server doing some complicated task.

After computation of δ (which can also be negative), the server modifies the error factor of the clock by δ units. Repeating this, we can synchronize the N clocks and the variance of the clocks will keep on reducing.

Distributed Systems Simulation: Due to practical constraints, you have to simulate distributed system on a single machine. You can consider a 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 clock synchronization requests to one of the servers and the receive event is the response from that server. The send events will be at the control of the user, whereas the receive will execute in any concurrent order.

In addition to these threads, additional threads will be used for implementing drifting clocks as explained below.

3 Drifting Clocks

The concept of drifting clocks has been explained above. The clock drift of each process is governed by the value of *driftFactor*, as explained in the snippet below. This variable is local to each process. Only the threads of that process can increment the value of that variable.

```

atomic<double> driftFactor = 0;

class local_clock()
{
 int error_factor;

 /* Invoked by processes (and their threads) to read the current time */
 Time read()
 {
  return( System_Clock() + error_factor + driftFactor );
 }

 // Used by processes to correct their clock
 update( int optimal_delta )
 {
  error_factor = error_factor + optimal_delta;
 }

 // Runs in a separate thread
 int IncrementDriftFactor()
 {
  while(true) {

   clock_drift=exponential_distribution($\lambda_{drift}$);

   /* clock_drift is exponentially distributed wrt $\lambda_{drift}$ and can be positive as well as
       negative
   */

   driftFactor += clock_drift;

   // Sleep to give some break between the drifts
   reqSleepTime = exp_rand($\lambda_{wkDrift}$);
   sleep(reqSleepTime);

  }
 }
};

/*
Process P synchronizes with process Q using the following methods:
synchronize_request() other for synchronize_reply()
*/

synchronize_request(Q)
{
 T1 = read();

 // Process Q replies back T2, T3
 request(T2, T3) from Process Q;

 T4 = read();

 calculate optimal_delta using T1,T2,T3,T4

 update(optimal_delta);

 // Sleep before synchronizing again with exponential average of $\lambda_p$
 reqSleepTime = exp_rand($\lambda_p$);
 sleep(reqSleepTime);
}

// Process Q replying for synchronization requests
synchronize_reply()
{
 // Received sync_request() from Process P
 T2 = read();

 /* Sleep  with exponential average of $\lambda_q$
    This sleep simulates some work done by process q.
 */
 respSleepTime = exp_rand($\lambda_q$);
 sleep(respSleepTime);

 T3 = read();

 // This reply contains values T2,T3
 reply(T2, T3) to Process P

}

/* Processes use send and receive functions described below to send and receive messages
    respectively. */
void send(destination_id)
{

 /* Sleep with exponential average of $\lambda_{snd}$
 This sleep simulates some work done by the sender process.
 */
 sndSleepTime = exp_rand($\lambda_{snd}$);
 sleep(sndSleepTime);

 network_send(destination_id);
}

void receive(sender_id)
{
 network_receive(sender_id);
 ...
 // computations

}
```

Pseudocode 1: Outline for Drifting Clock

Note: For obtaining time with much higher precision(nanoseconds), you can use clock_gettime.

4 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 :$N, K, \lambda_p, \lambda_q, \lambda_{snd}, \lambda_{drift}$.
N - no. of processes or time servers.
K - no of iterations/rounds of modification of error factor between any 2 clocks.
λ_p - parameter for exponential wait between 2 successive synchronization request.
λ_q - parameter for exponential wait between 2 successive synchronization reply.
λ_{snd} - parameter used to simulate exponential delay in message send.
λ_{drift} - parameter for exponential drift of local clock time. $\lambda_{wkDrift}$ - parameter to give some time lag between clock drifts.

Please note that you have to choose λ values in such way that the distributed associated with it will have values in milliseconds.

The input should contain the values of $N, K, \lambda_p, \lambda_q, \lambda_{snd}, \lambda_{drift}$, each value separated by a tab/white space. For instance, if $N = 5, \lambda_p = 2, \lambda_q = 3, \lambda_{snd} = 3, \lambda_{drift} = 2, \lambda_{wkDrift} = 3$, the input is as follows:
5 2 3 3 2 3

5 Output

Your program output should demonstrate the clock synchronization of different time servers and the offsets calculated. To demonstrate this, you have to output the contents of all the events onto a common output log file, out-log.txt.

The contents of output log file consists of two parts. The first part is the log of all the events. Although, in an actual distributed system, the processes don't have access to common clock. But since you are simulating the algorithm on a single machine, the threads have access to a common system clock which can be used for in logging. You need to display the log in a manner as shown below.

```
Server1 requests 1st clock synchronization to Server 2 at 10:00
Server2 receives 1st clock synchronization request from Server 1 at 10:01
Server2 replies 1st clock synchronization response to Server 1 at 10:03
Server1 receives 1st clock synchronization response from Server 2 at 10:04
Computing 1st delta between server1 and server2: ...
.
.
.
```

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

The contents of the second part of the log file is formatted as follows:

	$Round_1$	$Round_2$	$Round_3$	...	$Round_K$
P_1	t_1^1	t_1^2	t_1^3	...	
P_2	t_2^1	t_2^2	...		
P_3					
...	...	...	...	...	...
P_N					
	$mean_1, var_1$	...	...	...	$mean_K, var_K$

Here the notation, t_i^j is process P_i's time after the synchronization in round j. Finally, $mean_x, var_x$ are the mean, variance of the times of all processes at the end of round x with K being the final round.

6 Report

You have to submit a report for this assignment explaining the design and the implementation details. You must run this algorithm multiple times and compute the following graphs.

Graph 1: X-axis will vary the number of processes or time servers varying from 5 to 10 with increments of 1. K will remain fixed, say K=10. y-axis should demonstrate the value of the variance among N clocks after

K synchronizations by all the processes.

Graph 2: X-axis will have the numbers of processes, N=10 as fixed but will vary the value of K from 5 to 10 with the increments of 1. y-axis should demonstrate the value of the variance among N clocks after K synchronizations by all the processes.

7 Deliverables

You have to submit the following:

1. The source file containing the actual program to execute i.e CS-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: CSync-Assgn1-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