Distributed Computing - II: CS5320

Programming Assignment - I

External Clock Synchronization

Assignment Report

Sagar Jain - CS17BTECH11034

February 17, 2020

Contents

Program Design

The following points begin from the ***main*** function, describing the high level ideas of the code:

1. Initially we open the I/O files, read input from the files and allocate memory for the data structures used in the code.

2. We then create **N** threads, each of which simulates one distributed system. This happens in the function ***serverProcess***.

3. In each distributed system we have three threads:

   (a) Receiver Thread

   (b) Sender Thread

   (c) Clock Drift Thread

4. Each server thread first launches threads for receiving and drifting of the clocks.

5. After this the sender thread **waits for all receiver threads to begin listening**, this is done using a while loop checking the value of a variable, which is being updated by **reciver thread in the critical section**.

6. The port number given to the ith server is 50000 + i, to make sending requests easier.

7. The sender process then begins sending requests to all the other ports to synchronise with them.

8. **After one round every sender thread waits for all the other server threads to finish their respective rounds**, so that they can all enter their times into a table at about the same time. This is ensured, again using a **mutex lock** and variable, whose value is being checked by all the sender threads using a while loop.

9. The receiver thread, sets up a receiving socket and then updates a varibale within a lock so that the sender processes can know when all the receiver threads are ready.

10. The receiver thread on receiving a synchronisation request, get the server id from the message and then responds with the values of T2 and T3, back to the port 50000 + requestinServerID.

11. All the printing to the file happens using **fprintf** since it does not have the concurrency issues that cout does.

12. We have a class for the localClock of each thread.

13. We finally use the function writeSyncTableToFile, to print the entire table to the file in the requisite format.

Program Output

The program outputa one filea, **out-log.txt**. The following is an example.

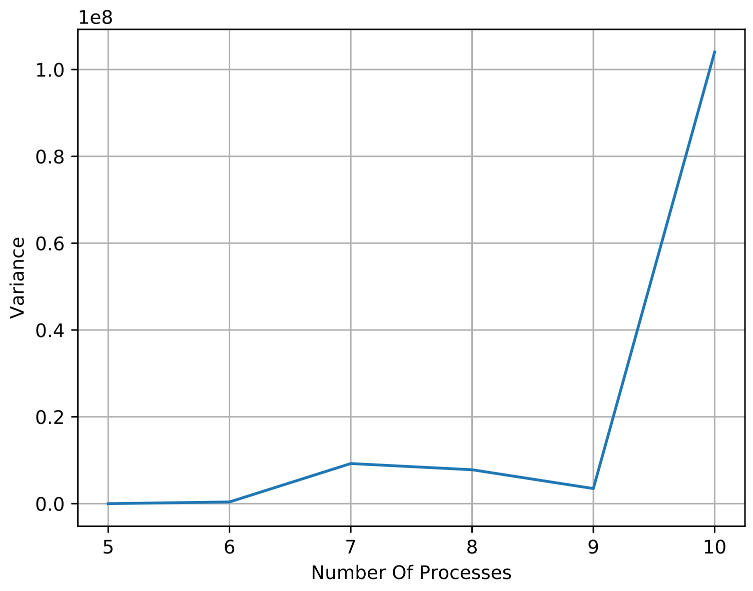
Example Output:

Log File:

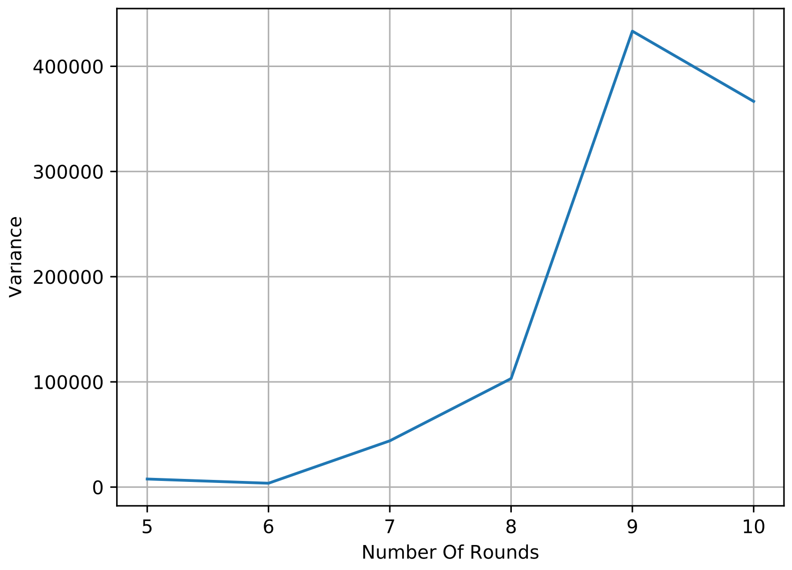
```
Server 0 replies 1th clock synchronisation response to Server 6 at 00:04:22.
Server 1 receives 1th clock synchronisation request from Server 0 at 00:04:22.
Server 0 requests 1th clock synchronisation to Server 1 at 00:04:22.
Server 5 requests 1th clock synchronisation to Server 0 at 00:04:22.
Server 0 receives 2th clock synchronisation request from Server 6 at 00:04:22.
Server 3 requests 1th clock synchronisation to Server 0 at 00:04:22.
Server 4 receives 1th clock synchronisation response from Server 0 at 00:04:22.
Server 0 replies 2th clock synchronisation response to Server 6 at 00:04:22.
```

Results & Graphs

Variance with Constant Rounds



Variance with Constant Processes



Explaination of Results

The following are a few points to note / observations about the above graphs:

1. The first graph shows that with a constant number of rounds the variance is high for a more number of processes. This makes sense, it is due to the fact that while a few processes can be synchronised in less number of rounds, if we would like to synchronise more number of processes we need more rounds of synchronisations.

2. For a constant number of processes we see that initially the variance increases from 0 and then at the end starts to decrease, this happens because of two simple reasons:

   (a) Initially the clocks are actually in-sync as they are all the system clock, so the initial variance is 0, but with increase in time, the drift and unnecessary accounting for round trip times begins to diverge the clocks, so we see the clock variance increasing.

   (b) As the rounds increase, the synchronisation begins to take place so we see that at the end the variance begins to decrease.