CS5320: Distributed Computing, Spring 2020

Programming Assignment 1: External Clock Synchronization

SAI HARSHA KOTTAPALLI

CS17BTECH11036

GOAL

Implementing clock synchronization using P2P communication on a Distributed System. and to synchronize the clocks, despite communication delays.

DESIGN

Let there be n time servers. Their channels are in such a way that the associated graph is a fully connected graph. The communication via channels can be done through sockets.

I introduce an error factor and drift factor, local to each time server so there is an offset between each server. The drift factor would be updated as time goes on continuously while the error factor is something that I can update.

Given, Input: number of servers, number of rounds of synchronization, and sleep duration distribution parameter for drift, request, reply, and channel delay, I then log the local times after each Synchronization.

For each time server, it should be able to send requests to other time servers for synchronization while also being able to respond back with its local time when asked for.

After the calculation of offset(x) and round trip delay(y), Optimal offset is randomly chosen in the range [x - y/2, x + y/2].

Here, x = $\frac{t2 - t1 + t3 - t4}{2}$ and y = $(t4 - t1) - (t3 - t2)$

By synchronizing with each server during each round, the error factor would be updated as some linear combination of other server's timestamp which would hence, make sure the error factor is updated such that I can synchronize the clocks as much as possible.

IMPLEMENTATION

Class **TS:**

This class is used to simulate each local time server. Hence, each time server will store the following:

- variables
    - Total -> Total number of time servers
    - K -> Number of rounds of synchronization
    - Id -> thread identifier (starts with index 2)
    - Runs -> number of messages sent
    - Exponential_distribution
        - For simulating work done after synchronization, request, reply and channel delay.
    - driftFactor -> how much the clock drifts every time

    - errorfactor -> used for minimizing error after each synchronization. The initial value doesn't matter as updates are accordingly anyway.
- Functions
    - Constructor and assign_attr -> Here I bind the socket for listening to connections for reading local clock time
    - Read -> Reads system clock and adds errorfactor and driftFactor to construct local timestamp
    - Update -> updates error factor with optimal delta
    - IncrementDriftFactor -> updates drift which runs in parallel
    - Timespec_t2_from_string -> decodes string to timespec
    - Send_channel -> simulates message being sent over the channel
    - Synchronize_requests -> every round, send requests to other servers for t2,t3 timestamps for calculation of optimal_delta
    - Handle_replies -> listens for requests and serves replies accordingly using Synchronize_reply.
    - Synchronize_reply -> encodes data to be sent and send it via send_channel to simulate message via the channel.

Global Variables:

Since the concept of shared memory does not exist in distributed systems, I made sure to not use any info from global variables for the simulation of local clocks. They are strictly used for capturing the global state.

I have used only the following vars with explanation for doing so:

- log_file
    - Represents the global state in our simulation where all the events info is recorded.
- mutex
    - Write_lock

        - Since I have to log the timestamps of specified events and tables, I used mutex so that race conditions don't occur while storing them in the log file. Since the log file represents the global state, it is fine to use a global mutex.
    - End_lock
        - used to ask all running threads to stop the execution of the program. In actual time servers, it keeps running, which is not what is required in the assignment.
- data
    - Since I am asked to make a table with local timestamps with corresponding mean and variance every round, a global variable is needed to do so, since, in doing so I am essentially capturing global state.
- default_random_engine
    - This is used for the generation of value from an exponential distribution for sleep to simulate some work done.
- local_clock_mut
    - An array of mutex for doing atomic updates on error factor. Each time server uses it's own index's mutex and in doing so, there is nothing shared here.
- start_time
    - Store start time so that the timestamp used for table generation is relative to the start of the program.

Other Functions:

- currTime
    - Returns Timestamp as a string, used for logging purpose

- Generate_table
    - Generate timestamps(in seconds) for every round for all processes. Also, prints mean and variance for each round.

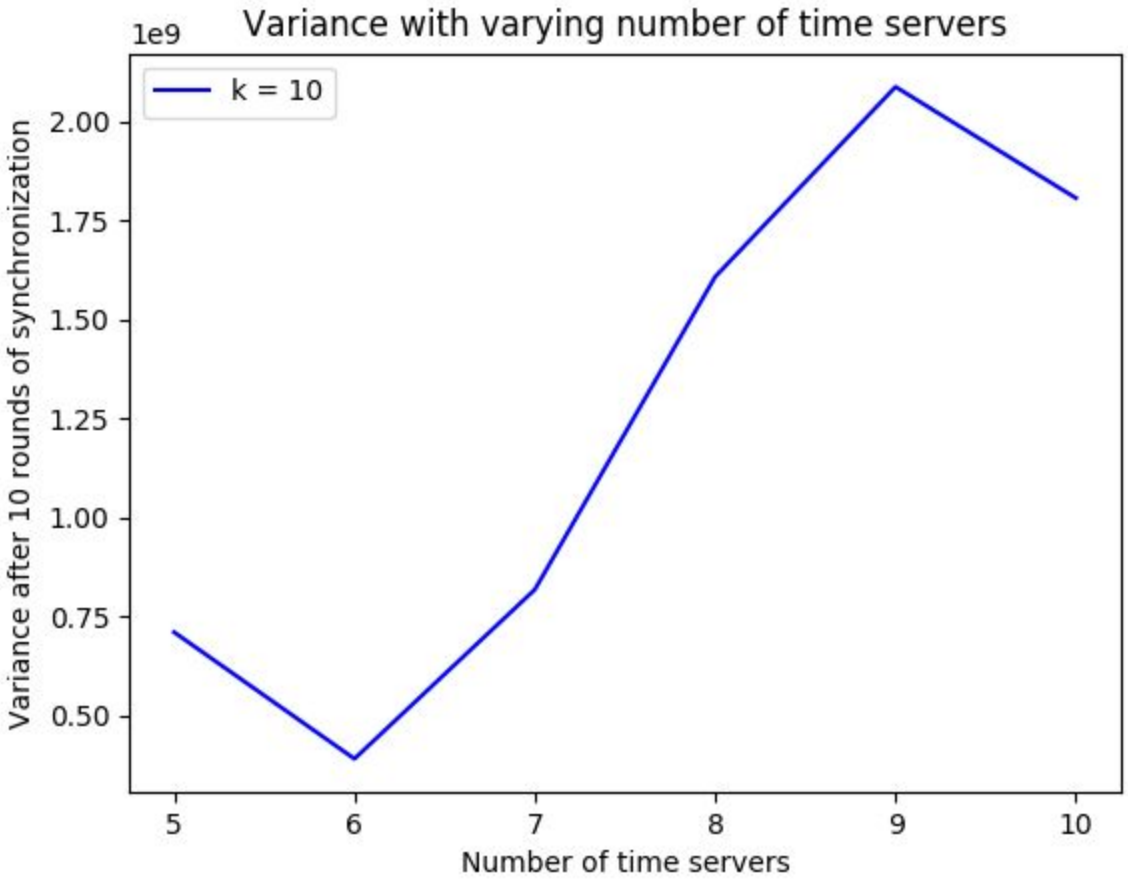
EXECUTION

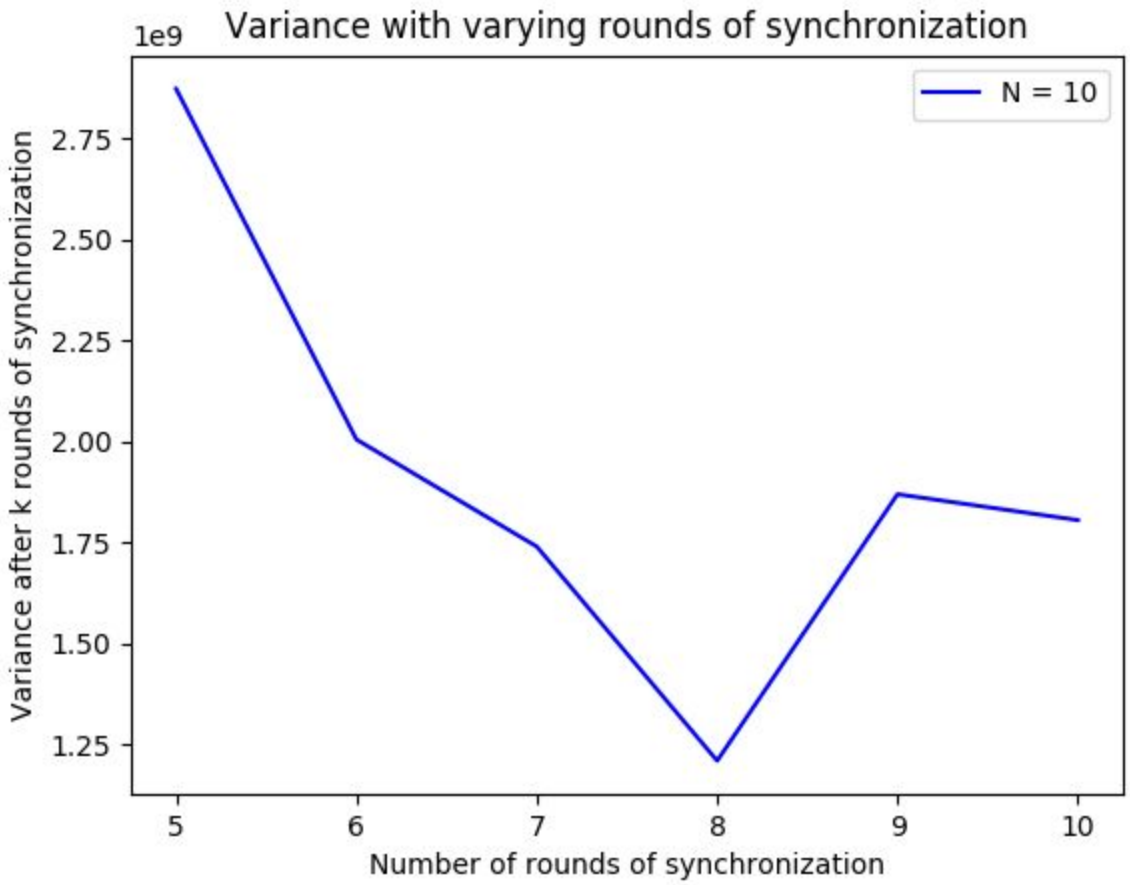
- Parse input from inp-params.txt and initialize the required number of TS(local clocks).
- In initialization, construct required exponential distributions and bind socket accordingly. (Set 127.0.0.x as server IP address) where x ∈ [2, number of time servers + 1]
- Open the log file and populate “data” vector for saving values once the simulation is done
- Store the start_time, so that calculation of timestamps for the table is done relative to this.
- Spawn threads equal to the number of time servers, where it, in turn, spawns 3 threads which are th_send, th_recv and th_drift.
    - Th_send
        - The thread which handles sending requests to other time servers for k rounds.
        - Here, let the thread wait until the receiver can set itself to handle other requests. Doing so, some drift in the local clocks is generated too.
        - For each request, construct a client socket which will then be used to communicate to the respective server using the said IP nomenclature (above).
        - Store t1 timestamp here.

        - Send client ID in the message via a channel, which will serve as a signal for request as well as the ID of the server to which it has to send a reply. The round number is also sent for logging purposes. Encoding has been done with ";" as the delimiter.
        - Once, a reply has been received, now, decode from the buffer for timestamp t2 and t3.
        - Store t4 timestamp here
        - Now with the help of t1, t2, t3, t4, calculate offset(x) and round trip delay(2* y)
        - Calculate optimal delta and update the local clock's errorfactor accordingly.
        - Store a new synchronized timestamp to a global vector so that a table is created to analyze in the last round.
        - Now, wait until all local clock's ith round synchronization is done and then proceed.
    - Th_recv
        - The thread which handles requests from other servers and replies with t2,t3 timestamps encoded using *synchronize_reply*.
        - Limit the maximum queue to number of servers
        - In *synchronize_reply*, decode the message to get client ID and round number. Then, get the timestamps t2 and t3, encode using ";" as delimiter into a string and send it back to client using the client ID.
        - Once, it has given max number of possible replies (total servers - 1) * k rounds, it stops.
    - Th_drift
        - This runs until the end of execution in intervals(using an exponential distribution to set time of interval).
        - This is responsible for simulating drift in the clock.

        - Once, it has given max number of possible replies (total servers - 1) * k rounds, it stops.
- Generate_table
    - Print into the log file with the required table format, and also calculate mean and variance each round.

GRAPHS





OBSERVATIONS

- For large values of n(in the first graph) or k (in the second graph) we notice the variance is getting smaller.
- The above is probably not observed for smaller values as we forcibly introduced some random wait time (work simulation) from an exponential

distribution when in reality there is an almost non-existent delay over communication through sockets. This is due to the case that we simulated the program on one computer instead of a distributed system.

- The above reasoning might be the case why we can't interpret the results for smaller test cases.