This assignment is based on the principles of clock consistency and drifts in a distributed system. You have to create a simulation, running on a single machine, of a simple distributed system involving a master object (MO) and four process objects (PO). The MO and each PO will contain a logical clock, a concept first proposed by Lamport.\(^1\) The concept of the logical clocks along with the following technique, which is based on the Berkley Algorithm,\(^2\) will attempt to resolve the clock consistency in this system:

1. Each event (send or receive) in the system is associated with a time-stamp, based on logical clocks.

2. Each PO, \(P_i\), will have an associated logical clock, \(LC_i\). This logical clock is implemented as a simple counter that is incremented between any successive events executed within that PO. Since a logical clock has a monotonically increasing value, it assigns a unique number to every event. The time stamp of an event is the value of the logical clock for that event. Hence, if an event A occurs before an event B in \(P_i\), then \(LC_i(A) < LC_i(B)\).

3. The MO will also contain a logical clock, that will be incremented periodically and as indicated in the next point.

4. After every \(t\) units of logical time (a parameter that can be varied), each PO will send its current local time (i.e., value of its logical clock) to the MO. Once a MO receives such a message from any PO, it will average all five values (i.e., logical clock values of four POs and its logical clock value) and will consider it to be the correct logical clock value. It will then calculate an offset (either positive or negative) for each PO’s logical clock and send it to each PO. It will also adjust its logical clock to that correct clock value.

5. A PO will advance its logical clock when it receives a message from the MO containing an offset. If \(P_i\) receives a message from MO with an offset \(t_i\) (could be positive or negative) then \(P_i\) should adjust its clock such that \(LC_i = LC_i + t_i + 1\).

6. A \(P_i\) will randomly decide, in addition to sending a message to the MO, if it wants to send a message to one or more other \(P_j\)s. Upon receipt of such a message, each \(P_j\) will also advance its logical clock if the time stamp associated with the incoming message is greater than the current value of its logical clock, i.e., if \(P_j\) receives a message (event B) from \(P_i\) with a time stamp \(t\) and \(LC_j(B) \leq t\) then \(P_j\) should advance its logical clock such that \(LC_j(B) = t + 1\).

7. Any \(P_i\), in the system, may exhibit Byzantine or arbitrary failures.

\(^{2}\)http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&carnumber=29484
Your task is:

1. Propose the interaction and failure models for this system. Discuss the pros and cons of your design.

2. To simulate this entire environment in C++ or Java using threads. Allow the simulation to reach a steady-state, i.e., run the program for a large number of iterations. During each iteration and at the end of the simulation compare the values of the logical clocks of POs with that of the MO. Print out the clock drifts (i.e., difference between the logical clocks of the MO and each PO) for all POs. Repeat the simulation with different probabilities and access their effects on the clock drifts.

   Please employ good software engineering principles in your design and implementation. Ensure that your program runs on pegasus.cs.iupui.edu. Provide adequate documentation of your programs. Create a makefile for your program. Submit all the source files (including the readme, input/output and make files) by using submitd command on Pegasus.