CSCI 53700 – Fall 2017

Assignment Number 1

Due Date: October 24, 2017

This assignment is based on the principles of clock consistency, associated drifts, inter-process communication and the end-to-end argument. It will also make you familiar with Java and its networking API.

You have to create a simple distributed system in Java, 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 and discussed in our class.[1] The concept of the logical clocks along with the following technique, which is based on the Berkeley Algorithm,[2] will attempt to resolve the clock consistency in this system:

1. Each event (send or receive or an internal computation) 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. A P_i will randomly decide (with some probability – a parameter that can be varied) if it wants to carry out an internal event or send a message to the MO or receive a message from the MO. If it decides to send a message, then it will attach the value of its logical clock to that message.
5. Once a MO receives such a message from any PO, it will average all current 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.
6. 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 is equal to $LC_i + t_i + 1$.
7. Any P_i, in the system, may exhibit Byzantine or arbitrary failures.
8. Inter-process messages containing logical clock values need to be encrypted, using some simple technique, such as the security API provided by Java, before the transmission.

[1] http://research.microsoft.com/en-us/um/people/lamport/pubs/time-clocks.pdf
[2] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=29484

Your task is to:

1. Propose the interaction and failure models for this system. Discuss the pros and cons of your design. Your design should consider the principles of the end-to-end argument. You must explicitly indicate the design decisions that you have taken, in order to achieve the above mentioned functionality, that have been influenced by the end-to-end argument. Also, the consequences of these decisions should be discussed.

2. Develop your system using the *stream sockets* available in `java.net` package.

3. Deploy the POs and MO on different machines from the list indicated below:

   ```
   in-csci-rrpc01.cs.iupui.edu 10.234.136.55
   in-csci-rrpc02.cs.iupui.edu 10.234.136.56
   in-csci-rrpc03.cs.iupui.edu 10.234.136.57
   in-csci-rrpc04.cs.iupui.edu 10.234.136.58
   in-csci-rrpc05.cs.iupui.edu 10.234.136.59
   in-csci-rrpc06.cs.iupui.edu 10.234.136.60
   ```

4. Run your system for a long time and periodically compare the values of the logical clocks of POs – this comparison should indicate the clock drifts and their synchronizations. Repeat the system with different probabilities and access their effects on the clock drifts.

5. Create a brief report that indicates the aforementioned models and the analyses of the results of your system.

Please employ good software engineering principles in your design and implementation. Provide adequate documentation of your programs. Create a *makefile* for your program. These files (source files and makefile) should be submitted via the *submitd* command on tesla.cs.iupui.edu in a zipped folder with the following format (LastNameA1.zip) - e.g., RajeA1.zip. Also, hand-in a hard-copy of the report at the beginning of the class on the due date.