15-440 Distributed Systems
Homework 1

Due: October 9, In class.

October 6, 2012

1. In this scenario, three nodes, named `Schenley`, `Frick`, and `Carnegie` are working on a job for a user. The sequence of events is given:

```
e1) Carnegie sends sync request to Frick
e2) Frick receives sync request from Carnegie
e3) Carnegie sends sync request to Schenley
e4) Frick sends sync request to Schenley
e5) Schenley receives sync request from Carnegie
e6) Schenley sends sync acknowledgement to Carnegie
e7) Carnegie receives job from user
e8) Carnegie receives sync acknowledgement from Schenley
e9) Frick sends sync acknowledgement to Carnegie
e10) Carnegie sends work to Schenley
e11) Schenley receives work from Carnegie and begins processing
e12) Carnegie receieves sync acknowledgement from Frick
e13) Schenley sends sync acknowledgement to Frick
e14) Schenley sends work to Frick
e15) Frick receives work from Schenley and begins processing
e16) Schenley completes processing and waits for results from Frick
e17) Frick completes processing and sends results to Schenley
e18) Schenley receives results from Frick
e19) Schenley combines results and sends them to Carnegie
e20) Carnegie receives results
e21) Carnegie sends close to Schenley
e22) Carnegie sends close to Frick
```

   Using this sequence of events:

   (a) Write out the Lamport Clock representation of each timestep, using the notation L([event]) = [Lamport Timestep]. For example, L(e1) = 1.

   (b) Write out the vector time representation of each timestep.

2. In class, we examined the problem of *distributed mutual exclusion*: Guaranteeing that only a single process can be in a particular critical section at one time.

   The lecture notes discuss four algorithms: The use of a central lock server; Ricart & Agrawala's algorithm; Lamport's Algorithm; and a Token Ring.

   (a) Easy warm-up: What are the two requirements we discussed in class?

   (b) In Lamport's algorithm, a node first puts its lock request in its own queue. It then sends a request to every other node and waits to hear replies from all of those nodes. Messages must be processed in-order. A node won't grant itself the lock until it has heard REPLIES from all other nodes containing a timestamp greater than the timestamp of its own request.

   What is the fairness provided by Lamport's algorithm compared to the fairness provided by the token-ring algorithm?

   (c) The solutions presented in class did not consider the problem of a machine failing. Using Lamport's Distributed Mutual Exclusion algorithm, there are two obvious consequences of a machine failure: A machine dies holding a lock forever; and the inability to make forward progress because a machine fails to respond to messages.

   How would you solve the first problem, of a machine dying and holding a lock forever?

   (d) Token Ring is also vulnerable to a node failure: Node N will try to send the token to node N+1. If its connection to N+1 fails, N+1 fails, the token will get "stuck" at node N. Improve this algorithm so that it is robust to connection failures and to the failure of a node that is *not* currently holding the token.

Give a description of your algorithmic changes and why they work. Code is not needed. You may assume that the network is modestly-sized enough that you don't need an O(1) algorithm - using up to or less than O(N) memory and CPU is fine.

(e) If you were particularly concerned about the death of one machine preventing progress, which of the discussed solutions to distributed mutual exclusion would you use? Explain briefly why.

3. Dropbox is, at its heart, a distributed filesystem. See this page in the dropbox docs: `https://www.dropbox.com/help/36/en`.

   (a) Allowing such conflicts to happen is a deliberate design decision in Dropbox. Briefly explain how this differs from AFS, and how AFS handles concurrent modifications to a file. (Briefly: 3-4 sentences)

   (b) Why do you think Dropbox chose its design over AFS's design? What advantage does it confer to Dropbox?

   (c) In what way is caching in dropbox more similar to AFS than it is to NFS?

4. Some questions about RPC.

   (a) RPC is designed to reduce programmer effort in writing distributed systems by making remote function calls as easy to use as local function calls. Identify *three* different ways in which this abstraction does not hold true.

   (b) Hit the web to answer this one: Briefly identify two advantages and one disadvantage of using Google's "protobuf" format instead of the JSON format you're using for marshaling in Project 1.

   (c) Your code in Project 1 handles the problem of preventing duplicate RPCs by using what networking people call "Stop and Wait": It has one message outstanding at a time. If a message with a sequence number lower than the next-expected sequence number is received, the system will throw it away.

   Imagine using your system to talk between CMU and the machine at UW from micro-quiz 1. Assume the RTT from here to UW is 70ms. How many RPC calls can you execute per second using a single client connection from here to UW?

   (d) Briefly sketch out how you would improve this design to handle having multiple RPCs in flight at a time.