Brian Bauman

CSC 536

Homework 7

1.

(a)

One such scenario would involve a process whose application code requires sending additional messages to a particular process or shutting down, as shown below in a group with nodes (A, B, C, D):

* Node A R-multicasts message m to node B, but crashes before sending to nodes C or D.
* Node B receives the message and processes it, requiring a separate synchronous message, n, to be sent immediately to node C and blocks until it receives a reply.
* Node C receives message n and processes it, requiring a separate synchronous message, n`, to be sent immediately to node B and blocks until it receives a reply.
* Repeat ad infinitum — nodes C and D never receive message m…

Or, what if, in the same scenario, node B processes message m, which the application code determines requires a program shutdown. The process terminates (no faulty behavior, this is expected) without ever forwarding the message.

(b)

FIFO ordering: All messages sent by a node are marked with a node-specific sequence number. If a process receives a message from another node that has skipped ahead by one or more sequence numbers, it waits to process the latest message until all prerequisite messages have been delivered.

Causal ordering: All messages sent by a node are marked with a node-specific vector timestamp. If a process receives a message from another node that has seen more messages from any given third node, the process waits until those prerequisite messages are received before processing the current message.

Total ordering: This can be implemented using Lamport timestamps and acknowledgements as discussed in the first week of class. I’m not sure that this algorithm will work in the presence of a single faulty node, but because that means that the message would never be processed, the principle of atomicity is maintained.

2.

(a)

The election safety property holds because, for a given term, a majority of votes is required to be elected leader. Two candidates cannot both receive a majority of votes in the same leader election term.

(b)

Scenario (e) could occur in a situation where that node was down for the 2nd and 3rd terms and the leader in term 4, but was unable to commit the last two log entries in that term. Then, the node was down in the 5th and 6th terms.

Scenario (d) could occur if the node was down for terms 2 and 3 and was the leader for term 7 but crashed before committing its log entries.

Scenario (c) could occur if the node was down for terms 2 and 3 and was the leader for term 6 but crashed before committing its last log entry.

(c)

If a candidate cannot be elected because its log is out of date relative to another node, this implies the ability to sequence the nodes in order or “up-to-date”ness. As each node will eventually (and quickly) attempt to become the leader after their timeout expires, the most “up-to-date” node will soon make a candidate request for votes. As it is the most up-to-date node, the election will not fail for this reason.

(d)

This assumption can be made due to the log matching property. If a log entry was committed by leader T, then all intervening leaders between T and U must also have committed that log entry, or else they wouldn’t have been elected due to out-of-dateness.

(e)

The broadcast time must be considerably less than the election timeout, or else servers would initiate elections constantly when, in reality, they just haven’t yet received their heartbeat message from a leader. The election timeout and time to process a new leader election must be considerably less than the MTBF, or else servers will be spending the majority of their uptime holding elections, which slows the progress of the intended system purpose.

3.

For my final project, I plan to implement a general, distributed MapReduce framework (option 1).