Homework 4

CS425/ECE428 Spring 2019

Due: Wednesday, April 24 at 11:55 p.m.

1. (a) (4 points) Modify the token ring mutual exclusion algorithm to support reader/writer-style mutual exclusion. As a reminder, a read lock can be shared with other nodes, but conflicts with a write lock.

   (b) (4 points) Analyze the best- and worst-case client and synchronization delays for N nodes. Note that there are two types of client delays: read and write client, and three types of synchronization delay: read/write, write/read, and write/write. Assume that the one-way communication delay between two nodes in the ring is T and there is no processing delay.

   (c) (2 points) Does your algorithm create a possibility of reader or writer starvation? Justify your answer. (Note: you will not lose any points for an algorithm that can lead to starvation, nor gain any points for an algorithm that does not.)

2. (a) (4 points) Describe a modification of the Bully protocol that elects a leader and a vice-leader, which should be the two nodes with the highest and second highest identifiers among all the live nodes. Note that all nodes should know which is the leader and vice leader. Write down the pseudocode for the algorithm, similar to what was contained in the lecture slides for Bully.

   (b) (2 points) Analyze the worst-case time for an election among N nodes, assuming no nodes fail after an election has been called. Assume that the one-way communication delay between any nodes is T and there is no processing delay.

   (c) (3 points) How would you modify Raft to use a Bully-style election algorithm? What might be an advantage and disadvantage of this approach as compared with the Raft election?

3. Consider an implementation of a bank account transaction participant. It supports five RPCs: deduct, which is called during a transaction, and canCommit, doCommit, and doAbort, which are called during two-phase commit. Assuming a transaction always uses the correct txid and only calls deduct once per transaction.

   Below is Python-like code implementing the RPCs:

```
def deduct(txid, amount):
  account.lock()
  will_commit[txid] = account.balance > amount
  account.unlock()

  saved_amounts[txid] = amount

def canCommit(txid):
  return will_commit[txid]

def doCommit(txid):
  account.balance -= saved_amounts[txid]

def doAbort(txid):
  pass  # do nothing
```

   (a) (3 points) Which of the ACID properties does this implementation violate? Explain your answer.

   (b) (3 points) Consider an alternate implementation:

```
def deduct(txid, amount):
  account.lock()
  account.balance -= amount
  will_commit[txid] = account.balance > 0
  account.unlock()

  saved_amounts[txid] = amount

def canCommit(txid):
  return will_commit[txid]

def doCommit(txid):
  pass # nothing

def doAbort(txid):
  account.lock()
  account.balance += saved_amounts[txid]
  account.unlock()
```

Which of the ACID properties does this implementation violate? Explain your answer.

(c) (4 points) Write pseudocode that correctly implements all ACID properties

4. Consider a system with 5 replicas. The latency to access each replica is given in the table:

Node	Latency
A	1 ms
B	3 ms
C	5 ms
D	30 ms
E	35 ms

(a) (2 points) What would you set the sizes of a read quorum (R) and write quorum (W) to be if you wanted to minimize write latency?

(b) (2 points) What would be the read and write latency using those quorum sizes, assuming optimal choice of nodes for the quorum?

(c) (2 points) What would be the read and write latency using those quorum sizes, assuming optimal choice of nodes for the quorum, if A has failed?

(d) (4 points) Imagine D and E are located in a remote data center. How would you change the number of votes each node gets, and the quorum sizes, to ensure that each write is replicated outside the data center, while still minimizing write latency? What would be the write and read latency in your configuration?