HW3 Solutions: CS425 FA15

1. (Solution and Grading by: <Alex Zahdeh>)
   Assume the processes are \( p_i \) with \( 1 \leq i \leq 5 \). Consider the scenario when, at each round \( i \) where \( 1 \leq i \leq 5 \), \( p_i \) crashes after sending its value array to only \( p_{i+1} \). In that case, after five rounds, \( p_6 \)'s set will be \{1,2,3,4,5,6,7,8,9,10\} whereas every other living node’s set will be \{2,3,4,5,6,7,8,9,10\}. This is because \( p_1 \)'s value is not forwarded due to the successive five failures. So \( p_6 \) will decide on 1, whereas all other processes will decide on 2 (remember that the “min” taken in the final step of the algorithm is min by id, not min by value proposed). Therefore consensus is not achieved.

2. (Solution and Grading by: <Guangxiang Du>)
   a. It is safe, because \( \frac{2}{3} \) is larger than 50%, 2 groups of \( \frac{2}{3} \)s would intersect.
      Therefore, there could be at most one leader and one decided value.
   b. Eventual liveness is satisfied.
   c. The new version slower than majority version because it takes longer to receive more votes (\( \frac{2}{3} > 50% \)).

3. (Solution and Grading by: <Qi Wang>)
   a. They are serially equivalent. All conflicting pairs follow order \( T_1 \rightarrow T_2 \).
   b. They are not serially equivalent:
      conflicting pair \( \text{read}(a,T_1) \) and \( \text{write}(a,\text{baz},T_2) \) has order \( T_1 \rightarrow T_2 \), but conflicting pairs
      \( \text{write}(b,\text{baz},T_2) \) and \( \text{write}(b,\text{caz},T_1) \), \( \text{write}(a,\text{baz},T_2) \) and \( \text{read}(a, T_1) \) have order \( T_2 \rightarrow T_1 \)
   c. They are not serially equivalent:
      conflicting pair \( \text{read}(a,T_1) \) and \( \text{write}(a,\text{baz},T_2) \) has order \( T_1 \rightarrow T_2 \), but other conflicting pairs have order of \( T_2 \rightarrow T_1 \)
   d. They are not serially equivalent:
      conflicting pair \( \text{read}(b,T_2) \) and \( \text{write}(b,\text{caz},T_1) \) has order of \( T_2 \rightarrow T_1 \), but other conflicting pairs have order \( T_1 \rightarrow T_2 \)

4. (Solution and Grading by: <Guangxiang Du>)
   No. The system only ensures object are accessed by one transaction at any given point of time (mutual exclusion), but it does not have mechanism to ensure all conflicting operations from 2 transactions occur at the same order.
A counterexample would be: (2 conflicting pairs of operations are not in the same order)

<table>
<thead>
<tr>
<th>Time sequence</th>
<th>Transaction 1</th>
<th>Transaction 2</th>
</tr>
</thead>
</table>
| 1             | T1 acquires lock for object a  
T1 reads object a  
T1 releases lock for object a |               |
| 2             |               | T2 acquires lock for object a  
T2 writes object a  
T2 releases lock for object a |
| 3             |               | T2 acquires lock for object b  
T2 reads object b  
T2 releases lock for object b |
| 4             | T1 acquires lock for object b  
T1 writes object b  
T1 releases lock for object b |               |

5. (Solution and Grading by: <Ayush Jain>)

I-read: Allows holder to lock descendant nodes in I-read/read modes
I-write: Allows holder to lock descendant nodes in I-read/I-write/read/write modes
read: read access to the node and its descendants
write: exclusive write access to the node and its descendants

To request read/write access to a particular node in the hierarchy,
   a. Before requesting read/I-read access to a node, all ancestors must be held in I-read/I-write mode
   b. Before requesting write/I-write access to node, all ancestors must be held in I-write mode.

The compatibility of these locks is shown in the table below - the columns show the lock being requested for a node and the rows show the current lock for the node. A “YES” indicates that the requested lock can be granted.
6. (Solution and Grading by: <Qi Wang>)
Please give explicit answers as false positive or not false positive.

\{0,2,4,8,10,12,14,16,20,22,24,28\}
\[h(x) = [h1(x),h2(x),h3(x),h4(x)]\]
\[h(1969) = [18,4,22,8]\]
\[h(1971) = [28,24,20,16]\]
\[h(1972) = [4,8,12,16]\]
\[h(2015) = [0,0,0,0]\]
\[h(2030) = [18,4,22,8]\]

If we assume “positive” as “the Bloom filter code thinks all the following (for example 1969) are present in it”, the results are:

a. false positive
b. not false positive
c. not false positive
d. not false positive
e. false positive

Otherwise the results are:
a. not false positive
b. false positive
c. false positive
d. false positive
e. not false positive

7. (Solution and Grading by: <Ayush Jain>)
False Positive Rate (FP) = \( \left( 1 - e^{-\frac{m}{n}} \right)^k \)

where \( k = \# \) of hash functions
\( m = \# \) of bits
\( n = \# \) of items in Bloom Filter

For the captain’s suggestion, plugging \( k = 4, m = 64, n = 5 \):
\[
\log \text{FP}_1 = -5.2613
\]
For your approach, two bloom filters have to give out a false positive simultaneously. So the probability of a false positive now is

\[ FP_2 = \left(1 - e^{-\frac{m}{k}}\right)^k \times \left(1 - e^{-\frac{m}{n}}\right)^k \]

with \( k = 4, m = 32, n = 5 \)

\[ \log FP_2 = -6.1302 \]

Since \( FP_2 \) is lower, your approach gives a lower false positive rate.

8. (Solution and Grading by: <Alex Zahdeh>)
   a. Passive replication since it would reduce client latency at times when failures are less likely and therefore replication is less needed
   b. Active replication to prevent data loss during high failure times, and to prevent leader election overhead.

9. (Solution and Grading by: <Yi Zhang>)
   a. Fore partition provides availability and aft partition only provides availability for read. It violates consistency since a write from the fore partition can not be seen by a read in the aft partition.
   b. Fore partition only provides availability for write and aft partition provides availability. It violates consistency since aft replication can not read writes by fore partition.
   c. Both partitions provide availability for read. Only partition with quorum number of servers provide availability for write. It violates consistency since the write can not seen by the read in the less number of servers partition.
   d. It violates the consistency. Since the quorum is measured only within that partition, it is possible that both partitions could execute read and write. It violates the availability. It is possible that neither partition is able to read and write.
   e. Both partitions provide consistency but they violate availability for write.
   f. It provides consistency since only one partition is able to read and write. It violates availability since the other partition can not read or write.

10. (Solution and Grading by: <Yi Zhang>)
    a. Eventual consistency
    b. Linearizability and Sequential
    c. They both care about giving an illusion of a single copy. Linearizability cares about time. Sequential consistency cares about program order. With sequential consistency, the system has freedom as to how to interleave operations coming from different clients, as long as the ordering from each client is preserved. With linearizability, the interleaving across all clients is pretty much determined already based on time. See [http://www.cs.cmu.edu/~srini/15-446/S09/lectures/10-consistency.pdf](http://www.cs.cmu.edu/~srini/15-446/S09/lectures/10-consistency.pdf)

11. (Solution by: Indy. Grading by <Sanchit Gupta>) All statements a-d are true, and you're good to land.