HW4 Solutions: CS425 FA18

1. (Solution and Grading by: <TA, Beomyeol>)
   Conflict operations:
   read(a, T1); read(a, T1); write(a, baz, T2);
   write(c, foo, T1); read(c, T2);

a. read(a, T1); read(b, T1); read(b, T2); write(d, bar, T2); read(a, T1); write(a, baz, T2); read(c, T2);write(c, foo, T1);
   ⇒ ***Not*** *serially equivalent. Conflict operations for 'a' have (T1, T2) order. However, conflict operations for 'c' have (T2, T1) order.*
b. read(b, T2); write(d, bar, T2); read(a, T1); read(b, T1); write(a, baz, T2); read(a, T1); read(c, T2); write(c, foo, T1);
   ⇒ ***Not*** *serially equivalent. Conflict operations, the first read(a, T1) and write(a, baz, T2), have (T1, T2) order. However, the second read(a, T1) and write(a, baz, T2) have (T2, T1) order.*
c. read(a, T1); read(b, T1); read(b, T2); write(d, bar, T2); read(a,T1); write(a, baz, T2); write(c, foo, T1);read(c, T2);
   ⇒ *Serially equivalent. Conflict operations for 'a' have (T1, T2) order. Also, conflict operations for 'c' have (T1, T2) order.*
d. read(b, T2); read(a, T1); read(b, T1); read(a, T1); write(d, bar, T2); write(a, baz, T2); write(c, foo, T1); read(c, T2);
   ⇒ *Serially equivalent. Conflict operations for 'a' have (T1, T2) order. Also, conflict operations for 'c' have (T1, T2) order.*

2. (Solution and Grading by: <TA, Faria>)
   If locks are acquired in order of decreasing lexicographical order of object ID, deadlocks will not occur. We know that each object is associated with a unique word/ID.

   We build a wait-for graph for all transactions in the system. According to our algorithm, we acquire locks in decreasing order of ID. We assume that all transactions obey the algorithm.

   Proof by contradiction:
   There is a deadlock in the system. This means that there must be a cycle in the wait-for graph. We traverse the wait-for graph. If transactions obey the rules, each transaction is waiting to acquire a lock on a object with a *lower* ID than the objects it has already acquired locks on. However, if there is a deadlock, this means that the object is waiting to acquire a lock on an object with a *higher* ID i.e. there is a cycle in the graph.

This is not possible as all transactions obey the algorithm.
Thus, we have a contradiction and there cannot be a cycle in the graph. QED.

3. (Solution and Grading by: <TA, Zhuolun>)
    a. No. The result of following example is a=2, b=1, which violates serial equivalence.

Transaction 1	Transaction 2
Lock(a) Write(a,1) Release_lock(a) Lock(b) Write(b,1) Release_lock(b)	 Lock(a) Lock(b) Write(a,2) Write(b,2) Release_lock(a) Release_lock(b)

    b. Yes, it can deadlock. The same counterexample for two-phase locking can be used here.

Transaction 1	Transaction 2
Lock(a) Lock(b) // DEADLOCK	Lock(b) Lock(a) // DEADLOCK

4. (Solution and Grading by: <TA, Rui>, reviewed by Zhuolun)
    a. Consistency is satisfied because no reads are possible.
    Availability for read & write are both violated because they are not allowed.
    b. Consistency is violated because DC could not get the latest value.
    Availability for read & write are both violated because they're not allowed in DC.

c. Consistency is violated because DC could not get the latest value.
   Availability of read is satisfied
   Availability of write is violated because it's not allowed in DC.
d. Consistency is violated because the partition with less than a quorum number of servers could not get the latest value.
   Availability of read is satisfied.
   Availability of write is violated because the partition with less than a quorum number of servers could not write.
e. Consistency is violated because the three partitions can have inconsistent values of the same file (Write hello to F1 in NJ partition.Write bye to F1 in DC partition).
   Availability of read is violated because the partition with less than a quorum number of servers could not read.
   Availability of write is satisfied.
f. Consistency is not guaranteed because if more than one partition is eligible to write, they can have inconsistent values of the same file.
   Availability of read & write are not guaranteed because the partition with less than a quorum of servers responsive could not read or write.
g. Consistency is satisfied because of no updates before partition is repaired.
   Availability of read is satisfied.
   Availability of write is violated because no write is allowed.
h. Consistency is guaranteed because there is at most one partition has a quorum number of servers. If it has, the client could only read from this one partition. Thus, all clients read the same value. If there no such partition, no one could read value, still guaranteeing consistency.
   Availability of read & write is violated because the partition with less than a quorum number of servers could not read or write.

5. (Solution and Grading by: <TA, Faria>, reviewed by Rui)
   A publish subscribe system has publishers which publish data e.g, servers publishing sports scores, and subscribers receiving the data e.g, clients receiving sport scores. There is little to no processing of data in publish subscribe systems. Kafka is a pub-sub system, but also performs additional work.

   Pub-sub systems may be topic-based or content-based. Examples are: topic-based -- Get me all scores for match "Nadal vs Federer", and for content-based -- get me all stocks whose values are more than $50.3.

Optional: Kafka is a pub-sub system that can be connected to data sources. Recent versions of Kafka also support some stream processing. Typically though, it is used to feed data into stream processing engines.

https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern
https://kafka.apache.org/documentation/#intro_topics

6. (Solution and Grading by: <Federico>)
   Answers may vary. Key points are summarized.

   ZooKeeper Atomic Broadcast (ZAB)

   - It elects a leader, synchronizes replica nodes, and the leader broadcasts updates
   - ZAB guarantees **reliable delivery**, **total ordering** of messages, and **causal ordering** of messages through TCP FIFO channels between the servers
   - Messages are done in 2 phases:
     a. Leader activation -- The leader initializes the correct state of the system and prepares proposals
     b. Active messaging -- The leader accepts messages to propose and organizes message delivery
   - Protocol
     a. A server sends a request to the leader and the leader creates a proposal with timestamp *zxid* (the ZooKeeper transaction ID)
     b. The leader broadcasts the proposal and the receiving servers log the transaction (which isn't used until committed by leader)
     c. A receiving server sends an ACK to the leader node (because of TCP)
     d. The leader commits the transaction to the servers when a quorum of them ACK the proposal, and the servers update their database

   Differences between ZAB and Paxos

   - Paxos cannot handle multiple outstanding transactions like ZAB; if 2 transactions have order dependency, Paxos can only have at most 1 outstanding transactions because FIFO ordering is not guaranteed
   - ZAB is strict with using FIFO ordering
   - ZAB uses transaction IDs (*zxid*) to order the transactions, and Paxos requires executing Phase 1 on a new primary for all unknown/unrecorded previous sequence numbers of transactions for ordering

- Unlike Paxos requiring sequential agreement for state updates, “ZAB replicas can concurrently agree on the order of multiple state updates without harming correctness” -- Apache
- ZAB’s leader activation uses epochs to skip uncommitted proposals and to avoid issues with duplicate proposals with the same *zxid* -- Apache

7. (Solution and Grading by: <TA, Rahul>, Reviewed by Federico)
   a. I) This approach is direct and the data is available immediately.
      II) The load distribution is more uniform than that of I
      III) Less load on base station compared to I and II
   b. I prefer the approach II as data is available within a reasonable timeframe compared t to approach III and does a better job distributing the load compared to approach I. Other answers are accepted if the reason is justified.
   c. Each mote should maintain the running average and count, it will pass to the parent mote the running average and count.
   d. quartile(25th percentile) is not amenable to in-network aggregation, so each mote needs to relay all the individual measurement values to the base station.
   d. If the total number is known in advance the lowest 25% can be forwarded. (BOTH solutions accepted)

8. (Solution and Grading by: <TA, Le>)
   a. P3 is the owner of the page, and is the only one holding it (in Read Mode)
   P3 switch to Write mode and write

   b. P3 is holding the page in Read mode and P4 is holding it in Write mode and P3 is the owner

   Not possible – If there is a process in W mode then the process has to be the owner

   c. P4 is the owner and is holding the page in a Write mode

   Invalidate P4’s copy (P4 gives up ownership), fetch copy, P3 becomes owner, P3 switch to W mode

   d. P1 and P2 are each holding the page in a Write mode, and P3 is the owner

   Not possible – can’t have two processes holding a page in W mode at the same time

e. P1, P2, P3 are each holding the page in a Write mode, and P3 is the owner

Not possible – can't have multiple processes holding a page in W mode at the same time

f. P1, P2 and P3 are currently holding the page in Read mode, P2 is the owner

Invalidate all copies for other processes, P2 gives up ownership, P3 becomes owner, P3 switch to W mode and write

g. P2 and P4 are holding the page in Read mode, and P4 is the owner

Invalidate all copies for other processes, P4 gives up ownership, P3 fetches all copies and becomes owner, P3 switch to W mode and write

h. P3 and P4 are both holding the page in write mode

Not possible – can't have multiple processes holding a page in W mode at the same time

i. P1 is holding the page in write mode

Invalidate P1's copy, P1 gives up ownership, fetch P1's copy, P3 becomes owner, switch to W mode and write

j. P4 and P5 are each holding the page in a Read mode, and P4 is the owner

Invalidate all copies, P4 gives up ownership, fetch all copies, becomes owner, switch to W mode and write

9. (Solution and Grading by: <TA, Shegufta>)
    a. **[M, Hash(K_{Apriv}(M))] : Incorrect.** Since receiver does not know K_{Apriv}, there is no way to calculate "K_{Apriv}(M)" on the receiving end.
    b. **[M, Hash(K_{Apub}(M))] : Incorrect.** "K_{Apub}" is publicly known. Therefore anyone can generate the signature and pretend to be Alice
    c. **[M, K_{Apub}(Hash(M))] : Incorrect.** "K_{Apub}" is publicly known. Therefore anyone can generate the signature and pretend to be Alice
    d. **[M, K_{Apriv}(Hash(M))] : Correct.** K_{Apub}(K_{Apriv}(Hash(M))) = Hash(M) which can be easily verified in the receiver side.
    e. **[M, K_{Apriv}(M)] : Correct.** K_{Apub}(K_{Apriv}(M)) = M
    f. **[M, K_{Apub}(M)] : Incorrect.** "K_{Apub}" is publicly known. Therefore anyone can generate the signature and pretend to be Alice

g. Best option is d, “[M, K_{Apriv}(Hash(M))]”. Hash(M) reduces the size of the message. Therefore, it is both efficient and safe.

10. (Solution and Grading by: <TA, Rahul>)
    a. No, There are two problems:
        i. **Being Stateless** : Since this is a distributed systems we need to support fault tolerance, in UNIX this is not required as all processes would crash along with the OS. If the server stores information per client this is lost when the server holding this crashes.
        ii. **Auto advancing read-write pointers** : Auto advancing read-write pointers would deviate from idempotent behavior. There could be inconsistency when there are multiple writes in case of lost ‘ACK’ from packet loss.
    b. Moving read-write pointer to client side, where :
        i. Read write would use absolute positions and would no require file descriptors.
        ii. Information regarding position need not be stored on the server side.