Solution to Homework 3

Problem 1. Two transactions T1 and T2 have instructions as follows (a and b are variables at the server):

T1: write(a, foo, T1); read(b, T1); read(a, T1);
T2: read(b, T2); write(b, bar, T2); write(a, baz, T2);

For each of the following interleavings, explain if it is serially equivalent and why:
(a) write(a, foo, T1); read(b, T1); read(b, T2); write(b, bar, T2); read(a, T1); write(a, baz, T2);
(b) read(b, T2); write(b, bar, T2); write(a, foo, T1); read(b, T1); write(a, baz, T2); read(a, T1);
(c) write(a, foo, T1); read(b, T2); write(b, bar, T2); read(b, T1); write(a, baz, T2); read(a, T1);
(d) read(b, T2); write(a, foo, T1); read(b, T1); read(a, T1); write(b, bar, T2); write(a, baz, T2);
Answer:
(a) Serially equivalent. The results are equivalent to executing T1 then T2.
(b) Not serially equivalent. Write write conflict (write(a, foo) & write(a, baz)).
(c) Not serially equivalent. Write read conflict (write(a, baz) & read (a)).
(d) Serially equivalent. The results are equivalent to executing T1 then T2.

Problem 2. Given the following real-time execution ordering:
W1(x, 3), R2(x), R3(x), W2(x, 2), R1(y), W2(y, 1), R3(y), R2(z), R3(z)
where W2(x, 3) means that transaction 2 sets the value of x to 3. The original values of x, y, z are 4, 5, 6, respectively and the transaction number serves as the transaction's timestamp. Assume a transaction commits immediately after its last operation, show how the following concurrency control schemes affect the ordering of the reads and writes in the execution history, and the value returned from each read operation.
(a) Strict 2PL locking
(b) Timestamp ordering
Assume that if a transaction aborts, it will be restarted as a new transaction with its reads and writes added to the end of the execution history. The new transaction will be assigned the next available transaction ID.
Answer:
(a) W1(x,3), R1(y=5), R2(x=3), R3(x=3), R3(y=5), R3(z=6), W2(x,2), W2(y=1), R2(z=6)
(b) W1(x,3), R1(y=5), R2(x=3), R3(x=3), 2 wants to write to x and aborts because rts(x)=3>2, R3(y=5), R3(z=6), R4(x=3), W4(x=2), W4(y=1), R4(z=6)

Problem 3. Explain how three phase commit protocol avoids delay to participants during their "uncertain" period due to the failure of the coordinator or other participants. Assume that communication does not fail.
Answer:
In the two-phase commit protocol, the "uncertain" period occurs when a participant has voted yes to commit but has not yet been told the final commit/abort decision. At this time, since it has voted yes, it can no longer abort unilaterally and must wait for the coordinator's decision.

In the three-phase commit protocol, a participant's "uncertain" period lasts from when the participant votes yes to commit (the participant is now in the Ready state) until it receives the PreCommit request (which allows the participant to move to the PreCommit state). At this stage, no other participant can have committed. Therefore if a group of participants discover that they are all "uncertain" and the coordinator cannot be contacted, they can decide unilaterally to abort.

Problem 4. Compare the following five consistency models: eventual consistency, linearizability, causal consistency, FIFO consistency, and sequential consistency. Among these:
(a) Which is the "fastest" (most available) models among these?

(b) Which are the two "most consistent" models among these?
(c) What is the difference between the two most consistent models among these?
Answer:
(a) Eventual consistency
(b) Linearizability and sequantial consistency
(c) Sequential consistentcy requires that the result of any execution is the same as if all operations of all clients were executed in the same sequential order and that the ordering from each client is preserved. However, it does not say anything about the interleaving of operations from different clients. Linearizability enforces that operations from different clients be interleaved according to a loosely synchronized global clock.

Problem 5. (adapted from Chapter 7 Problem 17 in Tbook)
Consider a variant primary-backup protocol where clients interact with the primary server to perform update operations through non-blocking RPCs. Clients' read operations can be performed by any server, i.e., primary or back.
(a) Does this scheme guarantee sequential consistency?
(b) Does this scheme always provide read-your-writes consistency?
Answer:
(a) Yes. All updates are ordered according to when the primary server received them. Order of updates from a same client are also correctly preserved as there is only one primary server. These updates are then applied to the primary and all backup servers in the same order.
(b) No. Since the primary server uses non-blocking RPCs, it will return an acknowledgement to the client once it has accepted the update. However, having received an acknowledgement does not necessarily mean the update operation has been successfully completed on the primary server or other backup servers. If the client moves to another replica, e.g., a backup server, immediately after receiving the acknowledgement from the primary server, it is possible that it will read obsolete data from the backup server.

Problem 6. Consider a key-value store that uses quorum for consistency. The total number of replicas, N, for a key, is fixed. However, N may be different for different keys. Each read has to access at least R replicas, while each write has to write to at least W replicas. For each of the following design choices, say whether it (by itself) does or does not prevent write-write conflicts. For each case, briefly explain why or why not. (You can assume there is no caching.)
(a) $R + W = N$
(b) $W = N$
(c) $R + W > N/2$
(d) $R + W > 3N/2$
(e) $R + W > 3N/2$, $W > 2N/3$
Answer:
(a) No. It is possible that $W < N/2$, which allows two simultaneous write operations.
(b) Yes. Only one write operation can take place at a time.
(c) No. It is possible that $W = 1$ and $R = N/2$, which allows more than one simultaneous write operations.
(d) Yes. This guarantees that $W > N/2$.
(e) Yes. Same as (d).

Problem 7. What is a virtual node and why does Dynamo use virtual nodes in consistent hashing over physical nodes?
Answer:
In Chord-like DHT systems, a physical node is mapped to a single point on the ring, and it is responsible for all the IDs between its predecessor and itself. In Dynamo, however, each physical node is assigned multiple points on the ring, and these points are called virtual nodes. Each virtual node is responsible for IDs between its predecessor virtual node and itself. In this way, instead of being response for a continuous set of IDs, each physical node is responsible for multiple, non-continuous sets of IDs on the ring.

Using virtual nodes has the following advantages:
1. If a physical node becomes unavailable, the sets of IDs handled by this physical node can be dispersed to other available nodes, i.e., the virtual nodes the failed node was responsible for can be moved to other available nodes. If virtual node is not used, then all the IDs the failed node was responsible for will be moved to its successor on the ring.
2. Similarly, if a new node joins the system, it can take over a roughly equivalent amount of load from other nodes.
3. Physical nodes that have higher capacity can by assigned more virtual nodes. This allows the system to take heterogeneity into consideration while making the assignment.

Problem 8. Both node failures and concurrent updates may lead to conflicting versions of an object. How does Dynamo determine whether two updates are causally related or concurrent?
Answer:
Dynamo uses vector clocks to capture causality between different versions of an object. We can determine whether two updates are causally related or concurrent by looking at the vector clocks associated with the two versions of the object after the updates.

If the counters of a first object's vector clock are smaller than or equal to the counters of a second object's vector clock, the first object is considered the ancestor of the second object and the first object can be forgotten. Otherwise, they are conflicting versions and must be reconciled.

Problem 9. Dynamo allows an application to customized its storage system to meet the desired level of performance, durability, and availability SLAs by allowing them to tune the parameters N, R, and W. Explain what these parameters are, and how they can be tuned.
Answer:
N is the number of nodes that a data item is stored at. R is the minimum number of nodes that must participate in a successful read operation. W is the minimum number of nodes that must participate in a successful write operation. Setting W+R>N yields a quorum-like system.

Setting N to a big number can make the data item more durable since even if N-1 nodes fail, we can still access the data item. Setting W to a small number make increase the availability of the system. For example, by setting W to 1, the system will never reject a write request as long as at least one node in the system can process the request. However, this comes at the cost of high risk of inconsistency and high read cost. Setting W to a large number can reduce the risk of inconsistency, but the latency of write operations may be increased by the slowest node selected in the quorum.

Problem 10. Dynamo uses a "sloppy" quorum mechanism. Explain what "sloppy" quorum is, and what implication it has on Dynamo's consistency model.
Answer:
In usual quorum-like protocols, read and write requests of a data item are sent to a fixed set of N nodes. In the event of network partitioning, it is possible that some of these N nodes may not be reachable, thereby a valid read or write quorum may not be formed. This makes the system unavailable. With a "sloppy quorum" mechanism, a quorum may be formed on the N healthy nodes. And these N healthy nodes may not be the same set of N nodes that typically serve the data, i.e., the first N nodes encountered while walking clockwise the consistent hashing ring.

With "sloppy quorum", the system remains available for read/write even in presence of network partitioning. However, this sacrifices the consistency guarantees that can be provided by Dynamo. It potentially allows concurrent writes to the same data item. Data written to one partition may not be immediately visible to the other partition. As a result, Dynamo is a eventually consistent data store.

Problem 11. Consider a 32-bit Bloom filter, and 3 hash functions h_1, h_2, and h_3, where $h_i(x) = ((x^2 + x^3) * i) \bmod m$.
(a) Show the Bloom filter bits following each of the following six insertions in order: 2013, 2010, 2007, 2004, 2001, 1998.

(b) For the Bloom filter obtained after part (a), find one value that is not among the six inserted values, but is a false positive.

Answer:

(a)

After inserting 2013: $h_1(2013) = 14$, $h_2(2013) = 28$, $h_3(2013) = 10$. Bits set: 10, 14, 28

After inserting 2010: $h_1(2010) = 12$, $h_2(2010) = 24$, $h_3(2010) = 4$. Bits set: 4, 10, 12, 14, 24, 28

After inserting 2007: $h_1(2007) = 24$, $h_2(2007) = 16$, $h_3(2010) = 8$. Bits set: 4, 8, 10, 12, 14, 16, 24, 28

After inserting 2004: $h1(2004) = 16$, $h_2(2004) = 0$, $h_3(2004) = 16$. Bits set: 0, 4, 8, 10, 12, 14, 16, 24, 28

After inserting 2001: $h_1(2001) = 18$, $h_2(2001) = 4$, $h_3(2001) = 22$. Bits set: 0, 4, 8, 10, 12, 14, 16, 18, 22, 24, 28

After inserting 1998: $h_1(1998) = 28$, $h_2(1998) = 24$, $h_3(1998) = 20$. Bits set: 0, 4, 8, 10, 12, 14, 16, 18, 20, 22, 24, 28

(b) The answer to this problem is not unique. As long as the number hasn't been inserted but all the bits the three hash functions map to are already set, it is a false positive. For example, 3200 will be a false positive since all of its hash will map to the zeroth bit which has already been set by 2004.

Problem 12. Consider a Cassandra deployment using the `RackInferring` snitch. Every data is stored at 3 replicas. Given the following IP addresses for the replicas, say in each case whether the allocation is: fault-tolerant to the failure of: i) a single server; ii) a single rack; iii) a single datacenter.

(a) 123.231.111.222, 123.231.112.222, 123.231.113.222

(b) 123.231.111.222, 123.230.1.2, 123.3.4.5

(c) 123.231.111.222, 123.231.111.223, 123.231.111.212

(d) 1.231.111.222, 1.231.112.222, 1.231.113.222

(e) 1.2.3.4, 1.2.3.4, 1.2.3.5

Answer:

(a) a single server and a single rack, but not a single datacenter

(b) all three

(c) a single server only

(d) a single server and a single rack, but not a single datacenter

(e) a single server only

Problem 13. It is said that HBase provides strong consistency over high availability. Please briefly explan how HBase handles a RegionServer failure.

Answer: In HBase, each Region is served by only one RegionServer. That is, all clients' requests (get or put) of all key-value pairs in a Region are served by the only one RegionServer. If the only RegionServer is down, the Regions it serves are not availble for any requests any more. That is, no availability for any Regions served by the down RegionServer.

Once the RegionServer failure is detected, HMaster will re-assign the unavailable Regions to other RegionServers. These RegionServers will replay HLogs to restore Regions' MemStores and locate HFiles. HLogs and HFiles are stored in HDFS with a replication factor of 3.