Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.824 Distributed System Engineering: Spring 2015

Exam II Solutions

I Serializability

You are using a database that implements serializable transactions. That means that the results of running a set of transactions concurrently is the same as running the transactions one at a time in some order.

The database stores records x, y, and z, with the following initial values:

```
x=1
y=10
z=100
```

Here is the code for three transactions that read and write those records.

```
T1:
  x = z + 1

T2:
  y = x + 1

T3:
  z = y + 1
```

1. [7 points]: The three transactions execute, starting at about the same time. None abort or deadlock. There is no other database activity. After they all commit, the results are x=4 y=2 z=3. For this execution, show a one-at-a-time order of the three transactions that yields the same results.

Answer: T2 T3 T1

2. [7 points]: Explain why, if they execute and all commit and none abort or deadlock, the final results cannot be x=101 y=2 z=11.

Answer: Here are the six possible orders and their outcomes; none of them are x=101 y=2 z=11.

```
123: x=101 y=102 z=103
132: x=101 z=11 y=102
213: y=2 x=101 z=3
231: y=2 z=3 x=4
312: z=11 x=12 y=13
321: z=11 y=2 x=12
```

II Thor

Consider a Thor system as described in *Efficient Optimistic Concurrency Control Using Loosely Synchronized Clocks*, by Adya *et al.* The system has two clients and a single Thor server. The server's database consists of two records, x and y, with these initial values:

```
x = 1
y = 0
```

Here are two transactions that use x and y:

```
T1:
  x = x + 1

T2:
  y = x
```

One client executes T1, and the other client executes T2. Both transactions request to commit at the same time. They send their validation and installation information to the server; both sets of information arrive at about the same time. T1's information says that it read x, wrote x, and that it wants to install 2 as the value for x. T2's information says it read x, wrote y, and wants to install 1 as the value for y.

There's no two-phase commit in this situation, but the server must still validate the transactions. The server validates transactions one at a time. While there might have been other transactions in the past, no other transactions are running concurrently with T1 and T2, and the server's VQ and invalid set are empty just before the time at which validation/installation information from T1 and T2 arrive.

> **3. [7 points]:** Explain how Thor's rules in the paper's Sections 3.2/3.3 could commit both T1 and T2.

Answer: The following sequence could occur, and results in both committing. The server processes T2 first. There's nothing in the VQ, so the later-conflict check is OK. There's nothing in the VQ, so there's nothing to do for the Section 3.3 checks. So the server validates T2. Now T2 is in the VQ. Now the server processes T1. T1's timestamp is higher than T2's. There's nothing in the VQ with a higher timestamp, so the later-conflict check is OK. Only Section 3.3's check number 1 applies, and it does nothing. So the server validates T1.

4. [7 points]: Explain how Thor's rules in Sections 3.2/3.3 could abort one of T1 or T2.

Answer: The following sequence could occur, and results in the server aborting T1. The server looks at T1 first. There's nothing in the VQ, so the checks in Sections 3.2 and 3.3 OK. So the server validates T1. Now T1 is in the VQ. T2 arrives second, with a higher timestamp. 3.2 has nothing to do, since there are no later transactions in VQ. 3.3 check 1 doesn't apply. 3.3 check 2 does apply (T1 wrote x; T2 read x). 3.3 check 2(a) fails! So the server aborts T1

Now consider a situation with the same starting values (x=1 and y=0). This time transaction T1 successfully commits, setting x's value in the server to 2. Then transaction T2 starts, but has not yet tried to commit.

5. [7 points]: Could T2 read 1 as the value of x? If yes, how could that happen? If not, why not?

Answer: Yes, T2 could read 1 as the value of x. It could happen if some previous transaction ran on the same client as T2, read x=1, and caused x=1 to be cached on T2's client. Then T1 could run on a different client. T1 commits but its invalidation message hasn't yet reached T2's client. T2 runs and sees the stale cached value.

Now a third scenario. There are two servers, S1 and S2. S1 stores x, and S2 stores y. The initial values, and the transactions:

```
x = 0    -- stored on S1
y = 0    -- stored on S2

T3:
  x = y + 1

T4:
  y = x + 1
```

These transactions require two-phase commit because they use records on more than one server. Each server validates transactions one at a time. While there might have been other transactions in the past, no other transactions are running concurrently with T3 and T4, and both servers' VQs and invalid sets are empty just before the time at which validation/installation information from T3 and T4 arrive.

The two transactions execute at about the same time on different clients. S1 and S2 receive and process the two-phase commit prepare messages in different orders, with the following validation/installation information:

```
S1 receives and attempts to validate a prepare from T4, then T3:
  T4: ReadSet is x, WriteSet is empty, no installation
  T3: ReadSet is x, WriteSet is x, installing x=1

S2 receives and attempts to validate a prepare from T3, then T4:
  T3: ReadSet is y, WriteSet is empty, no installation
  T4: ReadSet is y, WriteSet is y, installing y=1
```

6. [7 points]: Will both transactions end up being committed, just one, or neither? Explain how Thor's algorithms lead to this outcome.

Answer: One will commit – the one with the lower timestamp. The situation is symmetric, so let's assume T3 has the lower timestamp. S1 will validate T4, since it has an empty VQ. S1 will NOT validate T3, since it fails Section 3.2's later-conflict check. S2 will validate T3, since it has an empty VQ. S2 will validate T4, since Section 3.3 check 1 applies. Thus T3 will abort and T4 will commit.

III Kademlia

Recall the paper *Kademlia: A Peer-to-peer Information System Based on the XOR Metric* by Maymounkov and Mazières.

Kademlia supports a `store(key,value)` request to insert a key/value pair into the DHT, and a `find_value(key)` request to retrieve a previously-stored value. For some applications, it would be convenient if the DHT also supported a `delete(key)` operation, which caused subsequent `find_value(key)` requests to not return any value (except if there is a subsequent `store(key)`).

> **7. [7 points]:** Explain why it would be difficult for Kademlia to provide a `delete(key)` operation.

Answer: Kademlia caches key/value pairs along lookup paths, and it would take a lot of effort to find all the cached pairs corresponding to the key to be deleted.

IV MapReduce

MapReduce (*MapReduce: Simplified Data Processing on Large Clusters*, by Dean and Ghemawat) stores the Map output on the local disk of the Map worker. A Reduce worker that needs a Map worker's output fetches the data over the network directly from the Map worker. In contrast, the Map inputs and Reduce outputs are stored in the GFS replicated distributed file system.

> **8. [7 points]:** MapReduce could have been designed so that Map workers write Map output to GFS, and Reduce workers read the Map output from GFS. Explain the main performance and fault-tolerance advantages and disadvantages of switching from local disk to GFS for Map output.

Answer: Performance would be reduced, since the Map output would have to be sent twice over the network for GFS replication. Fault-tolerance would be improved, since a Map worker failure before the fetch of all its Map output would not result in the Map output having to be re-computed.

V Facebook/memcache

Have a look at Section 3.2.1 of *Scaling Memcache at Facebook* by Nishtala et al. Suppose the following sequence of events occurs in a system with just one region:

- Client C1 asks for key "x" from memcache, and memcache reports a miss.
- Client C1 asks the database for the value of "x"; the database returns 100.
- Client C2 writes the database, setting the value of "x" to 200.
- Memcache receives and processes the invalidation for "x" from McSqueal (Figure 6) caused by C2's write.
- Client C1 sends a set("x", 100) RPC to memcache. This is the third message in the left-hand part of Figure 1.

9. [7 points]: C1 has asked memcache to store a stale value. How does the paper's design avoid serving this stale value to a get("x") issued by a different client immediately after C1's set finishes?

Answer: memcache gave C1 a lease token along with the miss notification. memcache marks that token as invalid when the invalidation request from McSqueal arrives. C1 sends the token along with its set(x, 100). memcache ignores C1's set because the token isn't valid. A subsequent get(x) will miss and consult the database.

Ben Bitdiddle observes that his system, built as described in the paper, benefits from caching but never has the "thundering herd" problem. The reason is that, for his application, different get()s for the same key are always separated from each other by at least a few dozen seconds. Ben, who doesn't fully understand the lease mechanism, decides to disable it by changing the lease interval to zero seconds (rather than the default 10 seconds).

10. [7 points]: Explain how Ben's change will affect performance for his application.

Answer: It's not clear what the answer to this question is. Perhaps PNUTS expires each lease after the lease interval, in which case Ben's change will prevent clients from caching anything in memcache. Or perhaps PNUTS doesn't expire leases, and merely issues an additional lease after each interval; in that case, Ben's change won't affect performance. The former seems more likely, but the paper doesn't say.

VI PNUTS *vs.* Dynamo

Consider PNUTS and Dynamo (*PNUTS: Yahoo!'s Hosted Data Serving Platform* by Cooper *et al.* and *Dynamo: Amazon's Highly Available Key-value Store*, by DeCandia *et al.*).

Suppose there are 10 data centers, evenly distributed over the surface of the earth. You are trying to decide whether to use PNUTS or Dynamo to store your data across these data centers. For these questions, assume the load on the system is low, so that the CPUs, disks, and network are not busy. Assume no failures except as noted. Assume Dynamo uses N=3 R=2 W=2. Assume that clients use randomly-selected keys.

11. [7 points]: If the PNUTS client uses `read-any`, which of PNUTS or Dynamo will take less time for a client to do a read? Why?

Answer: PNUTS will take less time. A Dynamo read has to wait for the network-round trip to two (of three) random data centers; each will probably take dozens of milliseconds. A PNUTS read-any can be served from the local data center's replica of the data, in roughly zero time.

12. [7 points]: If the PNUTS client uses `read-latest`, which of PNUTS or Dynamo will send fewer inter-data-center messages for a read? Why?

Answer: PNUTS will send fewer messages. PNUTS has to consult just the master for the record, which is usually one inter-data-center RPC. Dynamo consults three nodes (because N=3), which usually requires three inter-data-center RPCs.

13. [7 points]: Which of PNUTS or Dynamo will be least affected by a power failure at a single entire data center? Why?

Answer: Dynamo will be less affected. If one data center fails, then 10% of the PNUTS keys can't be updated and can't be the target of `read-latest`. For Dynamo, a single data center failure probably won't cause any keys to be inaccessible.

VII Bitcoin

The Bitcoin software automatically adjusts the hardness of the proof-of-work so that it takes about ten minutes before the first Bitcoin server finds a nonce that produces a valid new block extending the block chain. After a server finds a new block, it floods the new block over the Internet to all the other Bitcoin servers; assume this flooding takes about a minute. When each Bitcoin server hears a flooded new block, it checks whether the block is valid extension of the current chain (or longest fork); if so, the server stops its current proof-of-work and switches to working on a successor to the newly arrived block.

Ten minutes is a long time for some users to wait for confirmation of their transactions. Suppose the Bitcoin developers persuaded all Bitcoin users to simultaneously switch to a new version of the software and protocol identical to existing Bitcoin except that the targeted inter-block time is 10 seconds instead of 10 minutes.

14. [7 points]: This turns out to be a bad idea. What would go wrong?

Answer: Most nodes would not have seen the most recent blocks, and thus would be working on new blocks that would form forks, rather than working on extending the true end of the block chain.

VIII 6.824

15. [1 points]: Which were the most and least interesting papers in the second half of the course?

The contestants:

```
MapReduce (again)
Spark
Spanner
Memcached/Facebook
PNUTS
Dynamo
Akamai/Hubspot
Argus
Thor
Kademlia/BitTorrent
Bitcoin
AnalogicFS
```

16. [1 points]: What topics should we add to 6.824, or delete from it?

End of Exam II