Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.824 Distributed System Engineering: Spring 2013

Quiz II Solutions

I Lab 4

Recall that Lab 4 shards, or partitions, the keys over a number of Paxos-replicated groups. Suppose that initially group G1 is serving shard S2. The shardmaster creates a new configuration in which G3 is responsible for S2. The servers in G3 learn about the new configuration via a Query RPC to the shardmaster before any servers in G1 learn about the new configuration. That is, there is a window of time during which **both** G1 and G3 believe that the latest known configuration makes them responsible for S2.

1. **[10 points]:** Explain why at most one of G1 and G3 will serve client requests for S2.

Answer: No server in G3 will start serving until it gets a complete copy of S1 from G1. No server in G1 will send S1 to G3 until G1 has an entry in its Paxos log indicating that the handoff is taking place. No server in G1 will return a successful reply to a client for a put or get on S1 if the put/get appears in the Paxos log after the handoff's position in the log. The result is that, if G3 could answer a client get/put on S1, all replicas in G1 will reject gets/puts on S1.

II Bayou

The Tasty Diner decides to implement its own reservation system using the Bayou distributed database. To keep the problem simple, assume that there are just ten seats in the diner (numbered 1 to 10), and that each seat can be reserved just once per day, and that we're concerned with a single day. The Tasty's central computer runs the Bayou "primary replica." The diner has several sales agents answering phone calls from hungry customers. Each sales agent's laptop keeps a replica database. The laptops are ordinarily not connected to any network, but the agents can choose to perform anti-entropy exchanges between pairs of laptops or between a laptop and the primary replica. The laptops and primary replica do not have synchronized clocks.

A Bayou "write" in this system consists of a dependency/merge procedure along with the customer's name. When anti-entropy copies writes among replicas, it is the procedure and name that are copied, not the seat assignments. The procedure finds the lowest-numbered open seat and, if one is available, marks it as reserved for the customer. Here is the dependency/merge procedure:

```
merge_procedure(char *customer) {
  int s;
  for(s = 1; s <= 10; s++){
    if(seat s is not reserved){
      reserve seat s for customer;
      return;
    }
  }
  /* there are no seats left */
}
```

An example call in a replica's log could be merge_procedure("Smith"), which would result from an attempt to reserve a seat for customer Smith.

The Tasty has three sales agents, Ack, Nack, and Sack, and they sit next to each other. Each agent has a laptop containing a database replica, so there are four replicas in total. Each laptop displays, for each seat number, the name of the customer who has reserved it.

2. [5 points]: As our scenario opens, no seats are reserved. Then Agent Ack reserves a seat for Neha, and then a seat for Robert. Then Agent Nack reserves a seat for Charles. Then Agents Ack and Nack perform anti-entropy between their replicas. Their laptops now display seat reservations; a possible display might be "1: Neha; 2: Robert; 3: Charles." Only the actions described above have occurred. Does Bayou (the complete system with all mechanisms) guarantee that both laptops display the same assignments? Answer yes or no; if yes, explain the specific mechanisms that enforce the guarantee; if no, explain specifically what could cause them to differ.

Answer: Yes. Since the laptops have performed anti-entropy, they have identical logs. Furthermore, they will order their logs the same way, using Bayou's timestamps. Thus when they replay the logs after anti-entropy, they will replay the operations in the same order, and get the same results.

3. [5 points]: List all seat assignment displays that are possible.

Answer: There are three possible displays:

```
Neha Robert Charles
Neha Charles Robert
Charles Neha Robert
```

4. [5 points]: Now Professor Strongly Consistent approaches the agents and demands to be given seat 1. Agent Ack says "My laptop won't let me do that." Professor Consistent won't take that for an answer, and exclaims that surely Agent Sack can get him seat 1. Who is right: Agent Ack? Professor Consistent? Both? Neither? And why?

Answer: Both are right. Seat 1 is already assigned in Ack's laptop's database. However, seat 1 is not assigned on Sack's laptop. Thus Sack can tentatively assign seat 1 to the Professor. If Sack then performs anti-entropy with the primary replica, the assignment of seat 1 to the Professor will be irreversible.

III Dynamo

A Dynamo system is configured to use N=3 and W=3 (see Sections 4.5 and 4.6 in the paper). Suppose at time T1 all replicas of key "x" contain value "y". The system operates until time T2 arrives. No put()s or failures occur after T2. After T2, client C1 notices that it gets a different result from get("x") depending on whether it uses R=2 or R=3.

5. [10 points]: Describe a specific sequence of events between T1 and T2 that would cause this behavior.

Answer: Suppose server S10 is responsible for key "x", and that the servers after S10 on the ring are S11, S12, S13, and S14. After T1, S10 and S11 are temporarily disconnected from the network. A client executes put("x","a"), which is sent to S12, S13, and S14. Just before T2, S10 and S11 are re-connected to the network. Now a get("x") with R=2 will yield the value "y", while a get("x") with R=3 will yield "a". The reason the latter get() won't yield "y" along with "a" is that the version vectors will reveal that "a" supersedes "y".

IV Two-Phase Commit or Paxos?

During the Argus lecture, the question came up of whether one could use Paxos instead of two-phase commit to provide atomic distributed commit. That is, instead of participant machines answering "yes" or "no" to a two-phase commit prepare message depending on whether they are willing to commit, the participants could propose "yes" or "no" in a Paxos agreement.

6. [10 points]: Why doesn't that work?

Answer: Atomic commit is required to abort if any of the participants doesn't want to commit. Paxos is free to choose any of the proposed values for agreement. So if some participants propose "yes" and some propose "no", Paxos may choose "yes", even though atomic commit requires the outcome to be "abort."

V Argus

Consider the bank example in Figures 6 and 7 of Liskov's 1988 Argus paper. Suppose the bank has three branches (B1, B2, and B3), each with its own computer running a branch guardian. The bank also has two front ends, F1 and F2, each running a front-end guardian. There are three bank accounts, A1, A2, and A3, which are located at branches B1, B2, and B3, respectively.

Front-end guardian F1 starts a transfer topaction from account A1 to account A2 (see Figure 7). The withdraw() sub-action, sent to B1, succeeds. However, B2's network connection is broken, and is never fixed, so that B2 never responds to the deposit() message from F1.

Meanwhile, F2 starts a transfer topaction from A1 to A3. F2 starts this transfer after F1's withdraw() from A1 has completed.

All the accounts exist and have sufficient funds for the transfers.

7. [10 points]: Will F2's transfer topaction be able to complete and commit successfully? If it possible for it to commit, explain what has to happen before it can commit. If it is not possible for it to commit, explain why not.

Answer: Yes, F2's transfer will be able to commit. Initially F1's transfer has account A1 locked, so F2 cannot proceed. After a while the RPC from F1 to B2 will time out and fail, causing F1 to abort, which will release the lock on A1. Then F2 can proceed and commit.

VI MapReduce

Suppose you modify MapReduce so that when a worker finishes each Map task, the worker pushes each intermediate file generated by that task to the worker that will run the corresponding Reduce task, and then deletes the intermediate file to save disk space. Your new design will require changes to the way the manager assigns reduce tasks to workers. Keep in mind that a MapReduce computation is divided into many more map and reduce tasks than there are worker machines (Section 3.5).

8. [8 points]: Explain why this change would interact badly with the way that MapReduce load-balances tasks across workers.

Answer: MapReduce as described in the paper balances the reduce work across workers by allocating reduce tasks to workers lazily, as previous reduce tasks finish. But this change requires that map tasks be able to determine the worker for each reduce task before any of the reduce tasks execute. That would require the system to assign reduce tasks to workers ahead of time, making load balance harder.

9. [8 points]: Explain why this change would force MapReduce to do more work to recover from a failed reduce worker.

Answer: If a reduce worker failed, the only copy of the intermediate files for the reduce tasks it was responsible for would be gone. Regenerating them would require re-executing map tasks; in general all map tasks might have to re-executed.

VII Bitcoin

John sells tee-shirts on his web store, and he accepts Bitcoin. John also runs a Bitcoin peer, located in his apartment; he checks only this peer to learn about transactions (for example, payments from his customers). For a while everything is fine: John's peer hears new blocks and new transactions from the other peers in the Bitcoin system, and he successfully receives payments from his customers. One day John's ISP (Internet service provider) is taken over by Dark Forces who wish to trick John into accepting invalid Bitcoin transactions. The Dark Forces have complete control over the packets that John's Bitcoin peer exchanges with other Bitcoin peers; the Dark Forces can read these packets, modify the packets, and generate new IP packets that appear to be to/from other Bitcoin peers. The Dark Forces have no other special powers (for example, they cannot break cryptographic schemes, they have considerably less total CPU power than the rest of the Bitcoin network, and they can't break into John's computers).

> **10. [10 points]:** Explain how the Dark Forces can use their control over John's peer's network connection to trick John into accepting a payment that uses a Bitcoin that has already been spent.

Answer: Suppose the Dark Forces own bitcoin B, and that the latest block known to John's peer is BL100. The Dark Forces can transfer B to someone other than John, but only send that transfer to bitcoin peers other than John's peer; the Dark Forces would hide all blocks after BL100 from John's peer in order to conceal that transfer. Then the Dark Forces could create an alternate B101' containing a transfer of B to John; it might take a while for them to compute the required nonce, or require them to have many CPUs, though they don't have to have as many as the legitimate Bitcoin network. The Dark Forces could generate further alternate blocks B102', B103', etc. containing subsequent real transactions, to prevent John from noticing anything wrong. This attack "forks" the block chain, with one fork visible in the main bitcoin peer network, and the other fork visible only to John.

VIII Memcache at Facebook

Ben is building a new social networking web site. He reads the Scaling Memcache at Facebook paper, thinks the paper's design is much too complex, and invents a scheme which he believes will be simpler and faster. Ben's scheme involves just one cluster of front end and memcached servers; there are no regions, and no other clusters. As in the paper, Ben stores the "real" data in a set of database (DB) servers, and caches data in memcached for speed. Unlike the paper, when one of Ben's web servers needs to change some data, it first updates the data in the DB, and then writes the new data to memcached. That is, where the paper deletes the data from memcached (right half of Figure 1), Ben's scheme updates the data in memcached. Ben's DB servers do not send deletes to memcached (that is, Ben's scheme does not have the McSqueal mechanism shown in Figure 6). Ben's web servers read data just as described in the paper (see the left half of Figure 1).

Think of a specific situation in which Ben's scheme would serve stale data to a web server, but in which the paper's design would serve fresh data.

11. [8 points]: Describe the situation.

Answer: The following is for key "k". Client C1 writes value "x" to the DB, then writes "x" to memcache. Client C2 writes the value "y" to the DB, then writes "y" to memcache. If C1 is slow with its write to memcache, the system will end up with "y" in the DB but "x" in memcache. This inconsistency won't go away until the next time a client writes "k".

12. [8 points]: Explain why the paper's design would serve fresh data in that situation.

Answer: With the paper's design, C1 and C2 delete key "k" from memcache, rather than updating it. So if C1 is late with its delete, that may delete a valid value from memcache, but the next reader will see nothing in memcache and fetch the correct value "y" from the DB.

IX 6.824

13. [3 points]: How could we improve the 6.824 project experience? Should there be more time for the project (and less for the labs)? Or would you prefer to see no project and more labs?

Answer: Votes for more emphasis on projects: 9. Votes for more emphasis on labs: 25.

End of Quiz II Solutions