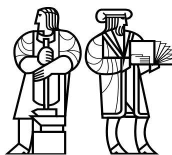


Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.824 Distributed System Engineering: Spring 2010

Quiz II Solutions

All problems are open-ended questions. In order to receive credit you must answer the question as precisely as possible. The quiz is designed to take 80 minutes, but you have two hours to complete it.

Write your name on this cover sheet AND at the bottom of each page of this booklet.

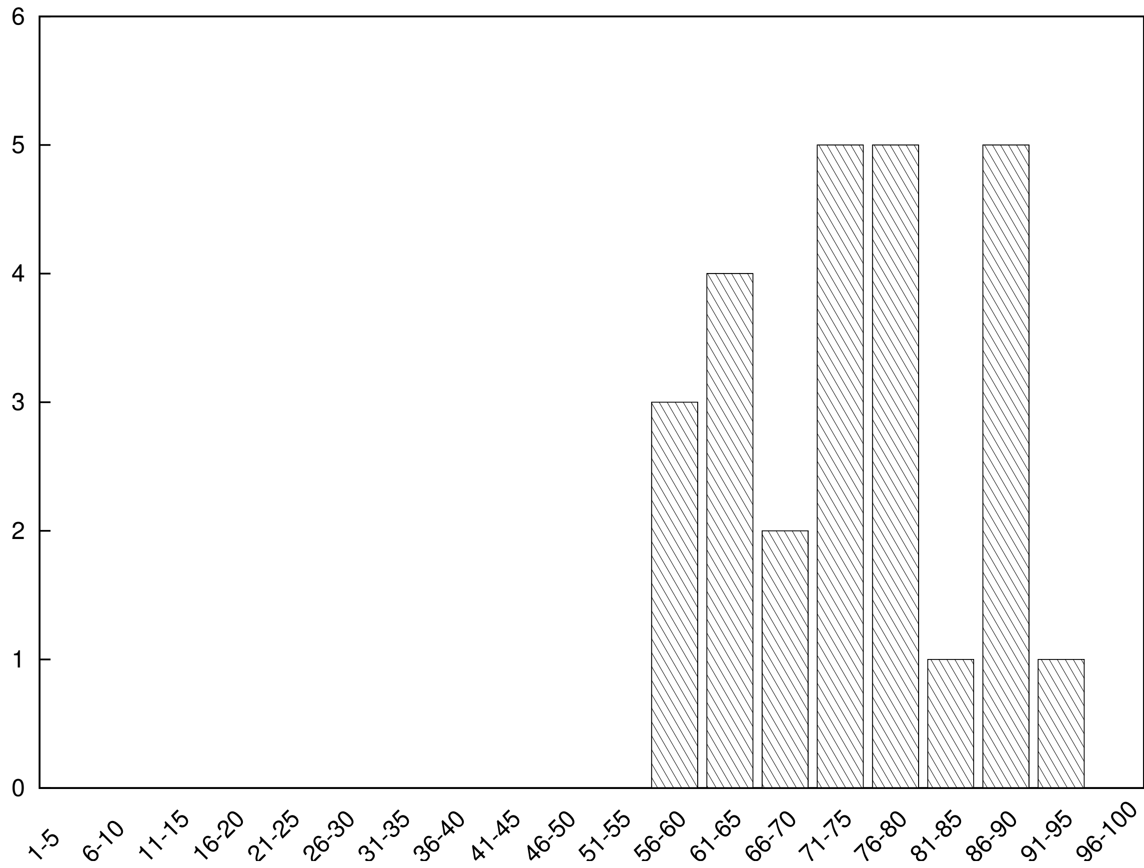
Some questions may be much harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ.

I (xx/36)	II (xx/14)	III (xx/48)	IV (xx/2)	Total (xx/100)

Name:

Grade histogram for Quiz 2



$$\begin{aligned} \text{max} &= 92 \\ \text{median} &= 75 \\ \mu &= 74.7 \\ \sigma &= 9.9 \end{aligned}$$

I YFS

1. [6 points]: Ben is working on Lab 6 and is having problems with his `create` function. Here it is omitting error handling code:

```
int yfs_client::create(inum di, std::string name, int dir, inum &ci)
{
  std::string extent;
  lc->acquire(di);
  ci = ilookup(di, name);
  if (!ci) {
    ci = random() & (~0x80000000);
    if (!dir)
      ci |= 0x80000000;
    ec->get(di, extent);
    /* add name and ci to extent */
    ec->put(di, extent);
    ec->put(ci, "");
  }
  lc->release(di);
  return OK;
}
```

Ben's code passed all test cases when he turned in Lab 5, but with his Lab 6 additions he can't create files correctly. Ben starts two `yfs_clients`, yfs1 and yfs2, creates a file in yfs1 and verifies that it is created correctly; however, he can't open the file in yfs2. What is he doing wrong? Why did his code work in Lab 5 and not in Lab 6?

Answer: Ben must add `lc->acquire(ci)` and `lc->release(ci)`, so that on an invalidate for lock `ci` the corresponding extent will be flushed.

Name:

2. [6 points]: Here is the heartbeat handler function that responds to a heartbeat RPC:

```
paxos_protocol::status
config::heartbeat(std::string m, unsigned vid, int &r)
{
  assert(pthread_mutex_lock(&cfg_mutex)==0);
  int ret = paxos_protocol::ERR;
  r = (int) myvid;
  printf("heartbeat from %s(%d) myvid %d\n", m.c_str(), vid, myvid);
  if (vid == myvid) {
    ret = paxos_protocol::OK;
  } else if (pro->isrunning()) {
    assert (vid == myvid + 1 || vid + 1 == myvid);
    ret = paxos_protocol::OK;
  } else {
    ret = paxos_protocol::ERR;
  }
  assert(pthread_mutex_unlock(&cfg_mutex)==0);
  return ret;
}
```

Below is a list of events triggered by node C trying to join. The starting view is 2 and contains A and B where A is the primary;

- C asks A to join.
- A starts Paxos to decide on view 3 with A,B,C in the view; A goes through the prepare and accept phases.
- In the decide phase, A sends a `decidereq` to itself and moves to view 3.
- B's heartbeat timer expires and the config code sends a heartbeat to A.
- A sends a `decidereq` to B.
- A receives the heartbeat from B and replies with ___________.
- B receives the `decidereq` and moves to view 3.
- B receives the response to the heartbeat from A.

Fill in the blank, what will A's heartbeat RPC handler (the function above) return? Circle the line of code that assigns the return value to `ret` in the code above.

Answer: It will reply with `OK`, since A is running Paxos, and `pro->isrunning()` will return true.

Name:

3. [8 points]: The high load on athena.lcs.mit.edu this year has revealed a bug in the YFS provided code. The problem is a race between the heartbeat RPC and the Paxos decide RPC. Below is a list of events that leads to this race. The starting view is 2 and contains A and B where A is the primary; node C is trying to join.

- C asks A to join.
- A starts Paxos to decide on view 3 with A,B,C in the view; A goes through the prepare and accept phases.
- In the decide phase, A sends a `decidereq` to itself and moves to view 3.
- A's heartbeat timer expires and the config code sends a heartbeat to B.
- A sends a `decidereq` to B.
- B first receives the heartbeat and replies with `ERR`, since it is still in view number 2.
- B then receives the `decidereq` and moves to view 3.
- A receives the response to the heartbeat from B and starts Paxos again to remove B from view 3.

How should the YFS code be changed to address this race so that B is not eliminated from the view?

Answer: A few different solutions are possible. For example, A can sends the `decidereq` to itself last, avoiding the race. As another example, A can note that it is running the proposer and return `OK` to B's hearbeat in that case.

4. [8 points]: Ben suggests eliminating sequence numbers in the YFS RSM protocol. He plans to remove sequence numbers sent in the RPCs between the lock server primary and the lock server backups. Give an argument why Ben's proposed change is correct or incorrect.

Answer: The sequence numbers in the replication protocol between the primary and backups can be eliminated, because the primary won't send the next request to the backups until it has received results from the backups for the current request.

Answer: The sequence numbers on lock requests cannot be eliminated, because the backups may have processed the lock request, but the client may not learn about the result because the primary may fail before responding to the client. In that case, the client will retry and resend its request. The replicated lock service must be able to detect that this request is a retry, so that it can respond appropriately.

5. [8 points]: After every view change, the YFS lock servers choose the primary according to these rules: If the primary in the previous view is a member of the new view, then it will stay the primary. Otherwise, the node with the lowest-numbered ID of the previous view will be the new primary. In either case, the new primary will be a node from the previous view.

Ben proposes to simplify the rule so that the lock servers always choose the node with the smallest-numbered ID as the primary. Give a scenario that shows why the simplified rule isn't a good idea. (Draw a timing diagram with a brief explanation.)

Answer: Assume 3 nodes, A, B, and C, with A being the lowest ID. A, B, and C are in a view processing requests. A fails. B and C form a new view, and process new requests, resulting in a lock table L. A rejoins, it will be the new primary and with B's simple rule, B and C download A's lock state, losing the state of L.

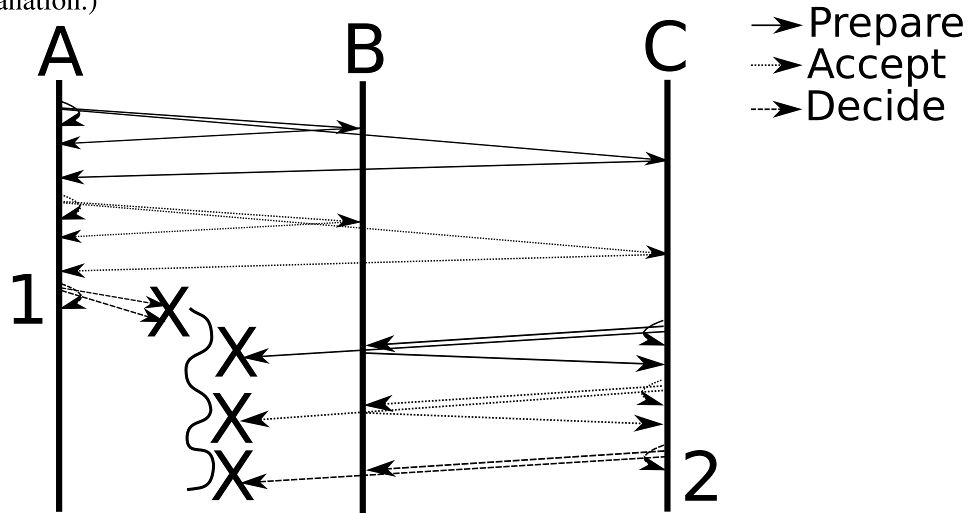
II Paxos in YFS

Here are the proposer `run` and `prepare` functions omitting all error handling code:

```
1  bool proposer::run(int instance, std::vector<std::string> c_nodes,
2                     std::string newv) {
3    std::vector<std::string> accepts, nodes;
4    std::string v;
5    bool r = false;
6    pthread_mutex_lock(&pxs_mutex);
7    /* call setn() and set any global state */
8    v.clear();
9    if (prepare(instance, accepts, c_nodes, v)) {
10     if (majority(c_nodes, accepts)) {
11       if (v.size() == 0)
12         v = newv;
13       nodes = accepts;
14       accepts.clear();
15       accept(instance, accepts, nodes, v);
16       if (majority(c_nodes, accepts)) {
17         decide(instance, accepts, v);
18         r = true;
19       }
20     }
21   }
22   pthread_mutex_unlock(&pxs_mutex);
23   return r;
24 }
25 bool proposer::prepare(unsigned instance, std::vector<std::string> &accepts,
26          std::vector<std::string> nodes, std::string &v) {
27   prop_t highn;
28   highn.n = 0;
29   highn.m = me;
30   for (unsigned i = 0; i < nodes.size(); i++) {
31     paxos_protocol::preparearg a;
32     paxos_protocol::prepareres r;
33     /* setup argument a, zero out r */
34     pthread_mutex_unlock(&pxs_mutex);
35     handle(nodes[i]).get_rpcc()->call(paxos_protocol::preparereq,
36                              me, a, r, rpcc::to(1000));
37     pthread_mutex_lock(&pxs_mutex);
38     if (r.oldinstance || !r.accept) {
39       /* handle old instance or not accepted */
40     } else {
41       accepts.push_back(nodes[i]);
42       if (r.n_a > highn) {
43         highn = r.n_a;
44         v = r.v_a;
45       }
46     }
47   }
48   return true;
49 }
```

Name:

6. [6 points]: Ben removes lines 42 through 45; Alyssa thinks that this change is a bad idea. Give a scenario that shows it is a bad idea to remove those lines of code. (Draw a timing diagram with a brief explanation.)

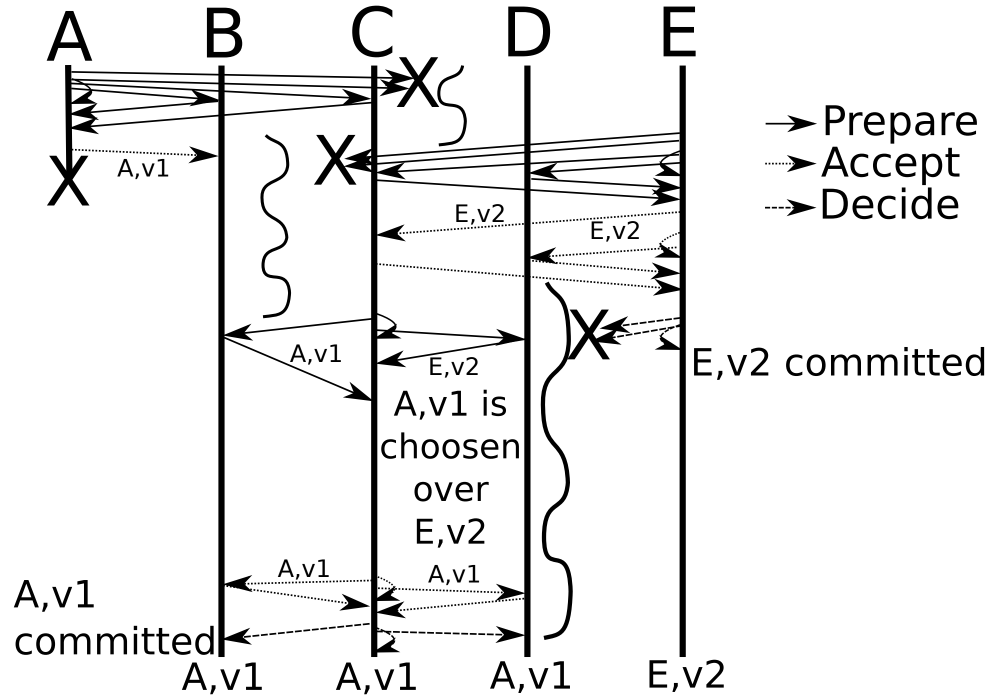


Answer:
The scenario needs to include two proposing nodes. In the above diagram, node A proposes value $v1$ and C value $v2$. A proposes and sends accepts, and nodes A, B, C accept. A sends decides to A, B, C but is partitioned and only A decides on $v1$ (marked as time 1 on the diagram). Now C proposes $v2$ to A,B,C. B and C reply with $v1$ in their proposes reply messages, but node C ignores them and sends accepts on $v2$; node B, C accept. C sends out decides to B, C (marked as time 2 on the diagram). Now when the partition heals, A and C have decided on two different values.

7. [8 points]: Ben rethinks the situation and replaces lines 42 through 45 with these lines instead:

```
if (r.n_a.n > 0)
  v = r.v_a;
```

Give a scenario that shows that Paxos fails to meet its specification with Ben's change. (Draw a diagram showing the Paxos messages.)



Answer:
The scenario must again involve two concurrent proposers. With the change the protocol will not stick to the highestest-numbered proposal. One proposer may chose a lowered-number proposal while another proposer may chose a higher-numbered proposal. As a a result, proposers commit to different values. The timing diagram illustrates an example.

Name:

III Short paper questions

8. [6 points]: Consider the mailer system written for Argus as described on pages 398–400 of the paper. Suppose Alyssa sends an email to a list (alyssa, bob, and charles), while at the same time a system administrator is creating an account for charles (using `add_user`). Given that mailboxes for each user may be on different machines and Charles's mailbox may exist or not, what are the possible outcomes of who will receive Alyssa's message? (Explain your answer briefly.)

Answer: The mailer will deliver to all three or none, because of the two-phase commit protocol and the subactions that Argus uses.

9. [6 points]: A local bank runs a hypervisor-based fault tolerant system as described by Bressoud and Schneider. One night the cleaner trips over the cables in the machine room and accidentally disconnects the primary from the backup (but both can still communicate with the shared disk). Does the system keep operating correctly? (Explain your answer briefly.)

Answer: No. Because the communication is broken, the backup will assume the primary has failed, and thus become a primary. Now the system runs with two primaries, both writing to the shared disk.

10. [6 points]: The same cleaner trips over a wire in a Frangipanni system (as described by Thekkath et al.) and partitions the network. Could more than one file server, in different partitions, successfully acquire the same lock concurrently and perform updates? (Explain your answer briefly.)

Answer: No. Only the partition with the majority of the nodes will have a lock server that responds to lock requests, and the server in that partition is able to acquire the lock. In addition, the system's use of leases on locks ensures that locks will be "released" after the lease time expires.

11. [6 points]: Given the protocol in the Chord paper, is `lookup(k)` guaranteed to return the IP address of the node that stores k? (Explain your answer briefly.)

Answer: No. During stabilization, a node responsible for a key may not have that key yet. The application is supposed to retry when a lookup fails.

12. [6 points]: Alyssa modifies the PBFT protocol described by Castro and Liskov as follows. Instead of sending a set of 2f+1 responses to the client, she sends only f+1. Describe a scenario under which Byzantine nodes can fool the client. (Explain your answer briefly.)

Answer: The paper describes a slightly different protocol than the lecture notes, but they boil down to the same thing. With f+1 responses, the client cannot decide what the valid response is from the f+1. The f faulty nodes can outvote the valid response.

13. [6 points]: Alyssa adds a SUNDR-like feature to PBFT. Clients know the public keys of all other clients that are using the system. A client signs the data before sending it to the PBFT RSM—you may assume that clients are not compromised. After a client receives n responses from the RSM, the client verifies the responses with the public key of the client. How many responses at minimum must a client receive before the client knows it has received a correct response (i.e., meeting the specs of a non-faulty replicated state machine)? (Explain your answer briefly.)

Answer: The client can verify which response has valid data (i.e., signed by another client.), but the client must still receive 2f+1 responses to ensure that it can determine the order for responses.

14. [6 points]: Alyssa logs in to Yahoo!, updates her entry in the user database, and views her entry. Sketch the application code for Alyssa's update and read operations in terms of the primitives listed in Section 2.2 of the PNUTS paper. The code must meet the user database specification described in Section 4. (Sketch the pseudocode.)

Answer: Alyssa update to her profile:
v = write(profile)
read-critical(v, profile)

15. [6 points]: Continuing from the previous question, describe a scenario under which Ben might not see Alyssa's change, even if he looks up Alyssa's page after Alyssa made the change. (Draw a timing diagram with a brief explanation.)

Answer: Ben's code uses read-any(Alyssa's profile). If Alyssa updates her profile, and if the network between Ben and Alyssa partitions, any-read may read Alyssa's profile from a storage node in Ben's partition that hasn't been updated yet.

Name:

IV 6.824

16. [2 points]: Did you learn anything in 6.824? Please give a score on a scale from 0 (nothing) to 10 (more than I had hoped for), and briefly explain.

Answer: Thanks for taking the class! Enjoy the sumer.

End of Quiz II