Computer Science Department

New York University

G22.3033-006 Distributed Systems: Fall 2008

Quiz II

All problems are open-ended questions. In order to receive credit you must answer the question *as precisely as possible*. You have 110 minutes to answer this quiz.

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/20)	II (xx/20)	III (xx/30)	IV (xx/15)	V (xx/15)	VI (xx/5)	Total (xx/100)

Please write down your name and NYU ID:

I Two-phase commit

Ben Bitdiddle and Alyssa Hacker are arguing about the relative merits of the classic two-phase commit protocol and the modified version used in Sinfonia. For concreteness, they examine a scenario where client C initiates a single transaction involving two servers A and B (servers are also referred to as memory nodes in Sinfonia).

1. [10 points]: Suppose the *one way* delay between any pair of nodes is r, how long does it take for client C to learn of the transaction's result using the classic two-phase commit protocol? How long does it take using Sinfornia's protocol? (Draw a message time diagram to illustrate both cases)

2. [5 points]: Enumerate all scenarios under which the classic two-phase commit protocol fails to make progress (i.e. the transaction neither succeeds nor aborts). Similarly, enumerate all scenarios when Sinfonia's 2-phase protocol fails to make progress.

3. [5 points]: How can you augment the basic two-phase commit protocol so that the system does not block under any single server's failure. (Hint: there is no need to design new algorithm or new protocol. You should be able to apply existing techniques from this class for the augmented design.)

II Replicated state machine

Ben finds Paxos very complicated and decides to devise a simpler protocol to handle the master failure in RSM. In Ben's proposal, the RSM replica set consists of a *fixed* list of n servers ($S_0, S_1, ..., S_{n-1}$). There are no new nodes joining the system and there are no nodes leaving the system. Note that his setup is different from that in Lab 6 (the RSM of Lab 6 allows new nodes to join an existing replica set). The RSM starts with the manually configured master S_0 at all servers. His proposed plan works as follow:

- The master processes clients' requests in the same fashion as Lab 6: it assigns sequence numbers to each client request and sends *invoke* messages to all servers. The master processes a client's request and returns the reply to the client if at least half of the servers in the fixed replica set have processed the request.
- When a client fails to contact the current master, it issues its request to the current master's successor. Recall that a successor is a node whose ID is the closest to the node in the wrapped-around ID space (i.e. S_0's successor is S_1. S_1's successor is S_2. S_{n-1}'s successor is S_0). A server that receives a client's request immediately sets its status to be the master and processes the client's requests as usual.

4. [10 points]: Assume the network is asynchronous (i.e. it can partition nodes and delay messages arbitarily). does Ben's proposal guarantee that all RSM operations are performed in a single total order? Explain and illustrate with an example.

5. [10 points]: Frustrated, Ben decides to revert to the RSM design in Lab 6. However, he changed how the master processes a client's request. His modified master processes a client's request locally first and returns the result to the client before waiting for replies from other replicas. Does his modified RSM still guarantee that all RSM operations are performed in a single total order? Explain and illustrate with an example.

III Paxos

In the appendix of this quiz, you can find the Paxos pseudocode as described in Lab5.

6. [10 points]: Ben proposes to remove the first round of message exchange in Paxos (i.e. skip phase involving `prepare` messages). In his proposal, a leader starts Paxos by sending `accept(vid+1,my_n,v)` to all nodes in the current view. If a majority of nodes reply with ”yes”, the leader declares the new view as being formed and proceeds to send out `decide(vid+1,v)` messages to all nodes. Is his protocol still correct? (Give a brief explanation and illustrate your answer with an example.)

7. [10 points]: In Paxos, each replica must log the Paxos state involving `n_a`, `v_a` and `n_h` as well as log every view change. Will Paxos remain correct if `n_a` and `v_a` are not logged? Explain briefly with an example.

8. [5 points]: Paxos does not guarantee liveness. In other words, it is possible that Paxos never terminates under a sequence of events. Give a concrete example in which Paxos never reaches agreement on a new view after a couple of retries.

IV Byzantine Fault Tolerant

Ben thinks the PBFT (Castro and Liskov) protocol is too expensive for his taste and decides to reduce the number of messages sent in the BFT protocol.

9. [10 points]: A quorum is a subset of nodes among n total nodes that return matching replies. The PBFT protocol requires a node to obtain a quorum of size $\lfloor 2n/3 \rfloor + 1$ for prepare and commit messages. Is it still correct if Ben modifies PBFT to use a quorum size of majority nodes (i.e. quorum size $\lfloor n/2 \rfloor + 1$). Briefly explain and give a concrete example to illustrate your argument.

10. [5 points]: In the A2M system proposed by Byung-Gon Chun et al., it is enough for a node to obtain a majority quorum while defending against no more than $\lfloor n/2 \rfloor - 1$ Byzantine nodes. Briefly explain why this is the case and illustrate why the example you gave in problem 9 is no longer a concern when using A2M.

V SUNDR

11. [5 points]: In SUNDR, each file is represented as a 20-byte hashed file handle. Is it necessary for a client to digitially sign individual file blocks to ensure the untrusted server does not maliciously modify them? Explain. Is it necessary for a client to digitially sign its version structure? Explain.

12. [10 points]: In SUNDR, why does a client increase its version number in the signed version structure upon a *fetch* operation (which does not modify data)? Briefly explain and provide an attack that violates *fork consistency* if a client does not increase its version number for *fetch* operations.

VI G22.3033-006

13. [5 points]: Did you enjoy learning the materials in this class? What aspects of the class can be further improved for the next offering?

VII Appendix

The following is the pseudocode of Paxos, as described in Lab 5.

```
state:
  n_a, v_a: highest proposal # and its corresponding value this node has accepted
  n_h: highest proposal # seen in a prepare
  my_n: the last proposal # the node has used in this round of Paxos
  vid_h: highest view number we have accepted
  views: a map of past view numbers to values
  stable: ‘‘false’’ when running Paxos, ‘‘true’’ when agreement has been reached for the cu

on each view change, initialize state
  n_a = 0
  n_h = 0
  my_n = 0
  v_a = () // empty list

when a node initiates Paxos (upon join or detecting failures),
  stable = false
  proceed to Paxos Phase 1

Paxos Phase 1
  a node (maybe more than one.) decides to be leader (need not be in current view):
    my_n = max(n_h, my_n)+1, append node ID  // unique proposal number
    sends prepare(vid_h+1, my_n) to all nodes in views[vid_h], initial contact node, itsel

  if node receives prepare(vid, n):
    if vid <= vid_h:
      return oldview(vid, views[vid])
    else if n > n_h:
      n_h = n
      stable = false
      return prepareres(n_a, v_a)
    else:
      return reject()

Paxos Phase 2
  if leader gets oldview(vid, v):
    views[vid] = v
    vid_h = vid
    view change
    restart paxos
  else if leader gets reject():
    delay and restart paxos
  else if leader gets prepareres from majority of nodes in views[vid_h]:
    if any prepareres(n_i, v_i) exists such that v_i is not empty:
      v = non-empty value v_i corresponding to highest n_i received
    else leader gets to choose a value:
      v = set of live nodes (including self)
    send accept(vid_h+1, my_n, v) to all responders
  else:
    delay and restart paxos
  if node gets accept(vid, n, v):
    if vid <= vid_h:
      return oldview(vid, views[vid])
    else if n >= n_h:
      n_a = n
      v_a = v
      return acceptres()
    else
      return reject()

Paxos Phase 3
  if leader gets oldview(vid, v):
    views[vid] = v
    vid_h = vid
    view change
    restart paxos
  else if leader gets acceptres from a majority of nodes in views[vid_h]:
    send decide(vid_h+1, v_a) to all (including self)
  else:
    delay and restart paxos

  if node gets decide(vid, v):
    if vid <= vid_h:
      return oldview(vid, views[vid])
    else:
      views[vid] = v
      vid_h = vid
      view change
      stable = true
```

End of Quiz II — Enjoy the winter break!