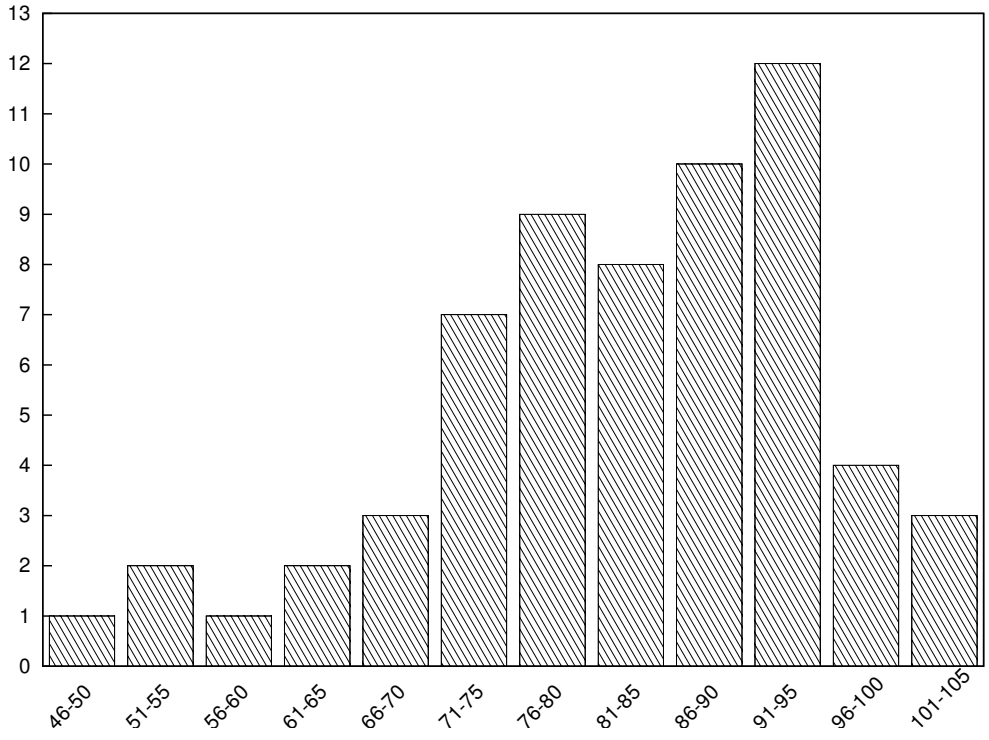
Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.824 Distributed System Engineering: Spring 2012

Quiz I Solutions

Grade histogram for Quiz I



min	=	49
max	=	104
median	=	84
average	=	82
σ	=	12

I Plan 9

1. [10 points]: The Plan 9 paper says (on page 2) that one of the principles underlying the design is that there is a standard protocol (9P) used to access all resources. It would have been more traditional to have each kind of resource (window system, file server, name server, remote login, etc.) define its own protocol. Does Plan 9 use a single protocol just for convenience, or is it a critical part of the architecture? Explain your answer.

Answer: Having the single 9P protocol is critical. It allows all resources to appear as files, and to be manipulable with standard file utilities as well as with file-system namespace tools such as exportfs.

II Dinner Server

Bifur and Bombur, graduate students in Course 12, go out to dinner together every evening. They sit in different offices, so they need a protocol to agree on the destination. They decide to write software that runs on their computers and communicates over the network, using the 6.824 lab RPC system. Each starts a `dinner` program on his computer in the morning. The program contains both a client and a server (it is a "peer-to-peer" system). Here is the software; note that on each computer, the client and the server code are using the same `state` and `m`.

```
enum rets { OK = 0, CONFLICT };
enum states { IDLE, SUGGESTING, AGREED } state;
pthread_mutex_t m;

int
client::suggest(std::string name)
{
  int r = -1;
  pthread_mutex_lock(&m);
  if(state == IDLE){
    state = SUGGESTING;
    cl->call(dinner_protocol::suggest, name, r);
    if(r == OK){
      state = AGREED;
      printf("decision: %s\n", name.c_str());
    } else {
      state = IDLE;
    }
  }
  pthread_mutex_unlock(&m);
  return r;
}

int
server::suggest(std::string name, int &r)
{
  pthread_mutex_lock(&m);
  if(state == IDLE){
    state = AGREED;
    printf("decision: %s\n", name.c_str());
    r = OK;
  } else {
    r = CONFLICT;
  }
  pthread_mutex_unlock(&m);
  return r;
}
```

Just two computers are involved (Bifur's and Bombur's). Each computer makes at most one call to `client::suggest`.

In a typical execution, Bombur gets hungry first and calls `client::suggest("Emma's")`. Bombur's client code sends an RPC to Bifur's program, causing Bifur's `server::suggest` handler to print `decision: Emma's` and return `OK`. In response to the `OK`, Bombur's client also prints `decision: Emma's`.

However, sometimes it takes a few minutes for the client `suggest` routine to complete. Bifur turns on RPC debugging output, and sees that the client RPC system is reporting timeouts, and that `call()` eventually gives up and returns an error. Bifur thinks perhaps the network is dropping packets, but on further examination it turns out that every packet is received. Neither computer ever crashes or malfunctions.

2. [10 points]: Explain the cause of the symptoms that Bifur sees.

Answer: The problem occurs when Bifur and Bombur call `suggest()` at about the same time. The client code on each of their computers locks the mutex, and then sends an RPC. The RPC handlers on both machines block waiting for the mutex. After two minutes, the RPC system gives up (since it has received no reply) and returns from `call()`. The client `suggest()`s then release the mutexes, allowing the server handlers to complete.

III Bayou

Victor is using a tablet computer that runs the meeting room scheduler described in Section 2.1 of the Bayou paper. He makes a reservation for 10:00, with 11:00 as the fallback time. After that, Vera makes a reservation for 10:00 on her own tablet, also with a fallback time of 11:00. Victor then performs anti-entropy with Vera's tablet, and sees that the scheduler application displays Vera's reservation at 10:00 and Victor's at 11:00. Both are marked tentative. Victor really wants 10:00, and he knows that the Bayou primary server commits operations in the order in which it sees them. So he hurries to the primary server before Vera and performs anti-entropy with it. All the computers involved execute Bayou as described in the paper, and no-one other than Victor and Vera is trying to reserve the meeting room.

3. [10 points]: At what time will Victor's committed reservation be? Explain why.

Answer: 11:00. His tablet knows about Vera's reservation, and knows that it is ordered before his reservation. When Victor performs anti-entropy with the primary server, his tablet will send the primary server all the tentative Writes it knows about, in order; this means his tablet will tell the primary about Vera's reservation before his. As a result the primary will assign Vera's reservation a lower CSN, and Vera's reservation will come first in the committed order as well as the tentative order.

As described on page 6 of the paper, each Bayou server uses a logical clock to time-stamp its new Writes, in order to preserve causal ordering. Vera hears about version vectors in 6.824, and in particular that version vectors can record causality more precisely than logical clocks. She considers modifying Bayou by replacing each server's logical clock with a version vector (the server's "vector clock"), and time-stamping new Writes with a version vector instead of with a logical clock value.

A version vector in Vera's new design would have an entry for each server in the system. When it learns about a new Write via anti-entropy, a server would set its vector clock to the element-wise maximum of the new Write's version vector and the server's current vector clock. When originating a new Write, a server would first increment its own element in its vector clock, and then set the Write's version vector equal to the vector clock. Thus the i'th element of a version vector indicates the number of Writes that server i has originated.

These version vectors are not the same as the Timestamp Vectors (O, C, and F) in Figure 4.

Vera's new rule for ordering tentative Writes is that Write x comes before Write y if $x_v < y_v$, where x_v and y_v are the version vectors associated with the two Writes. $x_v < y_v$ if every element of x_v is less than or equal to the corresponding element of y_v, and at least one element of x_v is strictly less than the corresponding element of y_v. Vera likes her new rule because $x_v < y_v$ only if the node that originated Write y had actually seen x. Logical clocks do not have this property.

Vera has not completed her design, but we'd like you to help her understand some of the implications of her ideas before she proceeds.

4. [10 points]: Describe a situation which would lead to two Writes having no order under Vera's new rule; that is, a situation in which her rule would order neither Write before the other.

Answer: If two Writes are created on different tablets without an intervening anti-entropy session, they will have version vectors that are neither less than nor greather than each other.

5. [10 points]: If two Writes have no order under Vera's rule, would it be OK for each server to use whichever order it likes? Explain why or why not.

Answer: It would not be OK. One of Bayou's goals is that two computers that perform anti-entropy, and thus have identical sets of Writes, should end up with identical databases. That would not happen if they ordered the same set of Writes in different ways.

IV Argus

Refer to the 3rd paragraph of page 385 of the Argus paper, which discusses Argus' use of two-phase commit. Suppose a top-level action successfully completes phase one (all participating guardians have responded positively), but then one guardian crashes and quickly restarts before the start of the second phase. Nothing else goes wrong.

6. [10 points]: Will the top-level action abort or commit? Explain your answer.

Answer: It will commit. Once a participant agrees to commit, it cannot change its mind, since others may then proceed to commit. Thus before a guardian responds positively in phase one, it must write to disk all information that would be required if the action ends up committing, including all modified values. The guardian writes the modified values to a temporary area on disk, not to the permanent locations, since the action has not yet committed. The guardian must record which locks are held in order that they still be held after restart.

An Argus sub-action's commit is conditional – it doesn't really commit unless all its ancestor actions commit (see the first paragraph on page 386). Suppose, instead, Argus sub-actions fully committed when they finished, so that a sub-action's updates were visible even if the parent action aborted.

7. [10 points]: Describe a concrete example where having sub-actions commit even if their ancestors don't would be an awkward design, and explain why.

Answer: Suppose a bank's guardian defines handlers for debiting and crediting accounts. It would be good to be able to combine them into a top-level action that transfers money from one account to another. However, if the debit sub-action commits and the credit sub-action aborts, the account balances will be left in an inconsistent state.

V Tra

Suppose each element in Tra's synchronization time and modification time version vectors held a sequence number rather than a time. That is, suppose that the i'th element in file f's modification time version vector indicated the number of times that replica i had modified file f.

8. [10 points]: This idea turns out not to work. Explain what would go wrong with the scheme in the paper's Section 3.4.1 if version vectors were defined in this way.

Answer: One problem is that Section 3.4.1's definition of directory modification time as the element-wise maximum of the childrens' modification times does not work: different files' version vectors do not have comparable elements if they hold sequence numbers.

This question is a little confusing because it does not state a definition of synchronization time.

VI Piccolo

The Piccolo paper says, in the 3rd paragraph of page 5, that `graph`, `curr`, and `next` are grouped together in Figure 2. This means that all three tables are partitioned across computers by key in the same way, so that a given key in all three tables is in the same partition. Here is a copy of Figure 2, with checkpointing and restore code omitted:

```
tuple PageID(site, page)
const PropagationFactor = 0.85

def PRKernel(Table(PageID, double) curr,
               Table(PageID, double) next,
               Table(PageID, [ PageID ]) graph_partition):
  for page, outlinks in
      graph.get_iterator(my_instance()): <<<--- (B)
    rank = curr[page]
    update = PropagationFactor * rank / len(outlinks)
    for target in outlinks :
      next.update(target, update) <<<--- (A) (B)

def PageRank(Config conf):
  graph = Table(PageID, [ PageID ]).init("/dfs/graph")
  curr = Table(PageID, double).init(
                graph.numPartitions(),
                SumAccumulator, SitePartitioner)
  next = Table(PageID, double).init(
                graph.numPartitions(),
                SumAccumulator, SitePartitioner)

  GroupTables(curr, next, graph)

  for i in range(last_iter, 50):
    Run(PRKernel,
          instances = curr_pr.numPartitions(),
          locality = LOC_REQUIRED(curr),
          args = (curr, next, graph))

    Barrier()

    swap(curr, next)
```

9. [10 points]: Identify all the places in Figure 2 where reads or modifications of tables require data to be moved between computers, assuming that every partition is on a different computer. Mark these places in the code on the previous page with a circled "A".

Answer: The call to `next.update()`.

10. [10 points]: Suppose the partitioning of `graph` were changed so that its keys were *not* grouped together with those of `curr` and `next`. With this new partitioning, identify all the places in Figure 2 where reads or modifications of tables would require data to be moved between computers, assuming that every partition is on a different computer. Mark these places in the code on the previous page with a circled "B".

Answer: The call to `next.update()`, and the call to `graph.get_iterator()`.

VII 6.824

11. [2 points]: What aspect of 6.824 would you most like to see changed?

Answer: Too many papers. Please answer the paper question in every lecture.

12. [2 points]: What is the best aspect of 6.824?

Answer: The labs.

End of Quiz I Solutions