6.824 2021 Midterm Exam Answers

MapReduce

A. Better performance: reducers can collect intermediate files while mappers are still running. Since network bandwidth is a bottleneck and the files are large, allowing workers to start early and read the intermediate files from remote machines as they become available can be a significant improvement: by the time a reducer worker can runs a reduce task some of its inputs already have been copied over the network.

B. When there are idle workers with no map tasks to do, the coordinator can assign them to reduce tasks. Those reducer workers look for intermediate files and copy them over the network. Since mappers make the intermediate files available using an atomic instruction, the reducer will read the complete mapper output. The reduce worker can keep track of how many intermediate files it has copied, and once it has copied all of them, run the reduce task. There is an additional opportunity for performance improvement: the reduce worker can sort the intermediate input files incrementally, and merge in new files as they come in.

MapReduce (Lab)

A. One possible scenario:

- Worker 0 has completed map task 0 and sends a Finished Task to the coordinator.
- The coordinator updates `c.numCompletedMapTasks` to 1
- Worker 1 is assigned map task 1 by the coordinator.
- Worker 2 is assigned map task 2 by the coordinator.
- Worker 2 finishes map task 2. c.numCompletedMapTasks is updated to 2.
- Worker 1 fails and never executes map task 1, failing to write any of the `map-1-x` intermediate files.
- Worker 0 sends a GetTask RPC.
- The coordinator's `HandleGetTask` starts checking for tasks starting at task 2, and sees that `c.mapWhen[2]` has timed out. Worker 0 is sent task 2.
- Worker 0 finishes task 2 and sends a `FinishedTask` to the coordinator.
- The coordinator updates `c.numCompletedMapTasks` to 3, but this accidentally counted a task twice, while not noticing that particular task failed.

B. Zara should keep track of which *unique* map tasks have completed, since tasks can complete out-of-order. Instead of using an integer count to track completed tasks, which fails to distinguish which tasks have finished, the coordinator needs to track each individual map task's `Done` status, updating this status in `HandleFinishedTask`.

Zara should change the `HandleGetTask` task assignment loop to iterate through unfinished tasks, instead of starting from the count of finished tasks, and should only move on to reduce tasks when all map tasks are `Done`.

GFS

A. In linearizable implementation there wouldn't be any need for application-level checksums, unique identifiers, or checkpoints. Checkpoints may be needed when overwriting a file; checksums and unique identifiers may be needed in records to handle duplicates and padding during record-append operations.

B. This is to ensure that appends are atomic. If a record were to span a chunk boundary then the append would be broken up in two writes and those might go to different primaries.

VM-FT

In replicated state machine approach, the backup must always end up in an identical state as the primary so that the backup can take and produce the same results for client requests. If networks interrupts are delivered at different instructions, the backup's state could diverge from the primary and cause the backup to respond differently to client requests than the primary would once the backup takes over. For example, the network interrupt may deliver a packet to the OS that causes the OS to change the state of the OS and/or application (e.g., a file may come into existence).

Raft

Yes. They must have term 3. They couldn't have term 4 because 4 is committed, which would have committed the other entries of the last follower, which is inconsistent with the followers who committed 4.

No points were deducted for saying that 2 is a possibility **in addition** to term 3 with correct explanation: if the server didn't have any term 3 entries and received more term 2 entries when it was leader for term 2.

One common idea is that part of the server's log could have been deleted and replaced with the committed entries (when it got back in contact with the current leader), so any term >= 8 was possible. This is correct, so no points were deducted if this was **in addition** to stating term 3, but ultimately not what the question was asking.

Raft (Lab)

A. 5 No failures means that everything functions normally, each will send once.

Full credit was given for saying that a slow network would give 0 as well, if the leader receives the entry but another leader is elected before the original sends it out.

B. 0, 1, 2, 3, 4, 5

- 0 if the leader is partitioned and receives a Start that ultimately gets deleted when it rejoins.
- 3, 4 if 1 or 2 peers are partitioned forever and the rest of the peers make progress
- 5 if everything functions normally or partitioned servers rejoin and catch up
- 1 or 2 are possible if peers receive entries to append, reply to the leader saying they've successfully appended, but are partitioned prior to receiving the updated commit index from the leader in the next heartbeat.

C. Any value from 0 to infinity

Without snapshots, upon a crash the servers must replay the entire log and re-send all values through the applyCh. This means that a server can always crash right before sending, thus never sending (this is how you would get 0-5). Otherwise, anytime a server crashes it has to replay, meaning every crash is another send, causing the value to be unbounded.

Credit was given if certain assumptions (network partitions always heal, crashed servers eventually remain uncrashed long enough to catch up) were **explicitly stated** and the correct answers were given under those assumption.

Raft (persistence)

A: It's not correct, the entire length of the log, including uncommitted entries, is used as part of Raft's election restriction so this can break the Leader Completeness Property. Moreover, whenever a follower confirms an AppendEntries, the leader might decide to commit entries that the follower has not yet committed under the assumption that the follower will retain those entries. If these uncommitted entries are not persisted, State Machine Safety is broken.

A sequence of events that show this is not OK:

- There are three servers: S0, S1 & S2. S0 is the leader. All are up to date (same term & log)
- S0 gets a new command X from the client and replicates it to S1 & S2.
- Upon receiving confirmation, S0 decides to commit the entry and apply it (with index i).
- Before the next AppendEntries heartbeat (which would tell the followers to commit the last entry), all servers crash.
- S1 & S2 come back up but S0 doesn't. Either S1 or S2 is elected leader since they have the same log.
- The leader gets a new command Y from the client, replicates it to the other server and applies it.
- Since S1 and S2 did not persist the uncommitted entry X, the newer entry Y is applied at the same index i.

Raft peers have now applied different log entries for the same index, violating State Machine Safety.

Zookeeper

A. No. The read operations still run all through the primary.

B. Read f of client 2 isn't guaranteed to see the write of f by client 1, so possible values are 0 and 1.

6.824

A. Which papers/lectures should we omit in future 6.824 years, because they are not useful or are too hard to understand?

Papers to omit

- MapReduce: 1%
- Q&A Lecture MapReduce: 36%
- GFS: 10%
- VMware FT: 27%
- Raft: 1%
- ZooKeeper: 17%
- Q&A Lecture Raft: 13%

B. Which papers/lectures did you find most useful?

- MapReduce: 58%
- Q&A Lecture MapReduce: 13%
- GFS: 32%
- VMware FT: 25%
- Raft: 91%
- ZooKeeper: 39%
- Q&A Lecture Raft: 32%

C. What should we change about the course to make it better?

Summarized feedback is generally ordered from most-received comment to least.

General course feedback:

- What is good: Reading case studies, interactive Q/A in lectures, well-structured
- More hands on experience with some of the other papers we read
- Read more papers earlier in the class
- More OH (weekends OH, later)
- More guest lectures from industry; guest lecture by Diego Ongaro or John Ousterhout!
- Q/A lectures for psets not useful, recitation-style review of labs instead of Q/A lectures
- More Q/A lectures for psets
- More practice writing test cases for distributed systems
- Add reviews for exams, nanoquizzes
- No raw txt files for notes

Raft lab feedback:

- Labs too dependent on each other
- Failing tests should give more info, tests should be more comprehensive
- More starter code
- 2D control flow/spec difficult to understand
- More scripts provided in handout code
- More talk about implementation of lab in lecture
- More visualizations
- More collaboration possibilities