Stanford University
Computer Science Department
CS 240 Quiz 2 Answers
Spring 2006

June 14, 2006

This is an open-book exam. You have 50 minutes to answer 8 out of 10 questions. Write all of your answers directly on the paper. Make your answers as concise as possible. Sentence fragments ok.

NOTE: We will take off points if a correct answer also includes incorrect or irrelevant information. (I.e., don't put in everything you know in hopes of saying the correct buzzword.)

Question	Score
1 - 5	
6 - 10	
total	

Stanford University Honor Code

In accordance with both the letter and the spirit of the Honor Code, I did not cheat on this exam nor will I assist someone else cheating.

Name and Stanford ID:

Signature:

Answer eight of the following ten questions and, in a sentence or two, say *why* your answer holds (5 points each).

1. Will a guest OS be more or less susceptible to livelock when running on VMware? Let's say you are running the unmodified system in the livelock system (with screend) and that VMware can magically detect when the system is livelocked. What could it do to fix livelock?

   The guest will effectively runs slower, so will be more susceptible to livelock. When it detects livelock it can discard packets instead of delivering them to the guest OS, throttling the input rate to match the MLFRR point.

2. When a file is created on FFS, FFS synchronously writes the directory entry pointing to the new file to disk before returning from the `creat` system call. The purported goal of doing so is to ensure that if an application later `fsync`'s the file the directory entry for the file will be on disk as well. Give an end-to-end argument against this choice.

   Standard interpretation: not all applications need this ability (you've probably never knowingly written one that did) so forcing them to pay for it violates E2E. (Main idea is that optimization at the lower layer is unnecessary, as it can be done correctly at the higher layer without incurring a cost on the common case.)

3.
```
fd = open("foo");

write(fd, ...)
        <------ someone else writes foo.
read(fd, ...)
        <----- someone deletes foo
write(fd, ...);
close(fd);
```

   What happens here on the base NFS system? For the leases system? For LBFS?

   For the read: NFS paper isn't totally clear, if the read happens before the attribute cache expires, it appears NFS will use the old value; if after it will check the attribute and refetch. Since both leases and LBFS have open-close semantics they will both use the old version. For both NFS and LBFS the inode number will change so the write will fail for both: for LBFS at the close, for NFS at either the close or write depending. Its not really clear what will happen with leases.

4. LBFS. Client A opens file F and starts writing. Client B opens file F and starts writing. Client B closes F. Client A closes F. When are the contents A's F? When are they B's

F? When are they something else?

*Last writer wins as observed by the **server**. Even if A closes before B in absolute time, network delays can create a race condition where A's (B's) version is the final version left on the server, depending on which close is observed last. Because the commit of the temporary file upon close on the server to the permanent file is not really atomic (the destination file is first truncated, and then copied over with the contents of the temporary file; this copy is not atomic), a crash on the server during close will leave a version of the file that is neither A's nor B's, but some incomplete version.*

5. You look in an LFS segment and see a data block D immediately followed by an inode I that points to it. The LFS cleaner claims that the space used by I can be reclaimed. Does this mean D is dead as well? The cleaner claims that D is dead. Does this mean that I is as well? (Say why or why not for both.)

   First case: D may not be dead; a write to another data block in the file could cause the inode I to be moved. Second case: D is dead, this means that it was either written or the file was truncated/deleted, in all cases this means I would be relocated or deleted as well.

6. Given an intuitive estimate of the "big-O" cost of `fsck` for FFS without softupdates, for LFS? (I.e., `O(n)`, `O(`n^2`)`, etc where you define n.) Can you construct a specific example where FFS without softupdates obviously beats LFS by a significant amount?

   For FFS: fsck = O(metadata) since `fsck` *has to crawl over everything seeing what is reachable (not reachable = free) and fixing reference counts. For LFS,* `fsck` *= O(metadata in segments since last checkpoint) since it has to walk over grabbing inodes and running directory log.*

   Simple case: long running file system, no checkpoint, run "rm -rf". FFS will have trivial metadata (so `fsck` *will be instantaneous), whereas LFS will replay all segments to finally determine FS is empty.*

7. How would network traffic change if the Map/Reduce system was modified so that map tasks created their output files in the GFS? What would the effect be on Figure 2 and Figure 3(a)? Be specific. How could this change improve performance when a map tasked failed?

   There will be a lot more traffic: from the text, GFS replicates files 3x times which will put a lot of load on the network and slow down processing. This doesn't seem to be a trivial effect: as the paper states, for sort, the shuffle rate is higher than the output rate because of it. On the other hand if a map worker fails after the data escapes into

GFS the master won't have to restart the worker since the reducers can just get the data from the GFS.

We took a variety of answers for shape and such as long as the assumptions were somewhat reasonable (if incorrect, at least in an informed way).

*Note: the graphs show the *effective* data rate, not the raw number of bytes that get sent (e.g., when workers go down and have to be restarted, the lost work causes bandwidth counts to go negative. This can also be seen in the explanation as to why the shuffle rate is higher than the sort output.)*

8. Softupdates: (1) how do you use the freelist when creating a snapshot file? (2) when you write to a block on a checkpointed partition, what are the states the written block could be in and what do you do for each?

   Walk down freelist: mark all blocks not freed as "not copied", mark all freed blocks as "not used."

   When you write: (1) marked "not used" or "copied" = don't do anything, let modification proceed, (2) marked used, not copied = allocate new block, copy.

9. You take a RAID5 array with D=4 and mirror it (i.e., each disk in the RAID5 array has an identical copy). Give the expressions for large read, large write, large RMW, and small read, along with the intuition for why you picked these values. What is the maximum number of disks this RAID can lose before it fails?

   Should have roughly the same performance as mirroring (there is the C/G bit that will likely have to be worked in the formula, but we didn't require this): (1) large read = 2D/S (replication = 2x bandwidth), (2) large write = D/S (have to write all), (3) large RMW 4D/3S, (4) small read = 2D.

   Maximum number of failures: each mirrored array has 4 data disks, plus 1 check disk. Have to lose one array (5) + one check disk (1) + one data disk = 6 disks.

10. You know the future. *Concisely* describe three *important* optimizations you could do (and how) for the papers covered by this quiz *besides* predicting which cache entry to evict.

    One good one would be when things will fail: you can cause all data and meta data to be flushed out so system in a consistent state. This eliminates all sync writes and recovery costs. LFS: when things will die. GFS: when things will die so you can avoid (or even what things are slow). Leases: what the read/write usage and failure will be to set optimal lease term.