Stanford University
Computer Science Department
CS 240 Quiz 2 with Answers
Spring 2004

May 24, 2004

This is an open-book exam. You have 50 minutes to answer eight out of ten 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 - 2	
3 - 4	
5 - 6	
7 - 8	
9 - 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. Assume you could classify packets with what application they were intended for infinitely quickly. What could you improve in the livelock paper and how?

   Traffic differentiation: One big improvement would be to track which application a packet was for and if its queue was full or it had used too much CPU discard the packet. Right now if any application has a full queue all receive processing is shut down. I took off 1 point for people who still had a single queue, but only discarded packets for backlogged applications. This way a backlogged application can still starve others for queue space, just not completely.

2. Livelock: Why are infinitely large queues not a good idea (even assuming an infinite amount of memory)?

   Infinite queue = infinite latency if the incoming packet rate is high enough. Jitter is also a problem.

3. Because NFS operations are not idempotent, we want them to be executed exactly once. Jimbo says the way to do this is to send them over TCP just like in LBFS, since TCP provides a reliable, FIFO bytestream. Give an example and intuition for why using TCP does not completely solve the problem. (Hint: think about crashes.)

   This doesn't completely work: if the server crashes before it sends a response the client cannot tell if it crashes after it did the operation but before it sent the packet, or after it did the operation but before it sent the packet. I took off 2 points for answers that said something like "in the middle of" without clarifying that the problem occurred explicitly between performing the operation and the client receiving an acknowledgement.

4. VMware I/O: Send-combining has a threshold of 3 packets at which it will always switch back and transmit, no matter what. What would be the relative disadvantages of setting this threshold to 1? To infinity?

   threshold = 1 = no send combining. threshold = infinity = infinite latency and host NIC under-utilization.

5. You just brought the Boneh-in-a-box coprocessor, which can do an infinitely fast SHA-1 hash computation. You change the metadata of FFS and LFS so that each pointer to a data block (rather than meta-data block) also stores a SHA-1 hash of the pointed-to data. Before reading a block, you will compare the SHA-1 hash to the blocks in your buffer cache and skip the disk read if it's already there.

   How would SHA-FFS performance compare to old-FFS on reads? How does SHA-FFS compare to SHA-LFS on writes?

   Reads that hit in the cache would be better, reads that miss in the cache will be a bit worse most likely since the metadata will be "dilated" by the SHA hashes. If you share blocks that have the same hash, then reads that miss in the cache will be dramatically worse since you will have to seek all over disk rather than reading a contigous extent.

   SHA-FFS will be even worse than old-FFS when compared to SHA-LFS on writes since it has to modify two places on disk (both the hash and the data block) rather than just the one for FFS (the datablock).

6. You change the system in the previous questions to use Rabin-fingerprinting as in the LBFS paper. Why might this be a better idea on SHA-LFS than on SHA-FFS? What other changes do you have to make to the meta-data?

   The big problem this will cause is variable sized data blocks, which will increase fragmentation. This will require cleverness in FFS, but should be straightforward in LFS since you just blast a bunch of data into a single segment. You would have to include a size along with the data block, turning it into an extent (similar to segmentation vs

paging).

7. LFS: draw the ideal distribution of segments on a 20MB disk with 10MB of live data and state why you think your distribution is the best case. Make sure you label the x- and y-axis with the actual numbers you would see.

   LFS uses 2MB segments, so 20MB = 10 segments. The ideal distribution would have 5 segments completely full of data and 5 segments completely empty. I took off 1 point for people whose axes were labelled incorrectly or not at all.

8. LBFS: client machine C1 writes to a file that an application on client machine C2 is using. The server sends C2 a message revoking its lease. What exactly does C2 do? (Note, make sure you also say what happens to the application using the file.)

   LBFS uses close-open consistency, so a lease invalidation only causes the client to remove the attributes from its cache. Nothing happens to the application: once you open a file its yours, the data in it never changes in response to other people. I gave 3 points to people who mentioned explicitly their assumption that the file wasn't already open and 2 or 1 point to people who said something generally true about leases, but not true about LBFS' use of them.

9. GoogleFS: If the primary's lease expires during the record append operation what must happen and why?

   It must abort the record append. Once primary P's lease expires another chunk server P' could be elected primary and perform record appends as well. Since P and P' are not synchronized these appends can happen in different orders on different chunk servers, violating the invariant that mutations to a chunk happen in the same order. The paper is not very clear, but it also seems possible that both primary's would write to the same offset.

10. The Google file system paper claims that each chunk has an associated version number that is periodically incremented to detect stale replicas. Is it a problem if a replica goes down and comes back up between these increments? (Be very concrete as to why this does or does not cause problems.)

    This question was tricky, since the google-fs paper was not that explicit about how things work. However, if you read between the lines, it's clear that when the primary is elected it has a list of valid chunkservers. If any chunkserver in this list goes down a write will abort. Since version number is only incremented when leases are granted rather than on every write, then if the primary does not have such a list then (for example) a network partition will easily create stale chunks that are impossible to detect.

    So: assume the primary does have a list of chunkservers that are considered active when its lease was granted. Then a chunkserver that goes down and up between an increment will not cause problems. The write attempt will fail until either (1) the lease expires or (2) the node comes back up, in which case we can safely go forward since the relevent state has not been modified.)

    We gave three points if you said that it did cause problems, and gave a plausible scenario, or if you said it did not cause problems and mentioned something reasonable about the server detecting stale chunks when the chunkserver rebooted.