Stanford University
Computer Science Department
CS 240 Quiz 2
Winter 2005

March 9, 2005

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 - 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. Livelock: In 6-3, why does the output packet rate of "Polling (no quota)" drop to almost zero after 6K pkts/sec? What does the same thing not happen with "Polling, no feedback" in 6-4?

   When packets arrive at a sufficient rate, transmit does not get to run. In 6-3 , all work happens in the kernel and cannot get interrupted. In 6-4, screend runs in user space and so can get interrupted by transmit.

2. What would be the worst-case livelock-inducing load that LBFS could generate? Be very concrete, and (as with all questions) make sure you state your assumptions and justify your answer.

   This is a bit of a trick question. The naive answer is that LBFS can get back 1024 get hash at a time. If none of them are on the server, it will issue 1024 read requests. However, these are all over TCP, so in reality it shouldn't lead to livelock. The best bet is if there are lots of clients. Many people forgot about TCP's behavior...

3. You know the future. *Concisely* describe three *important* optimizations you could do (and how) in a realistic implementation of the systems covered by this test *besides* predicting which cache entry to evict.

   (a) LFS: What things die together.

   (b) Livelock: whether you will get livelock if you handle a packet. Plausible: if a packet will cause a queue overflow. More plausible: what packet an application is for.

   (c) LFS: when you are about to crash, can write out checkpoint.

   (d) Leases: when the next write to a leased item is going to happen.

4. LFS: draw the ideal distribution of segments on a 50MB disk with 30MB 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. Why does it look this way? *The best distribution is bimodal: in this case it would be 30 completely full segments (1MB apiece) and 20 completely empty segments (1MB apiece). LFS does *NOT* use 2MB pages, see the paper...*

5. Leases, Figure 1: re-draw the graph lines assuming that at least one machine holding a lease was down. Why does it look this way?

   We meant to ask about Fig2... For Fig one, there are no noticeable changes except for maybe a slight shift down in performance.

6. Leases: the paper states that "impact of leases grows more significant in systems of larger scale and higher processor performance." What is their reasoning?

   In their world, larger scale = larger latency and more probability that a node fails. The first rewards caching (which leases help), the second rewards recovery in the face of failure. Higher processor performance = more requests per second, which also rewards caching.

7. Assume LFS recovery works as the paper hypothetically describes it would. If you eliminate checkpoints, can you ever clean a segment? If you did have checkpoints, what potential bug could occur in LFS that would be the closest analogue to the log-ordering problem FiSC found in ext3, JFS, and ReiserFS?

   Assumption: crash recovery = rerunning the "log" from the beginning. Without checkpoints, we cannot cut any prefix off of the log (since we start at beginning), and so can never clean anything. LFS does not "clear" checkpoints, only overwrites them, so a close analogue bug: overwriting a checkpoint C with information that assumed segment S had been written in its entirety, but not ordering C and S.

8. You want to check NFS using FiSC: assuming there is only one client running, what would the StableFS be and why?

   As the paper states: ""because of stateless have to commit any modified data to disk before returning results". Thus the StableFS is trivial: you should always recover to a state equal to replaying all past operations in sequence, either up to, or including the last operation.

9. Assume you have a simple RAID3 system with four data disks (A,B,C,D) and one dedicated parity disk (P). Assume the OS was writing to some of the data disks (you don't know which) and then crashes. Explain (1) what state you should recover the RAID system to (its StableFS), and (2) how to do so. What happens if a disk blows up during recovery?

You have to recompute parity by xor'ing A, B, C, D together. If a data disk crashes during crash recovery you cannot recover: the parity can be out of date and so xor'ing in the remaining data disks will generate nonsense.

10. Rank (and explain) the seriousness of a node crashing that is running (1) a single map worker, (2) a single reduce worker, or (3) a single master.

    Least serious: crashing a reduce worker, since you can just rerun its remaining tasks on other machines. Next is a map worker: here you can also rerun, the problem is that this must complete before any reduce using its data can go, so forms a synchrous bottleneck (reduce are more modular). Worst would be crashing the master right before completion since it sounds like you lose the entire job.