Stanford University
Computer Science Department
CS 240 Final Spring 2009

June 12, 2009

!!!!!! SKIP 15 POINTS WORTH OF QUESTIONS. !!!!!

This is an open-book exam. You have 75 minutes. Cross out the questions you skip. 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
1-8 (40 points)	
9-15 (45 points)	
total (max: 70 points):	

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:

I Short answer

1. [5 points]: The LFS paper states the minimal write cost is 1, but the formula (section 3.4) seems to imply it's 2. Which is right and why?

The text is right: If the segment is totally empty you don't have to read it. Write cost = 1 (all new data).

2. [5 points]: Give an example in LFS where you want to write something (a checkpoint, a segment, etc) but in order to make sure things work correctly across a crash you have to write something else first.
If a checkpoint refers to data in volatile memory you have to flush this out (and wait for it to be entirely on disk!) before committing.

3. [5 points]: You rerun the experiments in Figure 6 and Figure 7 in the xsyncfs paper on top of LFS. How do you expect the performance to compare to that of xsyncfs?
Fig 6: LFS will be slower in than xsyncfs since it has to do fsyncs right away, whereas xsyncfs can defer them until their output must hit the screen (runs locally). LFS should essentially get some amount of group commit help though, so should be faster than ext3. Fig 7: network on LFS should be slightly faster than on xsyncfs since they don't add to latency.

4. [5 points]: You run the NFS system on top of xsyncfs. If you run one NFS client on the same machine as the server, what do you expect to happen to performance and why? If you run one NFS client on a remote machine what do you expect? If you run many remote clients what do you expect?
This is basically the DB experiment (Figure 6) but using NFS instead. Just like the DB, NFS has to flush the data to disk before replying. And just like the DB it will see a benefit for the local case. It should see not much benefit for the remote. But if you add multiple remote will perhaps benefit from group commit.

5. [5 points]: Since you scored so well on xsyncfs questions, the VMware guys ask you to build it into ESX so they can make some slow OSes faster. Explain how you would do so. What simple thing for xsyncfs becomes a much harder challenge when you do things at the ESX level?
Tricky thing: app A does a system call that creates a dependency with app B (such as writing data to a pipe). You won't be able to tell that B is the one getting the dependency at the ESX level. A simple conservative thing would be to treat the entire guest OS and all of its processes as a single process. Note this assumes ESX can monitor all device output (and defer it if need be).

6. [**5 points**]**:** Give three specific changes you could make to NFS paper's system so that it used leases to improve consistency or performance.
For files you can get stronger consistency, and higher performance since you don't have to keep asking the server and you will get notifications when someone wants to write to the data. They also have an attribute cache and a directory cache with a 3s and 30s timeout (respectively) that you can redo with leases.

Alot of people read this question as asking how to put leases in NFS. We didn't mark that down typically (unless the answer was too vague) since the question wording doesn't preclude that interpretation.

7. [**5 points**]**:** LBFS: You rerun all the experiments in the paper with four clients rather than one,. In which figure would you *most plausibly* see a 4x performance improvement for LBFS and why? ("most plausibly" = not necessarily but more likely than in the other figures.)
The most plausible is gcc in Fig 7(a) since Fig 7(b) shows it has around 15% bandwidth utilization: it still has plenty of slack to support the four clients. Unfortunately however, the question is pretty confusing so we had to accept a lot of different answers.

We also took some other answers since there was a lot of latitude for interpretation.

8. [**5 points**]**:** LBFS states that "file systems can also better tolerate network latency than remote login sessions." Explain what they mean (and why) and which figure demonstrates this the best (and why).
With an FS you do commands locally an ddon't have to wait for round trip delay after each newline. You do all reads locally, absorb all writes locally, and generally have a happier life. Figure 10 shows this.

9. [**5 points**]**:** Your program has a bug. Klee doesn't find the bug. Give three different reasons why. *All sorts of reasons:*

A. *Exponential paths: doens't hit it.*

B. *Constraint too expensive so can't go down path.*

C. *Hits assembly or any of the things not supported.*

D. *Bug in klee.*

E. *Is a correctness error but no assert checks for it.*

10. [**5 points**]**:** Livelock: Explain how the picture in Figure 3 maps to Figure 11, saying where each part begins and ends.
basic breakdown:

A. *0-142: put on the input Q.*

B. *191-440: forward to xmit Q.*

C. *520-: handle fact that xmit happened.*

11. [5 points]: Consider two packet forwarders running Digital UNIX. Forwarder P (for polling) has Mogul and Ramakrishnan's enhancements, while forwarder U (for unmodified) does not and is purely interrupt-based. You send each forwarder a burst of three packets. Which one will forward the third packet of the burst with the minimum latency?

A P will forward the third packet with less latency.

B Both will have the same latency for the third packet.

C U will forward the third packet with less latency.

Justify your answer:

C: the modified does not poke the xmit til it's done processing the quota of packets.

12. [5 points]: You get an infinitely fast CPU. Assuming you run the unmodified Digital Unix system on it: give a workload that will livelock or argue why that is not possible. Make sure to state any (reasonable) assumptions! (Hint: recall that the paper does not define livelock as merely losing some packets.)

If you assume an infinitely fast input stream then yeah, seems like you could still starve the application. If you assume finite input stream then I don't think you can get livelock.

13. [5 points]: Give the one line change in the MapReduce word count program (in the appendix of the paper) that would allow it to run correctly when run without a combining step, but would produce incorrect output when run with combining. (And, of course, make sure you say why this change causes things to break.)
Basically: counter++ rather than reading the actual value in.

II MapReduce mashup

14. [10 points]: Times have changed. Assume that LFS has changed with them and instead of writing segments across an I/O bus to a disk, writes its segments across a network to the Google file system alluded to in the MapReduce paper.

Explain how to use map reduce to do segment cleaning in such a way that it will *commonly* (though perhaps not always) get a speedup. Give an intuition for what the map and reduce phases do, what they take as input and produce as output, and any challenges to using this type of scheme. Make sure you state exactly the key, value pair that map passes to reduce since this is the trickiest part.
This was a pain to grade. Basically, you'd like segments to be around 16MB or whatever (not a big deal on modern machines given the size of main memory) so they map to the GFS chunksize. Each map would get some number of segments. It'd spit out records giving the inode number of the object as a key, and the DA, time, version as the value. Reduce would take this and merge everything to just have the most recent key. It can discard anything that doesn't match the version. You could probably take the inode map into the map task and discard inodes that are dead.

III KLEE

```
   #include <assert.h>

   int main(void) {
     unsigned i, t, a[3] = { 5, 3, 1 };

     make_symbolic(&i, sizeof i);
1:   a[i]--;          % error: mem overflow.  {i < 3}, test: i = 5
2:   t = i / a[i];    % error: div 0.  {i < 3 /\ i != 2} test: i = 2
3:   a[a[i]]++;       % error: mem overflow. (i < 3 /\ i != 2 /\ i != 0) test: i=0

4:   assert(i == 1); % passes
5:   return 0;        % test: i = 1
   }
```

15. [**10 points**]**:** Explain what KLEE does at each numbered line of code (1-5), including the constraints it generates on `i`, any errors, and how many test cases it generates. For each test case, give a concrete value it could produce.