Midterm
Fall 2000 15-712

Please write your name on the following line:

_____

This midterm is open book and open notes.

You should write brief, precise, and legible answers. Rambling brain-dump answers may be punished twice — you run out of time writing them and we run out of patience reading them. :–)

There are eight questions but we will only grade five. List the five questions we should grade:

_____

Name: _____

1. Soft Updates involves a significant amount of mechanism to propagate metadata changes to disk in a specific order when employing write-back caching. Explain why soft updates is not needed in a log-structured file system.

   In a log-structured file system, all writes go to a sequential log. Hence, all changes are ordered trivially by inserting them into the log in the right order. In FFS, metadata blocks have fixed locations on disk, so cyclic dependencies can occur between them. The undo/redo mechanism in Soft Updates is needed in FFS to break these dependencies, ensuring that changes are well-ordered and consistent.

Name: _____

2. Callbacks are server-maintained state. Describe how consistency may be compromised if the server reboots.

   A callback is a promise that the server will inform a client when the client's cache is out-of-date. Imagine a client has some cached data and the server holds a callback for the client. When the server reboots, the server loses the callback for that client. Now another client modifies the data that the first client has cached. The first client will continue to use stale data in its cache because the server didn't know to contact it and inform it that its data has become invalid.

Name: _____

3. NASD and xFS both focus on scalability. Explain why xFS is probably more scalable and why NASD is probably more cost effective (in some circumstances).

   xFS is probably more scalable than NASD because there is no one centralized component for any file system service. All of data service, metadata service (name lookup, etc.), and caching (cooperative caching) is distributed. However, in NASD, there is a centralized file manager that provides all metadata service and also issues capabilities to clients on file lookup. The file manager in NASD may become a bottleneck.

   NASD is probably more cost-effective than xFS. Each xFS node is a computer capable of providing any or all of the pieces of the file system listed above. Scaling up a system running xFS requires adding more of these nodes. After investing in a file manager machine, scaling up a NASD system only requires adding NASD devices. Hence, it is more cost-effective if, as the paper argues, NASD devices do not cost significantly more than traditional disks.

Name: _____

4. A server can use RPCs to contact clients and inform them that their callbacks are broken. Explain why doing this within a single server thread can be inefficient. What might you change to fix this problem?

   Using a single server thread to make RPCs that need not be ordered can be inefficient because there is no parallelism. Most of the time the server thread will probably be blocked waiting on network latency and the remote clients, which may be slow.

   A solution would be to make the server multithreaded and use each thread to concurrently notify a different client. By doing so, the system components will be less idle, which is the same as saying that they are being more efficiently used.

   Also, if the writer doesn't need to wait for all clients to invalidate their caches before committing changes, another solution would be for the server to break callbacks using message passing.

Name: _____

5. River and xFS both try to balance load across collections of servers. Explain how their foci differ and the resulting effects on their designs.

   xFS is a general-purpose file system that seeks to decentralize and enable scaling of all file system functionality, such as providing a hierarchical namespace, caching, data storage, cleaning, etc. Each xFS node can provide any or all of these functions. Since function is distributed, there are many levels of lookup maps that enable one part of the file system to locate and request service from another part.

   Instead of general file system service, River targets data-parallel applications where producer and consumer code is replicated across many machines. In particular, it focuses on performance imbalances among produces or among consumers by allowing each to take on load according to its relative capabilities.

Name: _____

6. In the Emerald system, programmers must decide what node each object executes on. However, Emerald does have a few ways to automate some migration decisions; explain two such migration mechanisms.

   Emerald will automatically migrate objects when: (1) they are arguments to a remote procedure and a cost-benefit analysis shows that they should move; and (2) when the objects are "attached" to other objects that are migrating. Note: although the second mechanism requires the programmer to tag objects as attached, it is still somewhat automatic in that the programmer isn't involved in making a case-by-case decision.

Name: _____

7. The Abacus function placement approach works well for data-intensive applications structured as a stack of objects. Emerald supports a more general communication architecture, where communication patterns could be any arbitrary graph. What problems would Abacus face if it tried to support less regular, graph-like applications?

   The more general the communication patterns, the harder it is for the Abacus run-time to collect sufficient history, model the system, and divine appropriate object locations. A stack of objects on two machines (client and server) is relatively simple. In this scenario, there are few potential ways to partition the objects. Further, the run-time can assume that all the objects except the two at the network boundary have zero communication costs. However, if you introduce more nodes and more complex communication patterns between the objects, then the problem reduces to general graph-cutting, which is hard.

Name: _____

8. The UCLA stackable FS infrastructure allows untrusted file system layers to be run in separate processes. Describe two approaches that avoid context switch overheads. Mention the consequence of each of your proposed approaches.

   In class, we've discussed interpretation and type-safe languages.

   Imagine running the file system layers inside of an interpreter. The interpreter may or may not be a general purpose language (contrast a restricted language for writing simple network packet filters with a language like Java). Safety is provided by design, because the interpreter doesn't have constructs that would allow one layer to attack another. Drawbacks: interpretation can be slow; and if the interpreter isn't a general purpose language, users may find it constricting to work in.

   Type-safe languages (like Modula-3 used in SPIN) provide safety because one layer cannot obtain a reference (analogous to address pointer) to objects in another layer. Although a point of debate, some believe that because these are high-level languages, they impose a big performance cost. Also, some argue that these languages are less pleasant to use.

   Three other approaches are: (1) sandbox the code using software fault-isolation; (2) supply the code with an unforgeable proof of its correctness as in the proof-carrying code work; and (3) sacrifice safety and put every layer in the same process.