Name: **AndrewID:**

15-440 Homework #2
Spring 2011 (Kesden)

1. Consider the quorum-based protocols that we discussed in class. Each of these protocols requires some type of read-write locking to ensure that the client is playing with the same version of the object at each of the participating replicas. Without this, two writes could interleave or a read and a write could interleave.

   Please describe an efficient locking protocol to address these concerns. Assume that the client can interact with all servers, except in the event of failure.

2. Reconsider your answer to question #1. This time, please consider the special case of a mobile client using a read-one/write-all quorum. Please design an efficient locking protocol to ensure concurrency control. But, please ensure that your protocol permits the lock to be acquired at one replica and released at another.

3. What are the advantages of using CRCs over Hamming codes or parity bits?

4. What property of an encoding makes possible the detection of erroneous transmissions and possibly the correction of the errors?

5. In data communications, simple odd- or even-parity is understood to provide only error detection, but in RAID storage systems, we see that a similar parity scheme allows for error correction. What is the critical difference in these two applications that allows more mileage per error-handling bit?

6. What is the biggest challenge involved in restoring a node from a tightly-coupled distributed system using checkpoints?

7. What is the biggest advantage of using logs, over checkpoints, as a recovery mechanism for a node within a tightly-coupled distributed system?

8. When can recovery log entries be deleted?

9. Consider a file system such as HDFS. In particular, consider a file system that manages truly huge files by scattering multiple copies of their constituent blocks across a few to several data stores scattered across various switches, possibly across various locations.

   Take two steps back and consider some of the unstated assumptions implicit in the present design and implementation of HDFS. Is it, in its present form, capable of serving as a truly global file system? Please provide and explain three specific examples of design or implementation decisions that support your conjecture. Note: Please focus your answer on aspects of the design and/or implementation other than dependability

10. The original Google paper, and the Apache HDFS implementation use a fixed number of replicas per block. Does this make sense as a solution for a high-volume production system? If so, why? If not, why not – and what strategy might be better, and why might it be better?

11. Does the Hadoop Map-Reduce framework, itself, recover and continue if a Master node dies? If so, how? If not, how much benefit would be conveyed by this feature? Why?

12. Is the Hadoop Map-Reduce framework able to complete a task, without intervention form the application, in light of the failure of a worker node? If so, how? If not, why is not transparent to the application?

13. HDFS currently provides no mechanism for fail-safe, fail-soft, or self-recovery for the failure of a NameNode. Please define a mechanism that would enable continued operation or automatic recovery after the failure of a single NameNode. Your solution should not significantly impact throughput in the normal case. You should describe normal operation, any transition after the failure, operation prior to the replacement of the failed NameNode hardware, and the transition back to operation with a full set of resources.

14. Please describe a strategy that can be implemented within Hadoop's Map-Reduce Framework to verifying that all records contained within one file <userid, street, city, state, zip> have keys found in another file <userid, ssn, yearsOfService, benefitsEligible, hourly>.