We'll post an outline of each recitation after the fact so you know what was covered in case you had to miss class that day. The recitation outlines are not a full replacement for recitation!

This week's recitation was largely a problem-solving session (good for exam prep). These are the problems we went over; some sections may not have gotten to every problem.

Intro to Transactions

- What is atomicity? Isolation? Why are they good?
- What is RAID for? How does it fit into the context of these transaction lectures?
- What is write-ahead-logging for? Why is it better than the shadow copies Tim showed in Lecture 16?
- What is conflict-serializability? Two-phase locking? What do they give us?
- How do GFS and MapReduce fit into these topics?
- What does GFS do? What kind of data do you think is stored on it? Why is reliability important for that use case?
- What about MapReduce? What does *it* do? What types of problems can you solve with MapReduce?

GFS and MapReduce

Question 1:

T/F If there are m map tasks, using more than m workers in the map phase may still improve performance beyond that achieved with m workers.
T/F To achieve locality, map workers always execute on the same machine as the input data that they consume.
T/F GFS optimizes for writes that append data to a file, rather than writes to random offsets.
T/F From the point of view of clients, GFS provides single-copy consistency
T/F GFS implements "at least once" semantics for all writes

Question 2

Peyton's MapReduce system consists of five machines: a controller machine and four worker machines, M_1 through M_4. Each worker machine has four cores, and can run one map or reduce job per core in parallel. Each worker machine M_i has access to its own local disk D_i. Peyton's MapReduce job starts with sixteen map tasks; the controller assigns one task per core.

After running for some time, all map tasks have completed except Task 1 running on M_1. What will the controller do in response? Circle the best answer.

(a) Nothing, unless M_1 fails
(b) Assign Task 1 to an available machine

(c) Subdivide Task 1 into multiple smaller map tasks, and assign those smaller tasks to available machines
(d) Kill the process running Task 1 on M_1 and begin to assign reduce tasks, ignoring the result of Task 1.

Eventually, M_1 fails. This is the only failure in the system. Circle True or False for each of the following.

(a) T/F M_1 will notify the controller of its failure
(b) T/F The controller will re-assign any jobs currently running on M_1 to another available machine
(c) T/F The controller will copy data on D_1 to another disk

Question 3

Phi is running a MapReduce computation on some machines in a datacenter. The datacenter is *incredibly* lightly-loaded during this process: Phi's single MapReduce computation is the only thing running. Phi is watching some of the network traffic as this computation runs.

Phi first observes the map phase of this computation. She knows that the basic process is as follows: the controller assigns map tasks to workers; workers read input data; workers perform the map computation; workers write output data.

During the map phase, Phi observes the controller assign map tasks to ten of the machines in the datacenter. She then observes a network transfer of input data from Machine 5 to Machine 6. However, at **no point** during the map phase does she observe this type of data transfer to Machine 7 (i.e., no machine in the network sends input data to Machine 7 during the map phase).

T/F Machine 7 must not have been assigned a map task.

After it finishes its map computation, Phi sees Machine 6 send a message to the controller. What might the contents of this message be? Select **all** that apply.

(a) A response to a ping from the controller.
(b) Output data from the map computation.
(c) Meta-information about the output data from the map computation
(d) None of the above

Phi knows that MapReduce utilizes GFS as one of its underlying file systems. In the context of MapReduce, what does GFS improve? Select **all** that apply.

(a) GFS makes it more likely that the **input data** will be accessible throughout the MapReduce job, compared to using MapReduce with a filesystem that doesn't replicate any data.
(b) GFS allows MapReduce to continue running the computation even if the controller fails.
(c) GFS makes the map and reduce computations atomic.
(d) GFS can decrease the amount of time it takes for a machine to read the input data, compared to using MapReduce with a filesystem that doesn't replicate any data.
(e) None of the above

Since Phi has gotten familiar with GFS through her use of MapReduce, she starts using GFS on its own as a file system to back a web service she's running. She is using GFS exactly as described in the paper, with a default replication factor of three for every file.

Phi wants to improve the responsiveness of her system to client requests by increasing the replication factor of some files. Which type of files should she increase the replication factor for? Select the **best** answer.

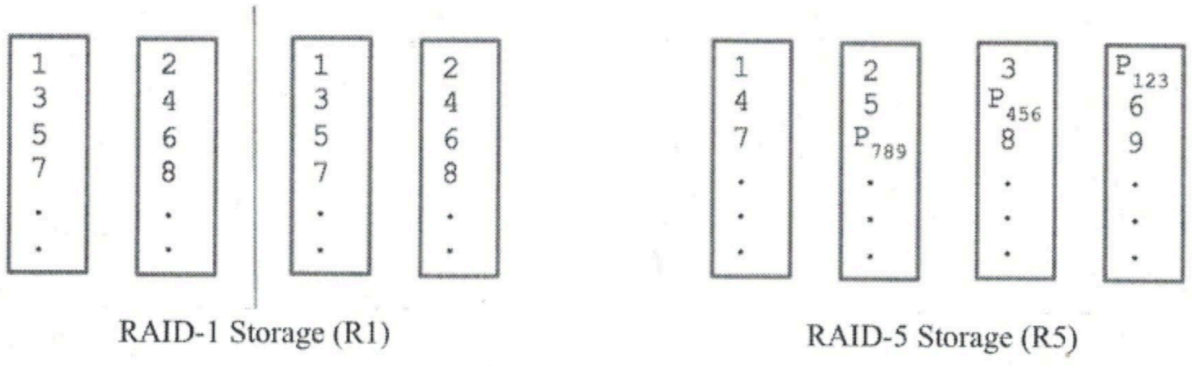
(a) Very small files
(b) Very large files
(c) Very popular files

RAID + Logging

Here's a RAID question and a Logging question, in case you need some backups.

Question 4

Consider the two storage organizations shown below:



One uses RAID-1 (R1) and the other uses RAID-5 (R5). Here, each rectangle represents a disk. The numbers represent blocks of data and show how the blocks are striped across different disks. P_ijk represents the parity block for data blocks i, j, and k. Assume that disk controllers

can detect an erroneous data block while reading it. If a disk fails, then all blocks on that disk become inaccessible.

What is the maximum number of distinct data blocks that can be read concurrently (where each read is processed at the same time on a different disk) by these two schemes?

How long is the longest sequence of consecutively numbered data blocks that can be read concurrently by these two schemes?

How many disk blocks are needed to store 6000 blocks of user data in these two systems?

In case a bad data block is detected while reading, how many disks have to be read to reconstruct the bad block in these two systems (excluding the read to find the bad block in the first place)?

Question 5

A transaction-based system is using write-ahead-logging. The log is backed by both on-disk cell storage and an in-memory cache. The cache is shared by all transactions. Writes go first to the log, then to the cache. Reads go first to the cache, and then to cell storage if the data is not present in the cache. Assume that all values are initialized to zero in cell storage, and that the cache is large enough to hold all data in question.

Consider the following log snippet. The format for UPDATE records is UPDATE var=<old value>; var=<new value>.

```
Log Entry  Transaction ID  Record
---------------------------------
              ...
    1          T1          BEGIN
    2          T1          UPDATE W=0; W=10
    3          T1          UPDATE X=0; X=20
    4          T1          COMMIT
    5          T2          BEGIN
    6          T3          BEGIN
    7          T3          UPDATE Y=0; Y=30
    8          T2          UPDATE Z=0; Z=40
    9          T2          COMMIT
```

Occasionally, all values in the cache are copied to cell storage. For the purposes of this question, assume that that happens exactly once: immediately before Log Entry 8 is written. Suppose the system crashes after Log Entry 9 is written.

Immediately before the crash, what are the values in the cache?

Immediately before the crash, what are the values in cell storage?

After the crash, the system goes through the recovery process. It first does an UNDO phase, followed by a REDO phase.

After the UNDO phase, but before the REDO phase, what are the values in cell storage?

After the REDO phase (i.e., after recovery is complete), what are the values in cell storage?