

Solutions are due at the beginning of class on the due date and must be **typed** and neatly organized. Late homeworks will not be accepted. You are permitted to discuss these problems with your classmates; however, your work and answers must be your own. All questions should be directed to the teaching assistant.

Problem 1 : A bunch of distributed file systems questions.

In this problem you will explore the strengths (and weaknesses) of two commercially successful distributed file systems – AFS and NFS. Specifically, you will look at the implications of state (or lack thereof) at the server when several clients are writing to the same file.

Recalling from the lecture the semantics of NFS server (the original version, i.e., no leases), it does not know when a client opens a file for writing nor when the client is done writing and closes the file. Because of that, NFS requires that the client writes thorough all changes i.e., every time a client writes a byte to the file, the client sends the appropriate block (e.g., 4 KB) to the server that writes it synchronously to the disk.

The AFS server, on the other hand, keeps a list of all clients that have a file open. The client modifies a local copy of a file and only when it closes the file are the dirty blocks sent to the server. The server then looks up all the clients that have a local copy of the same file and sends them an invalidation message via a callback. This message lists all the blocks that have been modified. The clients have to contact the server if they want to get the latest version of the data.

Assume a sequence of system calls (listed below) at four clients that all open the same file stored on a file server. The file is initially empty. The time increases such that $t_1 < t_2 < \dots < t_9$. The syscall `write(fd,data)` means that the `data` is written to the file whose descriptor is `fd`. Answer the following questions. Give all scenarios whenever possible and briefly explain your answer.

Time	Client 1	Client 2	Client 3	Client 4
t_1	<code>fd=open("f1")</code>	<code>fd=open("f1")</code>	<code>fd=open("f1")</code>	<code>fd=open("f1")</code>
t_2				<code>write(fd,4)</code>
t_3		<code>write(fd,2)</code>		
t_4	<code>write(fd,1)</code>	<code>close(fd)</code>		
t_5			<code>write(fd,3)</code>	
t_6	<code>close(fd)</code>		<code>close(fd)</code>	
t_7				
t_8				<code>close(fd)</code>
t_9				

- (a) Assuming file `f1` is stored on an NFS server, what is its content at t_9 ?
- (b) Assuming file `f1` is stored on an AFS server, what is its content at t_9 ?

Assume a network failure occurred between t_6 and t_7 and it never recovered(i.e., the `close` system call failed on client 4).

- (c) What is the content of the file at t_9 on an NFS server?
- (d) What is its content on an AFS server?

Assume a network failure occurred between t_4 and t_5 and it never recovered (i.e., the `write` failed on client 3 and `close` failed on clients 1, 2, and 4).

- (e) What is the content of the file at t_9 on an NFS server?
- (f) What is its content on an AFS server?

Now assume that the server went down between t_4 and t_5 , but everything else was functioning well. It took sufficiently long for the server to come back up so that all the clients finished their applications (since they didn't handle a failed `write` and `close` system calls) happily believing their data was saved.

- (g) What is the content of the file when the NFS server came finally up?
- (h) When the AFS server came finally up?
- (i) Hopefully, you have noticed the different contents of the file resulting from failures occurring at different times. Which semantics do you think are “better” – AFS or NFS? Of course, it depends what “better” means. So define your metric of goodness and justify your answer. We want you to think about the tradeoffs that the designers had to face when they were designing the distributed file systems. In answering this question think how often this situation arises.

Finally, a few more random questions to make it more difficult for Adam to grade these darned things:

- (j) Do two NFS servers provide a common, global name space? Explain how or why not.
- (k) Define *idempotency*. Why is it a useful property? For a distributed file system, give an example of an idempotent operation and a nonidempotent operation.
- (l) What is the function of an AFS callback? Why isn't there an “NFS callback”? What would be the advantage of designing one into the NFS protocol?
- (m) In general, access to local disks is faster than access to remote disks. Why bother with all the overhead of network attached storage? E.g., what's the “big win”?

Problem 2 : buzzwordsgalore.com.

- (a) The CEO of buzzwordsgalore.com (a hot new e-business Internet Startup whose upcoming product is a combination handheld personal video recorder/editor and potato masher) has decided that they need to keep their video data in a redundant format, unfortunately he is too cheap to buy a proper RAID 5 system so he came up with a redundancy plan of his own, which is shown in table 1. The B 's represent data blocks and the P 's represent parity blocks, where P_i is the parity block for data blocks $B_{(4i-3)}$ to $B_{(4i)}$.

What problems might arise from this arrangement of data blocks and parity blocks?

Disk 1	Disk 2	Disk 3	Disk 4
B_1	B_2	B_3	B_4
P_1	B_5	B_6	B_7
B_8	P_2	B_9	B_{10}
B_{11}	B_{12}	P_3	B_{13}

Table 1: Redundancy scheme used by buzzwordsgalore.com

- (b) After much debate the CEO decides to purchase a commercial RAID system using RAID level 1 (mirroring) or RAID level 5 (block interleaved, distributed parity). But now after much research he becomes worried that a power failure during a multi-block disk write may not be atomic.

Suggest a scheme for each (RAID 1 and RAID 5) that would allow the CEO to detect non-atomic writes, and if possible, suggest schemes that would allow them to recover from the failure.

- (c) Since RAID 1 is easier for the CEO to comprehend he decides to look into it further. However, he becomes increasingly worried about the mean time to failure (MTTF). His main engineers try to convince him that the mean time to data loss is much greater than the mean time to disk failure.

Suppose RAID 1 is being used with 2 disks and that there is a 5% chance per year that a disk will fail. The only time data can be lost if the second disk fails while the first disk has failed and is being replaced. What is the MTTF of the RAID 1 array (involving data loss) assuming the mean time to repair of a single disk is (MTTR) is 3 hours?

Disk 1	Disk 2	Disk 3	Disk 4	Disk 5	Disk 6	Disk 7	Disk 8
PG_1	PG_1	PG_1	PG_1	PG_1	PG_1	PG_1	PG_1
PG_2	PG_2	PG_2	PG_2	PG_2	PG_2	PG_2	PG_2
PG_3	PG_3	PG_3	PG_3	PG_3	PG_3	PG_3	PG_3

Table 2(a): One Standard RAID 5 Array of 8 Disks

Disk 1	Disk 2	Disk 3	Disk 4	Disk 5	Disk 6	Disk 7	Disk 8
PG_1	PG_1	PG_1	PG_1	PG_2	PG_2	PG_2	PG_2
PG_3	PG_3	PG_3	PG_3	PG_4	PG_4	PG_4	PG_4
PG_5	PG_5	PG_5	PG_5	PG_6	PG_6	PG_6	PG_6

Table 2(b): Two Standard RAID 5 Arrays of 4 Disks Each

Disk 1	Disk 2	Disk 3	Disk 4	Disk 5	Disk 6	Disk 7	Disk 8
PG_1	PG_1	PG_1	PG_1	PG_2	PG_2	PG_2	PG_2
PG_3	PG_3	PG_4	PG_4	PG_3	PG_3	PG_4	PG_4
PG_5	PG_6	PG_5	PG_6	PG_5	PG_6	PG_5	PG_6
PG_7	PG_8	PG_8	PG_7	PG_7	PG_8	PG_8	PG_7

Table 2(c): One RAID Array with Declustered Parity

Table 2: Different RAID 5 Parity Grouping Schemes

- (d) Mr CEO is now thinking that he wants more than one RAID array, since their video streaming business is booming, and two are always better than one. However, he begins to hear rumors that instead of having two individual arrays it may be worthwhile to merge the arrays together in some kind of scheme.

For this example we have 8 disks over which to distribute our parity groups. A parity group is defined as the set of data blocks over which parity is computed, plus the parity block itself, so in the case of RAID 5 a parity group consists of a stripe across the entire array.

- (i) Given that the MTTF of a RAID 5 array is: $\frac{MTTF(disk)^2}{N \times (G-1) \times MTTR(disk)}$
- $N = \#$ disks in the array
 - $G = \#$ disks in the parity group
 - $MTTF(disk) = 300,000$ hours
 - $MTTR(disk) = 3$ hours.

What is the MTTF for each of the schemes shown in Table 2?

- (ii) Given the following conditions, describe how each scheme in Table 2 performs:
- Average number of disks touched on a single block read (no failures).
 - Average number of disks touched on a single block write (no failures).
 - Average number of disks touched on a single block read (1 disk failure).
 - Maximum number of disk failures without data loss.
- (iii) Which scheme balances the load over the disks most evenly in the case of a single disk failure? Explain.

Problem 3 : AIR RAID.

- (a) Disk failures are not the only concern of RAID arrays – array controller system crashes can also cause the array to be put in an inconsistent state. (Assume that the system contains NO NVRAM).
 - (i) Describe a scenario that can leave the array's data in an inconsistent state after a controller crash.
 - (ii) Describe a brief solution to the scenario you described above.
- (b) When performing a small write (to a single unit within the parity stripe) in RAID 5, what is the approximate increase in response time (with no failed disks and no request queueing), as opposed to a single disk?
- (c) Choosing the best stripe unit size for a set of disks involves tradeoffs between two conflicting goals. Explain what both of these goals are, and how they conflict.
- (d) In terms of queueing times and throughput, how does striping affect performance (increase or decrease) over a single disk? Give a reason why each are affected in this manner.
- (e) In a database application where availability and transaction rate are more important than space efficiency, which RAID scheme would you use? Why?