
Name: _____

Instructions

There are three (3) questions on the exam. You may find questions that could have several answers and require an explanation or a justification. As we've said, many answers in storage systems are "It depends!". In these cases, we are more interested in your justification, so make sure you're clear. Good luck!

If you have several calculations leading to a single answer, please place a box around your answer.

Problem 1 : Short and concise answers. [55 points]

(a) Dr. Dimwit was thrilled to learn that his new disk drive has a MTBF of 1,000,000 hours. He immediately sold his backup devices, feeling confident that he is guaranteed years of fault-free storage. Why should he be nervous?

It's worth re-reading the MTBF reading even if you know the answer to this. MTBF makes sense when one is talking about failures in a population of disk drives, but cannot be used as a metric for estimating the time a single drive will fail.

(b) Bob, inspired by Greg's lectures about how large requests amortize overheads, proposes to use 20 MB requests whenever possible. Explain one difficulty he will face and one performance problem that may arise.

Difficulties that Bob will face: finding enough small buffers to coalesce into a big 20MB buffer. Or disk drive buffer size is smaller than 20MB, so problems matching media bandwidth with bus bandwidth. Performance problems: when reading small files, waste of bandwidth since have to read more than necessary. When writing, also waste of bandwidth since many read-modify-writes are necessary.

(c) Give two advantages of zoned recording?

Zoned recording increases disk capacity and media bandwidth.

(d) Although the C-LOOK scheduling policy gives slightly higher average response times then SSTF, it is still used in some application domains. Why does C-LOOK give higher response times and why is it still used sometimes, despite that?

C-LOOK gives higher response times since it moves the disk head one direction at a time. A request that may be closer to the disk head will not be considered if the disk head is moving at a direction away from that request. C-LOOK prevents starvation, however, and is fairer that SSTF.

(e) Upon coming home from a fun trip to Albania, Greg suddenly realizes that many of the pictures he took there will be hard to find in the future if he stores them in his hierarchical directory-based file system. Greg then contacts Eno to ask for a solution, and Eno tells him to install a database and store the pictures in that database. (1) Why would a database help in finding files? (2) What does Greg still need to do, even with a database, to ensure he'll find his files easily in the future? (3) Finally, why don't we get rid of file systems altogether, and use an off-the-shelf database to store files?

(1) A database allows an arbitrary number of metadata to be stored with a file. Metadata can include different attributes associated with a photo, e.g. time of day, GPS location, event, mood of photographer, etc. If all the photos are augmented with these attributes it's easier to find what one is looking for (e.g. "Give me all photos taken during daytime".

(2) Greg still needs to decide on the best attributes to use and insert them manually, for each photo, into the database.

(3) Performance has been the main reason file systems haven't disappeared. Databases case about atomicity and avoiding deadlocks and often induce more complexity in the path of a request, which increases latency.

(f) In POSIX systems, how can an application quickly read byte #1,234,567,890 in a newly opened file?

Use lseek.

(g) Heated discussions often arise when the log-structured file system is compared to shadow paging, as a mechanism for ensuring consistent updates to persistent storage. Briefly describe why log-structuring can be thought of as a form of shadow-paging or argue why it should not.

They are similar (one could argue they are logically the same). In shadow paging, a new update goes to a new page, and then the switch between the old and new pages is done atomically, usually by changing one bit. In LFS a new update to a file goes to a new location on disk, and the metadata for that file atomically changes...the old data can then be garbage collected.

(h) Prof. SmartyPants has invented a new storage interconnect technology that can transfer 100 GB/second. But, he is worried that it will take years for people to agree on a storage protocol for it. Why should he not worry?

Many possible answers here. The interconnect may be fast, but the disks are still slow, which means he shouldn't worry because his invention is useless. Another possible answer is that existing protocols like SCSI can still be run on that interconnect.

Problem 2 : Analyzing disk characteristics – the Skippy experiment. [25 points]

Raja has just come up with a brilliant idea (this is a very common occurrence). He is going to start a web-site devoted to the analysis of hard disks. Having just taken Greg's storage systems class, he decides that he will start by posting the results of the project one Skippy-based experiments for various disks on his web-site. Unfortunately, having not paid much attention in class, Raja is unsure as to how to run these experiments or interpret their results. Please help him out by answering the following questions.

(a) What three things must Raja remember to do on his lab machine before running the Skippy-based experiments? What will happen if he forgets to do these three things?

Raja must remember to:

- *Turn off the hard-disk read cache*
- *Turn off the hard-disk write cache*
- *Set up a raw device by which he can access the hard-disk. By accessing the hard-disk via the raw device, Raja will be able to bypass both the filesystem and the buffer cache when running experiments.*

Any caching of reads or writes to the hard-disk will mask the results that Raja wishes to see from the Skippy-based experiments (basically, he will observe only the time taken for the write or read to arrive at the cache, and not the time needed to actually go to disk). So, if he forgets to do these things, none of his experimental data will be correct.

Note: Though the above answers are what we were looking for, we also accepted pretty much anything else that had to be done before running the Skippy experiments (e.g., finding a free disk partition).

(b) Raja runs read-based Skippy and obtains the graph shown at the end of this handout.

(a) How many heads does this disk have? Label all of the head switches and cylinder switches on the graph.

The head switches are represented by the shorter spikes in the read-based Skippy graph, while the cylinder switches are represented by the larger spikes. The graph shows five head switches between each cylinder switch, so there are six heads.

(b) What is the rotational latency of this disk?

The rotational latency of the disk corresponds to the Y-intercept of the read-based Skippy graph. Looking at the graph, we see that the rotational latency is about 6ms.

(c) What are the MTM and STM values for this disk? Label the point on the graph that corresponds to MTM and STM.

The MTM value for this disk is about 0.25ms. The STM value is about 18 sectors.

(c) Raja now runs write-based Skippy (same as read-based Skippy, but reads are replaced with writes), but forgets to disable the disk's write-back cache. How would the graph differ from the correct output? How can Raja tell that he forgot to disable the write-back cache by looking at the graph?

Note: When answering this question, assume that the disk cache uses a FIFO replacement policy and does not coalesce multiple writes to the same sector.

If Raja forgets to disable the write-back cache, his write-based Skippy results will be translated in the positive x-axis direction by a stride distance of $\frac{cache\_size}{2}$ *sectors.*

To understand why the above answer is correct, lets have a look at how this experiment would proceed if the write-back cache were enabled:

- **When the write-back cache is not yet full**: *At the beginning of the experiment, when the write-back cache is not full, all writes will be cached. Since this is a write-back cache and* not *a write-through cache, the data contained in these writes will simply be buffered and not written to disk. Hence, the observed latencies in the graph for small stride distances will be equal and very small. Since each stride contains two writes, this will be the observed behaviour until a stride distance of* $\frac{cache\_size}{2}$ *sectors at which point the write-back cache will fill up.*
- **When the write-back cache has filled up**: *When the write-back cache fills up, the experiment will reach steady-state. In steady-state, each new write will cause an eviction of the oldest entry from the write-back cache. So, the first write in the stride pair of* $\frac{cache\_size}{2} + 1$ *sectors will cause the first write in the stride pair of one sector to be evicted, and so on. Thus, the observed latency for stride distances when the write-back cache has filled up will actually be latency necessary to write the stride pair that was just evicted to disk.*

(d) Raja is almost done running experiments! Thanks to your input, he re-runs write-based Skippy with the hard disk write-back cache disabled. However, he is confused by the fact that the MTM and STM values obtained from the write-based Skippy experiment are different from those that he obtained from the read-based Skippy experiment. Explain why you think these values should or should not be different. If you think they should be different, should the write-based Skippy MTM and STM values be higher or lower than the read-based Skippy MTM and STM values?

The write-based Skippy MTM and STM values should be higher than the corresponding values for read-based Skippy. This is because the hard-disk has to be much more careful with regards to positioning the disk head above a given sector when writing data versus when reading data. When reading data, the disk can attempt to avoid the cost of an extra rotation by trying to optimistically read the requested sector when the head is "almost" positioned correctly. The ECC codes written at the end

of each sector will inform the disk whether this optimistic read failed and whether an extra rotation is necessary in order to better position the head. Writes cannot take advantage of this optimization as writes are destructive; an optimistic write that writes the wrong sector will have disastrous consequences. As a result, the disk will never try to optimistically write a sector; it will always incur the cost of an extra rotation in order to guarantee that the disk head is positioned exactly *correctly.*

Problem 3 : File system layout. [21 points]

Many filesystem benchmarks specify strict rules that must be followed when the benchmark is run in order for the results to be considered valid. One such rule specifies that the underlying filesystem must be freshly formatted and empty before the benchmark is run. In a paper entitled *File System aging – Increasing the Relevance of File System Aging*, Keith Smith and Margo Seltzer argue that the results obtained from any benchmark that mandates this "fresh format rule" will show results that are atypical of real-world scenarios.

(a) Consider two identical Servers A and B that differ in only the age of their filesystems. Server A uses a FFS filesystem that has been freshly formatted, whereas Server B's FFS filesystem is six-months old and has seen steady usage during this time period. If the ACME benchmark is run against both, which will perform better? Explain your answer.

Server A will perform better. The difference in performance between both servers will be due to on-disk fragmentation. As an example, note that in order to minimize disk seeks when reading a single file, FFS tries to place the first twelve direct blocks of a file in the same cylinder group as the inode for that file. This placement policy can easily be realized in a new, empty, filesystem. However, in an aged filesystem, it may not be possible to find enough free data blocks in the cylinder group in which the file's inode is allocated. In such cases, the first twelve blocks will have to be split across different cylinder groups. This will increase the number of seeks necessary to read the file and hence decrease performance.

In addition to the example stated above, there are many other reasons why a new filesystem will perform better than an aged filesystem. Some of these alternatives are listed below:

(a) **Inode table fragmentation**: *FFS attempts to place the inodes for files within the same directory close to the directory itself. This placement policy attempts to utilize spatial locality in order to minimize seeks. However, as a filesystem ages, fragmentation in the inode allocation tables may prevent this optimal placement.*

(b) **Contiguous data allocation**: *FFS tries to allocate data blocks belonging to a file contiguously within a cylinder group. However, as blocks are created and deleted, it will become harder to find large enough contiguous free blocks.*

(b) If the two servers use LFS filesystems instead of FFS, which would perform better? Explain your answer.

Both filesystems should perform similarly if cleaning is not required on Server B. If all of Server B's log-segments have been used up and cleaning is required, then we would expect Server A to perform better.

LFS is not as susceptible to fragmentation problems as traditional filesystems because LFS always writes data out in contiguous chunks. However, LFS always writes data out to new locations and old versions of data are not immediately deleted. This free "versioning" of data yields many useful properties, but becomes a problem when LFS runs out of free log-segments. When there are no more free log-segments, LFS must activate a cleaner in order to delete old versions of data and free up enough contiguous space for new data. Depending on the cleaner implementation, LFS performance tends to decrease considerably when cleaning becomes necessary.

(c) Raja is running the ACME benchmark against his server which uses FFS. While doing so, he realizes that all of the files read and written by the ACME benchmark are 128KB in size. Raja knows that FFS uses a 8KB block size and inodes that contain 12 direct blocks references. Raja also knows that FFS changes the cylinder group in which a file's data is placed every time an extra indirect block is allocated. Suggest a change that Raja can make to FFS to potentially increase his server's performance on the ACME benchmark. Why will this change increase performance? Are there any potential disadvantages to this change?

*Only the first 96KB (12 direct blocks * 8KB/block) will be allocated within one cylinder group. The rest of the file will be allocated in another cylinder group, thus forcing a minimum of one seek when trying to read this file in its entirety. Any correct solution to this problem involved modifying FFS so as to avoid performing this seek when reading 128KB files. Some sample solutions are listed below:*

- **Change the number of direct blocks in an inode to be sixteen**: *This will cause FFS to try and allocate all 16 blocks of the ACME benchmark files in the same cylinder group. The disadvantage to this solution, however, is that the free space in individual cylinder groups will be used up more quickly. So this change will reduce the number of files for which FFS can achieve optimal data placement. Given these tradeoffs, we would expect Raja's server with this change to perform better on the ACME benchmark if the benchmark only creates a small number of files. If the benchmark creates a large number of files, performance will probably be worse.*
- **Increase the blocksize so that the entire file fits within the first 12 direct blocks**: *This will cause FFS to try allocate the entire file within the same cylinder group. However, in addition to the fact that it will increase internal fragmentation, it suffers from the same drawbacks as the previous solution.*
- **Use Extent lists instead of block pointers**: This answer isn't what we were looking for, but we accepted it as long as you mentioned that the goal was to minimize seeks by fitting all sixteen 8KB blocks within one extent (and hence within one cylinder group).

Problem 4 : Instructor trivia. [up to 3 bonus points]

(a) What should Greg's boys dress up as for Halloween?

The best answer we got for this question was "Hard-disk drives." Personally, if it were up to me, I'd say "Slimer" from "The Real Ghostbusters" and "Michaelangelo" from "Teenage Mutant Ninja Turtles."

We, of course, accepted absolutely anything for this question.

(b) Which TA has facial hair **on purpose**?

Even though Raja is known to resemble a gorilla at times (usually around deadlines), the correct answer to this question is: none of them.

(c) Which TA is from Albania?

Eno.

(d) Name Raja's favourite NFL football team (Hint: think of the hat he always wears during office hours).

The New York Jets.

(e) What is Greg's standard word for "bad performance", this term?

"Unfortunate."

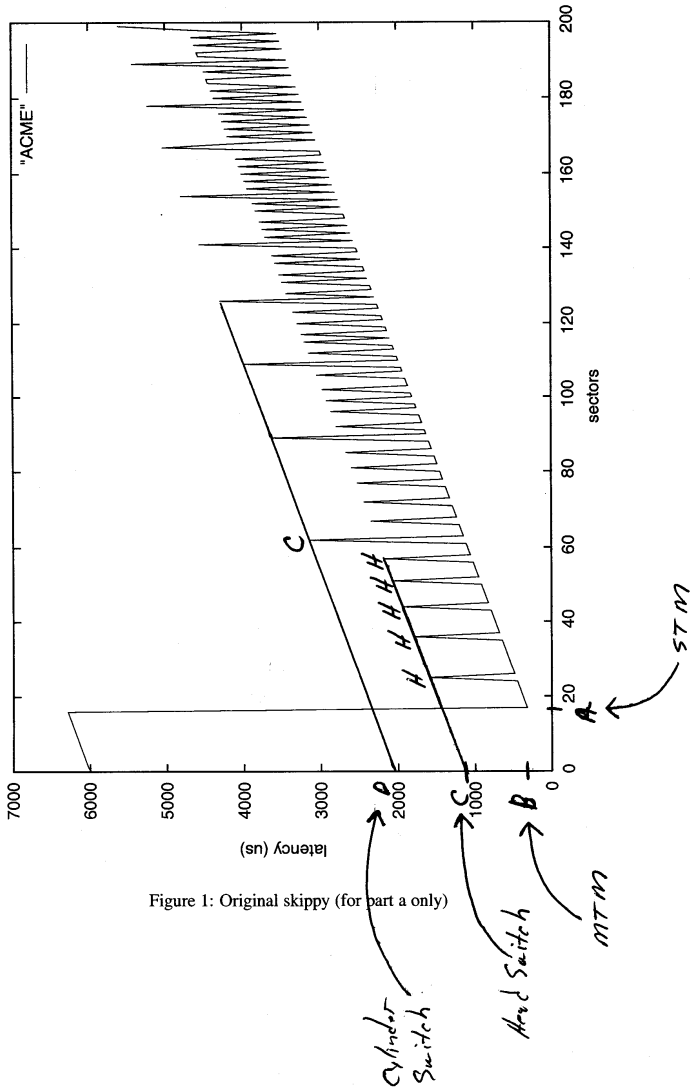

Figure 1: Original skippy (for part a only)

Figure 1: Raja's Read-based Skippy results