CMU 18–746 **Storage Systems** **20 Feb 2004**

Fall 2005 **Exam 1**

Name: SOLUTIONS

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 $\boxed{\text{box around your answer}}$.

Problem 1 : Short answer. [60 points]

(a) Large directories (i.e., those with many entries) in most file systems are very inefficient, because searching for specific entries is inefficient (O(n)). Give one simple approach to improving large directory efficiency without changing the on-disk directory format.

The key to this problem is improving directory efficiency without changing the on-disk directory format. The common directory format with O(n) lookup time is the unordered list. So, given a very large list of entries, what's the best way to organize them? These are answers that we accepted: sort the entries and perform lookups by doing a binary search, place the most frequently used entries near beginning of the directory, maintain a second internal data structure (e.g. a memory cache), and the most robust solution is to hash the directory entries.

(b) Log-structured file systems re-map modified file blocks into large segments, to achieve higher write efficiency. But, doing so also requires writing a new version of each modified files' inode. Why?

A log-structured file system writes new data to the end of the log and therefore changes the location of the file's data blocks. Because the inode points to the data blocks, if we change the location of the blocks we must also update the inode.

(c) Explain why a SCSI bus can only support up to 7 devices (in addition to the host's bus adapter).

Because each device attached to the bus is assigned one of the 8 data lines for selection, a SCSI bus can support at most 8 components (e.g., one HBA and 7 devices).

(d) What potential integrity issue arises from having a secondary index associated with a database table?

When adding or deleting an entry in the table, the index must also be modified. The two structures must be kept consistent with one another in the face of unpredictable system crashes.

(e) Explain why most device drivers are organized into a non-device-specific part and a collection of device-specific parts.

Most device drivers have a non-device-specific interface imposed by the operating system in order to abstract the details of each device driver from the rest of the operating system and to avoid redundant implementations of the non-device-specific parts.

(f) Most file systems allocate blocks of files near the directory that names them. Although this improves performance relative to random placement, it may not achieve the full performance of the disk when reading the set of files in a directory. Suggest two reasons why not.

Here are four reasons:

- *large files use up all the nearby blocks causing new files to live far away from the parent directory*
- *a file's blocks may not be accessed in the same order as their on-disk placement*
- *multiple files may be accessed in a different order then stored on disk*
- *files may still become externally fragmented within the "same cylinder"*

(g) Many file system caches hold dirty blocks for some time, allowing updates to be coalesced or even canceled (via deletion), but then flush them to disk. The longer they hold them, the bigger the potential performance improvement. What is the downside?

The longer you hold data in the cache, the greater chance you have of losing it if the system crashes before it is written out to disk.

(h) A common approach to flushing a file system cache is for a background process to periodically wake up and write out all dirty blocks. Explain why the average service time for these writes is often lower than that of cache misses.

Disk scheduling. Given the large number of write requests sent to the drive on a typically flush command, there are simply more opportunities to schedule. Individual cache misses (e.g., read request) do not often benefit from disk scheduling, because there is often only one (or few) outstanding request for an application (e.g., many applications block until they've read in the necessary blocks of data).

(i) Imagine a database table with 10000 records, each with 10 32-bit fields. Should the database system do the select first or the project first for the following query: SELECT <attr1,attr2> FROM <table> WHERE <attr2=FOO>? Assume 1% of records have <attr2=FOO>. Justify.

If we do the project first, we create a temporary table in memory consisting of the first two columns (attributes). The memory space required for this is $10K \times 8\ bytes = 80\ KB$*. If we do the select first, we are only selecting the rows where attr2 is equal to foo. This will require* $10K \times .01 \times 40B = 4\ KB$*. Doing the select is therefore a more memory efficient first step. Of course, the resulting memory usage from the select+project or project+select is the same.*

(j) Disk drive firmware designers rarely expect hits in a disk drive cache for data that was recently requested by the host. Why?

Data that is recently read by a host, if read again, will most likely be in the host's cache.

(k) Network-level flow control is used by Fibrechannel switches to prevent overflowing of their internal buffers, which can happen when multiple senders try to send data to the same receiver. Give an example where flow control is needed even when there is only one sender and one receiver.

The one sender could be sending multiple commands (the same effect as multiple senders), or a very large write command, faster than the receiver can accept the data. Flow control is necessary to speed-match between the sender and receiver.

(l) Some file systems store the data of very small files in the inode (using the space reserved for pointers to file blocks). Explain why they might be doing this.

To allow the system to read/write the inode and the data in a single disk operation.

(m) Imagine a disk that has a single zone. Even if it exposed the number of sectors per track, Joe Smarty claims that this is not enough information to compute exactly which track held each logical block. Support Joe's assertion with a concrete example.

Because of defect management (e.g., reserved space and block re-mapping to deal with bad sectors) the exact mapping of logical disk blocks is unknown.

(n) Why do modern disk drives normally only utilize one disk head at a time? (Rather than transferring from all heads in parallel.)

There's no guarantee that the tracks (on different surfaces) are aligned enough to allow the read/write heads to access data in parallel. Therefore, the heads would need to servo independently, but doing so would require an actuator per head which is expensive with current technology. The more cost effective solution is simply to buy more disks!

Problem 2 : Disk performance. [20 points]

Consider the following system: a computer running one application that sends 100 8KB random disk requests per second to a 10,000 RPM disk that has a 4.5ms average seek time and 325 sectors per track.

(a) What is the average service time?

$$\begin{aligned}
TimeRotation &= \frac{60\ sec}{10K\ RPM} \times 1000\ ms/sec = 6\ ms && (1)\\
Throughput &= \frac{(325 \times 512\ bytes/track)}{6\ ms/track} = 27.7\ MB/sec && (2)\\
TimeServer &= seek\ time + rot\ delay + transfer\ time && (3)\\
&= 4.5 + \frac{1}{2} \times 6 + \frac{8KB}{27.7\ MB/sec} = 7.8\ ms && (4)
\end{aligned}$$

(b) What is the utilization of the disk?

$$\begin{aligned}
Util &= \frac{arrival\ rate}{server\ rate} && (5)\\
&= arrival\ rate \times server\ time && (6)\\
&= 100\ requests/sec \times 7.8\ ms/request = .78 && (7)
\end{aligned}$$

(c) What is the expected queue depth for the disk?

$$\begin{aligned}
AvgTimeQueue &= AvgTimeServer \times \frac{Util}{1 - Util} && (8)\\
&= 7.8 \times \frac{.78}{1 - .78} && (9)\\
&= 27.65ms && (10)\\
AvgLengthQueue &= AvgTimeQueue/AvgTimeServer && (11)\\
&= 27.65/7.8 && (12)\\
&= 3.54 && (13)\\
& && (14)
\end{aligned}$$

(d) What is the expected response time?

$$\begin{aligned}
AvgResponseTime &= AvgTimeQueue + AvgTimeServer && (15)\\
&= 27.65 + 7.8 && (16)\\
&= 35.45 && (17)\\
& && (18)
\end{aligned}$$

(e) If the disk is attached via a 20MB/s SCSI bus, what is the new average service time?

If the disk and bus transfers are serialized then the transfer time is their sum. The time to transfer 8 KB of data over a 20 MB/s bus adds and additional .39 ms to our original 7.8 ms, giving us a new average service time of 8.19 ms.

If the disk and bus transfer can occur in parallel, then only the bus transfer will determine the transfer time, giving us a new average service time of 7.89 ms.

(f) What would happen if a second instance of the application were concurrently run on the same system?

The service time would remain the same, the server would be at full utilization, and the queue length and response time would grow without bound. In other the system explodes when the workload exceeds the server capacity.

Problem 3 : Instructor trivia. [up to 2 bonus points]

(a) What is the name of Greg's younger boy?

William

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

Adam

(c) How old will Timmy be this year?

4