March 28, 2016

A new technology is sweeping the storage-systems landscape. It is called *flash* and is the current best example of a more general trend towards solid state storage, in which electronics without any moving parts are used to persistently store data. The contrast, of course, is the more traditional storage approach of using hard-disk drives.
In these problems, the goal is to understand the impact of flash on file and storage systems.

A basic flash chip is composed of a large number of blocks; we'll assume that blocks are each 128 KB in size. Each block is further sub-divided into pages, each of which is 4 KB. Thus, a 1-MB flash chip consists of 8 blocks (128 KB per block X 8 blocks = 1 MB); each block consists of 32 pages; there are thus 256 pages in a 1-MB chip (8 blocks X 32 pages per block). Here is a diagram of such an arrangement:

```
|           Block 0             |                 Block 1          | ...
| Page0 | Page1 | ...| Page 31 | Page32 | Page33 | ...| Page 63 | ...
```

Each flash chip exposes three operations. The simplest is a `read(page)`, which reads a specific page and returns the data therein; this is quite similar to devices with which you are already familiar. Writing a flash is more complicated, however, and requires the use of two operations in tandem. Specifically, one must first call `erase(block)` to erase an entire block of the flash; erasing resets the contents of the entire block (thus losing all the data in the block!). Only after an erase can one then use the `program(page)` interface to write new data (one page at a time) to the block.

The cost of these operations is also uneven. Reads are quite cheap, taking on the order of 10 microseconds to read a page. Program operations are slightly more expensive, around 50 microseconds to program a page. Erase operations, however, are terribly expensive, and take 1 millisecond (1000 microseconds) to complete.

(PROBLEMS ON NEXT PAGES)

1) **Basic Flash Performance**.

   (a) Let's start with an easy calculation. Given the performance characteristics described above, how long does it take to perform 32 reads, of size 4 KB, to random locations on a flash device?

   (b) Now let's focus on writes. How long will it take to perform 32 writes, of size 4 KB, to different random locations on the flash device? Assume there is no live data anywhere on the device (and thus it is OK to erase a block without worry).

   (c) Now let's do 32 random4-KB writes to different random locations on the device, but with a key difference: the rest of the data on the device (and thus each block) is live. Thus, you can't just erase a block and then program a page within it, because that would lose data; instead, you need to perform a read-modify-write of any block you are updating. How long will these 32 writes take?

   (d) Let's do some sequential reads. How long does it take to read 32 consecutive 4-KB chunks from the flash? (assume the first read is aligned with the underlying flash block; i.e., all 32 4-KB reads will read from a single flash 128-KB block)

   (e) Our next focus is on sequential writes. How long does it take to write 32 consecutive 4-KB chunks to the flash? (assume the first write is aligned with the underlying flash block; i.e., all 32 4-KB writes will write to a single flash 128-KB block)

   (f) Finally, how long does it take to write 32 consecutive 4-KB chunks to the flash, in the worst case where you can no longer make any assumptions about alignment?

2) **File System Basics**. In this question, we now examine file system performance on flash devices and hard drives. Assume something like the very simple file system we first studied in class, with a single super block, a single data bitmap block (DB), a single inode bitmap block (IB), a series of inode blocks (I), and a series of data blocks (D); all blocks are 4 KB. Assume, for simplicity, that each inode is of size 4 KB (i.e., each inode takes up one full 4-KB block on disk, which is outrageous but makes your life easier).

   (a) Assume we are reading two files in the root directory, "foo" and "bar", each of which is 4 KB in size. In reading these two files in their entirety, which blocks will the file system read from disk? (assume that no file systems blocks save the superblock are in memory to begin with; assume also that blocks, once read, stay in the memory cache; assume there is no write traffic of any kind; finally, assume that the root directory only has these two files in it)

   (b) We are interested in how long this sequence of reads will take. Let's first assume we are running it on a hard drive (average seek of 4 ms, rotation of 15000 RPM, transfer rate of 100 MB/sec). Assuming we first read "foo", and then read "bar", how long will it take to read both files from disk?

(c) Now assume we are reading the files (first "foo", then "bar") from a flash device (10 microseconds to read a page, 50 microseconds to program a page, and 1 ms to erase a block). How long does it take to read "foo" then "bar" from the flash?

(d) Now assume we are reading the files from a smarter file system, such as the Fast File System (FFS). How much does performance change when reading "foo" then "bar" from a disk, using FFS instead of a simple file system? Assume the same disk (4 ms average seek, 15000 RPM rotation, 100 MB/sec transfer) as before. Estimate costs if you need to.

(e) Finally, assume we are running FFS on top of a flash. How much does performance change when reading "foo" then "bar" from a flash, using FFS instead of a simple file system? Is FFS needed on this device?

3) **FFS:** This question examines classic file systems and how they might need to change in order to adapt to new technologies.

(a) Classic file systems contain many policies that are tuned for hard disk drives. Consider the Berkeley Fast File System (FFS); what policies does FFS contain that are specific to hard drives?

(b) New computers are now being sold with flash-based drives instead of conventional rotating disks. Flash has this following performance property: excellent random I/O performance for reads (nearly the same as sequential). Assume that you run FFS on top of this raw flash device. What will the impact on read performance be? Just considering read performance, is FFS well-suited to flash?

(c) Flash also has this performance property: when writing to a page (assume a page size of 4KB), one must first erase an entire block (assume a block size of 128KB). Assume that you run FFS on top of this raw flash device. What will the impact on write performance be? Just considering write performance, is FFS well-suited to flash?

(d) Flash also has an unusual reliability property: when a particular block is erased "too many" (i.e., 10,000 or 100,000) times, it will "wear out" and thus become unreadable. Assume you run FFS on top of this raw flash. What will the impact on reliability be? Just considering reliability, is FFS well-suited to flash?