Meeting 12 :: SSDs

SSDs

Flash-based solid state storage devices (SSDs) are everywhere (phones, laptops, servers, ...), but unlike hard disk drives (HDDs), SSDs are difficult to reason about *externally*. With a hard drive, we can think about the costs of I/Os in terms of physical actions (moving the disk arm, a short rotational delay, and transferring as the platters rotate). With an SSD, our understanding is obscured by the presence of a **flash translation layer** (FTL). To truly know what is going on, we would need to know the internal state of the device as well as the (proprietary and super-secret) FTL's implementation details. For this reason, we mostly treat SSDs as a black box, and we try to give the SSD workloads that we hope are compatible with the things that the FTL is good at.

Learning Objectives

- Be able to describe the internal components of an SSD (banks, blocks, and pages)
- Be able to describe the various responsibility that the FTL performs (logical-to-physical page mappings, garbage collection, wear-leveling, error correction)
- Be able to describe the differences between SLC, MLC, and TLC and how those differences affect the characteristics/performance of the different types of NAND flash.
- Be able to describe the logical steps of writing data inside an SSD

Thought Questions

1. Given the write constraints of an SSD, one strategy that seems to make a lot of sense is to do log-structured writes (like LFS). In the LFS paper, we discussed garbage collection using the simple *greedy* or the *greedy cost-benefit* approach. Suppose you were asked to redesign garbage collection for an SSD. How would device wear-out affect the way you design your GC strategy? (You may want to think about things like initial data placement and victim selection)
2. One thing that you aren't always told about an SSD is its actual physical capacity. The difference between actual and usable capacity is called the over-provisioning percentage. Why might the actual capacity differ from the usable capacity (e.g., what types of things might an SSD designer use the over-provisioned blocks for)?
3. The reason you can't overwrite flash pages is because you can only program them precisely --- you must erase at block granularity. Suppose you know the contents of a flash page. Are there *any* modifications that you could make to update that page's contents without erasing it? Can you think of any clever data structure designs that would let you take advantage of the set of legal of overwrites? (This is an open question, with recent top-tier publications exploring new designs)
4. Flash pages have what is called a "page private area", that is available exclusively to the FTL. I recently read that a particular flash device with 16KiB pages has 2208 additional bytes per page set aside as the page private area. What types of things might the FTL use this page-private area to do? How many bytes to do you think each of those tasks might require? Dose 2208 bytes seem like a good ratio per 16KiB page?
5. To get peak performance on a high-end SSD, you need to exploit its parallelism. Expensive SSDs support very high queue depths --- the system must always have many outstanding I/Os to keep the device saturated and maximize the available bandwidth. What types of applications/workloads generate this type of traffic? Under those types of workloads, what other resources may become bottlenecks?
6. One thing that is important when evaluating different algorithms is the I/O amplification. The book defines write amplification as "the total write traffic (in bytes) issued to the flash chips by the FTL divided by the total write traffic (in bytes) issued by the client to the SSD. What seem like the main sources of write amplification given what you know about FTLs? Why is write amplification such a useful measure for evaluating FTL designer (e.g., what are the negative effects of high write amplification)?

7. Suppose you are tasked with writing a new file system because, let's face it, FAT is not the best. You are given two options: (a) you are granted direct access to the SSD internals, and you can access the banks/blocks/pages/cells individually using whatever API you define, or (b) you are given an SSD with an FTL and you access that SSD using the standard block interface. What are the tradeoffs? When would you choose (a) vs. (b)? What happens if you design your system and then are given a new SSD a year later as new advances are made in SSD hardware?
8. Why does your phone have an SSD instead of a hard drive?
9. What type of FTL do you think a typical commodity SSD has (page-mapped, block-mapped, or hybrid)?
10. Other commands/interfaces for SSDs are commonly available besides read/write.

- TRIM is one command that lets a user tell the device that a block is no longer needed. How might an FTL react to a TRIM command?
- Streams are a way an application (or FS) can "tag" a write. There are a fixed number of streams (suppose they correspond to the integers 1-16). The SSD is then presented with new data and an ID for that data. How might it use this information? What criteria might an application use when assigning stream IDs to its writes?