CHAPTER 8

SSD Firmware Design Considerations

In this chapter, we will discuss the design considerations for solid-state drive (SSD) firmware. We will start by discussing the different types of SSDs and their requirements. We will then discuss the different components of SSD firmware and how they interact with each other. We will also discuss the different challenges that need to be addressed in the design of SSD firmware.

Design Considerations

1. SSD host interface: SATA, NVME, SAS, USB, etc.
2. Cache (RAM) memory availability:
    - To transfer data to/from host/SSD
    - Mapping table
3. Number of processors and their internal memory availability
4. Number of NAND channel supports

G. Kuppan Thirumalai, *A Beginner's Guide to SSD Firmware*,
https://doi.org/10.1007/978-1-4842-9888-6_8

5. Type of NAND used, characteristics, and limitations: SLC, MLC
    - Limited number of program/erase cycles (SLC: 100,000, TLC: 3,000, QLC: 1,000)
    - Erase block-wise, write page-wise; erase before write.
6. Performance requirements
7. Benchmark requirements

At a high level, an SSD operates by using a series of memory chips to store data. These memory chips are organized into pages and blocks, with each page typically able to store around 16 kilobytes (KB) of data and each block consisting of multiple pages. In order to write data to an SSD, the firmware must first erase the designated block. This is necessary because NAND flash memory cells can only be written to if they are in the erased state. After the erase operation is complete, write the pages within the block in a sequential order. Recall that this serves as a basic foundation on which we can build.

Once a page has been written to, it cannot be overwritten directly. Instead, the firmware must first erase the block that the page is a part of, which will also erase all of the other pages in the block. Rather than erasing and rewriting the same block, a new erased block should be chosen with a similar P/E (program/erase) cycle. This process is known as wear leveling and is used to ensure that all of the blocks in a die have been written to an equal number of times, thus extending the lifespan of the SSD.

One of the key components of SSD firmware is the *mapping table*, which is used to keep track of the location of data on the drive. The mapping table maps logical block addresses (LBAs) to physical block addresses (PBAs), which represent the location of data on the drive. When data is written to the drive, the SSD firmware uses the mapping table to determine the location where the data should be stored.

When data is read from an SSD, the firmware uses the mapping table to determine the physical block where the data is stored. The firmware then retrieves the data from the physical block and sends it back to the host device.

One of the key challenges in designing SSD firmware is ensuring that it is able to efficiently manage the various processes involved in storing and retrieving data. This includes optimizing the wear-leveling algorithm to minimize the number of times that blocks have to be erased, as well as managing the mapping table to ensure that data can be retrieved quickly.

Another important aspect of SSD firmware is ensuring data integrity. Because an SSD has no moving parts, it is less susceptible to physical damage than a traditional hard-disk drive (HDD). However, SSDs can still fail due to a variety of factors, including the failure of memory chips or the corruption of data. To protect against data loss, SSD firmware typically includes error-correcting code (ECC) and other mechanisms to detect and correct errors.

Unexpected Shutdown

An unexpected shutdown in an SSD occurs when the power to the device is unexpectedly interrupted while the device is in operation. This can happen due to power outages, surges, spikes, sags, or brownouts, as well as by manually removing the SSD from the system while it is powered on. What happens to data in transit to the SSD when there is an unexpected power interruption is an item overlooked by many industrial Original Equipment Manufacturer (OEM) host system designers. Limiting the system's exposure to data loss should be high on the list of design priorities.

This power loss will not cause issues during an idle or read operation, but if a write operation is occurring, there is the potential for some data loss or worse. Power loss during a write is also known as Write Abort. The main consequences of an unexpected shutdown during a write operation are file-system corruptions and internal device data corruption.

File-system corruptions occur when the operating system is unable to update the file-system records before the power is lost. Most operating systems will perform a file-system repair operation on the next power-up. Or it can typically be repaired by running a command or utility on the next power-up.

Internal device data corruption is more severe, as it can result in the entire flash drive's becoming unusable due to the corruption of the SSD's internal metadata, requiring a low-level format, which results in the loss of all data on the drive. To minimize the risk of data loss due to an unexpected shutdown, system designers should, in the design process, prioritize recovering the system effectively after such events.

One option is to take frequent recovery points and implement algorithms to find and restore data up until the restore point. Additionally, a special algorithm can be implemented to find the last page that was successfully written in a block. This can help protect against power interruptions and reduce the risk of data loss without the use of capacitors, which are often used to provide a temporary power source during unexpected shutdowns. By implementing these measures in the SSD firmware, designers can effectively address the issue of unexpected power loss and ensure the integrity of data in transit to the solid-state drive.

Power-Loss Protection

To effectively handle unexpected shutdowns and protect against data loss, designers have several options. One approach is to use power-loss protection capacitors in the hardware design of the SSD firmware. These capacitors provide a temporary power source in the event of an unexpected power loss, allowing the firmware to complete any in-progress writes and save any buffered data to the NAND.

In enterprise computing, data-loss protection is considered to be much more critical than it is in client computing. During an unexpected power loss, the SSD firmware can detect the power loss using hardware support and take steps to ensure all the unsaved data in the SSD is saved to maintain the integrity of data. This may include completing any in-progress writes to lower or upper pages (TLC), dumping buffered writes from non-volatile memory into the NAND (using SLC for faster write speed), or using hold-up circuitry to preserve enough time and energy to save the Flash Translation Layer (FTL) mapping table and other un-flushed data to the NAND.

Power-Loss Design Considerations

The power-loss protection mechanism in SSD firmware is a vital aspect of ensuring data integrity and preserving content metadata during unexpected power failures. While the volatile RAM translation table facilitates fast data access and updates during normal SSD operation, it is susceptible to data loss in the event of power loss. To address this challenge, the firmware adopts a proactive approach by utilizing persistent data structures stored in the non-volatile NAND flash array. These data structures contain essential content metadata and enable the reconstruction of the translation table during the next drive initialization. The firmware employs error protection mechanisms such as error-correcting codes (ECC) to safeguard the stored metadata from potential corruption. During the power-loss handling process, the firmware detects power loss, stores content metadata in the NAND flash array either alongside user data or in a separate block, and subsequently reconstructs the volatile RAM translation table on SSD initialization. This comprehensive power-loss protection mechanism ensures data reliability, minimizes data loss risks, and contributes to the robustness and efficiency of SSDs.

Individual modules need to maintain persistent data for simplicity and efficient operation. To ensure data integrity and recoverability, this persistent data is periodically saved from volatile RAM to the non-volatile NAND flash at restore points. Each module is responsible for updating its respective data structures, allocated in designated sections in RAM. Restore points can be created after the first boot, following an unexpected shutdown, when the persistent data buffer is full, or in response to program errors.

During restoration after an unexpected shutdown, minimizing the read time for persistent data from NAND is crucial to achieve faster boot times. Hence, efficient data-retrieval mechanisms should be employed. Regardless of whether the shutdown was safe or unsafe, during every subsequent boot, all restore-point data structures should be restored to their previous state, ensuring the system's consistent operation. To safeguard against data corruption, these data blocks should be protected by robust error-protection mechanisms, such as error-correcting codes (ECCs).

In extreme scenarios, if error correction fails, the device should still be able to boot, albeit in a read-only (RO) mode, ensuring that the data remains intact and is not subjected to further risks. The combination of efficient restore points, error protection, and robust recovery mechanisms ensures the reliability and resilience of the system in handling unexpected events and contributes to an overall improved user experience.

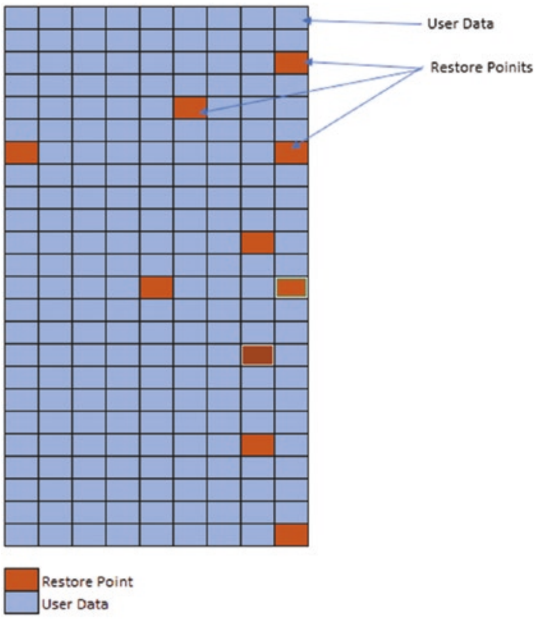

Figure 8-1. System restore point for unexpected shutdown handling

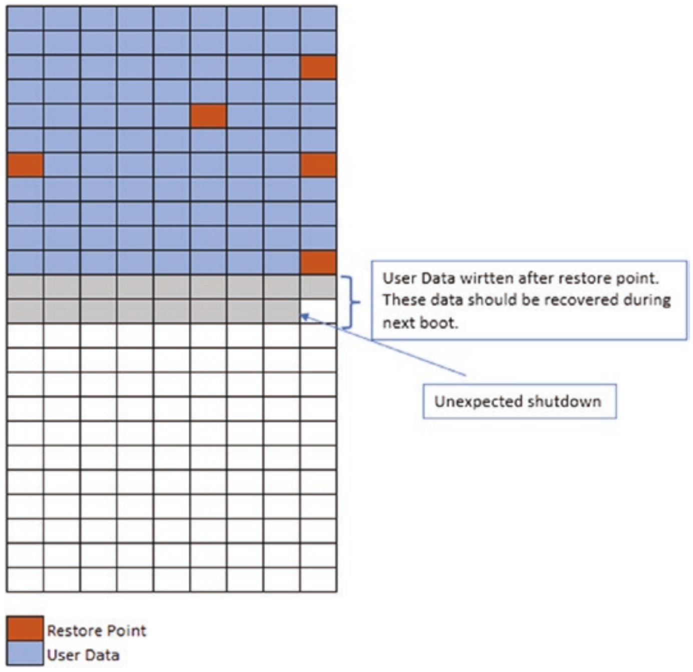
User Data wirtten after restore point. These data should be recovered during next boot.
Unexpected shutdown
Restore Point
User Data

Figure 8-2. *Unexpected shutdown during user data write*

The design considerations of SSD firmware are a complex process that involves optimizing various algorithms and data structures in order to maximize the performance and reliability of the drive.

Best Practices for Optimizing and Maintaining SSD Firmware

Next we will examine some key concepts for optimizing and maintaining SSD firmware, including reducing DRAM (dynamic random access memory, volatile memory) access, minimizing the code in the critical path of read and write operations, and managing firmware state snapshots. As fellow programmers, it is important that you understand the best practices for optimizing and maintaining SSD firmware. SSDs are becoming increasingly popular, and their firmware is complex. SSD firmware can have a significant impact on performance, reliability, and security. By following best practices, programmers can develop firmware that is more efficient, reliable, secure, and user-friendly.

One of the key considerations for optimizing SSD firmware is reducing the number of accesses to the DRAM on the drive. DRAM is a type of memory that is used by the SSD to store data temporarily, but accessing it can be slow and consume a significant amount of power. By reducing the number of accesses to the DRAM, it is possible to improve the performance of the SSD and reduce its power consumption.

One way to reduce DRAM access is to include less code in the critical path of read and write operations. The critical path is the sequence of operations that are performed when data is being read from or written to the drive. By reducing the amount of code in the critical path, it is possible to speed up these operations and reduce the amount of time that the drive spends accessing the DRAM.

Another approach to reducing DRAM access is to schedule read operations for data maintenance tasks, such as garbage collection and wear leveling. By performing these tasks during times when the drive is not being heavily used, it is possible to reduce their impact on the performance of the drive and minimize the number of accesses to the DRAM.

In addition to reducing DRAM access, it is also important to manage the firmware state snapshot data (management data). The firmware state snapshot is a copy of the firmware that is stored on the drive and is used to restore the firmware in the event of a failure. By managing this data carefully—i.e., keeping the management data as small as possible and writing only when necessary and when firmware is idle—it is possible to reduce the amount of space that is used by the firmware state snapshot, which can help to improve the overall performance and reliability of the drive.

Summary

In conclusion, optimizing and maintaining SSD firmware requires a careful balance of performance, power consumption, and reliability. By focusing on reducing DRAM access, minimizing the code in the critical path of read and write operations, and managing the firmware state snapshot data, it is possible to create SSD firmware that is optimized for performance and reliability.