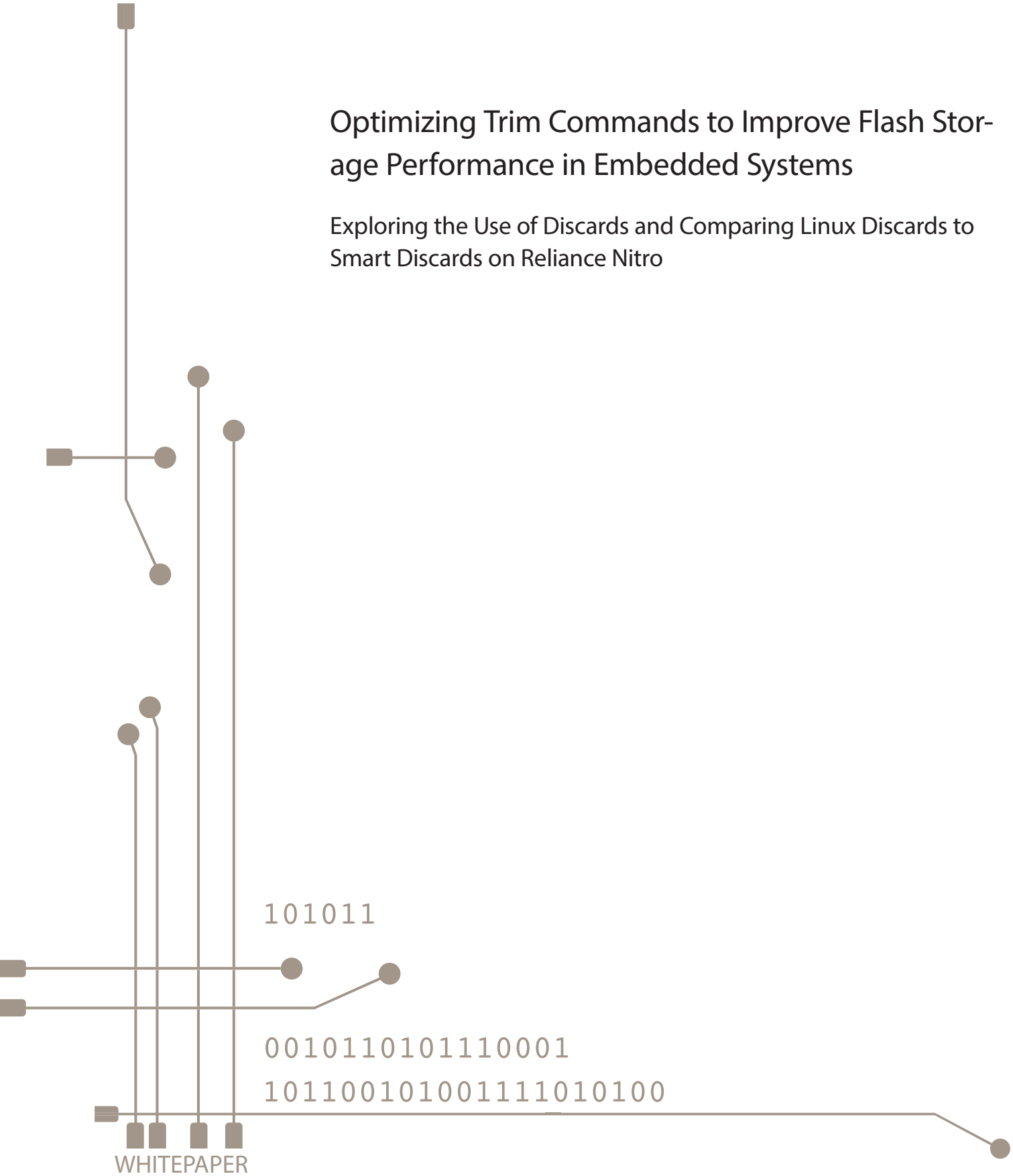
Datalight

Optimizing Trim Commands to Improve Flash Storage Performance in Embedded Systems

Exploring the Use of Discards and Comparing Linux Discards to Smart Discards on Reliance Nitro



WHITEPAPER

Introduction

Solid state drives (SSDs), and their embedded cousins, eMMC and UFS, are seen by many as the future of mass storage. However, steep declines in performance can arise if the NAND flash is not utilized properly – and doing that requires a steady diet of discards. What does that mean for the flash and for the file system, and how can you balance the performance and the cost? This paper will break it down for you.

Contents

Discards: A Brief History

Discard (also called trim, unmap, or erase in various command sets) is a command which informs the storage media that the contents of a sector or range of sectors are no longer important – the data has been deleted or replaced - and thus the sectors' contents do not need to be preserved until such time that the sectors are rewritten.

When a file is deleted from a file system, the content is marked obsolete but continues to occupy space on the media and more importantly, will continue to be managed by such tasks as wear-leveling and garbage collection. It is far better to notify the flash media that this data is no longer in use, a process Datalight pioneered in 1997 with the Discard command for CardTrick, a predecessor to FlashFX. This command was originally connected to a Datalight utility, FAT monitor on Microsoft Windows CE and Datalight ROM-DOS, then later became part of the Reliance file system.

The same type of interface was later utilized by some hard drive companies who used a DRAM cache for their rotating media – no need to keep the discarded blocks in RAM, or to run background defect management and error recovery processes[1]. On ATA, this was a trim command, and on SCSI, this was the unmap command. For SSDs, this command was referred to as a trim or discard, and we will use the term discard through the rest of this paper to refer collectively to any of these.

The first open source file systems to support discard commands were available in Linux on the 2.6.28 kernel, from December of 2008. These were followed by Microsoft (Windows 7 and above) and Mac OS X (10.6.8 and above), among others.

The Benefits of Discards

Why are discards now available in so many real-time operating systems (RTOS) and on so many devices? Primarily for the benefits available on NAND flash media, which may require a little explanation.

A flash memory device consists of one or more chips, which in turn consist of erase blocks, which in turn consist of pages. The only way to remove data is with the erase command, which resets every page in one erase block to the erased state. Only pages in an erased state can be written; a page, once written, cannot be rewritten without re-erasing the erase block and losing all the page data. Normally there is also a requirement that pages within an erase block are written sequentially. Most file systems are not designed to follow these rules, so a flash translation layer (FTL) is run atop of the raw flash memory and simulates a hard disk with rewritable sectors. The FTL can be software, like Datalight's FlashFX Tera, or it can exist in firmware, as with eMMC, SD cards, SATA SSDs, and so on.

When a modern high-performance FTL (a page-based FTL as opposed to the older block-based FTLs) gets a write request, it will write into an erase block with erased pages and maintain metadata to record the physical location of the written sectors. When a sector is rewritten, it gets written to a new page (this is referred to as copy-on-write) and the metadata indicating its location is updated. Eventually, after enough has been written, the FTL will begin to run low on erase blocks which have not been written. Many of the erase blocks will contain pages with obsolete data (dead pages), sectors that have since been rewritten. To reclaim space, the FTL will pick an erase block with dead pages, copy out the live (not dead) pages, and then erase the block so it can be reused. This process is called garbage collection (or compaction in FlashFX terminology). Garbage collection can be an expensive process, substantially slowing down writes to the disk. The more live pages there are, the longer it takes for garbage collection to reclaim space, because every live page must be read and programmed in a different block before the old block is erased.

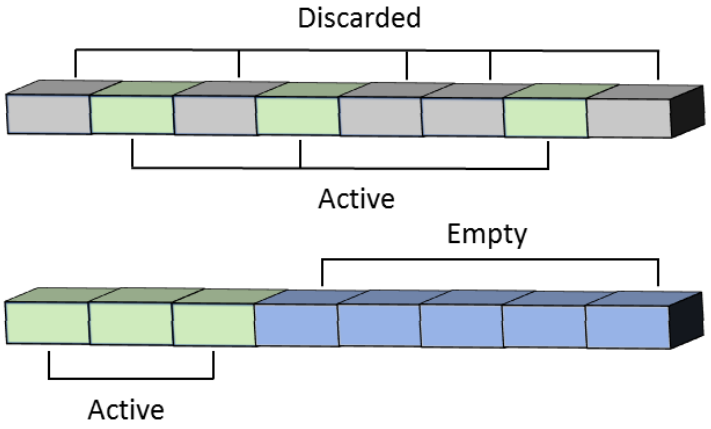


Figure 1: Compaction (Garbage Collection) on NAND flash media

Discards improve the process of garbage collection by decreasing the number of live pages. Each discard lets the FTL know that the contents of that page are no longer important. These pages are not copied when the erase block is garbage collected, which makes that process (and writes to the disk) faster. Discards also reduce FTL write amplification, which increases how long the flash memory will last before it wears out.

Discards have little, if any, positive impact until the disk starts garbage collecting. Attempting to gauge the impact of discards by formatting the disk (which usually discards the whole media) and running a short benchmark is almost always misleading, because the FTL has enough unused erase blocks to make it through the whole test without any garbage collection. Longer and smarter testing is required to evaluate the impact of discards.

Potential Drawbacks with Discards

Discard commands need to propagate through all layers of the code. As an example, an ext4 partition on top of LVM (Logical Volume Management), which is in turn on top of a dm-crypt volume will need discards enabled at all three levels in order to effectively reach the SSD.

Discards are not free. They can take time to execute, which can impose a performance penalty for those operations which issue discards. They also often happen synchronously, blocking other I/O requests.

Most FTLs need to write a record to indicate that a sector has been discarded; thus, in certain scenarios, discards can result in writing more to the flash memory instead of less. These downsides to discarding tend to be more pronounced when the discards are small; deleting or truncating or overwriting small or fragmented files can result in especially expensive discards. Some storage arrays will ignore discards that are smaller than a megabyte for this reason[2].

The severity of these drawbacks depends on the use case and how well the storage media processes discards. In some cases, it is more efficient to turn discards off and pay the penalty during garbage collection.

Demonstrating the Impact of Discards

Most file systems on Linux have the option to be mounted with discards on or off. Most developers simply add the option "discard" to the mount options in /etc/fstab. This enables the file system to report to the media when blocks are modified or files are deleted. In fact, this immediate notification can generate operations on the media, and cause unexpected reduction in performance.

To demonstrate this, we measured ext4 with and without the discard option. The test we created filled the media with a large sequential write, erased that data and then created a series of fragmented files. We then measured how long it took to delete those files. Following that, another large sequential file was created using the same space. We measured the throughput of the media while that create is taking place.

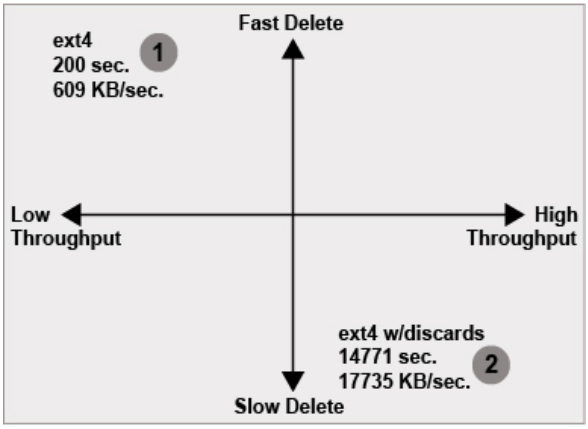


Figure 2: Test time and performance on ext4 with and without discards. Custom test using a Micron M500IT SSD and ext4 on Linux kernel 4.4

We found that with discards off, ext4 took 200 seconds to delete the fragmented files. These deletes happen in the file system, but the media does not erase the data – there has been no discard notification. The subsequent create thus pays a heavy compaction penalty, and the subsequent throughput is quite poor – 609 KB/sec. (item 1)

With discards on, the subsequent create has much greater throughput, nearly 18 MB/sec. However, the process of deleting the files issues a discard for each extent, which happens immediately and synchronously. This slows the overall process for deleting files from 200 seconds to over four hours! (item 2)

Performance Impact vs. Use Case

This use case neatly demonstrates the performance choices required. It may not describe the use case for your design, but elements of this use case may impact your work without you realizing it.

A third option that is gaining popularity on Linux is to leave discards disabled (which avoids performing the operation in real time) and instead running an occasional script with fstrim. This command will periodically inform the media which blocks are free[3].

Some system latency will be created when this script is run, and how much latency is dependent on how many blocks are free – ext4 discards all free blocks for fstrim, whether or not they were already discarded. In the time between when the operations are performed and when the script is run, blocks pending discard will be wear leveled and compacted. Finally, not all RTOS environments have support for fstrim, or the capability to run this as a regularly scheduled job.

Reliance Nitro's Smart Discards – A Balanced Option

The newest release of Reliance Nitro provides a better configuration option – Smart Discards.

This feature provides a better option for discards, particularly on managed NAND flash (e.g., eMMC, SD, SSDs, etc.) with page-based FTLs. This feature defers discard requests until they are larger, or until they can be performed in the background.

Utilizing the same test and hardware platform as earlier, we found Reliance Nitro with smart discards to have the best possible performance characteristics.

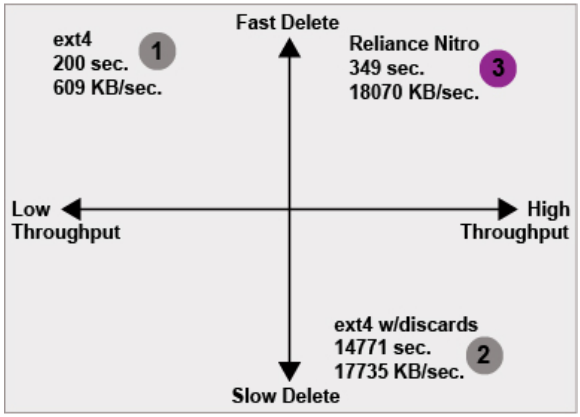


Figure 3: Test time and performance for ext4 and Reliance Nitro

With Smart Discards enabled, Reliance Nitro took 349 seconds to delete the fragmented files, comparable with ext4 without discards. The subsequent create had a throughput of just over 18 MB/sec, which is comparable to ext4 with discards enabled. (Item 3)

Further Impact – Erase Counts

Both the Micron eMMC and SSD showed very similar performance for this test. While running on the eMMC media, we also utilized a vendor command to investigate the erase counts. This allows us to understand the impact of a given file system discard solution on the media.

The erase counts for the initial provisioning portion of the test were very similar. Major differences began to appear in the portion that writes and overwrites the fragmented files, the erases required to delete the files, and the erases required by the subsequent sequential write.

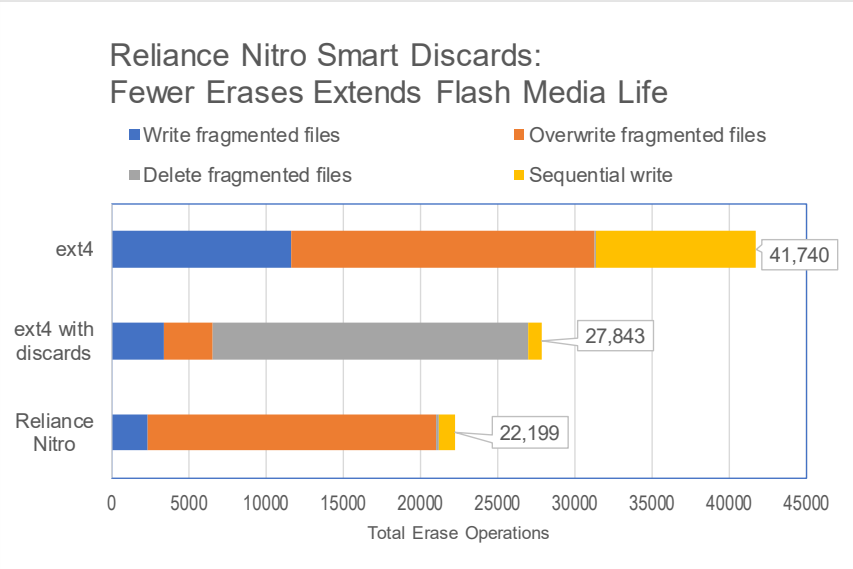


Figure 4: eMMC erases required by test portions, comparing ext4 with and without discards and Reliance Nitro. Tests performed on a Pandaboard running Linux 4.1 with Micron eMMC media.

We found that ext4 with no discards required a substantial number of erases for each of the write tests. Turning on discards for ext4 greatly reduced the erases required for write and improved the overall total (27,843 compared to 41,740), but the total of erases required for deleting the fragmented files (and the time those erases took) was considerably more than expected. Datalight's solution caused similar erases to ext4 while overwriting the fragmented files (understandable for a copy-on-write file system) but had the lowest overall total – 22,199 erases, or just 47% of the total required by ext4.

Summary

Discards are one method to help improve the lifetime and performance of flash based media, including eMMC and SSD. Enabling discards on Linux file systems can introduce system overhead. Utilizing Datalight's Smart Discards, available with the Reliance Nitro 5.0 release, provides the best balance of performance and lifetime – a 25% improvement shown with this test. This feature is also available for embedded designs across a wide range of RTOS environments beyond Linux.

References

1.) http://t13.org/Documents/UploadedDocuments/docs2008/e07154r6-Data_Set_Management_Proposal_for_ATA-ACS2.doc

2.) https://forums.freebsd.org/threads/56951/#post-328912

3.) https://blog.neutrino.es/2013/howto-properly-activate-trim-for-your-ssd-on-linux-fstrim-lvm-and-dmcrypt/

About Datalight

Datalight leads the industry in software technologies that manage data reliably in embedded devices. For more than 30 years, our focus on portable, flexible solutions has enabled customers to save money, reduce development time and get to market faster. Our customers have discovered that our solutions result in unparalleled interoperability and increased customer satisfaction. These accomplishments have helped Datalight earn a reputation as a provider of reliable and cost effective software solutions that are backed by a commitment to customer service and satisfaction.

For more information, call 425.951.8086 ext 100 or visit www.Datalight.com.

Datalight, Inc.
22118 20th Avenue SE, Suite 135
Bothell, WA 98021 USA
1-800-221-6630
www.Datalight.com