FlashVM: Revisiting the Virtual Memory Hierarchy

Mohit Saxena and Michael M. Swift
Department of Computer Sciences
University of Wisconsin-Madison
{*msaxena,swift*}*@cs.wisc.edu*

Abstract

Flash memory is the largest change to storage in recent history. Most research to date has focused on integrating flash as persistent storage in file systems, with little emphasis on virtual memory paging. However, the VM architecture in most of the commodity operating systems is heavily customized for using disks through software layering, request clustering, and prefetching.

We revisit the VM hierarchy in light of flash memory and identify mechanisms that inhibit utilizing its full potential. We find that software latencies for a page fault could be as high as the time taken to read a page from flash, and that swap systems are overly tuned towards the characteristics of disks.

Based on this study, we propose a new system design, FlashVM, that pages directly to flash memory, avoids unnecessary disk-based optimizations, and orders page writes to flash memory without any firmware support. With flash prices dropping exponentially and speeds improving, we argue that FlashVM can support memory intensive applications more economically than conventional DRAM-based systems.

1 Introduction

> ***Tape is Dead, Disk is Tape, Flash is Disk, RAM locality is King.*** –Jim Gray [6]

Flash memory is aggressively following Moore's law: it is cheaper than DRAM and faster than disks. With these trends, research has focused on integrating flash devices as a replacement to disks for storage [1, 24]. We assert, however, that flash also provides the underlying performance and price characteristics to back virtual memory. Specifically, its low read latency allows faster page access, and its relatively longer write latency is hidden through asynchronous page write operations.

While flash-based virtual memory has been previously investigated [11, 13, 19], its use has been debated. System administrators relying on anecdotal wisdom integrate flash disks as swap space to improve the responsiveness of their systems [3]. Others have advocated avoiding flash for swapping due to its limited write endurance [17]. In this paper, we investigate the truth behind these issues and revisit the VM hierarchy in light of flash memory.

Servers are often statically provisioned with enough memory to avoid swapping, but swap performance still matters for laptop and desktop systems. Workloads on these systems vary widely: running too many programs at once or working on a larger-than-normal data set can cause memory pressure and hence swapping. In both cases, swapping to flash can provide comparable performance at a lower price, or better performance if the system limits the amount of DRAM that can be installed.

Based on the price and performance characteristics of flash, we propose FlashVM, a restructured virtual memory system tuned for swapping to flash memory. FlashVM gives complete control over swapping to the VM system, rather than splitting it between the VM system and the block subsystem.

We also show how the VM hierarchy in most modern operating systems is overly tuned to the performance characteristics of disks. This has resulted from decades of disk being the *only* option for swapping [4, 15]. Since flash media behavior differs from disks, most of these disk-focused optimizations inhibit the optimal use of these devices. For example, flash devices have large performance differences based on write patterns (sequential is much better than random), but not read patterns.

This paper revisits the Linux virtual memory hierarchy and presents opportunities to improve its performance with flash memory. First, we find that software latencies vary widely and can be as high as the read access latencies of flash memory, and thus must be streamlined. Second, we find that the Linux VM system makes no attempt to optimize write behavior. With experiments on flash and disks, we show that with the current Linux VM system, memory intensive applications can execute from 60% slower to 65% faster when compared to traditional swap disk alternatives.

Device	Sequential (MB/s)		Random 4K-IO/s	
	Read	Write	Read	Write
HDD	56	45	120-300/s	
USB flash	11.7	4.3	150/s	20/s
SSD	250	170	35K/s	3.3K/s
PCI-e flash	700	600	102K/s	101K/s

Table 1: Hard disk and NAND flash memory characteristics.

2 Flash Memory: Now and Then

NAND flash memory technology has witnessed several big changes in the recent years. Many new manufacturers have joined the race to produce faster and cheaper flash devices [23]. In this section, we give a background on the key flash characteristics that distinguish them from modern hard disks.

Solid-state disks (SSDs) integrate firmware to provide a disk-like interface, such as SATA, on top of flash storage. This firmware, the Flash Translation Layer (FTL) [9], remaps the logical block addresses to physical flash addresses. It also provides wear leveling to increase write endurance. This layer is particularly designed for compatibility with existing file systems and storage interfaces, and may not be ideal for virtual memory. Furthermore, SSDs present a persistent storage abstraction, and also store the address mapping. This is unnecessary if using flash for virtual memory, as swap files are inherently temporary and volatile.

Transfer rates: Flash media differs from disks in terms of the performance variation between different devices, low read latencies, and their read/write performance asymmetry.

In contrast to disks, there is a wide range of performance for flash devices available in the market today, as shown in Table 1. Inexpensive devices such as USB flash sticks or camera memories offer moderate read bandwidth but have poor random-write performance. Solid-state disks (SSD), with a standard SATA interface, provide much better performance, about 3x better than the fastest hard disks, with 100MB/s sustained transfer rates. This results from intelligent block mapping schemes, parallel I/O accesses to multiple flash chips (similar to RAID) and write buffering [12]. High-end flash drives connected with the PCI-e interconnect interface are even faster, thereby enabling terabytes of virtual memory with speeds nearer to DRAM. Thus, a flash-enabled VM system must accomodate a variety of device performance.

Unlike disks, flash storage provides fast random read access (0.1ms vs. 8ms). Write access, however, is slower and rewriting a block requires erasing it completely, which may take up to 1.5ms. As the erase block is often larger than a file system block or memory page (128kB vs. 4 kB), FTLs often use log structuring to avoid rewriting flash pages in-place [1]. The asymmetry between read and write performance encourages usages of flash to optimize for asynchronous sequential writes, while the low latency of flash allows synchronous random reads.

Write endurance: Unlike disks, flash restricts the total number of writes to each block. Typical flash devices can sustain 100,000 to 1 million overwrites. Thus, 1-10 *petabytes* of data can be written to a 10GB flash over its lifetime. However, this assumes that the system writes to every block evenly. While file systems can greatly reduce write traffic through caching, paging loses its usefulness if pages are not written to storage and hence is less affected by caching.

To analyze whether flash devices have the overwrite capacity for paging workloads, we analyze a three-day block access trace for swapping traffic on a research workstation running Linux and configured with 700MB of physical memory. We use a pseudo device driver to intercept the block I/O requests to the swap device. Our measurements indicate a write rate of 948 MB per day for a swap partition of 4 GB. With this write rate, and factoring in wear leveling, a low-end 4 GB flash device with a limit of 100K overwrites can last over 700 years when writes are spread across the device. The maximum swap rate this device could support for five years is over 600 page writes per second.

Cost: Until recently, flash memory was far more expensive than either disk or DRAM. However, flash memory capacity and cost per unit is following Moore's Law much more aggressively than DRAM. SanDisk will be offering flash devices with prices around $2.5 per GB in capacities of 60,120 and 240GB; by mid of 2009 [8]. In contrast, 1GB of DRAM today costs ten times more. Some vendors like Samsung are even planning to hike DRAM prices [5]. Thus, it may be soon cost effective to populate a system with large quantities of flash rather than DRAM to satisfy memory intensive workloads. However, flash is not expected to reach the price of disk [18], so it is best used where its particular characteristics, such as low read latency, are critical.

3 Virtual Memory Management

Virtual memory paging has focused on swapping to disk for decades [4, 15]. As a result, the performance characteristics of disks, such as seek latency, have become ingrained in its design. In particular, we find that three characteristics of disks are assumed: (1) random read access is slow, (2) disk I/O latencies are much higher than other software latencies, and (3) swap devices are integrated as disk storage also being used by file systems. In this section, we describe the virtual memory hierarchy in Linux and show how it is overly tuned to these proper-

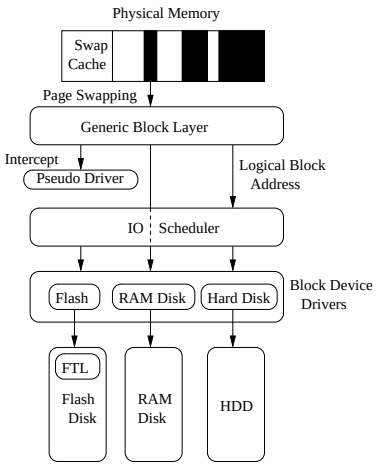

Figure 1: Linux virtual memory hierarchy.

ties of disks. Paging to disks follows a similar I/O path in other operating systems [16, 21].

Slow random reads: Virtual memory systems prefetch adjacent pages to improve read performance [2, 15]. Even though the Linux VM system makes no effort to allocate virtually contiguous pages together on disk, it prefetches 8 consecutive pages on disk by default. Thus prefetch, of effectively random pages, is free as the major cost of paging, seeking and rotational latency, must be paid for the first page. In the context of flash memory, where random access is cheap, prefetching has little benefit for amortizing seek latencies.

Long access latencies: Disks have access times, typically milliseconds, that are much longer than common software latencies. As a result, a paging request can pass through many layers of software without significantly impacting performance.

As shown in Figure 1, a swapped-out page passes through multiple layers in the VM hierarchy. The swap subsystem hands pages to the Generic Block Layer, which is responsible for the conversion of pages into block I/O requests, known as *bio* requests in Linux terminology [2].

These *bio* requests are then queued by the I/O scheduler. The I/O scheduler can reorder, merge or delay a request before passing it to the device driver. A request is delayed to merge it with other contiguous requests that arrive in the near future. This delay can range from 5-6ms [10] to minimize disk seek latencies while reading or writing. Lower in the stack, requests can be delayed or reordered again by the device driver. These additional software layers, which improve disk performance, substantially delay requests. However, flash devices do not suffer the same latencies as disks, so these delays can prove burdensome.

Shared with file system: One reason for these layers is that swap devices are generally shared with file systems. Thus, the OS must provide a common access interface for both file systems and virtual memory. Removing this requirement entails dedicating flash to virtual memory or providing a separate fast-path from the VM system directly to the flash device driver.

4 FlashVM: Revisiting VM Hierarchy

Based on the declining price of flash relative to DRAM and its increasing performance, we propose that future systems be configured with a large amount of dedicated flash to serve as backing store. This can be either attached directly to the system, or a reserved portion of a solid-state disk managed separately. Flash fits virtual memory behavior because (1) writes to free memory pages are asynchronous, while (2) synchronous random reads are fast, allowing frequent page faults.

However, this design stresses the virtual memory hierarchy, as paging may be far more frequent than in systems today. Hence, we revisit the virtual memory hierarchy and describe the design challenges for FlashVM. We also show why simply using flash devices for extending virtual memory in existing commodity operating systems is not going to tap their full potential.

4.1 Software latencies

As shown in Section 3, each swapped out page passes through many layers in the VM hierarchy. This is because the same I/O path below the generic block layer serves both the VM hierarchy and the storage stack. However, the low latency characteristics of flash devices mean that the standard I/O path adds additional software latencies for swapping.

We quantify this additional overhead by measuring the performance difference between hard and soft page faults to a RAM disk. A soft page fault need not copy data or access the I/O path; it adds a page back to the page table. In contrast, a hard page fault requires copying data from swap storage back into the memory and then adding it to the page table. Thus, the difference between the latency for the two faults, less the mandatory cost of copying the page by the RAM disk driver, gives the additional overhead.

We measure these overheads on a 2.5GHz Intel Core 2 Quad with a 512 MB RAM disk acting as a swap device. Soft page-fault latency averages 5μs, and copying a 4,096 byte buffer takes 10μs (when out of cache). In contrast, the average latency for a hard page fault is 196μs with a significant standard deviation of 5,660μs. Thus, the additional overhead associated with each hard page fault averages 181μs. This overhead is largely due to a small number of long-latency page faults that contribute a high percentage of the total execution time.

Write Pattern	ext2:HDD		ext2:Flash		nilfs:Flash	
	A	S	A	S	A	S
Seq. (MB/s)	39.4	25.3	20	1.1	16.8	0.63
Random (IO/s)	1,522	120	66	0.1	2,817	149

Table 2: Impact of log-ordering on write performance of flash (A: asynchronous, S: synchronous).

The bulk of this overhead is spent for CPU scheduling and additional delays introduced by the Linux swap management subsystem to avoid the possible congestion of paging traffic. For a disk, where the minimum read latency exceeds 500μs, these overheads are minor. However, this overhead is *nearly the same as the raw read latency of flash memory*. Thus, software overheads can double the time to swap in a page from a flash device. For FlashVM, the VM system must access flash directly, rather than pass through these generic block interfaces, to avoid this additional overhead.

4.2 Ordering Page Writes

As mentioned in Section 2, random writes perform poorly on flash memory. Log-structured file systems [20], or log-structuring in the FTL, improve the performance of flash disks by writing all blocks sequentially. However, the Linux VM system provides no support for swapping to sequential blocks on disk. Rather, the decision of *where* to swap is made independently of *when* to swap, leading to many random writes. Today, even for high-end SSDs that leverage log-structuring internally, there is a high disparity between the peformance of sequential and random writes (43,000 vs. 3,300 4K-IO/s). As we show below, flash devices in general may perform even worse than hard disks for random writes.

We measure the sequential bandwidth and random IO/s for synchronous and asynchronous writes (VM page writes are asynchronous in Linux). We use a 15.8 GB capacity IBM 2.5 inch SSD (Model 43W7617, SATA 1.0 interface), with a random read access latency of 0.2 ms and sustained read bandwidth of 69 MB/s. Table 2 compares the write performance of flash with ext2 and a simple log-structured file system, nilfs [14]) against a 7200 RPM disk with ext2.

At a high level, the hard disk is better at sequential access and the flash device is better at random access. However, the performance depends strongly on the file system layout and whether writes are synchronous or asynchronous. With a conventional file-system layout (ext2), the flash device performs *23 times worse* than the hard disk at asynchronous random writes. However, with nilfs' log structure the flash device performs *85% better* than the disk. Thus, sequential writes are critical to high performance.

In addition, synchronous writes are generally much more expensive for flash. These writes may incur high erase latencies and simultaneous cleaning overheads within the SSD. In contrast, asynchronous writes amortize these costs over a larger group of pages.

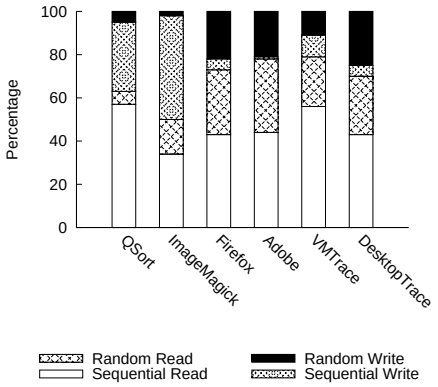

Figure 2: Block access pattern breakdown for different applications.

Thus, the FlashVM system should cluster writes into large sequential groups to optimize for write speed. While high-end flash translation firmware provides this clustering for file systems, they unnecessarily store the virtual-to-physical block address mapping persistently [7].

To explore these effects on real programs, we investigate the VM access patterns on unmodified Linux kernel 2.6.27 with a 4 GB swap partition on the SSD and 512 MB of physical memory. We profile the swap block-access patterns of 4 applications: a recursive Quick Sort of a large array of random integers, ImageMagick for resizing a large JPEG image, Firefox while surfing the web and streaming videos, and Adobe Reader while viewing pdf files of different sizes. We also trace the swap block accesses on a desktop machine for three consecutive days and on a VMWare virtual machine for 3 hours.

Figure 2 shows the access patterns of these workloads. We characterize as *sequential* those requests submitted to a block device that were adjacent to the previous request; otherwise we consider them *random*. All the access patterns are read-dominated, partially because Linux by default prefetches additional 8 pages on a page fault.

These traces illustrate two important points about the nature of swapping traffic. First, random reads are common, accounting for more than 20% of the I/O requests from the large programs and the whole-system traces. Flash disks greatly improve performance for these, and hence have the potential to dramatically improve application performance. For example, ImageMagick, with more than 10% random reads, improved execution time on flash by 65% compared to disk.

More importantly, random writes, which perform poorly on flash, are also common and at times exceed

20% of the accesses in these workloads. As shown earlier in Table 2, these perform 23 times worse than disk. They are mainly responsible for QuickSort peforming 60% slower on flash than on disk.

It follows from these results that simply applying an existing VM system to a flash device, despite better random read performance, may not improve paging performance. Rather, the VM system must be tuned for the particular characteristics of flash, in particular avoidance of random writes.

4.3 Prefetching

As mentioned in Section 3, Linux opportunistically prefetches 8 pages with each swapped-in page. Prefetching for disks is useful to amortize their high seek latency and to overlap I/O with other computation [22]. On the other hand, flash devices provide an order of magnitude lower read access latency and little penalty for random access. Therefore, prefetching functions embedded in the VM system must be updated for flash. We found, for example, that disabling prefetching for ImageMagick improved performance with flash by 27% and for QuickSort by 5%, which directly follows from our discussion on slow random reads in Section 3.

5 Conclusions

FlashVM is a promising alternative to existing virtual memory architectures. With flash memory getting cheaper than DRAM and faster than disks, we foresee systems populated with large quantities of flash rather than DRAM for satisfying memory-intensive workloads. However, the variable and different performance characteristics of flash memory as compared to disks require the re-design of memory management in modern operating systems. In this paper, we list the primary design challenges for FlashVM and demonstrate that it has the potential to provide significant performance improvements for real-world applications.

6 Acknowledgements

This work was supported by NSF Award CNS-0834473. Swift has a financial interest in Microsoft Corp.

References

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for ssd performance. In *USENIX*, 2008.

[2] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel, Third Edition*. O'Reilly Media, Inc., 2005.

[3] Dan's Data. Using flash-swap in Windows Vista. `http://dansdata.com/flashswap.htm`.

[4] E. W. Dijkstra. The structure of the the-multiprogramming system. *Communications of the ACM*, 11(5), May 1968.

[5] EE Times, Jan. 2009. `http://tinyurl.com/c9nevk`.

[6] J. Gray. Tape is dead, disk is tape, flash is disk, ram locality is king, Dec. 2006. `http://tinyurl.com/d2enxp`.

[7] A. Gupta, Y. Kim, and B. Urgaonkar. Dftl: a flash translation layer employing demand-based selective caching of page-level address mappings. In *ASPLOS*, 2009.

[8] InformationWeek. Sandisk Flash Devices. `http://tinyurl.com/dhebrr`.

[9] Intel. Understanding the flash translation layer (ftl) specification, Dec. 1998. Application Note AP-684.

[10] S. Iyer and P. Druschel. Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous IO. In *SOSP*, 2001.

[11] D. Jung, J.-S. Kim, S.-Y. Park, J.-U. Park, and J. Lee. Fass: A flash-aware swap system. In *IWSSPS*, 2005.

[12] H. Kim and S. Ahn. Bplru: A buffer management scheme for improving random writes in flash storage. In *USENIX FAST*, 2008.

[13] S. Ko, S. Jun, Y. Ryu, O. Kwon, and K. Koh. A new linux swap system for flash memory storage devices. In *ICCSA*, 2008.

[14] R. Konishi, Y. Amagai, K. Sato, H. Hifumi, S. Kihara, and S. Moriai. The linux implementation of a log-structured file system. *ACM SIGOPS Operating Systems Review*, 40(3), July 2006.

[15] H. M. Levy and P. H. Lipman. Virtual memory management in the VAX/VMS operating system. *Computer*, 15(3):35–41, 1982.

[16] R. McDougall and J. Mauro. *Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture, Second Edition*. Prentice Hall PTR, 2006.

[17] O. Narasimhan. Optimizing systems to use flash memory as a hard drive replacement. In *Sun BluePrints Online*, 2008.

[18] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating server storage to ssds: Analysis of tradeoffs. In *EuroSys*, 2009.

[19] S. Park, D. Jung, J. Kang, J. Kim, and J. Lee. CFLRU: A replacement algorithm for flash memory. In *CASES*, 2006.

[20] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1), 1992.

[21] M. E. Russinovich and D. A. Solomon. *Microsoft Windows Internals, Fourth Edition*. Microsoft Press, 2005.

[22] E. Shriver, C. Small, and K. A. Smith. Why does file system prefetching work? In *USENIX*, 1999.

[23] SSD-Reviews.com. Solid State Device review. `http://ssd-reviews.com`.

[24] M. Wu and W. Zwaenepoel. envy: A non-volatile, main memory storage system. In *ASPLOS-VI*, 1994.