NAND-Flash: Fast Storage or Slow Memory?

Kshitij Sudan[⋆‡] Anirudh Badam[†] David Nellans[‡]
University of Utah[⋆] *Princeton University*[†] *Fusion-io*[‡]

Abstract

NAND-flash storage is a preferred method to accelerate I/O intensive applications because of its high throughput and fast random-read performance. However, the high cost per GB of flash, relative to magnetic disks, prevents large datacenters from moving their data entirely onto the flash media. Normally, limited amounts of flash is used "in front of" magnetic storage to speed up accesses to the persistent storage subsystem. Recently, there have been proposal that flash should alternatively be used "behind DRAM", as a part of the volatile main memory system. In both configurations flash can be presented transparently to the application, making them competing architectural design points for where flash is to be best used in future systems. In this paper, we evaluate two state-of-the-art systems to compare the relative advantages of each approach. We present results for TPC-C, an on-line transaction processing benchmark, as database systems have been one of the largest adopters of flash technology due to the I/O intensive nature of the workloads. Our results show that when using flash for disk caching, application throughput improves by 56% and by 131% when using flash as part of a hybrid main-memory system.

1 Introduction

NAND-flash memory is gaining prominence in enterprise environments due its high bandwidth, and small access latency. As shown in Table 1 [2–4], these performance characteristics of NAND-flash are closer to DRAM than they are to rotational magnetic media, the storage devices which NAND-flash often replaces. Because of NAND-flash's non-volatile nature, using it to speed up persistent storage is a natural evolution of the Von-Neumann architecture. If the application's dataset size is larger than the system's flash memory capacity (which can occur due to device capacity limitations, or cost), an architect typically chooses to use flash as a fast cache for a magnetic disk based primary storage [1, 7]. There is however another architectural location for NAND-flash within the volatile memory hierarchy as part of a hybrid DRAM and NAND-flash main-memory system.

Using flash as a cache *in front of* magnetic disks, software identifies hot logical blocks on the slow backing store and places them on a fast NAND-flash device. This organization is very similar to an on-chip CPU cache which caches DRAM requests in SRAM to lower the average latency. With a few modifications to the operating system's storage sub-system, all requests destined for the backing store can be trapped to analyze data access patterns and provide intelligent caching of hot blocks. Depending on the caching policy the system implements (write-through vs. write-back), the cache may be a read-only cache, or also help speed up writes to the storage system. Since the caching is transparent to the application, which still sees a single block storage device on which it performs I/O operations, disk caching with NAND-flash requires no modifications to the user application.

Alternatively, using NAND-flash in a hybrid main memory means integrating both DRAM and flash technologies as volatile memory. In the past, systems have been proposed that allow the application to explicitly allocate memory from either DRAM or flash [6]. This however requires programmer knowledge about memory organization and modifications to the application, both of which hinder adoption rates. We propose that NAND-flash should be included in the virtual hierarchy such that it maintains a flat address space. This implies that the applications are exposed a uniform view of the address space irrespective of the memory technology backing that address space. In this organization, flash is used *behind DRAM* to create a large main memory sub-system.

Subsystem	Latency		Bandwidth	
	Read	Write	Read	Write
DRAM memory	51 ns	51 ns	13 GB/s	13 GB/s
PCIe NAND-flash	47 μs	15 μs	3.0 GB/s	2.6 GB/s
Disk storage	9 ms	9 ms	112 MB/s	45 MB/s

Table 1: Relative Performance of Storage and Memory Subsystems

In this work, we explore the benefits and drawbacks of implementing these two approaches for placing NAND-flash in a modern memory hierarchy. We compare two state-of-the-art systems which leverage an identical 160 GB flash device to accelerate a TPC-C like application. Fusion-io's directCache [1] disk caching system is used to place flash in front of a magnetic disk storage system. To build a hybrid main memory, we modified SSDAlloc [5] to be application transparent.

2 Hybrid Storage Systems Using NAND-Flash

When using NAND-flash as a cache for magnetic disk based primary storage, the caching software traps all requests to the primary storage. It then identifies heavily accessed logical blocks and caches them on the flash device. For subsequent requests, cached blocks are served from the faster flash device, both speeding up the requests, and decreasing traffic to the primary storage. The caching policy can be write-through, or write-back. In the former case, all writes are synchronously written to the primary storage, resulting in no speedup on write I/O. In write-back, the flash device buffers both reads and writes, de-staging dirty data to primary storage opportunistically, or when evicting dirty data.

A common model in enterprise systems is to serve all I/O traffic from a high availability Storage Area Network (SAN). This is done so that an individual machine failure does not result in data loss due to orphaned data within the failed machine. Using PCIe attached flash caching in write-back mode breaks this fault domain model as dirty data now is contained on the host side and may not be accessed until the machine comes back on-line. An alternative is to place flash caches within the fault domain of the SAN controller, and replicate data across multiple flash devices to maintain high availability and enable write-back caching.

Placing flash within the SAN however has negative performance implications. It adds hundreds of microseconds, if not milliseconds, of network latency to the native flash access time, which is only tens of microseconds. Additionally, the SAN now sees an increased load due to increased I/O rates across the network. This often requires costly network upgrades. Consequently, most enterprise systems accept the decrease in performance associated with read-only flash caches, rather than using a write-back cache.

3 Hybrid Main Memory Using NAND-Flash

While the majority of NAND-flash is being used behind the storage interface, it is possible to use flash within the virtual memory system as well. This capability has existed for many years as the SWAP subsystem of operating systems and allows virtually mapped memory pages to be stored on disks and loaded on-demand. Traditional SWAP implementations are optimized for disk-like latencies (10+ ms) and simply using flash as SWAP results in poor performance. Recently there have been several systems that allow the applications to use flash effectively within the memory hierarchy [5, 6]. These systems however are not transparent to the application, one of the major advantages of using flash as a disk cache.

In this work we extended SSDAlloc to create a hybrid main memory of DRAM and NAND-flash where tiering between these technologies is handled transparently to the application. This effectively makes DRAM a cache for a larger and slower main memory residing on flash. Using flash as a volatile main memory has several advantages. First, since the application views main memory as volatile, existing assumption about fault domains are unaffected. This means if an application server goes down, no dirty data will reside on the flash "memory" that is required for fail-over to another machine. Second, because of main memory's volatility, NAND-flash can be used to hold both clean and dirty data utilizing both the high read and write bandwidth of flash. Third, our system takes advantage of the fact that in-memory datastructures are built with finer granularity locks when compared to on-disk datastructures. This helps concurrent applications better exploit the parallelism in modern flash memory devices.

Finally, historically applications have been designed to avoid disk I/O that have a high latency. This trained programmers to be lavish with memory operations, but stingy with I/O operations. This however has been at odds with the limited main memory capacity. By expanding main memory capacity, programmers can extend their existing designs to use large amounts of main memory to further avoid the I/O subsystem.

4 Experimental Results

For this study we examined where NAND-flash should be added to a TPC-C like on-line transaction processing system for optimal benefit. Database systems were chosen because they are designed to be able to utilize all available memory (DRAM) in a system, in addition to relying on the storage system to durably store data with ACID guarantees. We used a baseline system consisting of 8 Intel Xeon cores, running at 2.3 GHz, 24 GB of DDR3 DRAM, and a 2-disk RAID-0 of 7200 rpm disks for storage. TPC-C throughput on a 200 GB database was measured after a 1-hour warm-up phase and 4 hours of continuous operation allowing both directCache and the hybrid memory system to warm up and reach steady state operation. A 160 GB Fusion-io PCIe flash card was used for both the directCache cache and our hybrid main memory system. In the all DRAM case, 160GB of additional DRAM was used in place of the Fusion-io NAND-flash card.

Figure 1 shows that an All-DRAM configuration is the highest performing, followed by our hybrid memory implementation, and then by disk caching. Cost is a major concern for an All-DRAM system as current commodity systems do not scale beyond 256 GB of DRAM, and those scaling up to 4 TB cost millions of dollars. Conversely, a desktop machine

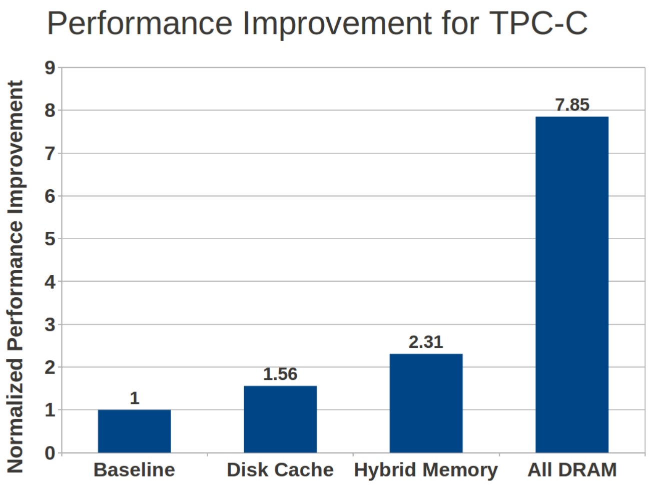

Figure 1: Throughput when using flash in different configurations

can easily be configured with 1 TB of flash, which costs just $10 per GB. Endurance of flash is a concern and we note that the state of the art devices are rated for nearly 17 PB of write endurance [2]. A sustained write rate of 2.5 GB/s (a worst case assumption) implies that the device will last for over 2 years as a main memory device. When used in a hybrid memory where hot-write pages typically reside in DRAM, the device endurance far exceeds the worst case expected lifetime.

5 Conclusions

NAND-flash has generally been considered to be too slow for use as a main memory technology. Our results indicate that using flash, along with DRAM, to build a hybrid memory system may be a compelling design point for future memory systems. We observed that using flash in a hybrid memory configuration for database applications has a 48% performance advantage over a disk cache configuration. This improvement comes primarily from the ability to use the flash write bandwidth, as well as the improved information about data usage (determining hot data that is then placed in the DRAM). This improvement can possibly be increased further by optimizing the flash devices for main memory access pattern. When using flash for main memory, the request access pattern exhibits a small request size (64 Bytes), whereas current flash devices are optimized for read bandwidth and large I/O block sizes. This is an encouraging result and implies hybrid memory systems need more investigation as a cost-effective way to build large memory systems of the future.

References

[1] Fusion-io directCache. `www.fusionio.com/images/data-sheets/directcache-sheet.pdf`.

[2] Fusion-io ioDrive2. `www.fusionio.com/data-sheets/iodrive2`.

[3] John McCalpin's Blog. `blogs.utexas.edu/jdm4372`.

[4] Seagate ES.2 Datasheet. `www.seagate.com/docs/pdf/datasheet/disc/ds_barracuda_es_2.pdf`.

[5] A. Badam and V. S. Pai. SSDAlloc: Hybrid SSD/RAM Memory Management Made Easy. *NSDI*, 2011.

[6] J. Coburn, A. M. Caulfield, A. Akel, R. K. Gupta, R. Jhala, and S. Swanson. NV-Heaps: Making Persistent Objects Fast and Safe with Next-Generation, Non-Volatile Memories. *ASPLOS*, 2011.

[7] J. Ren and Q. Yang. I-CASH: Intelligently Coupled Array of SSD and HDD. *HPCA*, 2011.