Virtualization and Adaptive Provisioning of Non-volatile Memory

Bhavesh Singh
143059003

Master's Thesis Project Stage I Report
submitted in partial fulfillment of the
requirements for the degree of
Master of Technology

under the guidance of
Prof. Purushottam Kulkarni

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
India – 400076

October 2016

Contents

List of Figures

List of Tables

1 Introduction

Virtualization is a technique used widely in datacenters for server consolidation and efficient resource utilization. To increase the utilization of resources like memory, CPU and storage, multiple servers are run as virtual machines on the same physical server. To ensure high availability and consistency of data in the datacenter, storage is consolidated into a centralized server, accessible over the network (called storage area network or SAN). In addition to high availability, storage area networks are easily scalable and allow for easy maintenance.

Non-volatile memory is a fast, persistent memory that has access latencies and cost per gigabyte between DRAM and magnetic hard disks. Figure 1.1 shows the position of non-volatile memory in the typical memory hierarchy. Because of its faster speed, it is being widely used as a host-side cache (figure 1.2) for the centralized storage to reduce access latencies [1, 4, 5]. Various caching policies are employed in these host-side caches like write-through [1, 4], ordered write-back [5], persistent write-back [12] to provide varying consistency guarantees and performance levels.

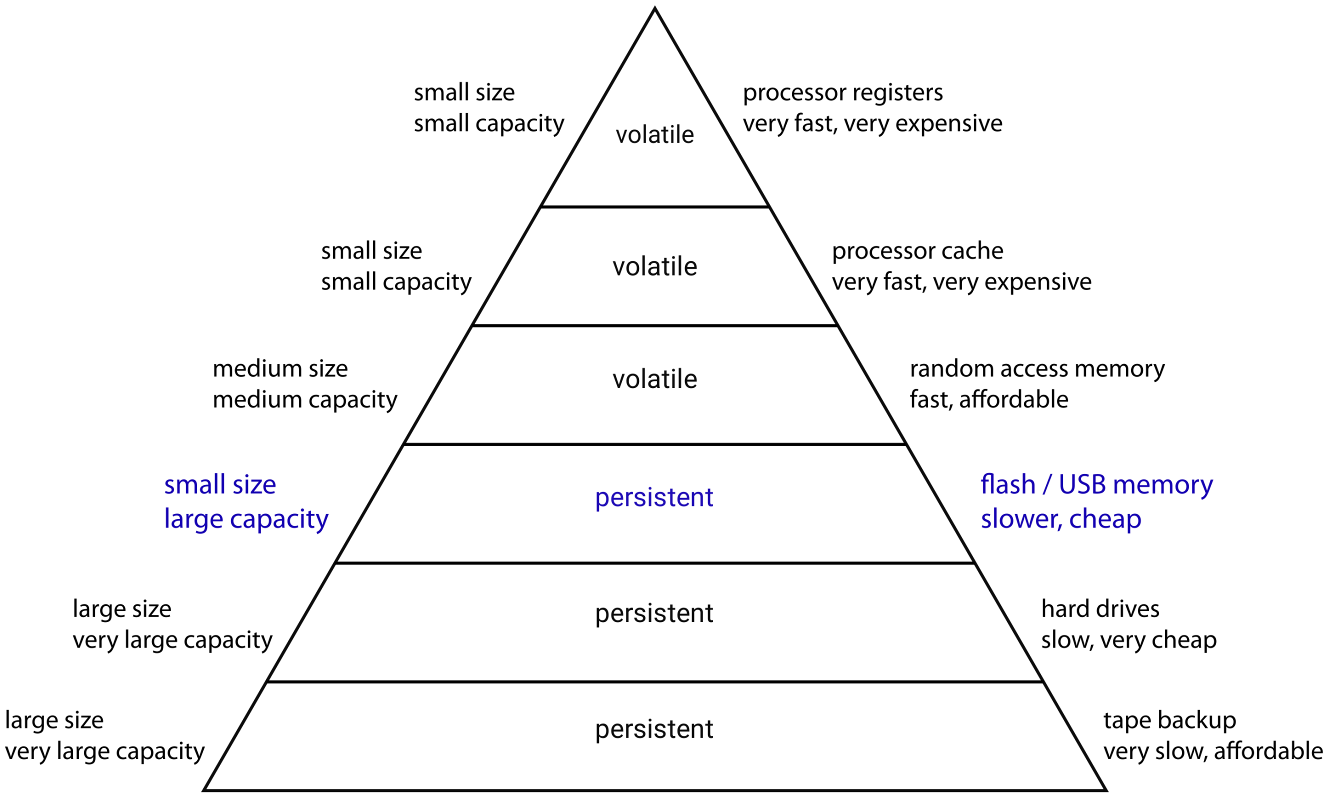

Figure 1.1: Memory hierarchy [15]

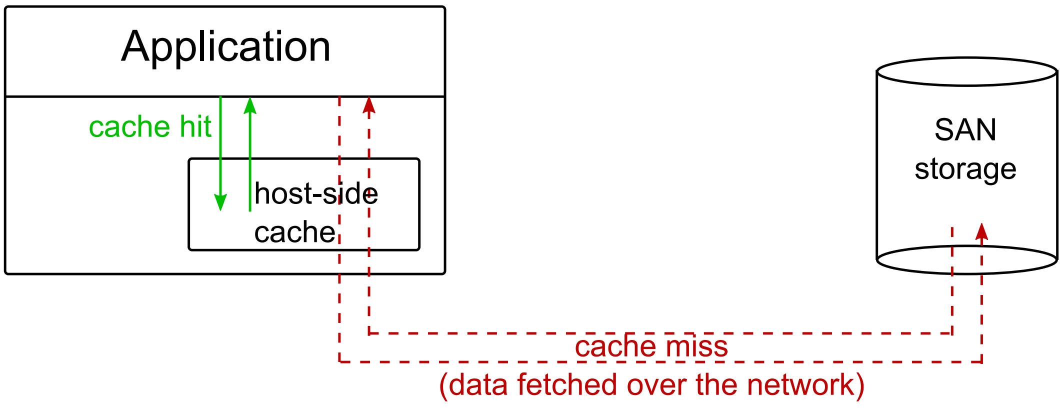

Figure 1.2: Host-side flash cache

1.1 Motivation

With enterprises consolidating more and more servers as virtual machines, two questions can be posed for non-volatile memory as a resource in the memory hierarchy.

1. **Is there a case for virtualizing non-volatile memory to VMs?**
2. **Are there ways to over-commit/multiplex this resource across VMs?**

1.1.1 Case for Virtualizing Non-volatile Memory

Current research advocates for using non-volatile memory, specifically flash memories, as hypervisor managed block/extended page caches. In both cases, the flash cache is dynamically partitioned by the hypervisor. The block cache has an inclusive cache setting i.e. the data is first brought into the block cache on a cache miss and then given to the file system page cache [6, 9]. In the extended page cache setting, the flash device is used as an exclusive second chance cache i.e. the pages are put in the cache only when they are being evicted from the page cache [13].

In both these cases, the guest is oblivious to the presence of non-volatile memory (figure 1.3). Thus, the guest has no control over the eviction policy. There is a case of guest-aware eviction policy in this case.

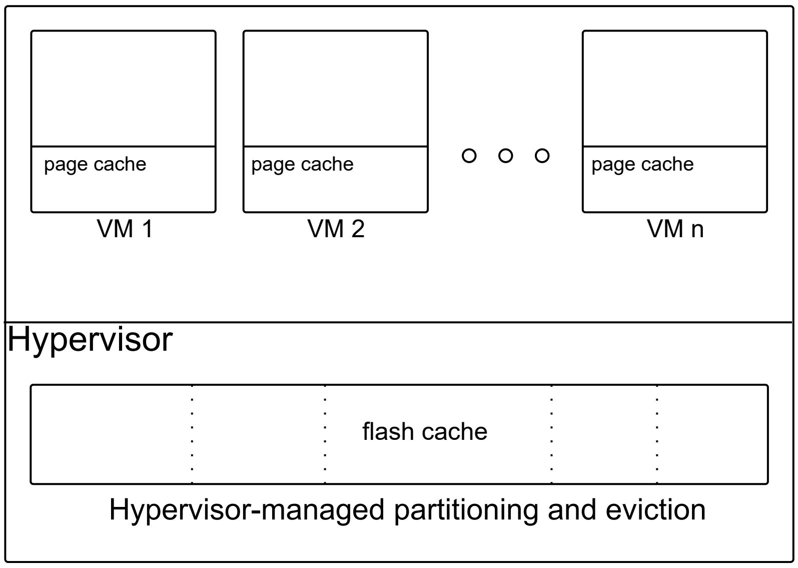

Figure 1.3: Hypervisor-managed host-side flash cache

1.1.2 Over-commitment/Multiplexing of Non-volatile Memory

Over-commitment or thin provisioning of non-volatile memory, especially flash memory, is an area of active research. But the motivation behind doing this is to increase the lifetime of the flash memory by avoiding duplicate writes or use a copy on write semantics [16]. Liang et al. [8] propose virtualization of non-volatile memory as both byte-addressable and block-addressable memory interface in the guests. They talk about efficiently virtualizing this memory by mapping hot guest data to DRAMs. Although they do not talk about over-commitment mechanisms explicitly, it can be easily handled due to the byte-addressable interface and hence use existing memory management techniques like ballooning.

1.2 Problem Statement

For the two questions posed above, this work proposes the following.

1. Exploring use-cases for virtualizing non-volatile memory to the guest.
2. Exploring use-cases for over-committing and dynamically resizing the partitions of non-volatile memory.

Host-side caching is a good candidate for both of the above cases. The hypervisor is responsible for dynamic partitioning of the flash memory while the guests are provided with a virtual flash device that can get resized dynamically. Each guest controls its eviction policy and can evict blocks on increase in memory pressure.

Probably to use a dynamically resizable disk, the guest filesystems need to be modified to keep data tiered (divided) in both the kinds of memory. Although this approach is akin to caching, it allows in-place updates to data residing in the faster memory.

2 Background

Non-volatile memory comprises a host of different types of memory which are all characterized by their speed (faster than magnetic hard disks and slower than DRAM) and persistence. Table 2.1 shows a comparison of various properties of different types of non-volatile memory.

	Access Granularity	**Read Latency**	**Write Latency**	**Erase Latency**	**Endurance**
HDD	512 B	5 ms	5 ms	N/A	$> 10^{15}$
SLC Flash	4 KB	25 ms	500 ms	2 ms	10^4 – 10^5
PCM	64 B	50 ns	500 ns	N/A	10^8 – 10^9
STT-RAM	64 B	10 ns	50 ns	N/A	$> 10^{15}$
ReRAM	64 B	10 ns	50 ns	N/A	10^{11}
DRAM	64 B	50 ns	50 ns	N/A	$> 10^{15}$

Table 2.1: Comparison of various properties of different memory types [10]

Among the three different types of non-volatile memories listed above, flash memories are most widely used. And hence a large part of research on non-volatile memory is actually on flash memory and its uses. Flash memories are available in two forms — block-addressable NAND flash-based memory (SSDs, USB sticks) and byte-addressable NOR flash-based memory. The former is used in an I/O interface while the latter can be attached to memory bus. The names NAND and NOR are just indicative of the way individual storage units on the chip (called cells) behave.

Flash memory provides read, write and erase operations. The read and write operations can be performed at a page/byte granularity (depending on NAND/NOR flash) but the erase operation can only be performed on an *erase block* of much larger size (typically 512 KB and above). Rewrites are not possible without erases. Thus, changes to a page are written to a new location and a mapping is maintained for logical to physical page numbers. This mapping is called the flash translation layer.

The cells have a limited number of program/erase cycles (P/E cycles) before they wear out and become unusable. The flash translation layer is used to provide wear-levelling. *Wear-levelling* is used to spread out the hot data across the device to prevent wearing out of the cells. A garbage collection routine erases the erase blocks and frees storage space. Any pages in an erase block which are still being referenced are copied elsewhere before erasing the block. This leads to the problem of write amplification. *Write amplification* occurs when a write triggers garbage collection due to lack of free space, which, in turn, might have to copy existing pages to new locations before erasing an erase block.

2.1 Related Work

We can classify the earlier work around non-volatile memories into two broad categories. These also include non-virtualized scenarios.

1. **Transparent to the application/guest**
2. **Visible to the application/guest**

2.1.1 Transparent to the application/guest

The application is unaware of the underlying non-volatile memory and just uses the facilities provided by the operating system/hypervisor.

1. **Block cache** [1, 4, 5] — flash memory is typically used as a block cache on the host. Although other types of non-volatile memory, notably NVDIMM also provide a block interface which can be used. In the virtualized setting, the hypervisor manages the cache and the guests are unaware of it.

2. **Second chance page cache** [13] — Venkatesan et al. suggest using non-volatile byte-addressable memory as an extension of the tmem interface. This is still managed by the hypervisor and not virtualized. NVM is used for frontswap while the DRAM is used for cleancache. This helps reduce the rate of swapping and thus improve performance. Due to the exclusive nature of the cache here, the eviction is still in the hands of the guest.

3. **Hybrid main memory systems** [3] — the hypervisors make dynamic decisions to map guest pages to either slower persistent memory or the faster DRAM and swap among the memories based on recency and frequency characteristics and reduce writes to non-volatile memory.

4. **File system for non-volatile memory** [7] — specialized file systems based on log-structured file system to cater to the asymmetry of reads and writes some non-volatile memories, notably flash, but also to facilitate faster I/O directly from the non-volatile memory.

2.1.2 Visible to the application/guest

The application is optimized to take advantage of the non-volatile memory.

1. **Libraries facilitating applications to place data in persistent memory** [2] — libraries (e.g. nvmalloc) allow applications to place data in non-volatile memory. The application-developers can utilize this to place larger data structures (whose size grows with the application data) in non-volatile memory so as to keep the DRAM relatively free for hotter data.

2. **Open-channel SSDs, which expose the parallelism in SSDs to the applications** [11, 14] — applications are designed to leverage the parallelism provided by open-channel SSDs. This is usually akin to a SAN storage exposing multiple logical block devices. These SSDs do not implement the flash translation layer inside their controllers and leave it to the operating system to do it.

2.1.3 Virtualizing Non-volatile Memory

Liang et al. [8] talk about virtualizing byte addressable non-volatile memory with the minimum overhead. Their design assumes persistent memory as the only type of memory available to the VM (with access characteristics similar to DRAMs). Their solution, called VPM, tries to map hot guest PM (persistent memory) pages to DRAM to improve performance. They guarantee the persistence property of guest pages, allowing crash recovery.

3 Design

Our design consists of exposing a dynamically resizable, block-addressable flash device to the guest. It is assumed that the guest will use it as block cache and can use any caching policy it wants to implement on it. The hypervisor is responsible for making decisions about dynamic partitioning of the physical SSD (see figure 3.1).

The virtual SSD will provide a normal block device interface. We are implementing a paravirtualized Virtio driver whose backend manages the partitioning, resizing and mapping of guest blocks to physical blocks.

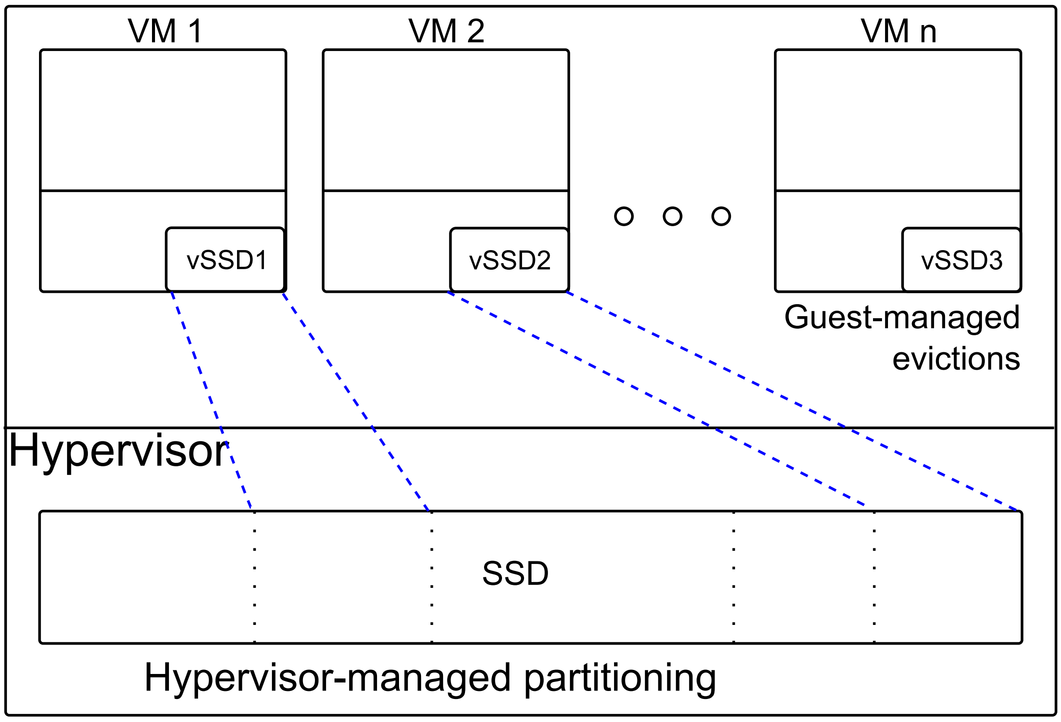

Figure 3.1: Our Proposed Soltion

3.1 Challenges

1. An efficient on-disk data structure to map physical blocks to virtual blocks across virtual SSDs.

2. The guest caching layer should be able to request the driver to increase the size of the disk based on the miss rate. But the kernel would not be able to detect this dynamic change in size.

3. The caching layer maps the disk blocks to guest blocks assuming some fixed size of the SSD. So the change in size should not affect the mapping logic.

4. The VirtIO backend driver should be able to communicate to the guest caching layer via the frontend driver that a resize event has occured and it needs to free `n` blocks from the virtual SSD.

4 Conclusion

In the first stage, we defined and motivated the need for a virtualized, dynamically sized, fast persistent store in the guest. We are implementing it for a very specific use-case of block caching in the guest. We intend to extend this work in the second stage to include the following.

4.1 Future Work

1. Evaluate the proposed design for efficiency (access latency for different workloads) vis-a-vis current hypervisor managed caching mechanisms using non-volatile memory.
2. Extend the semantics to a generic block layer and what could be possible use-cases for it in a virtualized scenario.

4.2 Acknowledgements

The idea for exploring this topic was suggested by Debadutta Mishra. He is always willing to help and his deep insight and valuable inputs and comments throughout the first stage went a long way to help me motivate the problem and define the problem statement clearly. The long discussions with my guide, Prof. Purushottam Kulkarni, were helpful in getting clarity and direction. I am grateful to both of them.

Bibliography

[1] Byan, S., Lentini, J., Madan, A., and Pabón, L. Mercury: Host-side flash caching for the data center. In *IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), 2012* (apr 2012), pp. 1–12.

[2] Dulloor, S. R., Roy, A., Zhao, Z., Sundaram, N., Satish, N., Sankaran, R., Jackson, J., and Schwan, K. Data Tiering in Heterogeneous Memory Systems. In *Proceedings of the Eleventh European Conference on Computer Systems* (New York, NY, USA, 2016), EuroSys '16, ACM, pp. 15:1—-15:16.

[3] Hirofuchi, T., and Takano, R. RAMinate: Hypervisor-based Virtualization for Hybrid Main Memory Systems. In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (New York, NY, USA, 2016), SoCC '16, ACM, pp. 112–125.

[4] Holland, D. A., Angelino, E., Wald, G., and Seltzer, M. I. Flash Caching on the Storage Client. In *Proceedings of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)* (San Jose, CA, 2013), USENIX, pp. 127–138.

[5] Koller, R., Marmol, L., Rangaswami, R., Sundararaman, S., Talagala, N., and Zhao, M. Write Policies for Host-side Flash Caches. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST 13)* (San Jose, CA, 2013), USENIX, pp. 45–58.

[6] Koller, R., Mashtizadeh, A. J., and Rangaswami, R. Centaur: Host-Side SSD Caching for Storage Performance Control. In *IEEE International Conference on Autonomic Computing (ICAC), 2015* (jul 2015), pp. 51–60.

[7] Lee, C., Sim, D., Hwang, J., and Cho, S. F2FS: A New File System for Flash Storage. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST 15)* (Santa Clara, CA, 2015), USENIX Association, pp. 273–286.

[8] Liang, L., Chen, R., Chen, H., Xia, Y., Park, K., Zang, B., and Guan, H. A Case for Virtualizing Persistent Memory. In *Proceedings of the Seventh ACM Symposium on Cloud Computing* (New York, NY, USA, 2016), SoCC '16, ACM, pp. 126–140.

[9] Meng, F., Zhou, L., Ma, X., Uttamchandani, S., and Liu, D. vCacheShare: Automated Server Flash Cache Space Management in a Virtualization Environment. In *Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC 14)* (Philadelphia, PA, 2014), USENIX Association, pp. 133–144.

[10] Mittal, S., and Vetter, J. S. A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems. *IEEE Transactions on Parallel and Distributed Systems 27*, 5 (may 2016), 1537–1550.

[11] Ouyang, J., Lin, S., Jiang, S., Hou, Z., Wang, Y., and Wang, Y. SDF: Software-defined Flash for Web-scale Internet Storage Systems. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2014), ASPLOS '14, ACM, pp. 471–484.

[12] Qin, D., Brown, A. D., and Goel, A. Reliable Writeback for Client-side Flash Caches. In *Proceedings of the 2014 USENIX Annual Technical Conference (USENIX ATC 14)* (Philadelphia, PA, 2014), USENIX Association, pp. 451–462.

[13] Venkatesan, V., Qingsong, W., and Tay, Y. C. Ex-Tmem: Extending Transcendent Memory with Non-volatile Memory for Virtual Machines. In *IEEE International Conferance on High Performance Computing and Communications, 2014* (aug 2014), pp. 966–973.

[14] Wang, P., Sun, G., Jiang, S., Ouyang, J., Lin, S., Zhang, C., and Cong, J. An Efficient Design and Implementation of LSM-tree Based Key-value Store on Open-channel SSD. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, ACM, pp. 16:1—-16:14.

[15] Memory hierarchy - wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Memory_hierarchy. Last accessed April 14, 2016.

[16] Weiss, Z., Subramanian, S., Sundararaman, S., Talagala, N., Arpaci-Dusseau, A., and Arpaci-Dusseau, R. ANViL: Advanced Virtualization for Modern Non-Volatile Memory Devices. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST 15)* (Santa Clara, CA, 2015), USENIX Association, pp. 111–118.