Improving I/O Performance using Virtual Disk Introspection

Vasily Tarasov,[1] Deepak Jain,[1] Dean Hildebrand,[2] Renu Tewari,[2] Geoff Kuenning,[3] Erez Zadok[1]
[1]*Stony Brook University,* [2]*IBM Research—Almaden,* [3]*Harvey Mudd College*

Abstract

Storage consolidation due to server virtualization puts stringent new requirements on Storage Array (SA) performance. Virtualized workloads require new performance optimizations that cannot be totally addressed by merely using expensive hardware such as SSDs. This position paper presents *Virtual Machine Disk Image (VMDI) introspection*—a key technique for implementing a variety of virtualization-oriented I/O optimizations. VMDI introspection gives SAs an understanding of guest file system semantics, such as determining object types and classifying read and write operations. We explore possible approaches for VMDI introspection and then describe a set of VMDI-introspection-based optimizations. Our prototype implementation with enhanced meta-data caching and placement shows 11% to 20× performance improvement.

1 Introduction

Analysis of today's IT trends suggests that by 2014 almost 70% of all x86 applications will be virtualized [2]. It has been shown that the workloads experienced by Storage Arrays (SAs) in virtualized setups are significantly different from those observed in physical setups [6, 16]. Most modern storage solutions are optimized for non-virtual workloads, but many optimizations such as traditional meta-data caching and read-ahead are not efficient in virtualized deployments.

Furthermore, widespread deployment of virtualization in modern data-centers, combined with the trend towards storage consolidation, demands increased performance from SAs. Solutions based on HDDs are limited by high latency and power consumption, while flash-based arrays are too expensive for large datasets. Tiered solutions involving multiple types of storage (e.g., persistent RAM, SSD, and HDD) reduce cost, but it is hard to consistently place data in the correct tier. Thus, exploring virtualization-aware optimizations is a reasonable alternative research direction.

Placing Virtual Machine Disk Images (VMDIs) on local, clustered, or distributed file systems is a common way to store per-Virtual-Machine data. VMDIs provide a convenient mechanism for VM isolation, data management, simple and efficient versioning, recovery, and mobility. Unfortunately, this increased manageability comes at the expense of performance. For example, SAs often apply different policies to data and meta-data (e.g., cache space and eviction, commit intervals). Such optimizations would benefit VMDIs, which have a complex internal structure that fits well into the meta-data-vs.-data classification. But from the SA's point of view, all VMDI blocks are the same—data blocks—so these optimizations cannot be applied.

We propose that SAs *introspect* VMDI files to understand their complex internal structure. This understanding can be leveraged to perform optimizations on all incoming I/O operations. For example, incoming writes to a directory or to inode blocks can be interpreted as file creations or updates. Meta-data-intensive workloads such as these can then be improved by caching appropriate portions of VMDI files in faster storage or having meta-data cached separately to avoid cache pollution.

Block-level introspection is a known technique [1,5,8, 11–15] but we are the first to apply it for optimizing NAS performance for virtual workloads. We begin this paper by discussing introspection techniques and the layers in the virtualized I/O stack where introspection can be implemented. We then analyze possible introspection-based optimizations, and finish by presenting a prototype implementation that demonstrates several possible improvements. Our general conclusion is that basic introspection optimizations can be added to an SA with relatively little effort, while providing performance improvements of up to 20× for some operations.

2 Introspection Techniques

We first discuss various approaches to storing VMDIs, then discuss techniques for introspection and their potential difficulties.

Storage Setups. Figure 1 depicts two typical storage configurations. In the first, an external Storage Array (SA) is connected to a hypervisor machine as a SAN or NAS. In the second (NoSAN) setup, VMDI files are stored locally to the hypervisor. In this paper we focus on the SA-based setup, although the NoSAN setup can also benefit from VMDI introspection. In either case there is a back-end file system such as VMFS, WAFL, GPFS, or ZFS that manages all VMDIs. Inside the VM, the guest OS's file system executes both meta-data and data operations, but with few exceptions the back-end file system views all operations as data accesses.

Introspection can be implemented either within the back-end file system itself, or at a layer above it (e.g., NFS/SMB/iSCSI server). The latter approach is less intrusive for complex back-end file systems and can independently support different file systems. However, this method precludes knowledge of back-end file system internals (e.g., on-disk layouts) and therefore may be restricted in the type of optimizations it can implement.

Guest-assisted approach. One way to perform introspection is to run a guest OS daemon that has access to the guest's file system structures and related operations. The daemon can pass relevant data to the SA on the network. Guest-assisted introspection allows errorless meta-data classification and incurs low overhead. However, in many cloud environments it is impossible to force all VMs to run special-purpose software, especially since VMs run different versions of different OSes, and accounting for the daemon's CPU cycles and I/O operations can be difficult. Thus, we do not believe that this approach is viable in most environments.

Black-box approach. Another way to perform introspection is to have the SA parse the internal structure of VMDI files without any assistance from the guest OS.

VMDI files come in a variety of formats: VirtualBox (`.vdi`), VMware (`.vmdk`), and Microsoft (`.vhd`), to name a few. Most of these formats have several flavors and multiple versions. Fortunately, their specifications are publicly available and there are libraries that can handle them all via a unified interface [10, 17]. VMDI files commonly contain several partitions and logical volumes (e.g., Linux LVM). To be practical, introspection must be able to parse these structures. There are currently 29 on-disk file systems supported by the Linux mainline, and all can potentially reside within a VMDI volume. Fortunately, only a few are in widespread use, and many use similar on-disk layouts (e.g., FFS-like). Thus, storage vendors can make an impact for the majority of their users by supporting introspection-based optimizations on only the most widespread file systems.

To implement introspection for simpler file systems such as ext2/3, we have found it easiest to develop introspection modules from scratch. For more complex designs, code from programs like `fsck`, `mkfs`, and associated libraries can often be reused.

To automate the process of introspection for any VMDI file, it is helpful that many modern SAs run Linux internally, which means that they have built-in support for almost all the file systems that can be in a VMDI file (even if the VM runs Windows). One challenge is that kernel file systems map meta-data and data blocks to device blocks, while introspection needs the reverse mapping to determine which blocks map to which file system structures. Fortunately, the recently added `FIOMAP` and `FIBMAP` `ioctl`s provide that functionality and can be extended to extract even more information useful for introspection. More generally, we speculate that in the future the VFS layer could be extended to support methods to ease introspection.

Security and reliability optimizations usually require absolutely correct answers from the VMDI introspection engine. However, for many *performance* optimizations, it is acceptable if introspection makes occasional mistakes. For example, errors in correctly predicting meta-data decrease the efficiency of meta-data placement and caching but do not break a system or corrupt the data. Thus, techniques that probabilistically detect file system layouts without prior knowledge of a file system's internal organization can be applied. For example, an analysis of the differences between blocks after every write might distinguish meta-data from data with acceptable accuracy. Alternatively, a one-time probe process could execute a controlled set of operations in the VM, so that block operations could be correlated with their file-system equivalents [15]. `Mkfs` is a natural example of such a probe, because it typically writes only to meta-data regions. These two methods cannot provide 100% accuracy (e.g., `mkfs` cannot detect indirect blocks), but for many introspection-based optimizations, such as intelligent caching, perfection is not required.

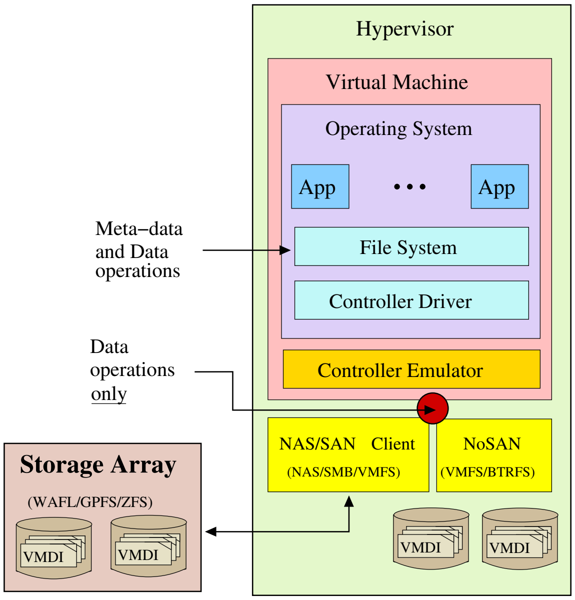

Figure 1: Two common server virtualization configurations: Storage Arrays (SAs) and NoSAN setups.

Difficulties. Write operations to a VMDI file may not be able to be classified until future writes are received. For example, for reliability reasons many file systems update data blocks before writing the associated meta-data. Some file systems (e.g., FFS-like) use fixed meta-data layouts and only the indirect blocks are allocated dynamically. But other file systems allocate most of their meta-data dynamically (e.g., Reiserfs, Btrfs). For such B-tree-based file systems, writes cannot be classified until the B-tree root is written. Thus, an introspection facility should support deferred write classification.

The introspection module needs to maintain a mapping between block offsets and file objects. Page-tablelike mappings can be used for block-based file systems, but for extent-based designs, more space-efficient data structures should be used.

In addition to technical challenges, there may also be legal and privacy concerns. It is possible that some users may not appreciate cloud providers scanning their VMDIs and analyzing incoming requests. Introspection therefore should be an optional, user-controlled feature.

3 Optimizations

Production data centers typically have many hypervisors (and many more VMs) accessing a single SA, leading to an imbalance between the sizes of the VM and SA caches. Therefore, it is critical for the SA to allocate cache space intelligently. A key benefit of VMDI introspection, leading to many possible optimizations, is the ability to distinguish between data and meta-data. For example, a straightforward approach is to pre-load a VMDI's meta-data into the SA cache for fast access. A slightly more involved method is to prioritize meta-data in the cache, avoiding interference between data and meta-data workloads within a single VM or between multiple VMs by applying intelligent eviction policies.

In hybrid and multitiered storage, efficient data relocation and migration is critical. For example, in non-virtual workloads, it is possible to place small and hot files on fast storage while large streams are redirected to large capacity storage. Technologies such as VMware vFlash are available for virtualized workloads, but it can be hard to know what to cache. Using introspection we can place data blocks on the correct tier by understanding their semantics. For example, in meta-data-intensive workloads, writes to meta-data and the file system journal can be redirected to SSD drives.

Guest file systems contain different meta-data types, such as journals, directory entries, inode tables, etc. Fine-grained meta-data classification allows sophisticated per-type optimizations. For example, the SA can maintain efficient file name indexes to speed lookups. Per-directory meta-data and inodes can be dynamically prefetched when a directory is accessed. Predicting cross-file access is also possible [9].

Guest file systems fragment as they age, but even in an unfragmented file system the SA's prefetching algorithms may not be optimal without an understanding of the complex structure of the VMDI. Sequential access in the guest may appear as random to the SA, and standard SA read-ahead algorithms may disable themselves or cache data that may never be accessed. Introspection can correct this problem by providing information about the location of virtually consecutive blocks in the VMDI.

Like meta-data, data comes in many flavors. Fine-grained data classification allows intelligent prefetching and eviction policies for data blocks. For example, during VM boot storms, the corresponding kernel image, initial RAM disk image, and init scripts can be prefetched to reduce boot times. These objects can be identified by file names (e.g., `/boot/vmlinuz-*` and `/boot/initrd-*`). Specific requests can indicate an upcoming VM start, e.g., VMware writes to a VMDI's lock file before starting a VM. After a VM is up, blocks required only for the boot can be evicted to save memory and eviction time later. If the SA implements deduplication and compression, it could decide how to compress certain data blocks based on their file type.

Introspection can also increase space efficiency. By recognizing which VMDI blocks are unused (such as those that have been deallocated by the guest), the SA could free those blocks in the back-end file system.

Introspection-based optimizations scale well to multiple identical or similar VMs, which often occur in real deployments. If a VM is a clone, the corresponding VMDI's internal structure is repeated and efficient sharing of introspection information is possible.

4 Implementation

Our prototype is implemented as an independent, flexible C library. For this prototype, we linked our library with the NFS-GANESHA user-space NFS server (git-commit-`041e4e`) [4]. We modified only 30 LoC in NFS-GANESHA's code to support our library. Our introspection library itself contains about 1,000 LoC. Our prototype introspects only ext2 and ext3.

The library provides three basic capabilities. (1) It can scan a VMDI file and create an internal representation of the file system's meta-data. This is usually used prior to the boot of a VM; if an inconsistency is detected between the VMDI and its in-memory state, the VMDI can be rescanned. (2) The library provides functions that allow the meta-data representation of a VMDI to be dynamically updated. These functions are invoked on the write path. (3) The library contains an API for discovering what objects exist in a region of a VMDI and for finding related file system information. This is used in both read and write path optimizations.

5 Evaluation

We used our new library to implement two introspection-based optimizations: (1) meta-data caching (2) redirecting meta-data writes to an SSD.

Experimental setup. We performed our experiments on two Dell PowerEdge R710 nodes, each with an Intel Xeon E5530 2.4GHz 4-core CPU and 24GB of RAM. We installed VMware ESXi 5.1.0 build 799733 on the hypervisor node. The NAS-based SA ran CentOS 6.3 and Linux kernel 3.7.3. A 250GB Fujitsu MHZ2250B hard drive and an 80GB Intel SSDSA2MH08 solid-state drive, both connected through a PERC 6/i Integrated controller, were used as storage. We formatted both drives with ext3 and used the SSD only to redirect writes. We used the NFS-GANESHA NFS server with the VFS File System Abstraction Layer.

Our experimental VMs were set up with a single CPU, 2GB of RAM, and an emulated LSI Logic Parallel controller. The guest OS was CentOS 6.3. Each VM had two VMDIs: one with a root file system created during guest OS installation (*OS VMDI*) and another used to conduct experiments with controlled directory trees (*data VMDI*). Both were 16GB VMDK flat files stored on an NFSv3 datastore hosted by the SA. The machines were connected using a 1Gb Ethernet.

The OS VMDI contained a 40MB boot and a 1.7GB root partitions. The data VMDI contained a file system generated by Filebench using the "file-server" personality with 90,000 files [3] (about 10GB of data in total).

Limitations. The first limitation of our evaluation is that we used simple workloads to stress our system, so evaluation of more complex workloads is useful. Second, we flushed the Linux page cache on the SA prior to every experiment. Consequently, with longer experiments the LRU policy used by the page cache may provide better results than what we observed in our experiments. Third, all experiments used a single VM. We believe that introspection can improve performance even more for multi-VM workloads, but the corresponding overheads need to be evaluated. We plan to address these limitations in the future.

Initial scanning. Before a SA can perform introspection, it must scan the VMDIs to collect mapping information. The initial scan of the OS VMDI took 60 seconds; for the data VMDI it took 260 seconds. When the VMDIs were stored in RAM, the time fell below 1 second for both images; clearly, I/O is the bottleneck for the scanning phase. The scans experienced an unusually low disk throughput of 4–6MB/s because our scanner performs single-threaded, non-sequential 4KB reads from the disk. Optimizing the scanning process through parallelization and by using better-suited I/O sizes is future work. Notice also that scanning can run as a background task and is not required for cloned VMs.

Our introspection library uses 24 bytes per 4KB block; so the maximum RAM usage is about 0.6% size of a VMDI file. In the future we plan to use extents instead of a block map to reduce memory consumption. In our experiments we cached all meta-data, so the mapping covered the whole VMDI. More sophisticated solutions can prioritize which regions to introspect based on access frequency. To reduce memory requirements when multiple VMs are introspected, mappings can be partially shared.

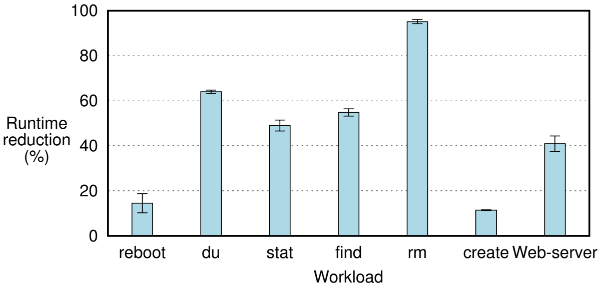

Figure 2: The impact of introspection-based optimizations on the runtime of several workloads. The Y axis represents the runtime reduction after enabling the optimizations.

Runtime. We first measured the runtime of several meta-data intensive workloads on a non-optimized SA. Then we measured the same workloads on the SA with both introspection optimizations enabled: in-RAM VMDI meta-data caching and meta-data write redirection to SSD. We did not migrate meta-data to SSD in advance; it was performed automatically as the writes were coming in. We experimented with the following workloads: (1) VM reboot; (2) disk usage as reported by `du`; (3) `stat` of every file in the file system; (4) `find` of a nonexistent file; (5) `rm` of all files; (6) multiple file creations; (7) Filebench Web server. For all experiments except VM reboot we introspected the data VMDI; for the reboot we introspected the OS VMDI.

Figure 2 depicts that the relative reduction in runtime for different workloads ranged from 11% to 95%. The reboot time of a single VM was reduced by 14%, which could be even better if several VMs are rebooted concurrently. Concurrent accesses to a disk drive can create seek storms, which greatly increase the service time. Concurrent accesses to cache, however, do not slow each other beyond queuing costs.

For `du`, `stat`, and `find`, the improvements were close (about 50%) because these workloads are similar: iterate over inodes. The `rm` workload was improved by over 20 times because it reads indirect blocks. Ext3 allocates indirect blocks dynamically, so they are spread randomly across the VMDI. By eliminating random accesses to the disk, meta-data caching significantly increases SA performance. In addition, `rm` benefits from both read and write optimizations. The file creation workload demonstrated an 11% improvement—modest but appreciable in large-scale deployments. The limited improvement is because writing to SSD was not significantly faster than to disk. Finally, the read-intensive Web-server workload improved by 40%, thanks to meta-data caching for many small files.

6 Related Work

Introspection was earlier applied by block-level storage arrays to implement effective exclusive caches [1], identify disk block liveness [13], perform graceful RAID degradation and quick recovery [14]. Semantically-smart Disk Systems (SDS) proposed fingerprint-based introspection mechanism for FFS-like file systems [15].

Using introspection for VMDIs instead of physical disks can be fairly useful. Several earlier studies explored VMDI introspection for intrusion detection [5, 7, 18], discovering VMs with buggy, infected, unlicensed, or outdated software [11], and locating the latest working VM snapshot [12]. Geiger applies introspection at the VMM level to identify live blocks, estimate working set size, and build a unified cache [8]. We use VMDI introspection at the NAS for general performance optimizations: data and meta-data prefetching, data placement, and storage tiering decisions.

7 Conclusions

VMDI introspection is a promising technique that allows SAs to improve performance by understanding the internal structure of virtual disk image files. Our prototype implementation of several introspection-based optimizations demonstrates up to 20× improvement for some common workloads. With emerging APIs to access file system formats, and the ease of implementation that we have demonstrated, we believe that using introspection to improve I/O performance in virtual environments shows promise.

Future Work. In addition to addressing the limitations of our current evaluation methodology, we plan to integrate our introspection module with the back-end file system of an SA; operating at this layer allows some optimizations to be developed more easily and efficiently than at the NFS level. We also plan to evaluate introspection for non-FFS file systems, specifically ext4 and NTFS—both popular as guest file systems. Eventually we plan to investigate complex formats nested inside a file system, such as docx, archives, and ISO files.

Acknowledgements. We thank the anonymous USENIX reviewers and our shepherd Andrea Arpaci-Dusseau for their valuable feedback. This work was supported in part by NSF awards CCF-0937833 and CCF-0937854, and an IBM PhD Fellowship.

References

[1] L. N. Bairavasundaram, M. Sivathanu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. X-RAY: A non-invasive exclusive caching mechanism for RAIDs. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, 2004.

[2] T. Bittman. *Virtual machines and market share through 2012*. Gartner, October 2009. ID Number: G00170437.

[3] Filebench. `http://filebench.sf.net`.

[4] NFS-GANESHA. `http://sourceforge.net/apps/trac/nfs-ganesha/`.

[5] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2003.

[6] D. Hildebrand, A. Povzner, R. Tewari, and V. Tarasov. Revisiting the storage stack in virtualized NAS environments. In *Proceedings of the Workshop on I/O Virtualization (WIOV)*, 2011.

[7] X. Jiang and X. Wang. "Out-of-the-box" monitoring of VM-based high-interaction honeypots. In *Proceedings of the International Conference on Recent Advances in Intrusion Detection (RAID)*, 2007.

[8] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Geiger: monitoring the buffer cache in a virtual machine environment. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2006.

[9] T. M. Kroeger and D. D. E. Long. Design and implementation of a predictive file prefetching algorithm. In *Proceedings of the Annual USENIX Technical Conference (ATC)*, 2001.

[10] Libvirt virtualization API. `http://libvirt.org`.

[11] D. Reimer, A. Thomas, G. Ammons, T. Mummert, B. Alpern, and V. Bala. Opening black boxes: using semantic information to combat virtual machine image sprawl. In *Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, 2008.

[12] W. Richter, G. Ammons, J. Harkes, A. Goode, N. Bila, E. de Lara, V. Bala, and M. Satyanarayanan. Privacy-sensitive VM retrospection. In *Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2011.

[13] M. Sivathanu, L. N. Bairavasundaram, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Life or death at block-level. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.

[14] M. Sivathanu, V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Improving storage system availability with D-GRAID. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, 2004.

[15] M. Sivathanu, V. Prabhakaran, F. I. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Semantically-smart disk systems. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, 2003.

[16] V. Tarasov, D. Hildebrand, G. Kuenning, and E. Zadok. Virtual machine workloads: The case for new benchmarks for NAS. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, 2013.

[17] Virtual disk development kit. `http://communities.vmware.com/community/vmtn/developer/forums/vddk`.

[18] Y. Zhang, Y. Gu, H. Wang, and D. Wang. Virtual-machine-based intrusion detection on file-aware block level storage. In *Proceedings of the International Symposioum on Computer Architecture and High Performance Computing (SBAC-PAD '06)*, 2006.