Towards Paravirtualized Network File Systems

Raja Appuswamy, Sergey Legtchenko, Antony Rowstron
Microsoft Research, Cambridge

Abstract

The virtualized storage stack used in enterprise data centers provides two mechanisms to enable virtualized applications to store and retrieve data, namely, virtual disks and network file systems. In this paper, we examine the pros and cons of using these two mechanisms to integrate emerging non-volatile memory devices, and show how neither of them provide low-overhead access to data without sacrificing compatibility with other popular virtualization-enabled features. In doing so, we present paravirtualized NFS, an alternate mechanism for accessing data, highlight its benefits, and outline research challenges involved in realizing it in practice.

1 Introduction

All modern enterprise data centers use virtualization-driven server consolidation to improve resource utilization. In order to meet the ever-increasing storage demands of virtualized applications, these data centers have long migrated from using node-local, direct-attached storage to consolidated, shared Network-Attached Storage (NAS) servers [12]. The virtualized storage stack used in these VM-NAS installations provides two mechanisms by which applications can access data stored on the remote NAS server, namely, virtual hard disks (VHD), and network file systems (NFS).

In this paper, we investigate these pros and cons of using these two mechanisms to access data stored in emerging non-volatile memory devices. In doing so, we show that there exists a performance–flexibility trade off in using these mechanisms, as one could either opt to use the overhead-free NFS mechanism while giving up compatibility with features like live migration, and client-side caching, or adopt the VHD mechanism and trade off performance for compatibility. Given this trade off, we present paravirtualized NFS, an alternate data access mechanism that offers the benefits of both VHD and NFS mechanisms without any of their associated disadvantages.

2 VM-NAS Data Access Mechanisms

Figure 1 shows the layers that constitute the I/O stack in a VHD-based VM-NAS setup. As can be seen, requests issued by application servers running within a guest OS are

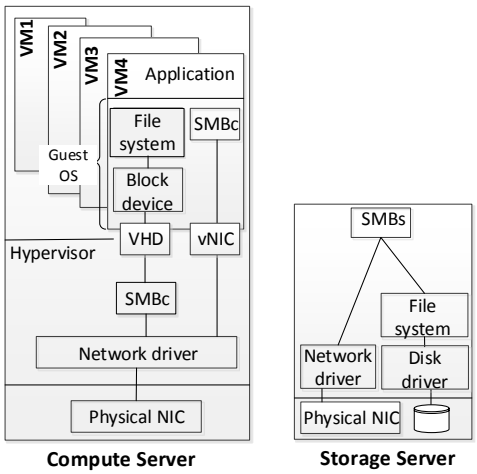

Figure 1: Data access alternatives in the VM-NAS stack

first mapped from files to blocks by the guest file system. Then, the guest block subsystem schedules, queues and issues these block requests to the virtual disk. These requests are intercepted by the virtual disk emulation subsystem in the hypervisor and routed to the network file system client (SMBc in Figure 1) which, in turn, forwards the requests across the data center network to the SMB server running within a NAS server. The SMB server converts these block request into file requests to the server-local file system which finally satisfies the requests using the server-local block subsystem.

An unintended side effect of using the VHD abstraction with NAS file servers is that I/Os issued by a virtualized application undergo a series of file–block–file transformations due to redundant nesting of file systems and block schedulers in the guest OS and the storage server. Recent studies have shown that this mutilayered transformation process increases the amount of random I/O on the server side causing performance issues in disk-based VM NAS servers [12].

An alternative to the VHD mechanism involves mapping file systems stored on remote NAS servers directly into the guest OS. In this VM-NAS setup, the file I/O requests issued by the virtualized application are handled by the network file system client in the guest OS which,

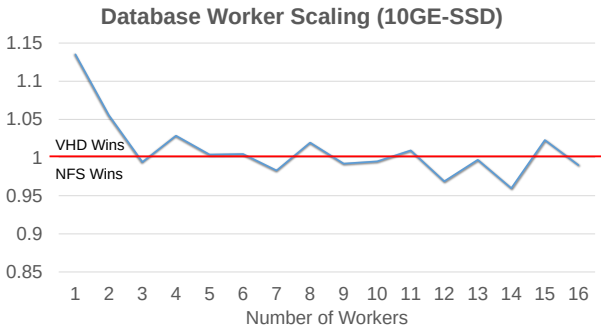

Figure 2: Performance of VHD configuration normalized by that of NFS in a test setup where data is stored in a SSD RAID array and accessed over a 1 GE network.

in turn, forwards it directly to the network file system server running on the remote NAS server (SMBs in Figure 1). The SMB server handles these file requests using server-local file systems as before. Thus, this approach completely side steps the performance issues inherent to the VHD mechanism by using NFS clients within the guest OS to bypass the guest file system and block layers.

2.1 VHD versus NFS: 1 Gigabit Ethernet-SSD

Despite the performance benefits offered by the NFS mechanism, it has not been widely adopted as the preferred data access approach due to two reasons. First, the approach of using NFS clients within the guest OS is incompatible with the other virtualization-enabled features like second-level SSD caching. Second, the software overheads inherent to use of VHDs are typically masked by high network access latencies in today's enterprise data centers that use 1 Gbps to connect compute servers with HDD/SSD-based storage servers. We will now present results from our experimental evaluation that quantifies this overhead using a hardware testbed that resembles a contemporary data center installation.

Test Setup: For these tests, we used two 4-socket Dell PowerEdge 910 servers, both equipped with four 8-core Intel Xeon E7-4820 2 GHz processors, connected using a 1 Gbps switch. Our compute server runs a VM that is assigned 16 vCPUs and 4 GB of memory (we chose 16 vCPUs to compare the performance of this setup with the one described in Section 3.) The storage server is equipped with eight 240 GB Intel 520 Series SSDs grouped in a RAID-0 configuration. A directory created on the SSD-based RAID array was exported as an SMB 3.0 share (with caching turned off) and was used as the data store. Windows Server 2012 is the guest OS within the VM as well as the host OS on both servers. Hyper-V is used as the hypervisor on the compute server.

Benchmark: We configured IOMeter to generate a workload that resembles a typical database server (100% random accesses, 67% reads, 8 KB block size, queue depth of 8) and used this workload to benchmark the performance of VHD and NFS configurations. In all cases, IOMeter operates on a single 10 GB file and runs within a VM as a virtualized application. In the first configuration, IOMeter operates on a file stored within a VHD, with the VHD itself being stored as a file on the storage server. In the second configuration (NFS), the SMB network share is directly mapped into the guest VM and IOMeter operates on a file stored on the network share. All tests were run three times and as the variation was less than 5%, we report only the median IOPS.

Result: Figure 2 shows the performance achieved by using by the VHD configuration normalized by the performance of the NFS configuration under the database workload while scaling the number of workers. As can be seen, despite the file–block–file translation, VHDs incur a negligible overhead. This is due to the fact that the 1 Gbps network dictates the observed performance in this hardware setup, as high network access latencies (300 µs average) dominate the observed response time when there are few workers, and the network bandwidth becomes the limiting factor when more than three workers are used under the database workload.

Thus, given that VHDs are capable of providing full-system virtualization with moderate overhead, they are certainly the right data access mechanism for today's VM-NAS installations.

3 NVM Data Access Options: Virtual Disk or Network File System?

Over the past few years, flash-based solid state storage devices have evolved from being used as SATA-based HDD accelerators to DIMM-slot-resident non-volatile memory devices [4]. Emerging non-volatile memory technologies, like Phase Change Memory (PCM), are expected to scale to capacities much larger than flash, while, at the same time, offering latencies and bandwidth guarantees comparable to DRAM [9].

Rivalling the rate of growth of these non-volatile storage devices are recent advances in data center networking. With the emergence of Converged Enhanced Ethernet (CEE), features like Remote Direct Memory Access (RDMA), which were once limited to low-latency networks like InifiBand and HPC clusters, have started to appear even over Ethernet (RoCE) [6]. Accordingly, network file system protocols have been extended to provide low-latency file access over RDMA [11].

Given these trends, there are two ways in which NVM devices can be used in VM-NAS installations, namely, as a persistent, client-side cache device in a compute server,

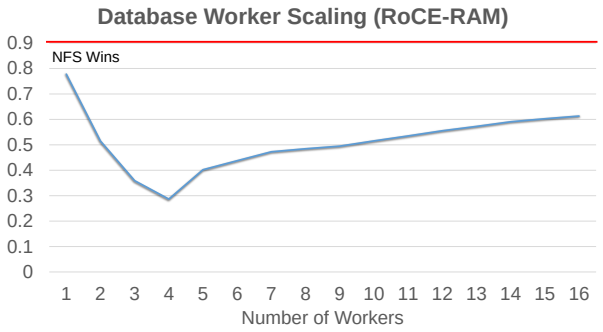

Figure 3: Performance of VHD configuration normalized by that of NFS in a test setup where data is stored in a RAM disk and accessed over a RoCE network.

or as a primary data storage device in NAS-based storage server. Ideally, the data access interface provided by the virtualized storage stack should provide low-overhead, high-performance access to NVM-based storage devices irrespective of the type of NVM integration used.

The NFS interface is capable of providing such low-overhead data access to virtualized applications. For instance, using DirectPath I/O (or SR-IOV), a NFS client running within a guest OS could issue RDMA requests directly to the physical NIC completely bypassing both the guest-OS's storage stack and the hypervisor. However, such hypervisor bypass makes it impossible to integrate NVM as a hypervisor-managed second-level cache. The VHD mechanism, on the other hand, is integration-independent, can be used in both types of VM-NAS installations, and does not impose any compatibility limitations. However, as we will show in this section, it suffers from a significant performance penalty due to overheads inherent to the file–block–file translation.

3.1 VHD versus NFS: RoCE - RAM Disk

Test Setup: For these tests, we used two servers, both equipped with 16 Intel Xeon E5-2665 2.4 GHz cores, 384 GB of RAM and a 40 Gbps RDMA-capable Mellanox ConnectX-3 NIC with a full-duplex port. The servers are connected to a Mellanox MSX1036B-1SFR switch using RoCE at the link layer. The hypervisor uses SMB 3.0 protocol over the SMB Direct RDMA transport. On the storage server, we use a 20 GB RAM disk to represent a NVM device. As before, a directory created on the RAM disk is exported over SMB.

Benchmark: In the VHD configuration, we run IOMeter in a single VM on the compute server as before using an identical setup (16 vCPUs, 4 GB memory). IOMeter reads/writes data from a file stored in a VHD, with the VHD itself being stored as a file on the RAM disk-based network share. Supporting the NFS mechanism requires using DirectPath I/O (SR-IOV) to provide a direct communication channel between RDMA-capable NFS clients and physical NICs. Unfortunately, the RDMA-based SMB Direct protocol currently supported by Hyper-V only works in the non-virtualized host environment, and hence, it is not possible for SMB clients in Hyper-V guest VMs to directly participate in RDMA-based data exchange with NAS servers.

For this work, we simulate the NFS setup by running IOMeter directly on the host as a non-virtualized application. Recent studies using other hypervisors, like VMWare ESX Server, have shown that applications running inside VMs empowered with DirectPath I/O are capable of achieving performance on par with non-virtualized case [3]. Thus, we believe that the results we obtain from using an SMB client in the host OS would be a close approximation of the performance of a NFS configuration.

Result: Figure 3 presents the normalized performance of the VHD configuration in our new test setup. Comparing this to Figure 2, one can clearly see that benefits offered by using the virtual disk abstraction are no longer free. Unlike the 1 Gbps setup, the VHD configuration consistently suffer from a significant performance penalty (3x in the worst case) when compared to the NFS configuration. This deterioration stems from several sources including file systems and block schedulers in the guest OS, translation from VHD blocks to network file requests in the virtual disk emulation layer, etc. Compare to VHD, the NFS approach side steps all these sources of overhead.

To summarize, the current VM-NAS storage stack presents a performance–flexibility trade off, as one could either use low-overhead NFS mechanism while sacrificing compatibility with several other hypervisor features, or use integration-independent, feature-compatible VHD mechanism while sacrificing performance.

4 The Case for Paravirtualized NFS

Given the drawbacks of VHD and NFS, it is obvious that a new mechanism that provides low-overhead access to data while enabling guest I/O interpositioning by the hypervisor is necessary. In this section, we will introduce paravirtualized NFS, highlight its advantages, and identify potential challenges that require further research.

4.1 Paravirtualized NFS

Conceptually, the paravirtualized NFS (PV-NFS) approach can be seen as extending the traditional NFS approach to support hypervisor interpositioning. Thus, similar to a traditional NFS client, a paravirtualized NFS client operates within the guest OS and provides a standardized network file system interface to virtualized ap-

plications. However, unlike the traditional NFS client, a paravirtualized client uses explicit guest–host communication mechanism provided by the hypervisor to forward all application file I/O requests to the hypervisor. Thus, the paravirtualized client would, in effect, act as a pass-through, request-forwarding proxy that delegates the task of communicating with the remote NAS server to the hypervisor.

As the hypervisor now intercepts all file I/O requests initiated by the application, it can perform various transformations on these requests to implement the file-level equivalent of all features traditionally supported using VHDs at the block level. Thus, the hypervisor would snapshot and clone file systems (instead of VHDs), cache files (rather than VHD blocks), and implement whole-file or subfile deduplication (rather than block-level deduplication).

4.2 Paravirtualized NFS: Advantages

The PV-NFS mechanism possesses the advantages of both VHD and NFS mechanisms without suffering from any of their associated drawbacks. Similar to VHDs, all guest I/O requests are intercepted and serviced by the hypervisor. Thus, the PV-NFS mechanism can be used to implement hypervisor-based features like second-level caching. However, unlike the VHD mechanism, and similar to the NFs mechanism, the paravirtualized NFS mechanism does not incur any translation overhead, as file I/O requests made by the application completely bypass the guest OS's storage stack.

Further, like the NFS mechanism, client applications can use file servers to not just store files but also share them with other clients. Virtual disks, on the other hand, force applications to adopt other means of sharing, as data stored in a virtual disk is private to the client (and the client's guest file system). In addition, as file servers store files with the PV-NFS approach, there is no need for using complex virtual-disk introspection techniques to extract semantic information. Rather, network file servers can use optimizations based on file typing information or file access patterns to improve the performance of even virtualized data center workloads.

4.3 Paravirtualized NFS:Challenges

The performance results presented in this paper both highlight the overhead of using VHD as a storage virtualization mechanism, and demonstrate the potential benefits of using the PV-NFS mechanism. However, more research is required to understand 1) the overhead of guest–host file I/O bypass mechanism, 2) the scalability of file-based client-side functionalities, and 3) usability complications due to lack of full-system virtualization. We will elaborate on these topics in this section.

One way of implementing the guest-NFS proxy is to not use paravirtualization at all; one could simply host a NFS server in the hypervisor and use unmodified NFS clients within the guest OS to forward file I/O requests to the hypervisor. Although the results are not shown here due to lack of space, we did evaluate the performance of this approach and we found that using SMB-based file I/O bypass incurs a 4x performance penalty compared to VHDs due to network virtualization and SMB protocol overheads. This clearly demonstrates the need for a low-overhead paravirtualization-based communication channel between the guest OS and the hypervisor. Recent research has shown how block I/O paravirtualization techniques can achieve performance comparable to that of DirectPath I/O (SR-IOV). Thus, an important area that requires further research is understanding whether these techniques can be extended to implement a paravirtualized NFS client [5].

Traditionally, hypervisors have exploited the block abstraction provided by the VHD mechanism to implement various features like snapshotting, caching etc. As we mentioned earlier, with a PV-NFS client, the hypervisor could implement semantically-aware, file-based versions of these features as it services file I/O requests, not just VHD block requests. For instance, consider second-level caching. An obvious approach to integrating NVM as a second-level cache is to have the NFS client resident in the hypervisor perform second-level data caching using NVM similar to the way SSDs are used today. The PV-NFS client operating within each guest performs first-level caching in DRAM similar to file system buffer caches.

Such an integration raises several important questions. First, can caching (and other client-side functionalities like deduplication) be implemented efficiently at the file level? How do we integrate such functionalities with other NVM-specific tasks like wear-leveling for PCM? How do we preserve the ordering of writes issued by the PV-NFS client? What is the right division of labor between hypervisors and NAS servers? Most modern NAS servers support snapshotting and cloning of multiple file volumes. Should the hypervisor exploit these features or should it reimplement them on the client side?

Despite several advantages, the PV-NFS mechanism does have one disadvantage compared to the VHD mechanism, namely, its inability to offer full-system virtualization. Unlike the VHD mechanism, where a single virtualized device can integrate both system and data volumes into a single unit of administration, the PV-NFS mechanism requires separate management of boot volumes (stored in VHDs) and application data volumes (stored as file systems in NAS servers). We believe that this limitation would not be an issue in the server consolidation scenario, as server VMs already use shared,

master VHD boot images that are not updated frequently, and rely on the scalability and reliability of NAS servers to store large amounts of data. However, more investigation is required to understand the impact of this limitation on usability and administration in VDI installations.

5 Related Work

While the work presented in this paper is, to our knowledge, the first study that examines the overhead of using current data access mechanisms in data centers with network-attached NVM, researchers have identified various issues associated with the usage of VHDs in other contexts.

Le et al. [8] measured the performance impact of file system nesting in installations where virtual disks are stored on local file systems. They observed that the use of two file systems, one in the guest and the other in the host, can cause unpredictable I/O performance and significant increase in end-to-end access latency. Similarly, Boutcher and Chandra [1] investigate the effect of nested I/O schedulers and show how the worst-case scenario could cause 40% throughput degradation. Recently, researchers also investigated the impact of storing virtual disks on NAS servers from a workload distortion point of view and have made the case for creating new NAS benchmarks explicitly targeted at VM-NAS deployments [12].

Based on the observation that virtualization is often used to achieve application compatibility, Jannen et al. [7] propose Zoochory, a redesign of the virtualized storage stack that changes the division of labor between guest and host file systems by isolating media management to host file systems and reducing guest file systems to mere API implementations. One could also achieve such a division of labor by abandoning traditional block abstraction in favor of a semantically richer alternative. For instance, one could use the Object Storage Device (OSD) abstraction and have the hypervisor provide each guest VM with an OSD, rather than a traditional block device. However, we believe that such an approach is too invasive as it either requires end-to-end support (at both client and server) for the OSD protocol, or forces the hypervisor to implement the mapping of objects to blocks (OSDFS), effectively relocating file system functionality from the guest to the hypervisor similar to Zoochory. The PV-NFS mechanism, in contrast, provides all the benefits of using the OSD abstraction while building on existing network file system protocols.

In contrast to Zoochory, Ventana [10] showed how a virtualization-aware distributed file system can solve several usability, management, and security issues associated with virtual disks. However, as the focus of that work was on proving that all VHD features could be implemented using distributed file systems, they used guest-resident NFS client to communicate with hypervisor-resident NFS server–an approach, which, as we demonstrated in our paper, possess a 4x performance penalty when used to access data stored in low-latency NVM.

A huge body of related research focuses on designing file systems for NVM devices [2]. As our focus is on integrating NVM devices in the virtualized storage stack, our work is orthogonal to this topic.

6 Conclusion

Fully exploiting the performance benefits of emerging NVM storage devices requires eliminating unnecessary software overhead in the virtualized storage stack. In this paper, we demonstrated the overhead of using VHDs as the primary data access mechanism in NVM-based VM-NAS installations. We showed how NFS offers a low-overhead alternative to VHD but is incompatible with other popular virtualization-enabled features. Given this performance–flexibility trade off, we made the case for using PV-NFS as a new data access mechanism, highlighted its benefits, and outlined issues that warrant further research.

References

[1] BOUTCHER, D., AND CHANDRA, A. Does Virtualization Make Disk Scheduling Passé. *SIGOPS Oper. Syst. Rev. 44*, 1 (2010).

[2] CONDIT, J., NIGHTINGALE, E. B., FROST, C., IPEK, E., LEE, B., BURGER, D., AND COETZEE, D. Better I/O Through Byte-addressable, Persistent Memory. In *Proc. of the 22nd ACM SIGOPS Symp. on Oper. Syst. Prin.* (2009).

[3] DAVDA, B. Ultra-Low Latency on vSphere with RDMA. In *VMWorld* (2012).

[4] DIABLO TECHNOLOGIES. Memory Channel Storage.

[5] HAR'EL, N., GORDON, A., LANDAU, A., BEN-YEHUDA, M., TRAEGER, A., AND LADELSKY, R. Efficient and Scalable Paravirtual I/O System. In *Proc. of the USENIX Ann. Tech. Conf.* (2013).

[6] INFINIBAND TRADE ASSOCIATION. Supplement to Infiniband Architecture Specification Volume 2 Release 1.2.2 Annex A16: RDMA over Converged Ethernet (RoCE), 2010.

[7] JANNEN, W., TSAI, C.-C., AND PORTER, D. E. Virtualize Storage, Not Disks. In *Proc. of the 14th USENIX Work. on Hot Topics in Oper. Syst.* (2013).

[8] LE, D., HUANG, H., AND WANG, H. Understanding Performance Implications of Nested File Systems in a Virtualized Environment. In *Proc. of the 10th USENIX Conf. on File and Storage Tech.* (2012).

[9] LEE, B. C., IPEK, E., MUTLU, O., AND BURGER, D. Architecting phase change memory as a scalable dram alternative. In *Proc. of the 36th Ann. Intl. Symp. on Comp. Arch.* (2009).

[10] PFAFF, B., GARFINKEL, T., AND ROSENBLUM, M. Virtualization Aware File Systems: Getting Beyond the Limitations of Virtual Disks. In *Proc. of the Third Conf. on Networked Syst. Design and Impl.* (2006).

[11] TALPEY, T., AND KAMER, G. High Performance File Serving With SMB3 and RDMA via SMB Direct. In *Storage Developers Conference* (2012).

[12] TARASOV, V., HILDEBRAND, D., KUENNING, G., AND ZADOK, E. Virtual Machine Workloads: The Case for New Benchmarks for NAS. In *Proc. of the USENIX Conf. on File and Storage Tech.* (2013).