Memory Interface Simplifies Storage Virtualization

Shuichi Oikawa, Gaku Nakagawa
Department of Computer Science
University of Tsukuba
Tsukuba, Ibaraki, Japan
e-mail: {shui,gnakagaw}@cs.tsukuba.ac.jp

***Abstract*—Using a simple library operating system (OS) as a guest OS of a virtualized environment is one of the current trends of cloud computing in order to reduce the overheads incurred by virtualization. Its persistent storage access, however, remains the same as that for the existing guest OSes; thus, it poses a problem of the long execution and data paths. This paper proposes virtualized memory storage that provides the memory interface for a library OS of a virtualized environment, and also discusses the two key benefits of virtualized memory storage, journaling acceleration by synchronous access and a modern implementation of single-level store. The proposed memory interface to virtualized memory storage can simplify both the execution and data paths, and it accelerates the access to persistent storage.**

Keywords-operating systems; virtualization; file systems; storage

I. INTRODUCTION

There is a trend of using a simple library operating system (OS) as a guest OS of a virtualized environment. A library OS typically satisfies a specific need to execute a target application; thus, its simple and light-weight implementation enables higher efficiency of application execution than the traditional OS, such as the Linux and the BSD (Berkeley Software Distribution) UNIX. While a library OS lacks the protection support between an application and the kernel, it is protected from another library OS by a virtualized environment. There are several library OSes that target such a virtualized environment. Exokernel [1], [2] is one of early work that realized the kernel functions as libraries, and its success stimulated the following work. Mirage unikernel [3] is a library OS, of which applications are executed on the OCaml language runtime. OSv [4] is another library OS, applications of which are executed on the Java language runtime.

While library OSes emphasize their high performance, library OSes employ the existing block interface to persistent storage. The block interface for a virtualized environment, however, poses a significant problem to achieve high storage access performance. Fig. 1 depicts the architecture of the common storage virtualization method. It provides the block interface for a guest OS kernel; thus, a guest OS kernel requires a block device driver to interact with the block interface. Because of asynchronous nature and a long latency of the block interface, it requires the page cache to accommodate the recently accessed data. A file system is placed upon the page cache and a block device driver and interacts with them.

The problem to provide the block interface for a library OS is that the mechanisms for the block interface, a file system, the page cache, and a block device driver, are duplicated in a library OS, and they exist both in the host OS and a library OS. Because of such duplication, both the data and execution path become very long. They have to go though the layer of a file system, the page cache, and a block device driver both in a library OS and the host OS. The execution overhead of going through the file access layer twice is huge, and also data needs to be transferred several times. While a library OS kernel simplifies its mechanisms by specializing them for target applications, there is no simplicity achieved in the block interface for a virtualized environment.

This paper proposes the use of memory interface to persistent storage for a library OS of a virtualized environment. We call it *virtualized memory storage*. This architecture provides the memory storage for a library OS, and a file system is constructed upon the memory storage. Since processors can directly access memory, there is no need to interpose a device driver between a file system and storage. Virtualized memory storage makes the layer of a file system, the page cache, and a block device driver for a library OS of a virtualized environment as simple as that of the existing OS kernel, and thus it significantly simplifies and also accelerates the access for a library OS to persistent storage, since it enables the direct access for a library OS to the page cache of the host OS kernel. While this paper is based on the past work [5], [6], its focus on a library OS is different from them.

The rest of this paper is organized as follows. Section II describes the virtual memory storage and its key benefits. Section III concludes the paper and describes the future work.

II. VIRTUALIZED MEMORY STORAGE

Virtualized memory storage provides the memory interface for a guest OS of a virtualized environment. Fig. 2 depicts its architecture. Virtualized memory storage constructs a single hierarchy of a file system, the page cache mechanism, and a block device driver, which is the same as the monolithic kernel, while only a file system resides in a guest OS. Virtualized memory storage consists of a memory image provided by a virtual machine monitor, and is backed by the page cache of the host OS. While such a memory image is the same as that for a main memory of a guest OS, the memory image of virtualized memory storage is backed by a persistent storage device or a file on it. Therefore, the written data on virtualized memory storage persists across the process of shutting down a guest OS and rebooting it. Virtualized memory storage is analogous to a memory image created for a user process by the mmap system call. The mmap system call maps a file

 ISBN: 978-1-61208-529-6

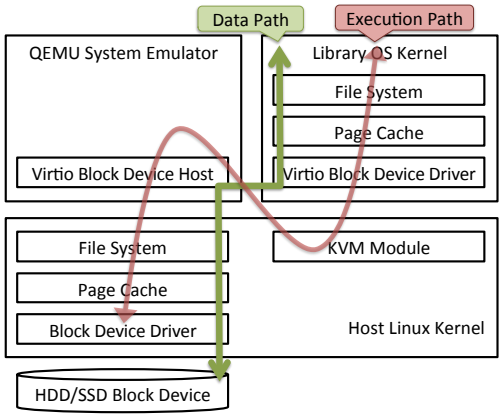

Figure 1. The common storage virtualization method that provides the block interface for a guest OS kernel.

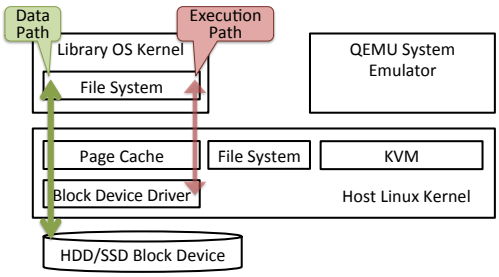

Figure 2. Virtualized memory storage that provides the memory interface for a library OS kernel.

on a virtual address space of a user process, and the user process can access a file through the mapped region of its virtual address space. In case of virtualized memory storage, the virtualization software maps a persistent storage device or a file on it on a physical address space of a guest OS, and the guest OS kernel can access the storage through the memory interface. There are several file systems that were designed to be constructed directly on memory storage. The persistent random access memory file system (PRAMFS) [7] and the storage class memory file system (SCMFS) [8] are such examples. They can be used on virtual memory storage in order to enable file access on it.

Virtualized memory storage best fits simple library OSes since it significantly simplifies the storage access architecture and also accelerates the access to persistent storage. The following are the benefits brought by virtualized memory storage: 1) No block device driver in library OSes and no device host in the host OS: A block device driver is a complicated software framework since it deals with the block and asynchronous interface of devices and also enables efficient access to them by utilizing the page cache mechanism. A block device driver in a guest OS requires a counterpart in the host OS, a device host, that emulates a block device. Virtualized memory storage gets rid of them; thus, it significantly simplifies the storage interface. 2) Simplified file system implementation: The implementation of a file system on a block device cannot be separated from the block interface even with the page cache mechanism that provides the memory interface because it needs to deal with block access natures and and to include their management. A file system on virtualized memory storage is greatly simplified since it does not include such block management and the page cache mechanism. 3) Zero copy data access: This is a great advantage of the integration of library OSes and virtualized memory storage. When an application accesses data on virtualized memory storage through a file system, the application obtains the address of the data on the virtualized memory storage. There is no need to copy from storage to buffer cache. 4) Efficient virtual machine migration: Virtualized memory storage is simply a memory image, and it can be treated in the same way as the main memory of a virtual machine. When a virtual machine is migrated from a host to another over the network, the main memory is copied between them. The same mechanism can be employed to transfer virtualized memory storage; thus, there is no specific shared storage necessary for virtualized memory storage to enable virtual machine migration.

Virtualized memory storage is secured by a virtual machine monitor since it is made independent form each other. The memory image of virtualized memory storage is created for each instance of a library OS, and its data is not shared by default. An instance of a library OS can only access its memory image but not the other images of the other instances since they are separated by the virtual memory mechanism that the virtual machine monitor sets up. Obviously, it is possible to create a shared memory image of virtualized memory storage. In this case, a whole memory image is shared; thus, all the files of the shared image are shared.

Virtualized memory storage simplifies the execution path to access storage; thus, such simplicity makes a system with it more reliable. While it removes the page cache and block device driver layers from a guest OS, it keeps the mechanisms in the host OS the same. An only difference is that it exposes the page cache of the host OS to a guest OS for data access. Only a part of the page cache is, however, exposed to a guest OS, and a guest OS does not have unlimited access to the page cache of the host OS. Thus, the introduction of virtualized memory storage does not increase security risks.

We discuss the two key benefits of virtualized memory storage below.

A. *Journaling Acceleration by Synchronous Access*

The journaling is a mechanism to guarantee the consistency of written data. It is known as write ahead logging (WAL) for database management systems. The journaling writes data twice, first in the journal and second in the destination place. The significant cost of the journaling is brought by a latency to complete writing. Logging must be completed and it must ensure the log becomes persistent before writing in the destination place. In other words, writing in the destination place must wait for the completion of logging. Each log tends to be small data, and writing small data in block storage is a typical

inefficient operation; thus, it causes a long latency to complete writing.

Virtualized memory storage employs synchronous access to storage; thus, logging does not suffer from a long latency caused by the block interface. Because the region for logging is typically preallocated and fixed, its page frames can be pinned down in the page cache of the host OS. Even without pinning them down, logging is frequently performed, and the region for logging is very likely on the page cache.

B. Modern Implementation of Single-Level Store

Virtualized memory storage can be considered as an implementation of the single-level store [9]. The single-level store is the model of storage where applications access data storage objects directly through the memory interface; thus, there is only a single storage level [10]. From the point of view of the single-level store, memory and disk storage are distinct parts of computer systems since memory is byte addressable while disk storage is block addressable; thus, processors cannot access disk storage directly, and data on disk storage must be brought to memory in order for the processors to access it. Files are the abstraction of disk storage, and file systems manage disk storage to provide storage spaces with users as files. Data in files is accessed through the file application program interface (API), which is designed to deal with block addressable disk storage.

Virtualized memory storage takes the memory management one step further towards the single-level store by involving the memory interface in the hierarchy of memory and storage. Library OSes can access data on memory storage directly since memory storage is byte addressable and a file system serves name and protection services.

III. Conclustion and Future Work

This paper proposed virtualized memory storage that provides the memory interface for a library OS of a virtualized environment in order to simplify and to accelerate the access to persistent storage. Because of a trend of using a simple library OS as a guest OS of a virtualized environment for higher performance, the proposed virtualized memory storage best fits the use cases of a library OS.

Our future work includes the implementation and evaluation of the proposed architecture. While we realized the basic mechanism of the virtualized memory storage [5], we are currently working on the implementation of a library OS on top of it.

References

[1] D. R. Engler, M. F. Kaashoek, and J. O'Toole, Jr., "Exokernel: An operating system architecture for application-level resource management," in Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles, ser. SOSP '95. New York, NY, USA: ACM, pp. 251–266, 1995.

[2] M. F. Kaashoek, et al., "Application performance and flexibility on exokernel systems," in Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles, ser. SOSP '97. New York, NY, USA: ACM, pp. 52–65, 1997.

[3] A. Madhavapeddy, et al., "Unikernels: Library operating systems for the cloud," in Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS '13. New York, NY, USA: ACM, pp. 461–472, 2013.

[4] Cloudius-Systems, "Osv: the operating system designed for the cloud," http://osv.io [retrieved: February, 2017].

[5] S. Oikawa, "Virtualizing storage as memory for high performance storage access," in Proceedings of the 12th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA-14), pp. 18–25, 2014.

[6] S. Oikawa, "Adapting byte addressable memory storage to user-level file system services," in Proceedings of ACM Conference on Research in Adaptive and Convergent Systems, ser. RACS 2014. ACM, pp. 338–343, 2014.

[7] "Pramfs: Protected and persistent ram filesystem," http://pramfs.sourceforge.net/ [retrieved: February, 2017].

[8] X. Wu and A. L. N. Reddy, "Scmfs: a file system for storage class memory," in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, ser. SC '11. New York, NY, USA: ACM, pp. 39:1–39:11, 2011.

[9] B. E. Clark and M. J. Corrigan, "Application system/400 performance characteristics," IBM Systems Journal, vol. 28, no. 3, pp. 407–423, 1989.

[10] J. S. Shapiro and J. Adams, "Design evolution of the eros single-level store," in Proceedings of the General Track of the Annual Conference on USENIX Annual Technical Conference, ser. ATEC '02. Berkeley, CA, USA: USENIX Association, pp. 59–72, 2002.

 ISBN: 978-1-61208-529-6