Towards New Interface for Non-volatile Memory Storage

Shuichi Oikawa
Faculty of Engineering, Information and Systems, University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki, Japan

Keywords: Operating Systems, File Systems, Storage, Non-volatile Memory.

Abstract: Non-volatile memory (NVM) storage is rapidly dominating the high end markets of enterprise storage, which requires high performance, and mobile devices, which require lower power consumption. As NVM storage becomes more popular, its form evolves from that of HDDs into those that fit the market requirements more appropriately. Such evolution also stimulates the performance improvement because it leads to the changes of the interface that connects NVM storage with systems. There is a claim that the further improvement of NVM storage performance makes it better to poll a storage device to sense completion of access requests rather than to use interrupts. Polling based storage can expand to become main memory based on NVM storage since there is no complex mechanism required to enable interrupts and access requests are processed one by one. This paper predicts that NVM storage will be in a form of main memory, and proposes constructing a file system directly on it in order to overcome its drawbacks when used simply as main memory. The performance projection of the proposed architecture is that accessing files on such a file system can reduce the overhead introduced by handling block devices.

SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Non-volatile memory (NVM) storage is rapidly dominating the high end markets of enterprise storage, which requires high performance, and mobile devices, which require lower power consumption. Flash memory[1] is currently the most popular NVM, and its storage is called SSDs (Solid State Drives). As NVM storage becomes more popular, its form evolves from that of HDDs (Hard Disk Drives) into those that fit the market requirements more appropriately. Such evolution also stimulates the performance improvement because it leads to the changes of the interface that connects NVM storage with systems. The examples of the currently employed interfaces are PCI Express and NVM Express.

The investigation results shown by (Caulfield et al., 2010) and (Yang et al., 2012) posed one interesting claim that the further improvement of NVM storage performance makes it better to poll a storage device to sense completion of access requests rather than to use interrupts. They investigated high performance NVM storage architecture and found that the existing block device interface of the operating system (OS) kernel does not always perform well with it. The reason of this claim is that by excluding the time required for process context switching and interrupt processing there is no time left for a yielded process to be executed if processing times of access requests shorten; thus, it is simply faster for systems to poll a device in order to wait for completion of access requests.

Such polling based storage device can simplify the mechanisms that constitute the device since there is no complex mechanism required to enable interrupts and access requests are processed synchronously. Synchronous processing of access requests corresponds to the functionality of main memory; thus, NVM storage can expand to become main memory in this aspect. Flash memory is, however, not byte addressable; thus, DRAM buffer is required in order to connect flash memory to the memory bus of CPUs, and address translation is performed to export the whole address space of flash memory. This paper calls this memory architecture *NVM storage based main memory*. eNVy (Wu and Zwaenepoel, 1994) is an example of such a memory architecture while it assumes that flash memory is byte addressable. This approach, however, has a significant drawback that there is no way to predict addresses of future access; thus, access locality is the only means to mitigate the long access latency of NVM.

This paper predicts that NVM storage will be in a form of main memory and connect with the memory bus; thus, the storage devices are byte addressable, and CPUs can simply access the data on them by

[1] In this paper, flash memory stands for NAND flash memory.

Oikawa S..
Towards New Interface for Non-volatile Memory Storage.
DOI: 10.5220/0004757901740179
In *Proceedings of the 4th International Conference on Pervasive and Embedded Computing and Communication Systems* (PECCS-2014), pages 174-179
ISBN: 978-989-758-000-0
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

using memory access instructions. The drawback of this architecture is that access latencies vary depending upon the locations of the accessed data. Based on the prediction, the paper proposes constructing a file system directly on NVM storage in order to overcome its drawbacks when used simply as main memory.

The proposed architecture utilizes NVM storage based main memory as a base device of a file system. The file system directly interacts with the device without the interposition of the block device driver framework and the page cache mechanism. This architecture can be a solution to avoid the drawback posed by NVM storage based main memory, and also has several advantages over the existing file system architecture based on block devices. The advantages include the simplification of the OS kernel architecture and faster processing of data access requests because of the simplified execution paths.

We show the performance advantage of the proposed architecture by performing two experiments. The experiments show the performance impacts imposed by the existing block device driver framework. The results show the significant overhead of the block device driver framework, and the proposed architecture has a lot of room to achieve high performance access to data on NVM storage.

The rest of this paper is organized as follows. Section 2 describes the background of the work. Section 3 describes the details of the proposed architecture. Section 4 describes the result of the experiments. Section 5 describes the related work. Section 6 summarizes the paper.

2 BACKGROUND

This section describes the background of the work. It first describes the OS kernel's interaction with block devices. It then describes, the implication of NVM storage performance improvement. It finally describes the overall architecture of NVM storage based main memory.

2.1 Interacting with Block Devices

Block devices, such as SSDs and HDDs, are not byte addressable; thus, CPUs cannot directly access the data on these devices. A certain size of data, which is typically multiples of 512 byte, needs to be transferred between memory and a block device for CPUs to access the data on the device. Such a unit to transfer data is called a block.

The OS kernel employs a file system to store data in a block device. A file system is constructed on a block device, and files are stored in it. In order to read the data in a file, the data first needs to be read from a block device to memory. If the data on memory was modified, it is written back to a block device. A memory region used to store the data of a block device is called a page cache. Therefore, CPUs access a page cache on behalf of a block device. Figure 1 depicts the hierarchy of CPUs, a page cache, and a block device as the existing architecture to interact with block devices.

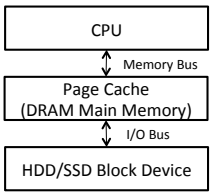

Figure 1: The existing architecture to interact with block devices.

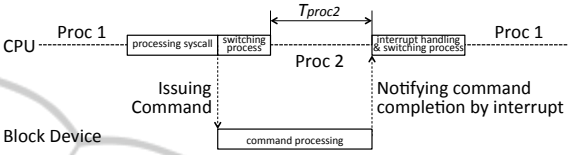

Figure 2: The asynchronous access command processing and process context switches.

Since HDDs are orders of magnitude slower than memory to access data on them, various techniques were devised to amortize the slow access time. The asynchronous access command processing is one of commonly used techniques. Its basic idea is that a CPU executes another process while a device processes a command. In Figure 2, Process 1 issues a system call to access data on a block device. The kernel processes the system call and issues an access command to the corresponding device. The kernel then looks for the next process to execute and perform context switching to Process 2. Meanwhile, the device processes the command, and sends an interrupt to notify its completion. The kernel handles the interrupt, processes command completion, and performs context switching back to Process 1. T_{proc2} is a time left for Process 2 to run. Because HDDs are slow and thus their command processing time is long, T_{proc2} is long enough for Process 2 to proceed its execution.

2.2 Implication of NVM Storage Performance Improvement

The performance of flash memory storage has been improved by exploiting parallel access to multiple chips (Josephson et al., 2010), and newer NVM

technologies inherently achieve higher performance. Such higher performance changes a premise that devised the asynchronous access command processing to amortize the slow access time of block devices, and can affects how the OS kernel manages the interaction with NVM storage.

One of such possibilities is a claim made by (Caulfield et al., 2010) and (Yang et al., 2012). The claim compares the asynchronous access command processing with the synchronous processing, and shows that the synchronous processing is faster. The claim was supported by the experiments performed by employing a DRAM-based prototype block storage device, which was connected to a system through the PCIe Gen2 I/O bus. For a 4KB transfer experiment performed by (Yang et al., 2012), the asynchronous processing takes 9.0 μs for Process 1 to receive data from the block device, and T_{proc2} is 2.7 μs. In contrast, the synchronous processing takes only 4.38 μs. The difference is 4.62 μs, which is longer than T_{proc2} of the asynchronous processing; thus, the synchronous processing saves the overall processing time.

As the I/O bus becomes faster, the command processing time of a device becomes shorter; thus, T_{proc2} also becomes shorter because the context switching time remains the same. Therefore, there will be no useful time left for another process while waiting for command completion.

2.3 NVM Storage based Main Memory

High performance NVM storage makes asynchronous processing of access requests meaningless as described above. Moreover, most of the current NVM storage devices equip with a certain amount of DRAM as a buffer to accommodate transient data. Though the sizes of DRAM buffer vary in accord with their targets and prices, devices with 1GB of DRAM buffer can be found among recent products.

Such facts easily make NVM storage expand to become main memory. By making it directly connect to CPUs through a memory bus, there is no complex mechanism required to enable the asynchronous access command processing, and access requests are simply processed synchronously. In order to make NVM storage work as main memory, the whole address space made available by NVM storage needs to be addressable by CPUs while NVM storage cannot be directly connected to CPUs. Since a DRAM buffer is connected to CPUs through an address translation mechanism, the address translation mechanism enables mapping of DRAM to NVM storage. This paper calls this memory architecture *NVM storage based main memory*, and Figure 3 depicts it. Making NVM

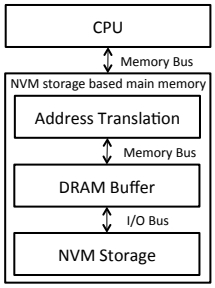

Figure 3: NVM storage based main memory architecture.

storage work as main memory is not a new idea as it was researched when flash memory appeared. eNVy (Wu and Zwaenepoel, 1994) is such an example.

Newer NVM technologies, such as PCM and ReRAM, are byte addressable and provide higher performance than flash memory. They, however, have limited write similar to flash memory; thus, they cannot simply replace DRAM, and a DRAM buffer is required to place frequently accessed data. Therefore, NVM storage based main memory is also legitimate for such newer NVM technologies.

This approach, however, has a significant drawback that access latencies vary depending upon the locations of the accessed data. If the data is on a DRAM buffer, the access latencies are comparable to DRAM. If the data is on NVM, the access latencies can be significantly longer. The problem is that there is no generally working way to predict addresses of future access. If addresses of future access can be predicted, the data of these addresses can be read ahead in the DRAM buffer in order to mitigate the long access latency of NVM. Main memory is accessed by physical addresses. Because physical addresses do not provide any information of higher abstractions that can be clue to predict addresses of future access, no useful information is available. Therefore, access locality is the only means to accommodate frequently accessed data in the DRAM buffer.

3 PROPOSED ARCHITECTURE

This section describes the proposed architecture that constructs a file system on NVM storage based main memory.

The proposed architecture utilizes NVM storage based main memory as a base device of a file system. NVM storage persistently stores the data of a file system, and CPUs access the data through a memory bus. The file system directly interacts with the device through the memory interface; thus, the block device driver for NVM storage is not required. Files

in the file system can be directly accessed and also mapped into the virtual address spaces of processes since files reside on memory; thus, there is no need to copy the data of files to a page cache. Therefore, the page cache mechanism is not required, either.

This architecture can be a solution to avoid the drawback posed by NVM storage based main memory. File systems are designed to allocate the blocks referenced by a single file as contiguous as possible since contiguous blocks can be accessed faster on HDDs. Such contiguous blocks make it easy to read ahead blocks in the DRAM buffer. Moreover, recent advances of file systems and storage architecture bring the concept of object based storage devices (OSDs), and higher abstractions are introduced and made available in storage. Such an architecture also enables to take advantage of the knowledge of higher abstractions for the prediction of future access. It should also be possible for user processes to give hints to NVM storage based main memory in order to read ahead blocks since user processes should have knowledge of their access patterns.

The architecture also has several advantages over the existing file system architecture based on block devices, such as the simplification of the kernel architecture and faster processing of data access requests because of the simplified execution paths in the kernel. The kernel architecture can be significantly simplified since the architecture does not require block device drivers and a page cache. File systems directly interact with NVM storage through the memory interface although additional command interface may be needed to communicate with it in order to exchange hint information. The simplified architecture can stimulate active development of more advance features that take advantage of NVM storage based main memory. Accessing data in a file becomes much faster since there is no need to go through a complex software framework that consists of a page cache mechanism and block device drivers in the kernel. Such a complex software framework paid off when block devices are as slow as HDDs. High performance NVM storage makes it outdated, and can rather take advantage of the simplified execution paths in the kernel.

4 EXPERIMENTS

This section shows the performance advantage of the proposed architecture. We performed two experiments in order to show the performance impacts imposed by the existing block device driver framework. We performed the experiments on the QEMU system emulator, and measured the number of instructions required to read and write certain sizes of a file from a user process by using read and write system calls on the Linux kernel, of which version is 3.4. The NVM storage based main memory is emulated by memory; thus, no extra overhead is considered. The user process allocates a 8KB buffer for reading and writing.

For the measurements of file reading, the file was read when its data is not cached; thus, the measurements include the costs of page cache allocation if applicable. For the measurements of file writing, the file was written when all the blocks of the file were allocated; thus, the measurements do not include the costs of block allocation.

4.1 Performance Impact of I/O Request Scheduling

The first experiment measures the performance impact of I/O request scheduling involved in the block device driver framework. The I/O request scheduling is especially necessary for HDDs. It queues access requests and sorts them, so that it can minimize seek times, which are the major factor of access latencies. The I/O scheduler is, however, unnecessary for NVM storage of which random access performance is much better than HDDs. Therefore, it is recommended for NVM storage to use the *noop* mode, which stands for no operation, of the I/O scheduler, in order to minimize the overhead. The noop mode still involves simple queueing.

The synchronous access of NVM storage allows the I/O scheduler to be skipped completely. When skipped, each access request is processed in the order of issuing. By comparing the noop mode of the I/O scheduler and the synchronous access mode, it is possible to reveal the performance impact of I/O request scheduling. We developed a ram disk block device driver that can choose the use the noop mode of the I/O scheduler or the synchronous access mode.

Figure 4 and 5 show the results of file reading and writing with and without the I/O request scheduling, respectively. We used the three file systems, Ext2, Ext4, and Btrfs, for the measurements in order to see the differences of the processing costs. In the figures, queueing means the I/O request scheduling is used.

The results show the significant overheads of the I/O request scheduling. For file reading, Ext2, Ext4, and Btrfs performs 4.69x, 4.75x, and 2.02x slower with the I/O request scheduling than the synchronous access, respectively. For file writing, Ext2, Ext4, and Btrfs performs 3.52x, 3.31x, and 1.48x slower with the I/O request scheduling than the synchronous access, respectively. The results advocate that the syn-

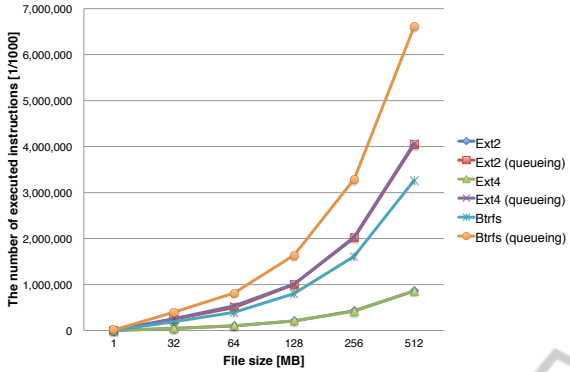

Figure 4: The experiment result of file reading with and without I/O request scheduling.

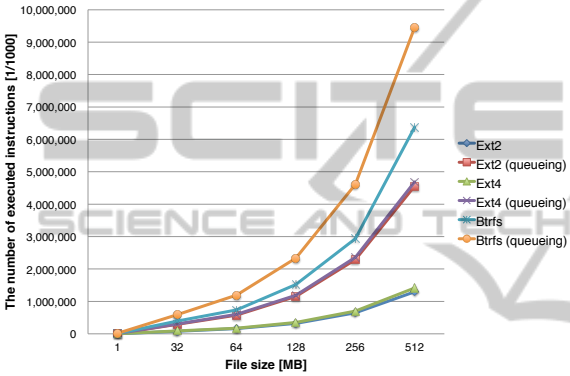

Figure 5: The experiment result of file writing with and without I/O request scheduling.

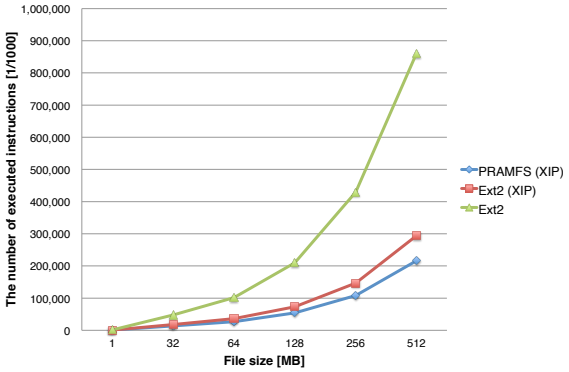

Figure 6: The experiment result of file reading with and without XIP.

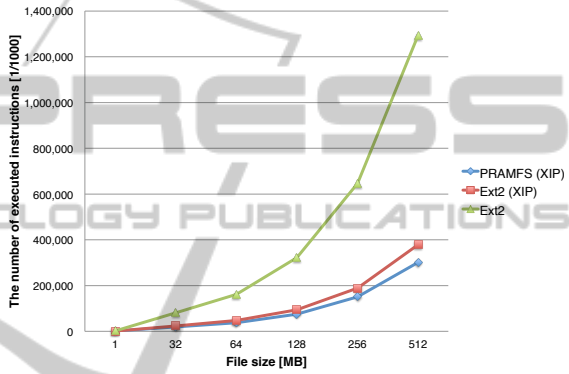

Figure 7: The experiment result of file writing with and without XIP.

chronous access of NVM storage can reduce the I/O processing cost significantly, and the processing cost of the I/O request scheduling can pay off only with slow storage devices.

The results also clearly show the differences of file system performance. Btrfs is the slowest among the three file systems that were used for the measurements although it is the most functionally rich file system. Ext2 is the fastest but the functionally simplest. Ext4 performs comparably to Ext2 with more functionality than Ext2.

4.2 Performance Impact of XIP

The second experiment measures the performance impact of the XIP (Execution-In-Place) feature in order to seek the further performance improvement. The XIP feature is for byte addressable devices, such as ram disks, and enables direct access to blocks without going through a page cache. Since the XIP feature does not require a page cache, there is no need to copy data to a page cache; thus, it incurs less processing cost. The XIP feature requires device drivers to support the direct access interface; thus, we implemented the interface in our ram disk driver.

Figure 6 and 7 show the results of file reading and writing with and without the XIP feature, respectively. We used the two file systems, Ext2 and PRAMFS. We chose Ext2 because it is the fastest file system as shown in the previous experiment. We also chose PRAMFS because PRAMFS is a file system that directly access memory and does not require a device driver. In the figures, XIP means the XIP feature is used.

The results show the XIP feature can accelerate the file access performance even further. For file reading and writing, Ext2 with XIP performs 2.92x and 3.42x faster than without XIP, respectively. PRAMFS is even faster than Ext2 with XIP. It is 1.36x and 1.26x faster than Ext2 for file reading and writing, respectively. The cumulative performance gains by both XIP and the exclusion of I/O request scheduling for file reading and writing are 13.69x and 12.03x, respectively, on Ext2. If further acceleration made possible by PRAMFS is included, the gains become 18.61x and 15.15x, respectively.

From the above experiment results, constructing a file system on NVM storage based main memory en-

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

ables significantly higher performance of file access than using the block device framework by removing the overheads of the framework.

5 RELATED WORK

eNVy (Wu and Zwaenepoel, 1994) proposes NVM storage based main memory. It assumes the utilization of NOR flash memory, which is byte addressable and of which read access latency is comparable to DRAM. Moneta (Caulfield et al., 2010) is a storage array architecture designed for NVM. While its evaluation revealed the necessity of reducing the software costs to deal with block devices, it does not consider the removal of the block device interface. (Yang et al., 2012) also investigated the software costs to deal with block devices on the premise that PCIe gen3 based flash storage devices will become even faster. While it proposed the synchronous interface to block devices, it is still based on the block device interface. (Tanakamaru et al., 2013) proposed a storage state storage device that is a hybrid of flash memory and ReRAM. Their hybrid architecture is similar to the NVM storage based main memory, they do not discuss the interaction with a file system. (Meza et al., 2013) describes the idea to coordinate the management of memory and storage under a single hardware unit in a single address space. They focused energy efficiency, and did not propose any software architecture. (Condit et al., 2009) and SCMFS (Wu and Reddy, 2011) proposed the file systems that were designed to be constructed directly on NVM. They, however, have no consideration of hybrid storage architecture.

6 SUMMARY AND FUTURE WORK

Non-volatile memory (NVM) storage is becoming more popular as its performance and cost efficiency improve. High performance NVM storage can expand to become NVM storage based main memory since polling based access is faster and its hardware can be simplified. This paper proposed constructing a file system directly on NVM storage based main memory in order to overcome its drawbacks when used simply as main memory. The performance projection of the proposed architecture is that accessing files on such a file system can reduce the overhead introduced by handling block devices.

We are currently developing a simulation environment of NVM storage based main memory based on a virtual machine. By employing a virtual machine environment, the simulation of NVM storage and its interface with the DRAM buffer becomes possible. It also enables more precise performance evaluation with realistic benchmark and workload.

REFERENCES

Caulfield, A. M., De, A., Coburn, J., Mollow, T. I., Gupta, R. K., and Swanson, S. (2010). Moneta: A high-performance storage array architecture for next-generation, non-volatile memories. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, pages 385–395, Washington, DC, USA. IEEE Computer Society.

Condit, J., Nightingale, E. B., Frost, C., Ipek, E., Lee, B., Burger, D., and Coetzee, D. (2009). Better i/o through byte-addressable, persistent memory. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, pages 133–146, New York, NY, USA. ACM.

Josephson, W. K., Bongo, L. A., Li, K., and Flynn, D. (2010). Dfs: A file system for virtualized flash storage. *Trans. Storage*, 6(3):14:1–14:25.

Meza, J., Luo, Y., Khan, S., Zhao, J., Xie, Y., and Mutlu, O. (2013). A case for efficient hardware-software cooperative management of storage and memory. In *Proceedings of the 5th Workshop on Energy-Efficient Design (WEED)*, pages 1–7.

Tanakamaru, S., Doi, M., and Takeuchi, K. (2013). Unified solid-state-storage architecture with nand flash memory and reram that tolerates 32x higher ber for big-data applications. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, pages 226–227.

Wu, M. and Zwaenepoel, W. (1994). envy: a non-volatile, main memory storage system. In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS VI, pages 86–97, New York, NY, USA. ACM.

Wu, X. and Reddy, A. L. N. (2011). Scmfs: a file system for storage class memory. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 39:1–39:11, New York, NY, USA. ACM.

Yang, J., Minturn, D. B., and Hady, F. (2012). When poll is better than interrupt. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, pages 1–7, Berkeley, CA, USA. USENIX Association.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS