Journal of ICT, Design, Engineering and Technological Science (JITDETS)
VOL. 3, NO. 1, pp. 20-24, 2019
DOI: https://doi.org/10.33150/JITDETS-3.1.5

ORIGINAL CONTRIBUTION

Acceleration of Storage Performance in Cloud Systems by Using NVRAM

Jisun Kim [1], Yunjoo Park [2], Kyungwoon Cho [3], Hyokyung Bahn [4*]

[1,2,3,4] Department of Computer Science & Engineering, Ewha W. University, Seoul, South Korea

***Abstract*— Non-Volatile Random Access Memory (NVRAM) is anticipated to be utilized as the performance accelerator of future cloud storage systems. This article aims to analyze how large performance improvement can be expected if we use NVRAM as the performance accelerator of legacy storage subsystems. Specifically, we try to find the best employment of NVRAM with respect to storage performances. To do so, we quantify the hot storage area and replace it with NVRAM to obtain the best performances. Our analysis shows that the source of hot storage traffic is not limited to any single particular storage area, but it is different for application types executed. Specifically, journaling accesses dominate in database applications, while swapping accesses dominate in memory-intensive applications. In some applications such as video streaming, file accesses dominate. Based on these findings, we suggest the usage of NVRAM to maximize the performance improvement of cloud storage systems.**

***Index Terms*— NVRAM, Storage, Cloud System, Cloud Application, Storage Accelerator**

Received: 13 April 2019; **Accepted:** 26 May 2019; **Published:** 26 June 2019

I. INTRODUCTION

With the rapid enhancement of NVRAM technologies like PRAM (phase-change random access memory) and STT-RAM (spin-transfer torque random access memory) [1, 2, 3, 4], it is expected that NVRAM will be used in the memory and storage hierarchies of future cloud servers [5, 6, 7]. In this article, we perform empirical analysis to see how much performance improvement can be expected if we supplement NVRAM as an additional component of cloud storage. Our goal is to optimize the usage of NVRAM when it is added to existing storage subsystems. To this end, we first see the storage access characteristics of different types of applications with respect to swapping accesses, journaling accesses, and file accesses. Note that all storage accesses can be classified into one of these 3 types of accesses. Because these 3 access types occur on separated storage areas, we can utilize NVRAM as a swapping area, a journaling area, or a file system area. Due to the high cost of NVRAM, we need to decide which area of the storage subsystems will consist of NVRAM.

The first observation of this article is that a bunch of storage accesses do not happen in the same storage area, but it is different for application types executed. For example, journaling accesses dominate in database applications as transaction handling causes significant write operations to the journaling area in database systems. On the contrary, swapping accesses dominate in memory-intensive applications because the memory size is not sufficient to load the full workload of the application. For multimedia streaming applications, file accesses dominate as these kinds of applications need significant file accesses on multimedia data.

Our second contribution is the investigation of expected performance improvement when we supplement NVRAM to different storage areas. We observed that the largest improvement by using NVRAM does not happen if we use it as a fixed storage area but we can improve performance further by utilizing it adaptively for different kinds of applications. For example, for memory-intensive applications, we can obtain the best performance when we adopt NVRAM as a swapping area. On the contrary, for database applications, the best performance can be obtained by adopting NVRAM as a journaling area. For video streaming applications, adopting NVRAM as a file system area leads to the best performances.

*Corresponding author: Hyokyung Bahn
†Email: bahn@ewha.ac.kr

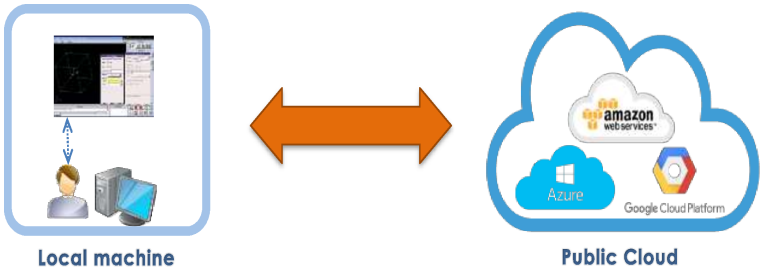

Fig. 1. Accessing cloud storage in a local machine.

Although previous studies on NVRAM have been performed, they focused only on a single usage such as database [8], smartphone [1], journaling [3], swapping [4, 9], or cache media [10, 11]. Unlike these existing works, this article has a novelty in that it considers various types of applications and storage layers, and performs optimized adoption of the given NVRAM storage.

The remaining part of this article is organized as follows. We show the analysis of application's storage access characteristics in Section II. Section III describes how to obtain the best performances by using NVRAM with an appropriate management policy. Section IV briefly summarizes previous studies related to this article. Finally, we conclude this article in Section V.

II. APPLICATION'S STORAGE ACCESS CHARACTERISTICS

Before describing our optimized adoption of NVRAM, we first analyze the characteristics of application's storage accesses. To do so, we extract storage access traces while executing different category of applications. We split our HDD subsystem into three areas, a swapping area, a journaling area, and a file system area, and collect storage access traces separately for observing relative intense of each area. We use the Ext4 file system, and the traces for journaling area are separately collected from the file system area by using the external journaling option. In our trace collection, three application categories are used: a memory-intensive application, a database application, and a multimedia streaming application. Figure 2 shows the number of storage accesses that occur on each storage area for the three applications. As we see from the figure, a bunch of storage accesses do not appear on a single particular area, but it is different for application types.

To be specific, swapping accesses dominate in memory-intensive applications because the main memory size is not enough to load the entire workload of the application. On the contrary, journaling accesses dominate in database applications as database has a sequence of transaction processing steps, which makes a large number of storage writes on the journaling area. In case of the multimedia streaming applications, file accesses dominate as video play services need reading of data from file systems.

III. OPTIMIZED ADOPTION OF NVRAM TO STORAGE

Figure 3 compares the traditional storage architecture with HDD and the proposed architecture consisting of both HDD and NVRAM. As can be seen from the figure, NVRAM is used as a part of the storage subsystem that may be adopted as a swapping area, a journaling area, or a file system area, based on the application's storage access characteristics. Since the size of NVRAM is limited due to the high cost per capacity compared to HDD, we need to decide the area of the storage subsystems that consists of NVRAM. Based on the storage access characteristics observed in Section II, we can predict how large performance improvement can be expected when we use NVRAM as specific storage functions such as file system, swapping, and journaling areas. For example, Figure 4 depicts a case that adopts NVRAM as a file system area located on main memory.

Figure 5 depicts the total storage access latency for executing 3 kinds of applications. As we see from this figure, the best performance by supplementing NVRAM is not resulted when it is fixed as a particular storage area but is different for application types. Specifically, in case of memory-intensive applications, using NVRAM as a swapping area consistently exhibits good results. Note that we use a graph drawing application for the memory-intensive application. In case of database applications, the results contrast for the NVRAM size. In particular, when the size of NVRAM becomes smaller than 30MB, utilizing NVRAM as a file system area exhibits the best performances. In contrast, according as the NVRAM size grows, utilizing NVRAM as a journaling area shows better results, while utilizing NVRAM as a file system area does not exhibit any more improvement. Note that this is a coherent result we observed in Section II. In case of multimedia streaming applications, utilizing NVRAM as a file system area exhibits the best performances. This is also coherent with the trace characterization studies in the previous section.

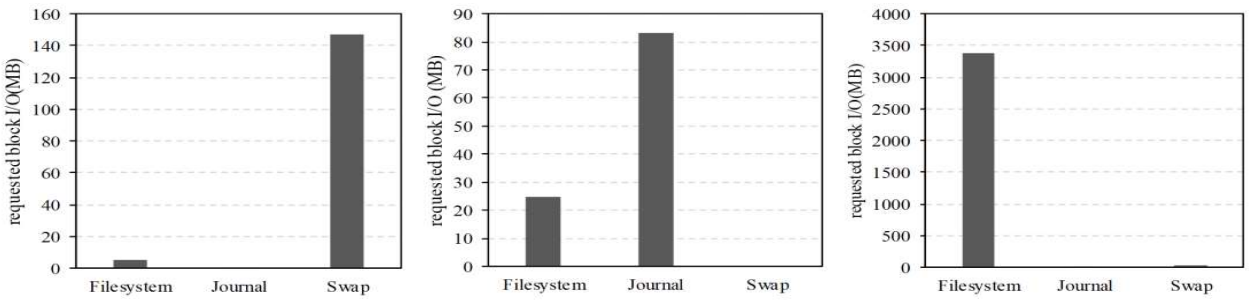

Fig. 2. Storage accesses that occur on each storage area (a) Memory-intensive (b) Database (c) Multimedia streaming

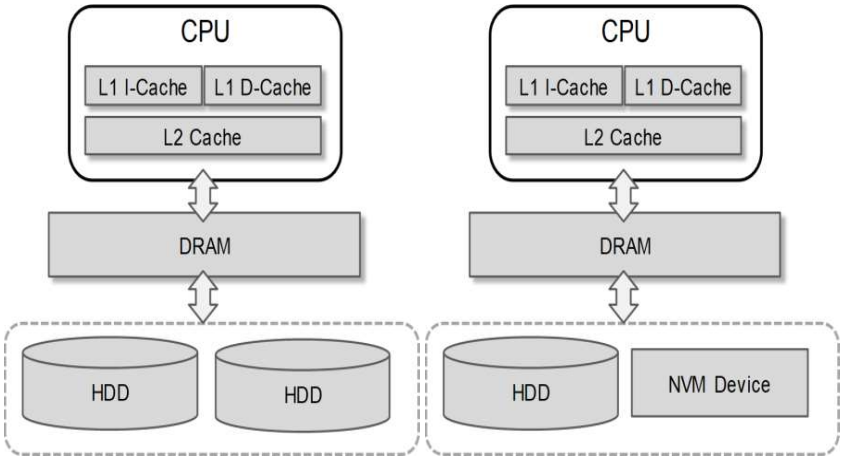

Fig. 3. Storage architecture with NVRAM (a) Traditional storage structure (b) Storage organization with NVRAM

In summary, our conclusion of utilizing NVRAM with the performance results in this section is as follows. For memory-intensive, database, and multimedia streaming applications, we recommend NVRAM to adopt as a swapping, a journaling, and a file system areas, respectively. In case of database applications, however, one can alternatively utilize NVRAM as a file system area or a journaling area based on the size of NVRAM provided.

IV. RELATED WORK

A. File System Buffer Cache and Journaling

To narrow the widening speed gap between main memory and secondary storage, modern operating systems use a file system buffer cache that stores requested file blocks in a certain portion of main memory, thereby servicing subsequent requests without accessing slow storage media. As traditional file system buffer cache uses volatile DRAM, the file system may enter an inconsistent and/or out-of-date state when the system crashes before the change is reflected to permanent storage. To overcome this problem, modern file systems adopt journaling or copy-on-write transaction mechanisms that prevent data corruption via out-of-place updates through periodic flushes.

Instead of writing modified data directly to its original location in the file system, journaling writes the changes to the journal area first and then reflects them to the original location later. In this way, journaling guarantees file system consistency even when the system crashes in the middle of storage updates [12]. Specifically, journaling groups data to be updated atomically and manages them as a single transaction. When a transaction is successfully written to the journal area, a commit mark is placed on the journal area. This mark indicates that the transaction will be reflected to storage even in case of a system crash. The committed data are periodically transferred to their permanent location by the checkpointing operation. Unlike the journaling operation that is performed frequently (e.g., every 5 seconds) to protect data from being corrupted, the interval between checkpointing is relatively long (e.g., 5 minutes).

Journaling guarantees more reliable file system states, but it generates additional storage writes. As a compromise between reliability and performance, journaling file systems often offer different journaling modes such as metadata journaling and full-data journaling. In metadata journaling mode, only metadata is journaled and regular data blocks are directly flushed to its original location. Data corruption is possible in this mode.

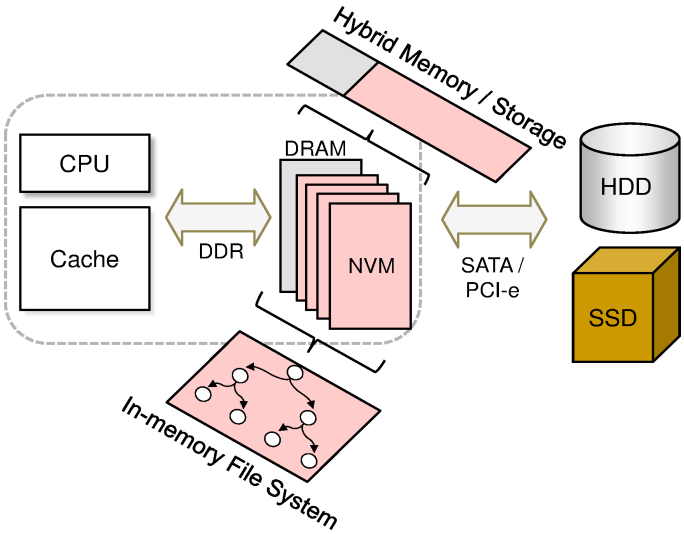

Fig. 4. Using NVRAM as an in-memory file system.

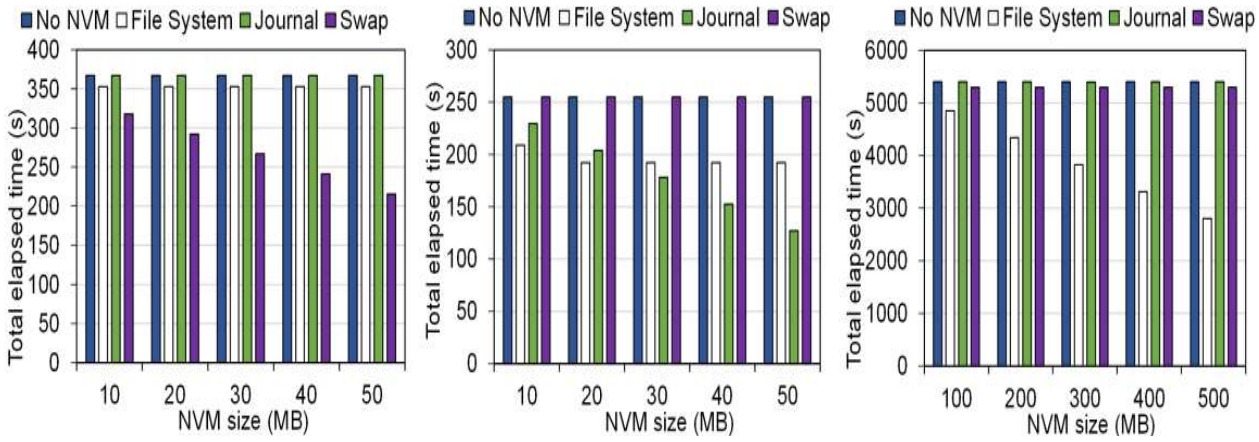

Fig. 5. Total storage access latency as the NVRAM size is varied. (a) Memory-intensive (b) Database (c) Video streaming

In full-data journaling mode, the file system journals both metadata and regular data. This mode guarantees complete consistency of a file system, but incurs a significant performance penalty due to the write-twice behavior of journaling. To balance between performance and reliability, Ext4 and ReiserFS provide the ordered mode where only the metadata is journaled but with regular data flushes preceding metadata journaling so as to reduce the chance of data corruption such as dangling pointers [13, 14].

Another approach for supporting file system consistency is the copy-on-write technique. Copy-on-write creates a copy of data when the data needs to be updated and all modifications are performed on this copy. Thus, the original file data blocks are preserved during modifications. Unlike journaling file systems, copy-on-write file systems do not write modifications back to the original locations but the copies become part of a new file system tree. Generating a new version of a file system tree is performed periodically, and this is also called commit. Similar to commit in journaling file systems, all modifications are handled as a transaction and a commit is completed by generating a new version of a root node. Btrfs and ZFS are examples of copy-on-write file systems of which the default commit period is set to 30 seconds. In copy-on-write file systems, both regular data and metadata are protected [15].

B. Virtualization Structures in Cloud Systems

There are two types of virtualization in cloud systems, full virtualization and para-virtualization, depending on the way guest systems are supported [16]. Full virtualization runs a guest operating system as an independent application on top of the host operating system. This is advantageous in that the host and guest operating systems can be run without modifications. In para-virtualization, the host is equipped only with the hypervisor, which contains minimal interfaces to access hardware, and a guest operating system is modified to properly run on the hypervisor. Para-virtualization can reduce the performance overhead of virtualization, but it requires modification of the guest operating system. VirtualBox [17], VMware [18], and KVM [19] are examples of full virtualization systems, while Xen is a well-known para-virtualization supporting hypervisor [20].

In a fully virtualized system, the guest storage device is usually managed as a single disk image file on the host system. This allows the allocation of the guest's storage capacity upon an actual access, achieving better space utilization. Supporting other functions such as snapshot and migration of guest storage also becomes easier when managing storage as a virtual disk file rather than a raw disk. Both the guest and the host have their own file systems and buffer caches. The host regards each guest as a user application and considers the buffer cache and the file system of the guest as user memory and a file, respectively. Upon request of a storage access, the guest first searches its own buffer cache and then, the hypervisor sends the request to the host via a system call. Then, the host checks its buffer cache and finally, the request is delivered to storage. As host and guest machines manage their own buffer caches, duplicated caching may degrade space efficiency. However, with a large host cache that acts as the second-level cache, guests can expect to reap performance benefits. Specifically, if many virtual machines, whose lifetime is difficult to estimate coexist, then managing a large shared buffer cache on the host side can be much more effective than allocating a large cache space for each individual guest a priori [21]. Due to this reason, while there are options to support host cache bypassing, hypervisors, by default, generally select to use host caching [21, 17].

V. CONCLUSIONS AND IMPLICATIONS

In this article, we characterized the storage access traces in order to determine the appropriate adoption of NVRAM for future storage subsystems. Our finding is that a bunch of storage accesses do not occur on a common single storage area, but it is different for the types of applications executed. Specifically, journaling accesses dominate in database applications, while swapping accesses dominate in memory-intensive applications. On the other hand, in multimedia streaming applications, file accesses dominate. With these findings, we suggested the usage of NVRAM for maximizing the performance improvement of future storage subsystem design. Recent patents on NVRAM described various types of storage organizations with NVRAM, meaning that NVRAM-based storage subsystems are imminent [22, 23] The limitation of this article is that we do not provide the automatic adoption of NVRAM for the target storage layer. For now, the system administrator needs to configure NVRAM for the target layer determined, and we will further study on this direction. We hope that our study will be helpful in designing future storage subsystems of cloud servers.

VI. ACKNOWLEDGMENT

This article was supported by the ICT R&D program of MSIP/IITP (2019-0-00074, developing system software technologies for emerging new memory that adaptively learn workload characteristics) and also by the Basic Science Research Program through the NRF grant funded by Korea Government (MSIP) (No. 2019R1A2C1009275).

Declaration of Competing Interest

The authors declare that there is no conflict of interest.

References

[1] Y. Park and H. Bahn, ``Challenges in memory subsystem design for future smartphone systems,'' in *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, 2017, pp. 255-260.

[2] S. Yoo and H. Bahn, ``An efficient page replacement algorithm for pcm-based mobile embedded systems,'' in *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. IEEE, 2016, pp. 183-188.

[3] E. Lee, H. Kang, H. Bahn, and K. G. Shin, ``Eliminating periodic flush overhead of file i/o with non-volatile buffer cache,'' *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1145-1157, 2014. doi: https://doi.org/10.1109/TC.2014.2349525

[4] Y. Park and H. Bahn, ``A working-set sensitive page replacement policy for pcm-based swap systems,'' *Journal of Semiconductor Technology and Science*, vol. 17, no. 1, pp. 7-14, 2017.

[5] J. Kim and H. Bahn, ``Optimized adoption of nvm storage by considering workload characteristics,'' *JSTS: Journal of Semiconductor Technology and Science*, vol. 17, no. 1, pp. 1-6, 2017. doi: https://doi.org/10.5573/JSTS.2017.17.1.001

[6] E. Lee, J. Kim, H. Bahn, S. Lee, and S. H. Noh, ``Reducing write amplification of flash storage through cooperative data management with nvm,'' *ACM Transactions on Storage (TOS)*, vol. 13, no. 2, p. 12, 2017. doi: https://doi.org/10.1145/3060146

[7] Y. Park and H. Bahn, ``Management of virtual memory systems under high performance PCM-based swap devices,'' in *39th Annual Computer Software and Applications Conference*, vol. 2, 2015, pp. 764-772.

[8] A. van Renen, V. Leis, A. Kemper, T. Neumann, T. Hashida, K. Oe, Y. Doi, L. Harada, and M. Sato, ``Managing non-volatile memory in database systems,'' in *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018, pp. 1541-1555.

[9] D. Liu, K. Zhong, X. Zhu, Y. Li, L. Long, and Z. Shao, ``Non-volatile memory based page swapping for building high-performance mobile devices,'' *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1918-1931, 2017. doi: https://doi.org/10.1109/TC.2017.2711620

[10] E. Cheshmikhani, H. Farbeh, S. G. Miremadi, and H. Asadi, ``Ta-lrw: a replacement policy for error rate reduction in stt-mram caches,'' *IEEE Transactions on Computers*, vol. 68, no. 3, pp. 455-470, 2018. doi: https://doi.org/10.1109/TC.2018.2875439

[11] H. Farbeh, A. M. H. Monazzah, E. Aliagha, and E. Cheshmikhani, ``A-cache: Alternating cache allocation to conduct higher endurance in nvm-based caches,'' *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2018. doi: https://doi.org/10.1109/TCSII.2018.2881175

[12] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, ``Analysis and evolution of journaling file systems.'' in *USENIX Annual Technical Conference, General Track*, vol. 194, 2005, pp. 196-215.

[13] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, ``The new ext4 filesystem: current status and future plans,'' in *Proceedings of the Linux symposium*, vol. 2, 2007, pp. 21-33.

[14] R. H and ReiserFS. (2007) The best linux blogging software. [Online]. Available: https://bit.ly/2ZsDhiM

[15] Btrfs. Btrfs design. [Online]. Available: https://bit.ly/2ZgzzO2

[16] S. Hajnoczi, ``An updated overview of the qemu storage stack,'' *LinuxCon Japan*, 2011. doi: https://doi.org/https://bit.ly/2NAlF2n

[17] Virtual box. [Online]. Available: https://bit.ly/1mQkaJC

[18] VMWare. Vmware tools for linux guests. [Online]. Available: https://bit.ly/2PeZ9ZM

[19] KVM. Kernal virtual machine. [Online]. Available: https://bit.ly/2I6Jc4n

[20] Xen Source. Progressive paravirtualization. [Online]. Available: https://bit.ly/30BFl9L

[21] K. Wolf, ``A block layer overview,'' in *KVM Forum*, 2012.

[22] H. Bahn, E. Lee, and S. H. Noh, ``Device and method for integrated data management for nonvolatile buffer cache and nonvolatile storage,'' Google Patents, US Patent 9,563,566, 2017.

[23] E. Lee, H. Bahn, and S. H. Noh, ``Buffer cache apparatus, journaling file system and journaling method for incorporating journaling features within non-volatile buffer cache,'' Google Patents, US Patent App. 13/862,597, 2014.