International Journal of Natural Sciences Research
2016 Vol. 4, No. 6, pp. 100-106
ISSN(e): 2311-4746
ISSN(p): 2311-7435
DOI: 10.18488/journal.63/2016.4.6/63.6.100.106

IMPROVING CLOUD SYSTEM PERFORMANCES BY ADOPTING NVRAM-BASED STORAGE SYSTEMS

Jisun Kim[1] --- Yunjoo Park[2] --- Sunhwa A. Nam[3] --- Hyunkyoung Choi[4] --- KyungWoon Cho[5] --- Hyokyung Bahn[6]†
[1,2,3,4,5,6]Ewha University, Seoul, Republic of Korea

ABSTRACT

NVRAM is being considered as an additional memory/storage component of future cloud computing systems. This paper investigates how much performance improvement can be obtained if we add NVRAM as the memory/storage media of cloud systems. As NVRAM is put on DDR slots, it is byte-accessible and hence can be used as a memory medium like DRAM. It can also be utilized as swap or journal devices if we use it as a block I/O device. We first consider NVRAM as a storage cache, and then, we measure the performance of systems that additionally use NVRAM as memory, swap, and journal devices. We use two workloads, I/O and memory intensive workloads. Our experiments show that using NVRAM as a journal device performs the best in I/O-intensive workload as it performs journaling I/O on NVRAM instead of slow storage. Using NVRAM as memory or swap devices does not show good results in I/O-intensive workload. However, in case of memory-intensive workload, NVRAM memory significantly improves the performance, and NVRAM swap also gains a certain level of improvement. We expect that our experiments will be helpful in the design of NVRAM-based cloud systems for memory or I/O intensive workload situations.

Keywords: Cloud system, NVRAM, Storage, Journaling, Swap.

Received: 8 November 2016/ **Revised:** 15 December 2016/ **Accepted:** 19 December 2016/ **Published:** 23 December 2016

Contribution/ Originality

The paper's primary contribution is finding the NVRAM's effectiveness on the performance of future cloud systems. Specifically, we show the effectiveness of NVRAM if it is used as storage cache, main memory, journal device, or swap device in the memory hierarchy of cloud computing systems.

1. INTRODUCTION

Due to the recent advances in semiconductor technologies, byte-accessible NVRAMs (nonvolatile RAMs) such as MRAM (magnetic RAM), PRAM (phasechange RAM), and FeRAM (ferro electro RAM) are emerging rapidly [1-3]. NVRAM is being considered as an additional memory/storage component of future cloud computing systems. This paper investigates how much performance improvement can be obtained if we add NVRAM as the memory/storage media of cloud systems. As NVRAM is put on DDR slots, it is byte-accessible and hence can be utilized as a memory medium like DRAM. However, if we want to use NVRAM as a swap or a journal device, it should be recognized as a block I/O device. Thus, we develop an NVRAM device driver based on the existing Ramdisk driver.

We first consider NVRAM as an additional storage cache and show how much performance improvement can be obtained if we adopt NVRAM cache. Then, we consider NVRAM as a fast storage medium as well as main

† Corresponding author

memory medium and measure the performance of the original system that uses DRAM memory and HDD storage, and new systems that additionally use NVRAM as memory, swap, and journal devices. We use two workloads, IOzone and a memory intensive workload. Our experimental results show that using NVRAM as a journal device performs the best in case of IOzone as it performs journaling I/O on NVRAM instead of slow storage. Using NVRAM as a memory or a swap device does not exhibit such good results as they have the effect of extending memory capacity but IOzone is an I/O-intensive workload. Using NVRAM as memory performs slightly better than that of a swap due to the buffering effect of I/O. However, in case of memory-intensive workload, NVRAM memory significantly improves the performance of the system as it extends the effective memory capacity. NVRAM swap also gains a certain level of improvement. Though the memory size itself is not extended, NVRAM swap performs well in memory-intensive workloads as it provides a high performance swap device. We expect that our experiments will be helpful in the design of NVRAM-based cloud systems for memory or I/O intensive workload situations.

The remainder of the paper is organized as follows. We present the formal definition of NVRAM caching problems in Section 2. Then, we explain an NVRAM storage cache architecture and present a new cache replacement algorithm on this architecture in Section 3. Section 4 shows experimental results obtained through trace-driven simulations to assess the effectiveness of the proposed scheme. Section 5 describes the performance results when we use NVRAM as additional memory or storage devices. Then, finally we conclude this paper in Section 6.

2. NVRAM CACHING PROBLEMS

Before mentioning NVRAM caching, we revisit the conventional caching problems. Let us suppose that S is the size of cache, r the total number of accesses, and h the number of in-cache accesses. Then, the cache management module needs to cache the currently requested block without exceeding S. In this article, we focus on non-lookahead, on-demand fetching algorithms. If the number of blocks in the cache exceeds the size of S, the replacement algorithm selects an eviction victim and removes it from the cache. The eventual goal of the algorithm is to maximize the number of blocks accessed directly from the cache after all the requests have been processed. The hit ratio, calculated by h/r, is a well-known performance measure to evaluate the performance of the replacement algorithm. In the conventional volatile cache structures, Belady's OPT algorithm is known to be optimal in terms of the hit ratio [4]. The OPT algorithm removes the block that will be accessed furthest in the future. OPT is not a practical solution as we cannot know future accesses in real systems. However, OPT provides the upper bound of performance to the system designers pursuing good practical algorithms.

Now, let us see the NVRAM caching problems. Suppose that both volatile and nonvolatile caches are used together. We also assume that all writes are performed in nonvolatile cache. Hence, reliability is always guaranteed. Writes to secondary storage occurs only when a block is removed from the nonvolatile cache. Also, we assume that all clean blocks reside in volatile cache. This assumption may be released in real systems, but we include it in theoretical analysis by following previous studies [5].

Suppose that S_{NV} is the size of nonvolatile cache, S_V is the size of volatile cache, r is the total number of read accesses, w is the total number of write accesses, and h is the number of in-cache accesses. Unlike conventional caching systems, it is known that the hit ratio, i.e., h/(r+w), is not a good performance measure for nonvolatile caching structure. Instead, the performance measure needs to be changed to the number of storage accesses [5]. The reason is that even a hit may incur storage access operations. For example, if a write access arrives and the accessed block resides in the volatile cache but not in the nonvolatile cache, it is apparently a cache hit. However, this incurs a write operation to nonvolatile cache. If there is no available space in the nonvolatile cache, the cache management module needs to remove a dirty block from the nonvolatile cache, which essentially causes a storage access operation.

The problem is then to reduce the total number of storage accesses. For read operations, blocks in both volatile and nonvolatile caches can be serviced directly. However, for write operations, the system should only use nonvolatile cache for the reliability reason. Thus, if a write request happens and the requested block does not exist in the nonvolatile cache, the replacement algorithm needs to make a space in the nonvolatile cache for this request. If the number of blocks in the nonvolatile cache is larger than S_{NV}, the replacement algorithm selects an eviction target and removes it, which eventually causes a write operation to storage. However, if a read request happens, a cache hit from either volatile or nonvolatile cache does not cause storage accesses.

Due to these changed situations, the optimality of the replacement algorithm is also defined differently for nonvolatile cache structures. An optimal algorithm, that we call NV-MIN, behaves as follows. Note that we do not provide the formal proof of the optimality, as it can be shown intuitively. If a read miss happens, NV-MIN selects a replacement block by the following procedures. If there is an empty space in volatile cache or if there is a space of which the next access is write in volatile cache, NV-MIN stores the new block here. Otherwise, NV-MIN evicts the block that will be read-accessed furthest in the future among those in volatile cache.

If a write miss occurs in nonvolatile cache, NV-MIN acts as follows. Note that this includes the case that a write-accessed block already exists in volatile cache but not in nonvolatile cache. If there exists an empty slot in nonvolatile cache, NV-MIN stores the requested block in this space. (If the block already exists in volatile cache, NV-MIN evicts it from volatile cache.) Otherwise, NV-MIN removes the block that will be write-accessed furthest in the future and writes it to the storage. If the evicted block will be read-accessed again in the future, it may be copied to volatile cache according to the following procedures. If there exists an empty space in volatile cache or if there is a block whose next request is write in volatile cache, NV-MIN stores the removed block to this space. Otherwise, NV-MIN compares the two blocks, namely the block that will be read-accessed furthest in the future among those in volatile cache and the removed block from nonvolatile cache, and replaces the block that will be read-accessed further in the future.

3. NVRAM CACHING ALGORITHMS

The storage and cache systems we consider are depicted in Fig. 1. Storage systems are composed of the secondary storage systems and two kinds of caches, namely volatile cache and nonvolatile cache. The secondary storage system is basically composed of hard disks, but NAND flash memory or other storage media can be adopted. The volatile cache consists of DRAM, and the nonvolatile cache consists of NVRAM such as PCM, ReRAM, or STT-MRAM. All writes requests are performed in nonvolatile cache. Thus, all dirty blocks reside in nonvolatile cache and hence reliability is always guaranteed. Writes to secondary storage occurs only when a removal from nonvolatile cache happens.

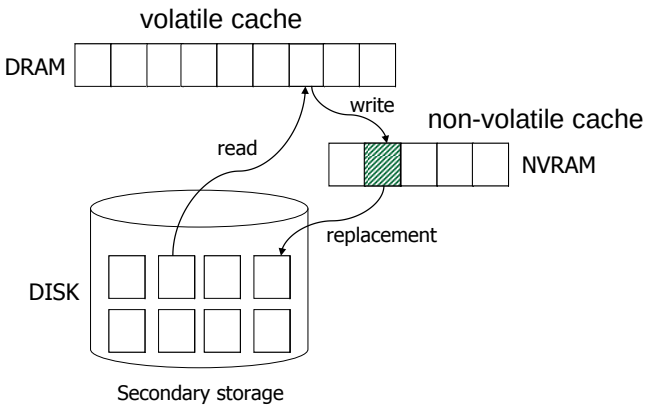


Fig-1. System structure of nonvolatile cache

Nonvolatile cache utilizes the read history and the write history of block accesses separately to predict future accesses precisely. To this end, nonvolatile cache uses two lists in order to maintain the recency history of accesses, namely the LRR (least recently read) list and the LRW (least recently written) list. Note that blocks in these two lists are not required to be identical to those blocks in the volatile and the nonvolatile caches, respectively. This separated management of metadata and actual data allows more efficient management of volatile and nonvolatile cache spaces. For example, if a block is recently read and also written, the metadata of the block can exist in both LRR and LRW lists, but actual data is only maintained in the nonvolatile cache. Hence, some blocks in the LRR list may not exist in the volatile cache. This enables volatile cache to preserve more blocks, leading to more read-hits. Nevertheless, we maintain the read history of the block in the LRR list as the block may return to the volatile cache if it is evicted from the nonvolatile cache.

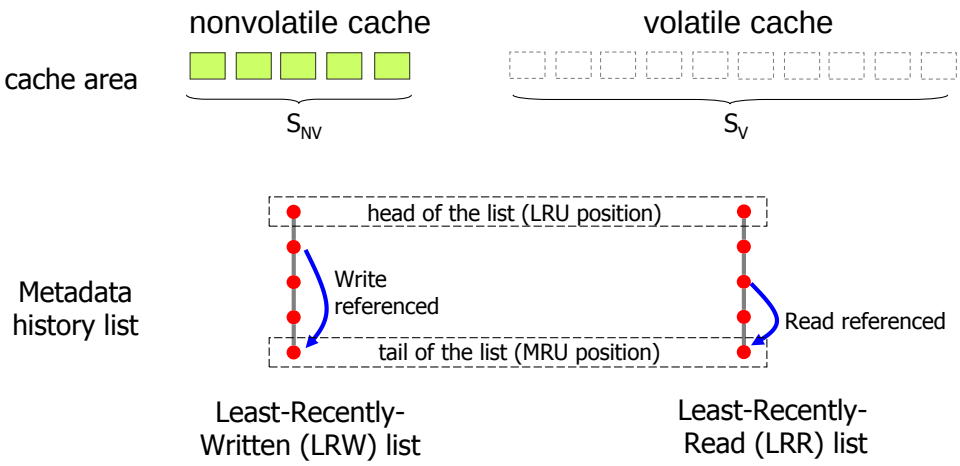


Fig-2. Data structures used in the nonvolatile caching algorithm.

4. PERFORMANCE EVALUATION OF NVRAM CACHING

We conducted trace-driven simulations to evaluate the effectiveness of cache replacement algorithms with respect to the total number of storage accesses. We used traces collected by Roselli, et al. [6] from the Hewlett-Packard series 700 workstations running HP-UX. These traces are classified as three categories, namely INS (instructional workload), RES (research workload), and WEB (Web server workload). For a comparison purpose, we designed and implemented NVLRU, which is a modified version of the conventional LRU algorithm. NVLRU holds dirty blocks in NVRAM, preserving file system consistency without periodical flush. Whenever it needs to store a new dirty block but there is no free space in NVRAM, it chooses and removes the least recently used dirty block from NVRAM.

Fig. 3 shows the experimental results of our NVRAM block management (NBM) algorithm with respect to the total number of storage access operations normalized by NVLRU. NBM performs better than NVLRU specially when NVRAM load is high. This is due to the major drawback of the NVLRU algorithm. NVLRU maintains dirty blocks that are recently read but rarely written in NVRAM as it maintains an LRU list based on not only write histories but also read histories for dirty blocks in NVRAM. In contrast, NBM recognizes dirty but rarely written blocks efficiently using an LRW list which keeps track of write histories only.

Furthermore, NBM also decreases total storage access counts when the volatile cache space is sufficiently large. As explained earlier, for every NVRAM replacement victim, NBM decides to move the block from nonvolatile cache to volatile cache if it is recently read. This helps the system cache to hold all the recently read blocks while consuming less NVRAM space. However, if volatile cache is under heavier memory pressure than nonvolatile cache, moving a block from nonvolatile to volatile caches may cause worse read hit ratio. NBM exhibits little performance

improvement for workloads that rarely reads from dirty blocks. For such workloads, using LRU or LRW for NVRAM blocks makes no difference. Hence, NBM and NVLRU work similarly. This can be seen in Fig. 3 (c).

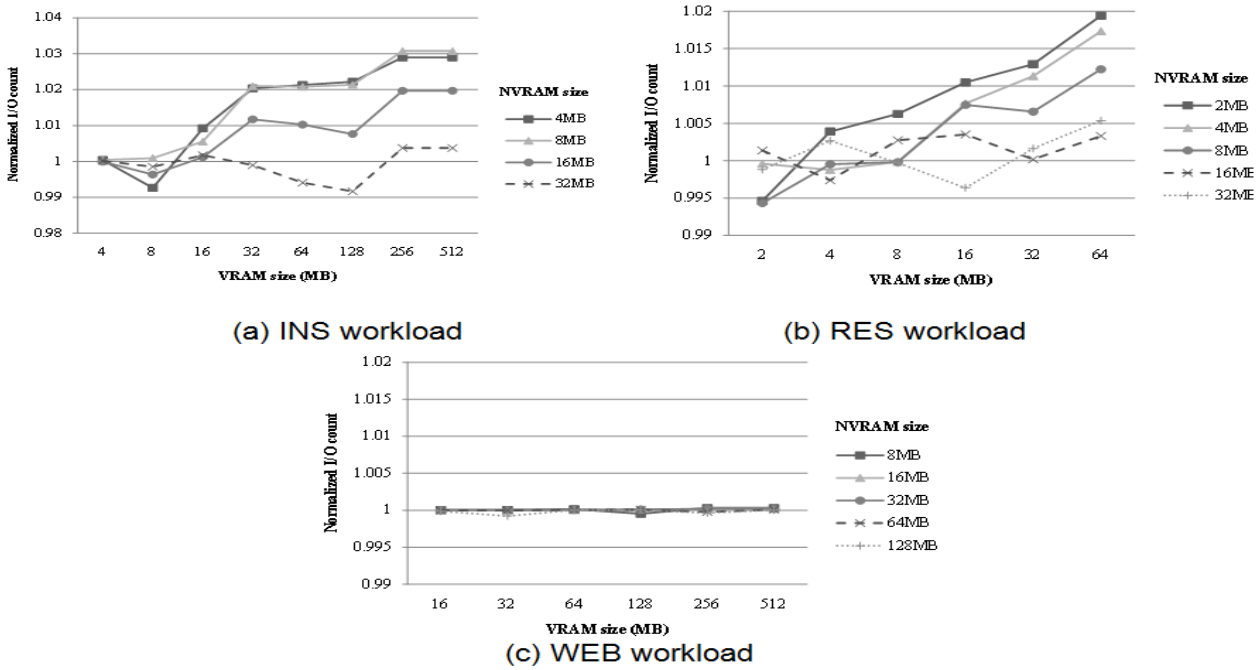
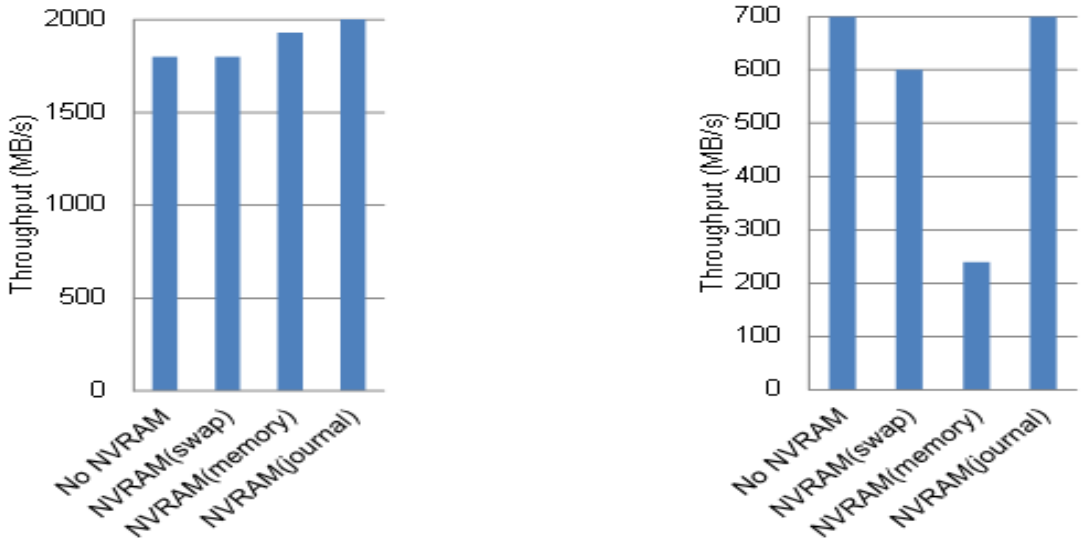


Fig-3. Performance of NBM with respect to total storage accesses normalized by NVLRU varying volatile and nonvolatile cache sizes.

5. USING NVRAM AS MEMORY/STORAGE DEVICES

In this section, we perform measurement studies to investigate how much performance gain can be obtained if we add NVRAM as a memory device or a storage device, specially focusing on a swap or a journal device. Our experiments are performed on Linux kernel 3.16.0 and Ext4. As commercially available NVRAM hardware is limited, we emulate it by making use of DRAM on DIMM slots with appropriate timing delays. As we want to use NVRAM as a swap or a journal device, it should be recognized as a block I/O device. Thus, we develop an NVRAM device driver based on the existing Ramdisk driver.

We measure the performance of the original system that uses DRAM only and new systems that additionally use NVRAM as memory, swap, and journal devices. We use two workloads: IOzone and Memzone for I/O and memory intensive workloads, respectively. IOzone generates a series of I/O accesses whereas Memzone increasingly allocates a certain range of memory space to the program. Fig. 4(a) shows the measured throughput of IOzone with four architectures: DRAM only, NVRAM(memory), NVRAM(swap), and NVRAM (journal). As shown in the figure, NVRAM(journal) performs the best as it performs journaling I/O on NVRAM instead of slow secondary storage. NVRAM(memory) and NVRAM(swap) do not exhibit such good results as they have the effect of extending memory capacity but IOzone is an I/O-intensive workload. NVRAM(memory) performs slightly better than NVRAM(swap) due to the buffering effect of I/O. Fig. 4(b) shows the measured execution time when Memzone is run. The results show that NVRAM(memory) significantly improves the performance of the system as it extends the effective memory capacity. NVRAM (swap) also gains a certain level of improvement. Though the memory size itself is not extended, NVRAM(swap) performs well in memory-intensive workloads as it provides a high performance swap device. The performance improvement of NVRAM(memory) and NVRAM(swap) over the original system is 65.6% and 15.2%, respectively.



(a) IOzone results (b) Memzone results

Fig-4. Performance measurement using NVRAM as additional memory, swap, or journal devices.

6. CONCLUSIONS

This paper investigated how much performance gain can be obtained if we add NVRAM as the memory/storage media of cloud systems. We first considered NVRAM as an additional cache and showed how much performance gain can be obtained if we use NVRAM cache. Then, we considered NVRAM as a fast storage medium as well as main memory medium and measured the performance of the original system that uses DRAM memory and HDD storage, and new systems that additionally use NVRAM as memory, swap, and journal devices. We used two workloads, IOzone and a memory intensive workload. Our experimental results showed that using NVRAM as a journal device performs the best in case of IOzone as it performs journaling I/O on NVRAM instead of slow storage. Using NVRAM as a memory or a swap device did not exhibit such good results as they have the effect of extending memory capacity but IOzone is an I/O-intensive workload. However, in case of memory-intensive workload, The results showed that NVRAM memory significantly improves the performance of the system as it extends the effective memory capacity. NVRAM swap also gained a certain level of improvement. We expect that our preliminary experiments will be helpful in the design of NVRAM-based cloud systems for memory or I/O intensive workload situations.

Funding: This work was supported by the ICT R&D program of MSIP/IITP (R-20160904-004151, Research on Autoscaling and Storage for High Performance Computing on Hybrid Cloud) and also by the Basic Science Research program through the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2016R1A2B4015750).

Competing Interests: The authors declare that they have no competing interests.

Contributors/Acknowledgement: All authors contributed equally to the conception and design of the study.

REFERENCES

[1] E. Lee, J. Jang, T. Kim, and H. Bahn, "On-demand snapshot: An efficient versioning file system for phase-change memory," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, pp. 2841-2853, 2013.

[2] E. Lee, S. Yoo, and H. Bahn, "Design and implementation of a journaling file system for phase-change memory," *IEEE Transactions on Computers*, vol. 64, pp. 1349-1360, 2015.

[3] E. Lee and H. Bahn, "Caching strategies for high performance storage media," *ACM Transactions on Storage*, vol. 10, pp.1-22, 2014.

[4] L. Belady, "A study of replacement of algorithms for a virtual storage computer," *IBM Systems Journal*, vol. 5, pp. 78–101, 1996.

[5] K. Lee, I. Doh, J. Choi, D. Lee, and S. H. Noh, "Write-aware buffer cache management scheme for nonvolatile RAM," presented at the IASTED International Conference: Advances in Computer Science and Technology, 2007.

[6] D. Roselli, J. R. Lorch, and T. E. Anderson, "A comparison of file system workloads," presented at the USENIX Annual Technical Conference, 2000.