WRITE-AWARE BUFFER CACHE MANAGEMENT SCHEME FOR NONVOLATILE RAM

Kyu Hyung Lee[*], In Hwan Doh[*], Jongmoo Choi[**], Donghee Lee[+], Sam H. Noh[*]
[*]Department of Computer Engineering, Hongik University, 72-1 Sangsu-dong, Mapo-gu, Seoul
[**]Division of Information and Computer science, Dankook University, San 8, Hannam-Dong, Yongsan-gu, Seoul
[+]School of Computer Science, University of Seoul, Jeonnong 3-dong, Dongdaemun-gu, Seoul
Republic of Korea
[*]{lkh, ihdoh, samhnoh}@cs.hongik.ac.kr, [**]choijm@dankook.ac.kr, [+]dhlee@venus.uos.ac.kr

ABSTRACT
Nonvolatile RAM (NVRAM) technology is advancing rapidly with 1-2Mb capacity single-chip prototypes becoming available from major semiconductor companies. We will soon see NVRAM become an everyday component of our commodity computers. This paper explores the use of NVRAM as part of the buffer cache. A nonvolatile buffer cache provides a computer system with a means to maintain complete consistency as well as improved performance. The results of this paper can be summarized as follows. First, we show that the hit ratio that has been a commonly used metric to measure buffer cache performance is no longer adequate for caches with NVRAM. Instead of the hit ratio, we need to count the number of disk accesses to assess user perceived cache performance. Second, we show that because of this change in performance metric, when managing a buffer cache with NVRAM, one can do better than when using the MIN replacement algorithm mainly by distinguishing read and write operations. With this, we show that there is room for improvement in efficiently handling caches with NVRAM. Finally, based on these findings, we propose a simple and practical buffer management technique that improves on using the LRU algorithm.

KEY WORDS
Computer Architecture, NVRAM (Non Volatile RAM), Buffer Cache, Hit Ratio, Performance Metric

1. Introduction

Nonvolatile RAM (henceforth, referred to as NVRAM) technology is advancing rapidly with 1-2Mb capacity single-chip prototypes becoming available from major semiconductor companies. Yet, there have only been a limited research interest in relation to NVRAM.

Assuming the computer system is equipped with NVRAM, this paper focuses on using NVRAM as part of the buffer cache. In this regard, we make three contributions. First, we show that the hit ratio that has been a commonly used metric to measure buffer cache performance is no longer adequate for caches with NVRAM. Instead of the hit ratio, we need to count the number of disk accesses to assess cache performance. Second, we show that because of this change in performance metric, when managing a buffer cache with NVRAM, one can do better than when using the MIN replacement algorithm mainly by distinguishing read and write operations. With this, we show that there is room for improvement in efficiently handling caches with NVRAM. Finally, based on these findings, we propose a practical buffer management technique that will improve the performance of conventional practical policies.

In the remainder of this section, we shortly describe the current changes that are happening in regards to NVRAM. Then, we review some of the related works that have considered NVRAM as a system component. As there are many facets in improving performance with the use of NVRAM, we clearly state the assumptions that we make in this study. In Section 2, we describe why the hit ratio is no longer the appropriate performance metric when using NVRAM cache. We then show that the MIN algorithm can be improved when the performance metric of interest changes. Based on these results, in Section 3, we quantify the benefit that is obtainable in a practical replacement algorithm, specifically, LRU. We then propose a scheme that is implementable in real life. Finally, in Section 4, we give a summary of the work and end with conclusions.

1.1 NVRAM, Related Works, and Assumptions

Studies in using NVRAM are not new. Research results on this issue were published in the early 1990's. However, NVRAM envisioned at the time was RAM with battery support. Hence, the use of NVRAM was not considered common.

Today, it is a different story. NVRAM is being realized in many forms. MRAM (Magnetic RAM), PRAM (Phase-change RAM), FeRAM (Ferro electro RAM) are some of the more commonly talked-about NVRAM **[10]**. They are being developed by major semiconductor companies such Texas Instruments, IBM, Samsung, Fujitsu, Motorola, etc. 1-2Mb chips are already being sold, and these chips are, supposedly, fully compatible with SRAM, meaning that installing them to everyday computer systems should not be a difficult task (though not to be done by a layman as if installing a new program) **[11]**. As semiconductor

technology continue to make progress we will soon see NVRAM become an everyday component of our commodity computers. However, research on what effects NVRAM has on computer systems and software and how to make efficient use of NVRAM have been limited.

Specifically, previous research on software issues related to NVRAM can be viewed as being directed in two directions. One is using NVRAM as an extension of storage and thus, maintaining metadata in this part of storage. Miller et al. introduces the HeRMES file system that makes use of MRAM to store metadata, while storing file data in disk **[6]**. Another file system, MRAMFS, uses a similar approach as HeRMES, but it utilizes compression on inodes in order to save NVRAM space as the authors assume that NVRAM is a scarce resource **[4]**. Finally, Conquest is another file system being developed with NVRAM in mind **[7]**. Conquest considers storing not only metadata, but also small sized files while leaving large files on disk.

The other direction of research with NVRAM considers one specific aspect of the file system, that is, the buffer cache. Since real systems generally tend to use the copy-back policy for writes due to performance reasons, there is always a window of time in which consistency of the file system may be compromised. By making writes to NVRAM, this window of consistency loss can be removed, and consistency can be maintained in full without compromising performance. It was shown by Baker et al. that write traffic can be significantly reduced with the help of NVRAM **[2]**. For management of the buffer space, they compare the LRU and random replacement algorithms and show that the two schemes show little difference in reducing write traffic to disk. Haining and Long propose algorithms for NVRAM write buffer management; specifically, LRU, shortest access time first (STF), and largest segment per track (LST) **[5]**. They not only consider the issue of replacement, but also consider the issue of staging, that is, when to clean the dirty blocks in cache. They report that in most cases LRU is most effective. Finally, Akyurek and Salem perform an extensive simulation study on managing NVRAM buffers **[1]**. They propose and categorize policies based on the actions taken upon a read miss and on write allocation.

The topic of this paper is also buffer management. To set the stage of our discussion we need to make clear the assumptions in which we are presenting the work. We will also present in more detail the LRU Volatile and LRU Global algorithms proposed by Akyuket and Salem as they are used to present subsequent results.

To ease the discussion, we use the following terminology throughout. The traditional form of buffer cache, that is, one with all volatile memory is called a volatile cache. A buffer cache where volatile memory and nonvolatile memory coexist, will be referred to as a nonvolatile cache. The volatile memory part of a cache is referred to as volatile space and the nonvolatile memory part is referred to as nonvolatile space.

The assumptions we make about the buffer is as follows.

1. [Write Nonvolatile Assumption] We assume that all writes are done in NVRAM only. Hence, consistency is always guaranteed. Writes to disk happens only when an eviction from NVRAM occurs. This is in line with the 'Update Rule' discussed by Akyurek and Salem **[1]**.
2. [Clean Separation Assumption] With nonvolatile cache, all dirty blocks reside in nonvolatile space and all clean blocks reside in volatile space only. Management of the two spaces is done separately and independently.
3. [No Staging Assumption] No form of staging is happening, that is, there is no extra effort to stage the dirty blocks. Hence, a dirty block is evicted to disk only when nonvolatile space overflows.

Let us now discuss the LRU Volatile and LRU Global algorithms. These algorithms are employed upon a read miss. (Henceforth, for simplicity, we will denote LRU Volatile and LRU Global as LRU-V and LRU-G, respectively.) With LRU-V, when a read miss occurs, the victim block to be evicted is the LRU block found in volatile space. Thus, the incoming block will reside in volatile space as well. With LRU-G, when a read miss occurs, the victim block is the global LRU block, that is, the LRU block among all blocks residing in volatile and nonvolatile space. Hence, note that the incoming (clean) block may reside in nonvolatile space if the global LRU block was found in nonvolatile space. This is in violation of the 'Clean Separation Assumption'. This is the only case where any of our assumptions will be violated, and we use LRU-G only to give motivation to this work.

1.2 Experimental Setup of the Study

Figure 1 depicts the experimental setup used in this study.

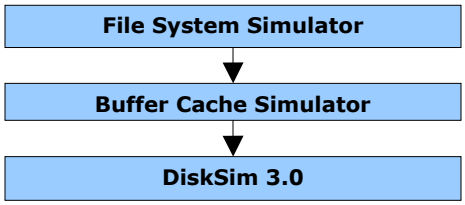


Figure 1: Organization of the simulation environment used to obtain the experimental results.

The simulator is implemented in three layers. The top layer is the file system simulator that simulates the System V UNIX file system. The buffer cache simulator layer runs separately from the file system layer. Various forms of buffer caches are implemented at this level to compare their effects. To consider user perceived response time performance, the DiskSim disk simulator layer is used. DiskSim 3.0, which is the current most version publicly available, is used **[9]**. We use parameters for the Seagate Cheetah9LP disk, which is of size 9.1GB, as this is the largest disk that can currently be simulated by DiskSim.

Table 1: Characteristics of the TPC-C trace and the reduced trace that is used in our experiments.

The trace used to drive the simulator is based on TPC-C [8]. Due to the size limitation of DiskSim and our simulation environment, however, we reduced the full TPC-C trace to 1/200 of the original. This was done by randomly selecting one reference out of every ten references, then taking the front part of the resulting trace. For all our experiments, the buffer cache is warmed up with the first 1/100 portion of the reduced trace before results are obtained with the following 1/20 part of the trace. The characteristics of the reduced trace as compared with the original TPC-C trace are shown in Table 1.

	TPC-C Trace	Reduced Trace
Total Request	99.4 GB	497.2 MB
Reference Set	16.9 GB	200.2 MB
Read Request	84.2GB(84.65%)	394.8MB(79.4%)
Write Request	15.2GB(15.34%)	102.4MB(20.6%)

2. Hit Ratio is No Longer the Adequate Performance Metric with Nonvolatile Cache

In traditional buffer cache replacement research the hit ratio is a common metric that is used to compare the performance of replacement schemes. Though the hit ratio metric may not be an exact representation of performance in real systems due to varies system implementation overhead, it does provide a close indication of what kind of performance to expect in reality. Hence, this metric has been widely used. In this section, we show that for nonvolatile buffer caches, however, the hit ratio is no longer the appropriate indicator of performance. We show that with nonvolatile caches, instead of the hit ratio, using the number of disk accesses directly is more appropriate. Based on this observation, we show that using the MIN replacement algorithm is no longer optimal (when the number of disk accesses is the metric of interest), but that improvements to using MIN can be made by distinguishing read and write operations.

2.1 Hit Ratio Not a Good Indicator of User Perceived Performance

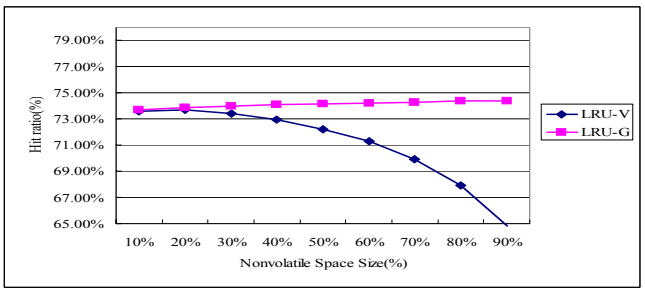
Figure 2 shows the performance of LRU-V and LRU-G. Figure 2(a) shows the hit ratio of the two algorithms, while Figure 2(b) shows the response time. The x-axis for both results is the proportion of nonvolatile space in nonvolatile cache ranging from 10% to 90% of the total cache size.

An interesting observation can be made in the 10% to 40% range of these two graphs. Figure 2(a) shows that the hit ratio of LRU-G is always higher than that of LRU-V. However, from Figure 2(b), we see that for the same range LRU-V shows better response time than LRU-G. This is rather peculiar as if the hit ratio is high, then one would expect the response time perceived by the user to be shorter. To understand the reason behind this discrepancy, let us review the actions taken when a reference is made to the buffer cache.

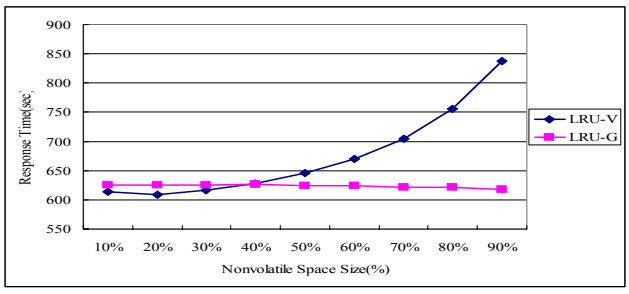
Table 2: Number of disk accesses for hit/miss with read/write operations in volatile and nonvolatile caches. (*Though the value in Hit-Write of Volatile buffer cache should be 1 to maintain consistency, systems generally use the copy-back policy for performance reasons. In this more common case, there is no disk access. Hence, we mark this space as 0.)

Number of disk accesses					
		Volatile buffer cache		Nonvolatile buffer cache	
Hit	Read	0		0	
	Write	0*		Hit from volatile memory	1
				Hit from nonvolatile memory	0
Miss	Read	1		Replace dirty (LRU-G only)	2
				Overwrite clean	1
	Write	Replace dirty	1	Replace dirty	1
		Overwrite clean	0	Overwrite clean (LRU-G only)	0

Table 2 compares the number of disk accesses that are made when hit/miss occurs on volatile and nonvolatile caches. From this table, we observe two peculiar phenomena. First, in the nonvolatile cache there is a disk access even upon a hit on a write. In traditional caches, when a hit occurs, no disk access will occur. (With the write-through policy one disk access will occur, but this policy is generally avoided due to performance degradation. Hence, we denote it as incurring zero accesses.) This disk access can be explained with Figure



(a) Hit ratio



(b) Response time

Figure 2: Performance comparison between the LRU-V and LRU-G nonvolatile management policies.

3(a). Consider a write request to Block 10, which is found in volatile space. This is a write operation, and since the 'Write Nonvolatile Assumption' states that blocks must be written in nonvolatile space, we need to move the newly written block to nonvolatile space. When this happens, a block in nonvolatile space needs to be removed and written to disk. Hence, a disk access happens even on a hit upon write.

The second peculiar phenomenon is that two disk accesses may occur upon a read miss. This is explained using Figure 3(b). Say Block 20 is being read and a cache miss occurs. Block 20 must be brought in from disk, hence one disk access. As one block is fetched, one block must be replaced. If the victim block to be evicted is found in nonvolatile space (which can happen only with

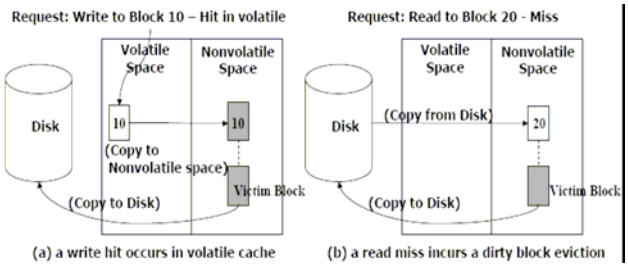


Figure 3: Actions taken in a nonvolatile cache.

policies such as LRU-G), this block must be written back to disk, and hence the second disk access. (Note that this violates the 'Clean Separation Assumption'.) Since in LRU-V all read requests are serviced only in volatile space, this situation cannot arise.

From these observations, we come to two important findings. The first is that unlike traditional caches, distinguishing read and write operations is important for nonvolatile caches. The second is that the number of cache misses is not synonymous with the number of disk accesses, which is a major performance indicator of cache performance.

The reason behind the performance discrepancy found in Figure 2 can be explained from these observations. should be used directly. This is evidenced in Figure 4, where Figure 4(a) is the same figure shown in Figure 2(b), that is, the response time when using the two policies, and Figure 4(b) is the number of disk accesses that occur due to each policy. We see that, indeed, the number of disk accesses is a good representation of user perceived performance.

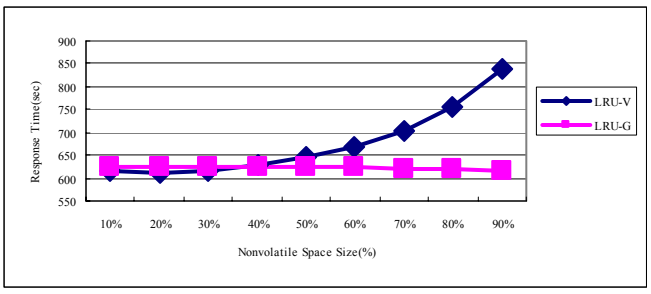
2.2 Performance Better than MIN in Nonvolatile Caches

Belady's MIN algorithm is the proven optimal replacement algorithm for traditional volatile cache management **[3]**. Using hit ratio as the performance metric, Belady proves that MIN will achieve the highest hit ratio when replacing the block that will be used furthest in the future. Though unrealizable because one cannot know future references, MIN has served the buffer management research community by setting a goal to reach, since traditionally the hit ratio is a close indicator of user perceived performance for volatile caches.

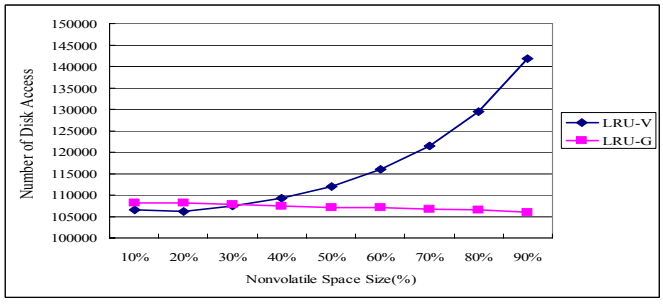
A similar question can be raised with nonvolatile caches. Given that we somehow know the future, what is the best management policy possible with nonvolatile caches? Is MIN still the best policy? In an attempt to answer these questions we use the MIN algorithm as the underlying policy. Recall, however, that MIN was optimized for the hit ratio metric. Since for nonvolatile cache the hit ratio is

no longer the appropriate performance metric we choose to use the number of disk accesses as suggested in the previous section. Also, because of the 'Clean Separation Assumption', we employ the MIN policy at each space separately

To minimize the number of disk accesses, again we start from Table 2. Here, we had observed two situations where an extra disk access occurs aside from the norm. The first is when a write hit occurs in volatile space and the second is when a read miss replaces a dirty block from nonvolatile space. Since the second situation can arise only when volatile and nonvolatile space is managed together, we do not consider this situation in this paper as



(a) Response time



(b) Number of disk accesses

Figure 4: Performance comparison between the LRU-V and LRU-G nonvolatile management policies

Even though the hit ratio using LRU-G is higher than that of LRU-V, the number of disk accesses was actually greater with LRU-G. Hence, for nonvolatile caches, the hit ratio is no longer the appropriate indicator of cache performance, but rather, the number of disk accesses this violates the 'Clean Separation Assumption'.

Let us now concentrate on the first situation. When a write hit occurs in volatile space, the dirty block must be moved to nonvolatile space. This move will force a replacement of a dirty victim block, resulting in a write

disk access. This disk access is inevitable. The question is what we can do to improve performance. The answer is that since a write hit in volatile space will lead to a miss anyway, anytime we need to find a victim block due to a miss we look for a block that will write hit in volatile space with its next reference. This will allow the would-be victim block (the block to be referenced furthest in the future) to not be evicted and be kept in cache longer, helping improve performance. We will refer to this policy as MIN+ to simplify the discussion.

Initial Cache State: 1 2 3 4 5 Reference String : 6(R), 3(W), 2(R), 1(R)

Reference (R:Read, W:Write)	MIN			MIN +		
	Hit / Miss	Cache State	Disk Access	Hit / Miss	Cache State	Disk Access
6(R)	Miss	6 2 3 4 5	1	Miss	1 2 6 4 5	1
3(W)	Hit	6 2 x 3 5	1	Miss	1 2 6 3 5	1
2(R)	Hit	6 2 x 3 5	0	Hit	1 2 6 3 5	0
1(R)	Miss	6 2 1 3 5	1	Hit	1 2 6 3 5	0
Total # of Disk Access	3			2		

Figure 5: Example of cache state evolution using MIN and MIN+.

Consider the example depicted in Figure 5. There are a total of five cache blocks in this nonvolatile cache, of which three comprise volatile space (represented by the clear blocks) and two comprise nonvolatile space (represented by the shaded blocks). Blocks 1, 2, and 3 are currently residing in volatile space and Blocks 4 and 5 are residing in nonvolatile space. The reference string references Blocks 6,3,2,1 with the R and W values in parenthesis indicating Read and Write references, respectively. Recall that volatile and nonvolatile spaces are being managed separately with MIN. The Hit/Miss column indicates the hit or miss incurred by the particular reference and the Disk Access column indicates whether a disk access is incurred with the reference.

Upon read reference to Block 6 a miss occurs with the MIN algorithm and of the 3 blocks in volatile space Block 1 is replaced since it is referenced furthest in the future. For MIN+, the same miss occurs, but instead of Block 1, Block 3 is replaced because the next reference to Block 3 is a write.

The next reference is a write to Block 3. For MIN, this is a write hit in volatile space, hence we move Block 3 to nonvolatile space leading to an eviction of Block 4. (Block 5 could have been evicted here without altering the discussion.) For MIN+, it is a miss to nonvolatile space. Hence, Block 4 is evicted and replaced with Block 3. We see here that a disk access due to Block 3 is inevitable for both MIN and MIN+.

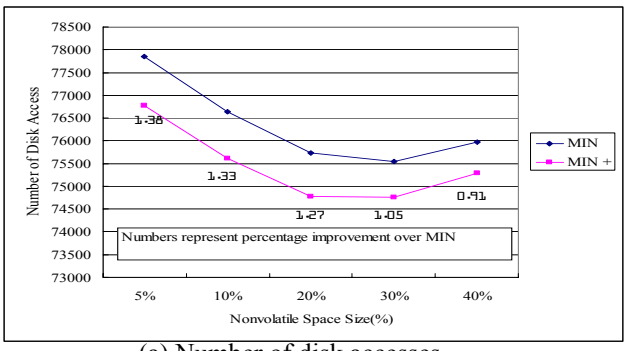
The third reference to Block 2 is a hit for both cases. Finally, the last reference is a read reference to Block 1. This is where MIN and MIN+ differ. By not having evicted Block 1 when Block 6 was referenced, and instead having had evicted Block 3, which was useless in volatile space anyway, MIN+ is able to save a disk access with this last access.

We see that even though the total hit/miss count for MIN and MIN+ is identical, we find that the number of disk accesses may be reduced by evicting blocks in volatile space that will write with its next reference, instead of evicting blocks that will be referenced in the furthest future, as is done with MIN.

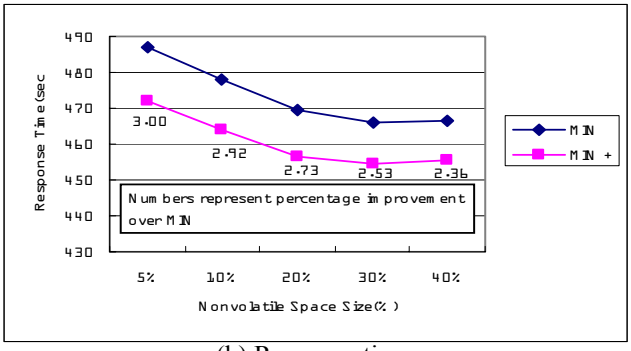
Figure 6 compares the performance of MIN and MIN+ obtained through simulations. We see that MIN can be improved by distinguishing read and write operations and evicting blocks with next write references from volatile space.

3. Putting Theory into Practice

In the previous section, we showed that performance of using the MIN algorithm may be improved (in terms of the number of disk accesses) by selecting a victim block from volatile space whose next reference is a write. In this section, we show that this is also applicable to practical algorithms in use today, specifically, the LRU algorithm. In the first part of this section, we use the crystal ball approach and quantify the improvement that is possible when future knowledge is available. In the latter part of the section, we present a scheme that is more practical in that it is based on acquired history.

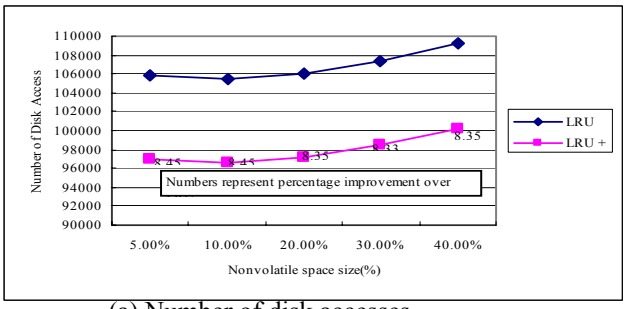


(a) Number of disk accesses

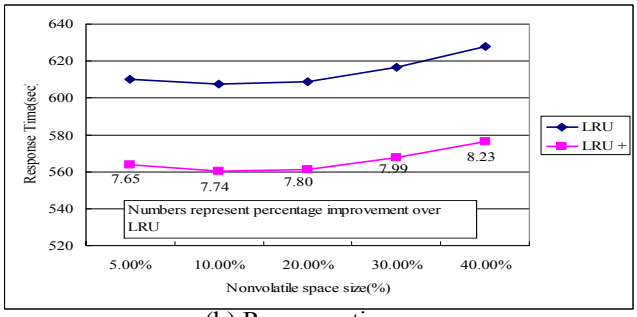


(b) Response time

Figure 6: Performance comparison of MIN and MIN+



(a) Number of disk accesses



(b) Response time

Figure 7: Performance comparison of LRU and LRU+.

3.1 Improving LRU with a Crystal Ball

We showed that for nonvolatile cache MIN could be improved by distinguishing reads and writes to volatile and nonvolatile space. Using the same technique used with MIN+, here, we show that similar improvements are possible with practical replacement algorithms such as LRU. The improvements are experimentally quantified in this section.

Assuming that we have a crystal ball that tells us whether the next reference is a write or a read, the LRU replacement algorithm can be augmented with the same technique used with MIN+ so that the block whose next reference is a write is evicted from cache. This is implemented in the following manner. (We call this modified algorithm LRU+.) Whenever a block is brought into volatile space due to a miss or is hit in volatile space, the block is checked to see if its next reference is a write or a read. If it is a read, then, as is normally done, we move this block to the MRU-end of the LRU list. If it is a write, then this block is the next candidate to be evicted. Hence, it is moved to the LRU-end of the LRU list.

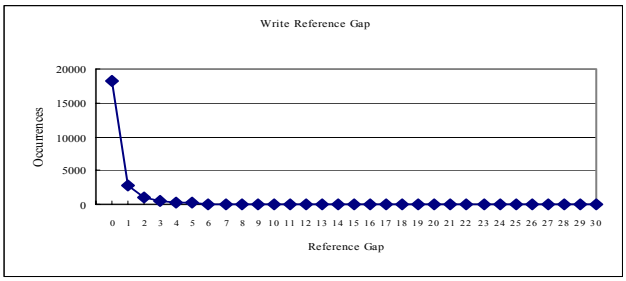
Figure 7 shows the performance difference between LRU and LRU+. The results clearly show that the number of disk accesses is an efficient indicator for user perceived response time performance and that there is room for improvement in the management of the buffer cache. For a wide range of nonvolatile space sizes, the improvement in performance is steadily in the 7 to 8% range.

3.2 Write History based LRU

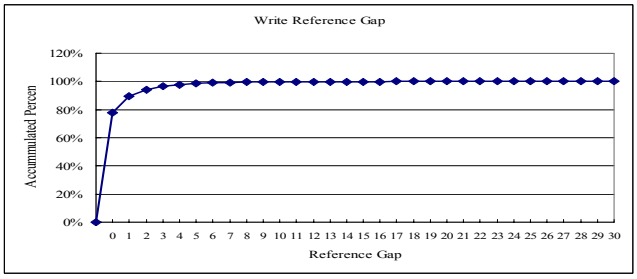
In the previous subsection, we showed that there is room for improvement in managing nonvolatile cache. Unfortunately, LRU+ required future knowledge, which, in reality, is not easy to come by. Here, we propose a more feasible scheme based on the observations of the previous sections. The key point in improving performance as described in the previous sections is that blocks in volatile space with next write reference should not hold a cache block as it will be of no benefit anyway. Hence, devising a scheme to correctly guess a write access is crucial.

Observe from Figure 8 the write reference behavior of blocks used in our experiments. Figure 8(a) shows the absolute frequency of write reference gaps. A write reference gap is the number of read references between two write references. That is, for every number of read references between two write references we count all such occurrences. These values are plotted in Figure 8(a). For example, there are roughly 18,000 occurrences where there is zero read reference between two write references. Figure 8(b) shows the relative cumulative frequency (accumulated percentage) of these counts. Note that the majority of writes come in succession and roughly 95% of writes have less than 3 reads between write references. This means that when a write reference occurs another write will occur in close proximity.

Based on this observation, we devise a crude, but simple history based scheme where we keep a list of blocks that have been write referenced, which we will call the write-history list. When a block that is headed to volatile space is referenced (whether on a miss or a hit) we check the write-history list to see if a write reference on that block had occurred previously. If it had, the block is placed at the LRU-end of the LRU list. This block is retained on the LRU side of the LRU list (even upon a subsequent read hit) until evicted. Otherwise, it is placed on the MRU-end as would be done normally. Cache management in nonvolatile space remains unchanged. We call this

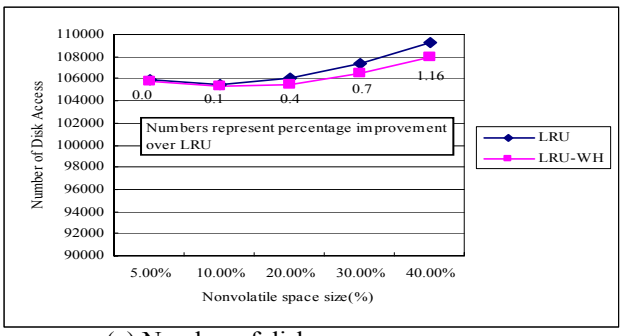


(a) Absolute frequency of write reference gaps

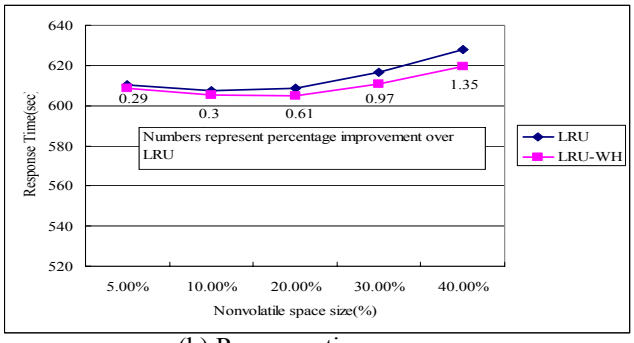


(b) Accumulated percentage

Figure 8: Write reference behavior observed for the trace used.



(a) Number of disk accesses



(b) Response time

Figure 9: Performance comparison of LRU and LRU-WH.

scheme LRU-WH (for Write History). Note that though this scheme is employed with the LRU policy for this study, the write-history scheme may be employed independently with any other replacement policy.

Figure 9 shows the performance results of LRU-WH as compared with simply using LRU. The improvement obtained in these experiments is in the 1% range. This is nominal compared to the achievable improvement shown in Figure 7. The significance of this result is that the improvement was achieved through a crude, first-time attempt. Research to come up with a better scheme is continuing even as we write.

4. Summary and Conclusion

A nonvolatile buffer cache provides a computer system with a means to maintain complete consistency as well as improve performance. The results of this paper in regards to nonvolatile buffer cache can be summarized as follows. First, we showed that the hit ratio that has been a commonly used metric to measure buffer cache performance is no longer adequate for caches with NVRAM. Instead of the hit ratio, we need to count the number of disk accesses to assess user perceived cache performance. Second, we showed that because of this change in performance metric, when managing a buffer cache with NVRAM, one can do better than when using the MIN replacement algorithm mainly by distinguishing read and write operations. With this, we showed that there is room for improvement in efficiently handling caches with NVRAM. Finally, based on these findings, we proposed a simple and practical buffer management technique, LRU-WH, that improves on LRU by a nominal 1%.

The results here have only set the groundwork for much more research to be done. Admittedly, the experiments were conducted with only one set of traces, hence we need to consider traces that have a variety of characteristics. The LRU-WH, though a good starting point, is not satisfactory. There certainly should be a way to do better, and this needs to be explored as well. The three assumptions of this paper are not stone-graved. They could be relaxed in one way or another. How this will influence the analytical results as well as the experimental results are some of the questions that are still open and need to be answered.

Acknowledgement

This work was supported in part by grant No. R01-2004-000-10188-0 from the Basic Research Program of the Korea Science & Engineering Foundation.

References

[1] S. Akyurek and K. Salem, “Management of Partially Safe Buffers,” IEEE Transactions on Computers, Vol. 44 No. 3 pp. 394-407, 1995.

[2] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer, “Non-volatile memory for fast, reliable file systems,” in Proc. of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 10--22, October 1992.

[3] L. Belady, “A Study of Replacement of Algorithms for a Virtual Storage Computer,” IBM Systems Journal, 5(2):78–101, 1966.

[4] N. K. Edel, D. Tuteja, E. L. Miller, and S. A. Brandt, “MRAMFS: a compressing file system for non-volatile RAM,” in Proc. of the 12th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, October 2004.

[5] T. R. Haining and D. D. E. Long, “Management policies for non-volatile write caches,” in Proc. of the IEEE International Performance, Computing and Communications Conference, pp. 321–328, Feb. 1999.

[6] E. L. Miller, S. A. Brandt, and D. D. E. Long, “HeRMES: High Performance Reliable MRAM-Enabled Storage,” in Proc. of the 8th IEEE Workshop on Hot Topics in Operating Systems, pp. 83-87, 2001.

[7] A. A. Wang, P. Reiher, G. J. Popek, and G. H. Kuenning, “Conquest: better performance through a disk/persistent-RAM hybrid file system,” in Proc. of the 2002 USENIX Annual Conference, June 2002.

[8] Y. Zhou, Z. Chen and K. Li. “Second-Level Buffer Cache Management,” IEEE Transactions on Parallel and Distributed Systems, Vol. 15, No. 7, July, 2004.

[9] The DiskSim Simulation Environment (Version 3.0). http://www.pdl.cmu.edu/DiskSim/index.html

[10]http://www.memorystrategies.com/report/EmergingMemories.htm, Emerging Memories: Applications, Device and Technology.

[11] http://www.ramtron.com/doc/Products/overview.asp