Non-Volatile Memory Based Cache Replacement Policy

Ziqi Fan, Alireza Haghdoost, and David H.C. Du
University of Minnesota-Twin Cities
{fan, alireza, du}@cs.umn.edu

Doug Voigt
HP Storage
doug.voigt@hp.com

***Abstract*—Currently, most computer systems consist of Dynamic RAM (DRAM) as main memory and Hard Disk Drives (HDDs) as storage devices. Due to DRAM's volatile nature, it suffers from data loss in the event of power failure or system crash. With rapid development of new types of non-volatile memory, it becomes promising that one of these technologies will replace DRAM as main memory in the near future.**

Considering non-volatile memory as main memory in our future systems, some mechanisms designed for DRAM based system should be modified to fully explore the advantage of the non-volatile characteristic of main memory. In today's DRAM based systems, newly updated pages will have to be flushed to HDD in every 30 seconds. In the future non-volatile memory based systems, any updated pages can be kept longer in main memory without the urgency to be flushed to HDD. Therefore, this opens opportunities to investigate new cache replacement policy.

In this paper, we investigate and propose a new non-volatile memory based cache replacement policy that giving priority to re-group and synchronize long consecutive dirty pages to take advantage of HDD's fast sequential access speed and non-volatile property of main memory. Specifically, with the help of a fixed or dynamic *Threshold*, we synchronize and evict qualified consecutive dirty pages out of cache, instead of evicting singular page at the least recently used position. We evaluate our scheme with various I/O traces. The experimental results show that our proposed cache replacement policies shorten system I/O completion time more than 10x and improve cache hit ratio varied from 0.5% to 10% when compared with DRAM based LRU policy.

I. Introduction

In the last several years, non-volatile memory has quickly developed into promising replacement for DRAM as main memory. Among them, flash memory has already being commercialized and been widely deployed in small devices [1]. However, due to its relatively slow access speed and low endurance, flash memory is typically used as storage devices or caching devices. Other new types of non-volatile memory such as Phase Change Memory (PCM), Memristor and SST-RAM, have rapidly developed into possible candidates for main memory in the future computer systems.

Dynamic random-access memory (DRAM) is the most common technology used for main memory. Despite DRAM's advantages of high endurance and fast read/write access speed, DRAM suffers from data loss in the event of power failure or system crash due to its volatile nature. The emerging non-volatile memory technologies may offer other advantages in addition to their non-volatile nature. For examples, Memristor and PCM can achieve higher density, Memristor and SST-RAM can have faster read accesses and lower energy consumption [2]. With non-volatile memory as main memory, dirty pages in main memory will not be lost in the occurrence of power failure or system crash. As a result, the frequency of dirty page synchronization from memory to storage can be cut down dramatically.

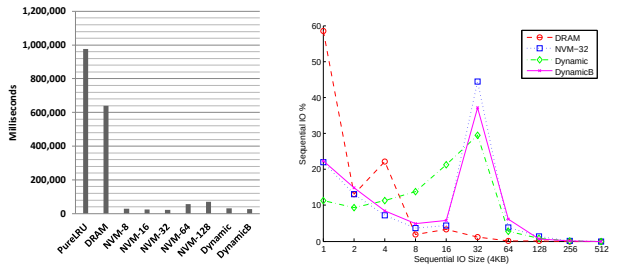
Due to limited capacity of main memory, only certain amount of data can be cached. When the memory is full and a new requested page needs to be cached, a victim page has to be evicted to reclaim space. To fully explore HDD's fast sequential access speed, we propose new non-volatile memory based cache replacement policies that giving priority to re-group, synchronize and evict long consecutive dirty pages out of cache. In this paper, we use the concept of *Threshold* to select long consecutive candidate pages. Compared with DRAM based environment, this non-volatile memory based main memory system does not require frequent synchronization with storage and some dirty pages can be kept in memory for longer duration [3]. This creates a bigger pool for selecting qualified long consecutive dirty pages. To mitigate the overhead of selecting long consecutive dirty pages, we propose to modify the current page cache hash table to identify long consecutive dirty pages more efficiently. We evaluate our proposed schemes with various I/O traces. The experimental results show that our proposed cache replacement policies shorten system I/O completion time more than 10x and improve cache hit ratio varied from 0.5% to 10% when compared with DRAM based LRU policy.

The major goals of our proposed replacement schemes are to decrease the amount of I/O traffic to storage and to shorten I/O completion time without reducing cache hit ratio, rather than to increase cache hit ratio. We have demonstrated such a replacement policy can effectively work together with existing cache schemes like LRU. We plan to demonstrate its applicability with ARC [4] in the near future.

II. Evaluation

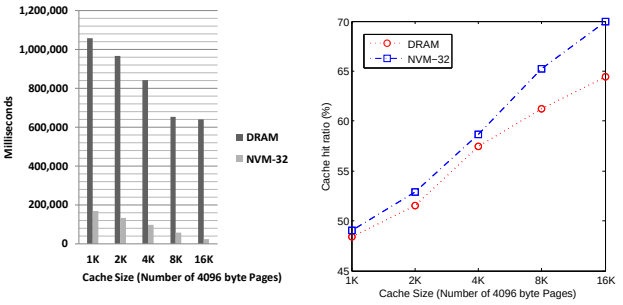
In this section, we evaluate our proposed cache replacement policies along with some baseline algorithms to compare their performance: 1) *PureLRU*: LRU algorithm without any auto

This work was partially supported by NSF Awards: 1217569 and 0934396. This work was also supported by Hewlett-Packard Company and University of Minnesota DTC/ADC Fellowship.



(a) I/O Completion Time (b) Percentage of Sequential I/O Sizes

Fig. 1. Different Scheme Performance of Trace wdev_0



(a) I/O Completion Time (b) Cache Hit Ratio

Fig. 2. Cache Size Variation Performance of Trace wdev_0

flush back from memory to disk; 2) *DRAM*: LRU algorithm with every 30 seconds flush back from memory to disk; 3) *NVM-Threshold*: Proposed scheme with fixed threshold varied from 8 to 128; 4) *Dynamic*: Proposed scheme with dynamic threshold. Threshold increases in step by one and decreases in step by half; 5) *DynamicB*: Proposed scheme with dynamic threshold. Threshold increases in step by one and decreases in step by one too.

To evaluate our proposed cache replacement policy, we implemented them on the basis of Sim-ideal [5] simulator and used DiskSim [6] simulator to test how long it will take to finish all I/O requests. We use MSR Cambridge traces shared from SNIA provided by Narayanan *et al.* [7] as input. Out of all the traces, we select the first volumes of traces from 9 different servers with intensive write requests and got similar results. In this paper, we only show the results of trace wdev_0 for space efficiency.

In order to compare system I/O performance between different schemes, we measure I/O completion time. As shown in Figure 1(a), scheme *NVM-32* has the shortest I/O completion time. Other *NVM-Threshold* (*NVM-8*, *NVM-16*, *NVM-64*, *NVM-128*), *Dynamic* and *DynamicB* schemes perform comparably. Theoretically, write requests with the longest consecutive pages to HDD produce the best results. However, for *NVM-64* and *NVM-128*, due to their *Threshold* is too large, few of consecutive dirty pages in the cache could meet this high standard. Counterproductively, short consecutive dirty pages from LRU position will be issued if no such dirty page sequence can be identified. On the other hand, if *Threshold* is too small, such as *NVM-8* or *NVM-16*, HDD's fast sequential access benefits cannot be fully explored. For *Dynamic* and *DynamicB*, they perform slightly worse than *NVM-32*. In the future, we would like to explore more intelligent dynamic threshold based schemes that can potentially achieve better performance facing variations of workloads.

As a baseline experiment, we have implemented a pure LRU scheme (*PureLRU*). It displays worst performance which takes 10x to 40x of time to finish all the write requests when compared with *NVM-32*. The reason for the bad performance of *PureLRU* is each write request is in length of one page which represents complete random accesses to HDD. Compared with *PureLRU*, *DRAM* performs slightly better because in 30 seconds interval, some short write request will be issued to HDD rather than complete random write accesses. However, *DRAM* still takes 10x to 30x of time to finish all write requests compared with *NVM-32*.

To give an intuitive explanation of what leads to the huge performance difference for these schemes, we plot the percentage of sequential I/O sizes in Figure 1(b). A given sequential I/O size means the storage write requests with the size in the range of two consecutive numbers of pages. Since *NVM-32* gives the best performance in the *NVM-Threshold* family, we only use *NVM-32* to compare with other schemes. Compared with *DRAM*, *NVM-32*, *Dynamic* and *DynamicB* all have low percentage of small sequential I/O sizes and high percentage of large sequential I/O sizes which can gain more benefits due to HDD giving better performance to large sequential I/O. Compared with other algorithms, *Dynamic* has a lower percentage of small sequential I/O sizes (1 and 2). But due to higher percentage of medium sequential I/O size (4, 8 and 16), its overall performance is not the best.

We compare baseline experiment *DRAM* and our proposed *NVM-32* on different cache sizes from 1K to 16K number of 4096 byte pages. The results are shown in Figure 2. For I/O completion time, both *DRAM* and *NVM-32* achieve better performance with larger cache sizes. On the other hand, *NVM-32* performs 5x to 20x better than *DRAM* along with cache size increased from 1K to 16K number of pages. Both *DRAM* and *NVM-32* have higher cache hit ratio with larger cache sizes. On the other hand, *NVM-32* achieves 0.6% to 5.8% higher cache hit ratio than that of *DRAM* for different cache sizes.

REFERENCES

[1] M. Murugan, and David H.C. Du. Rejuvenator: A Static Wear Leveling Algorithm for NAND Flash Memory with Minimized Overhead. In MSST '11, Denver, Colarado, USA.
[2] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable dram alternative. In ISCA '09, New York, NY, USA, 2009.
[3] S. Qiu and A. L. N. Reddy. NVMFS: A hybrid file system for improving random write in nand-flash SSD. In MSST '13, Long Beach, CA, USA
[4] N. Megiddo and D. S. Modha. ARC: a self-tuning, low overhead replacement cache. In FAST '03, Berkeley, CA, USA, 2003.
[5] Sim-ideal. git@github.com:arh/sim-ideal.git
[6] DiskSim v.4.0. http://www.pdl.cmu.edu/DiskSim/.
[7] D. Narayanan, A. Donnelly, and A. I. T. Rowstron. Write offloading: Practical power management for enterprise storage. In FAST '08, San Jose, CA, Feb. 2008.