WAlloc: An Efficient Wear-Aware Allocator for Non-Volatile Main Memory

Songping Yu, Nong Xiao, Mingzhu Deng,Yuxuan Xing, Fang Liu, Zhiping Cai,Wei Chen

State Key Laboratory of High Performance Computing
College of Computer
National University of Defense Technology
Changsha, China
we.isly@163.com

***Abstract*—The non-volatile memory （ NVM ） has the illustrious merits of byte-addressability, fast speed, persistency and low power consumption, which make it attractive to be used as main memory. Commonly, user process dynamically acquires memory through memory allocators. However, traditional memory allocators designed with in-place data writes are not appropriate for non-volatile main memory（NVRAM）due to the limited endurance. For instance, the number of write operations is merely 10^8 times per PCM cell. In this paper, we quantitatively analyze the wear-oblivious of DRAM-oriented designed allocator—glibc malloc and the inefficiency of wear-conscious allocator—NVMalloc. For example, the average imbalance factor (the maximum/the average) of memory allocation is about 7.5 and 3, respectively. Based on our observations, we propose WAlloc, an efficient wear-aware manual memory allocator designed for NVRAM, decouples metadata and data, uses Less Allocated First Out allocation policy and redirects the data writes. Experimental results show that the wear-leveling of WAlloc outperforms that of NVMalloc about 30% and 60% under random workloads and well-distributed workloads, respectively. In addition, considering the trade-off between space and wear-leveling, WAlloc reduces average data memory writes in 64 bytes block by average 1.5X comparing with malloc with almost 8% extra space overhead.**

Keywords—Non-Volatile Memory ; memory allocator ; wear-aware

I. INTRODUCTION

Dynamic Random-Access Memory (DRAM) has been the de facto component for the main memories of computer systems for past few decades. However, despite providing a very fast access speed, DRAM has a huge defects with respects of an increasing need of large main memory due to its density limitation [1,2] and its high energy consumption[3,4](about 10%−30% of total system energy). In response to this issue, the emerging Non-Volatile Memory(NVM) , such as Phase Change Memory (PCM) [5], STT-RAM [6], and memristors[7][35], incorporating a host of desirable features − access speeds comparable to DRAM, storage-like persistence, low power consumption, and byte addressability. These new types of memory, especially PCM, show the promise of being the candidate main memory with comparable performance and much higher capacity than DRAM.

Unfortunately, PCM suffers from limited write endurance. A typical PCM cell permanently fails after bearing 10^7 to 10^9 writes [8]. Write-intensive operations may break PCM in dozens of seconds, which are shared in data-intensive application, web applications, and so on. Consequently, without appropriate memory allocation schemes to tackle this obstacle, it is formidable to reap huge fruits of PCM.

In this paper, we propose a wear-aware allocator for non-volatile main memory (**WAlloc**). Specifically, WAlloc concerns two main folds of metadata memory writes-reduction and data memory writes-reduction. First, it decouples metadata and data management. With the Less Allocated First Out policy, equilibrium writes of metadata for allocated space are feasible under dynamic changing workloads; and using lazy copy-on-write technique to achieve the balance of leveling data writes and memory space footprint. Besides, WAlloc is an off-heap memory allocator and redirects data access instantly. At last, WAlloc, implemented as a runtime library, allows application to use with minimum code change in a transparent manner and keeps recoverable volatile metadata in DRAM avoiding unnecessary small writes to mitigate impairment to NVM further.

Our contributions can be summarized as follows:

1. We quantify the write damage to NVM with traditional allocator (malloc etc.), and propose three insightful observations: (1) Traditional allocation policy (e.g. FIFO、LIFO) could make memory areas hot spot;(2) metadata for space management brings extra wear; (3) in-place data writes do dreadful damage to NVM.
2. Based on the observations, we propose the WAlloc, which (1) decouples metadata and data management; (2) minimizes metadata writes through separating volatile metadata and non-volatile metadata; (3) organizes free memory spaces with careful allocation policy(Less Allocated First Out) to prevent some specific memory areas from becoming hot spot under changing workloads; (4) redirects data writes around the non-volatile memory space.
3. Considering compatibility and memory management flexibility, we implement WAlloc as a runtime library that allows application to use with minimum code change and manage NVM memory in an off-heap way.

978-1-4673-8590-9/15/$31.00 ©2015 IEEE

4. Evaluation shows that the wear-leveling of WAlloc outperforms that of NVMalloc[25] about 30% and 60% under workloads of random and uniform data size distribution, respectively; and the average data memory writes in 64 bytes block reduced by average 1.5X comparing with malloc and NVMalloc.

II. BACKGROUD AND MOTIVATION

A. Operation System Assumptions for NVM

New emerging memory technologies (e.g. PCM, Memristor, and STTRAM) have blurred the gap boundary between main memory and storage. They have been demonstrated fast, persistent and byte-addressable, and are widely regarded as candidate replacements or supplements for DRAM in the main memory.

Taking PCM for an illustration. PCM is a non-volatile memory built out of chalcogenide-based materials[38] (e.g., alloys of germanium, antimony, or tellurium). It stores bits by heating a nanoscale piece of chalcogenide glass with a large current and allowing it to cool down into forms with different electrical resistance enabling multiple bits, typically speaking, a crystalline state corresponds to "1" and an amorphous state corresponds to "0".

The other two alternatives technologies are also the promising memory substitutions. Although these technologies have both the merits of main memory and storage, they have several weaknesses as well, including an asymmetry of read and write costs and limited endurance. For example, PCM and Memristor offer 10^8 writes/cell compared to 10^{16} writes/cell of DRAM. Table 1 presents the parameters of NVM and DRAM.

In this paper, we focus on how to design a wear-aware allocator to improve the endurance of non-volatile main memory (NVRAM), assuming that PCM as the non-volatile main memory and the memory model is a combination with DRAM for the purpose of performance and reducing writes to NVRAM. The supports of operating system are shown below: (1) Volatile and non-volatile memory could be mapped into the process address space within virtual memory mechanisms and system call mmap() supports to get NVRAM pages from operating system[36].(2) The atomic memory writes at 64 bytes granularity and the basic unit of NVRAM is 4KB. (3) In order to take advantage of the non-volatility of NVRAM, a process must be able to map to the same NVRAM pages after system reboots; and persist memory pointers with mappings fixed[28], pointer swizzling[29] or binding each logical group of NVRAM pages in its own segment[11]. (4) it will be possible to have just one big giant share virtual memory address (2^{64}) in operating system which also provides access control to regions of NVRAM just like in file system.

TABLE 1 Characters of NVM and DRAM

	Read	Write	Endurance
PCM	50-85ns	150ns-1us	10^8-10^{12}
Memristor	100ns	100ns	10^8
STT-RAM	6ns	13ns	10^{15}
DRAM	60ns	60ns	$>10^{16}$

B. Motivation

To quantify the problems of main memory allocator, we compare the data and metadata writes patterns of two memory allocators with memory allocation and deallocation of random, uniformly distributed sizes between 8B and 500B of 10K iterations: one is the general glibc malloc allocator, the other one is NVMalloc[25], a wear-aware allocator for NVM, limiting writes of every memory block within time interval T(seconds). We instrument thees two workloads using Intel Pin[30] to record memory writes.

When it comes to memory management, traditional allocators (such as glibc malloc) are the best alternative. The malloc allocator is the most universal and general way to provide memory for user applications. As shown in Figure 1 and Figure 2, memory writes of malloc skew heavily both under allocation of uniform and random distribution of data size. The maximum deviation factor(the maximum divides the average value) of metadata writes of uniform and random distributed data size is around 9.6 and 5.4, respectively; and in-place data writes accelerate the speed of wear-out. This indicates that **malloc designed with the principles of in-place updates, coupling metadata and data to easy free memory space management, allocating memory blocks in LIFO (Last In First Out) way is not suitable for NVM with the facet of limited endurance**.

Hence, NVMalloc[25], a wear-aware allocator for NVM. It curbs write frequency of memory block no higher than 1/T. In the first place, it timestamps this memory with T nanoseconds and adds it to a FIFO freed block queue. Each time of memory allocation or release, NVMalloc checks the head of the FIFO queue; if it has been in the queue at least for T, than it become available for reallocation. Figure 3 shows the memory writes of NVMalloc with the timestamp of 7 microseconds, compared to malloc, it can spread memory writes more evenly and less (maximum number of data writes is 2700 and maximum number of metadata writes is 210). However, data writes still in-place deviates from metadata's heavily; furthermore, Figure 4 shows that the maximum metadata writes deviation factor is still around 3. Last but not the least, NVMalloc states that trading memory allocator performance and memory space for wear-level; the longer timestamp T is, the more space will be exhausted.

Overall, we observe that memory allocator for NVM should take wear-level into consideration. Traditional memory allocator and NVRAM allocator still needs further improvement in aspects of memory allocator policy and data wear-leveling.

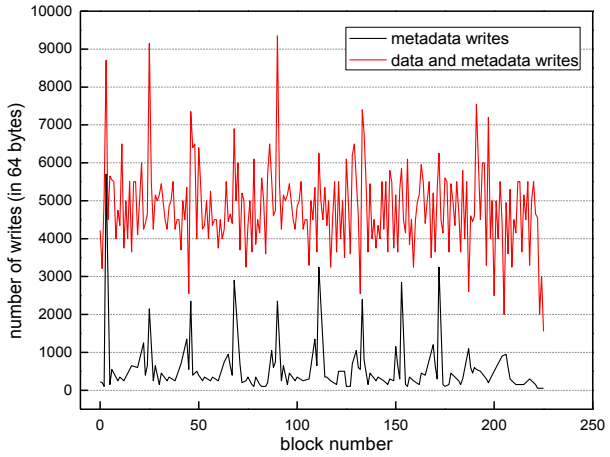

Fig. 1. memory writes pattern of uniform data distribution using glibc malloc

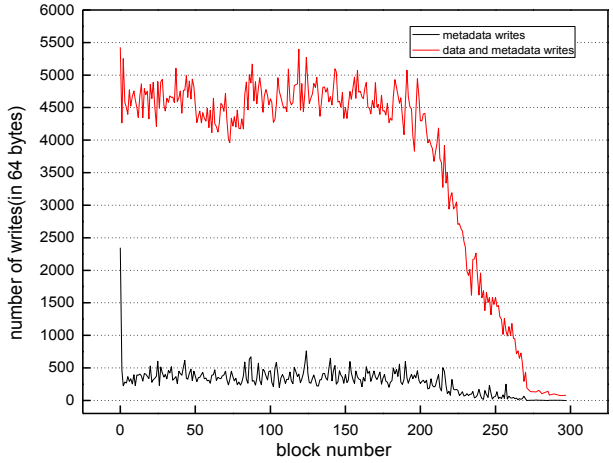

Fig. 2. memory writes pattern of random data distribution using glibc malloc

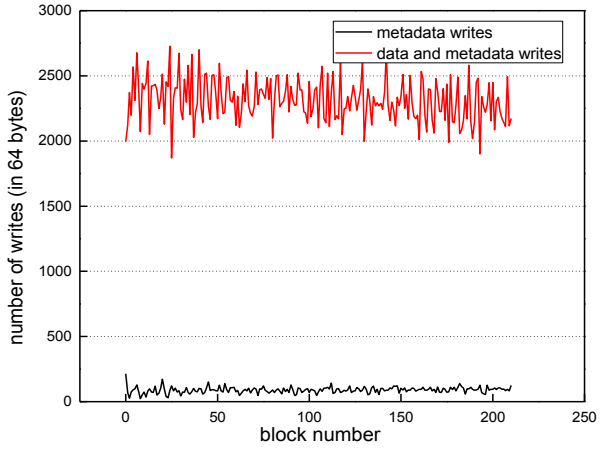

Fig. 3. memory writes pattern of data distribution using NVMalloc

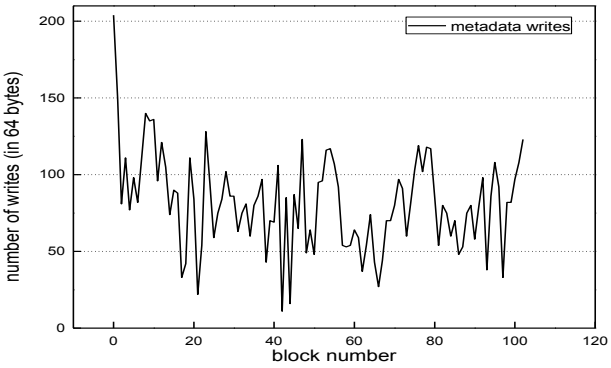

Fig. 4. memory writes pattern of metadata in NVMalloc

III. DESIGN AND IMPLEMENTATION OF WALLOC

In this section, we propose WAlloc, an allocator for Non-Volatile Main Memory with the elaborate design for wear-aware purpose.

A. Design Principles

P1. Decoupling volatile metadata management and data management with classifying metadata into volatile and non-volatile. Traditional allocators use pointers to chain free memory blocks together, small writes of pointers happen under malloc/free operations. As for NVM, due to the characteristic of non-volatile, WAlloc only needs to ensure non-volatile metadata of memory block state and data size to be persistent. The volatile metadata for free memory space stores in DRAM. WAlloc rebuilds these volatile information after a crash, similar to [25].

P2. The Less Allocted First Out allocation policy. Either LIFO in malloc, allocations are deallocated in the reverse order of their allocation or FIFO in NVMalloc, allocations are deallocated approximately in the order of their allocation could trigger hot memory space areas making some NVM cells wear-out easily. According P1, although minimizing the volatile metadata writes, wear-aware allocation policy to neutralize hot allocation spot is indispensable. WAlloc allocates free memory space in the Less Allocated First Out manner, meaning choose a free memory block of less writes to allocate.

P3. Dynamic off-heap memory allocator with spreading memory writes around. Generally, user process dynamically requests for memory space from heap areas and mmap areas. In malloc, program usually obtains heap memory through system call brk() allocating memory space in LIFO way; Based on P2, wear-aware allocation policy must be taken into consideration, so WAlloc , which does not be confined to LIFO policy, manages an off-heap memory with more flexibility support of allocation methods. Also, compared to metadata writes, the common data writes contribute much more wear to NVM. Metadata writes appear during every memory allocated or freed, while data writes always carry on the specified memory areas allocated. WAlloc redirects data updates to free or allocated read-intensive memory areas to attain data writes wear-leveling.

B. Logical OverView

WAlloc, as shown in Figure 5, is an allocator of off-heap memory areas which created through system call mmap() and the underneath hybrid physical memory architecture is DRAM and NVM. The DRAM memory is transparent for user program in WAlloc, and the only usage of it is to store the volatile metadata for NVM memory management: memory block state, metadata wear count(MWC), data wear count(DWC), free memory block lists(Less Allocated First Out lists) to organize released memory, and so on. Non-volatile metadata (valid flag, data size, timestamp, checksum) coupling with data stores at the head of every allocated memory space (dummy head only exists for small size allocation when large memory splits) and data writes are redirected to free memory areas. The memory space allocated in WAlloc is 64 bytes aligned.

In subsection C, based on our observation, coupling metadata for free memory with data will generate a host of writes to NVM under dynamic change workloads; the volatility of metadata will be elaborated and the design of the Less Allocated First Out policy will be discussed. In subsection D, in-place data writes accelerate the wear-out rate of NVM; data writes wear-leveling will be discussed.

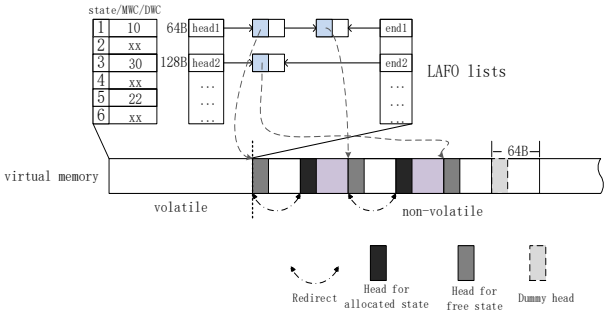

Fig. 5. The whole logical view of WAlloc. Two memory areas of six NVM lines depicted in the diagram are allocated

C. Metadata Wear-Leveling

Metadata in memory allocators typically are classified into two kinds: data description and space description. Specifically, data description includes data validity flag, data size, CRC for data check; space description includes memory block state, free memory space size, memory space start address, pre-free block, next-free block, wear counts record in 64bytes block. In WAlloc, data description should be persistent and space description is volatile, which could be recovery with data description except wear counts. Wear counts are the wear-leveling important parameters in WAlloc. The best case is that if the writes to NVM is equilibrated each time, then the wear counts rebuild is unnecessary. However, the truth we least like to see are those which usually appears. In fact, WAlloc checkpoints wear information of memory block to NVM at intervals.

Under dynamic change workloads, traditional allocators couple metadata with data often result in an ocean of small volatile metadata writes of changing pointers to chain its pre-block and next block in order to merge many small contiguous spaces into large free space for large size requests or split large space into pieces of small spaces for small size requests. Besides, FIFO or LIFO allocation policy (aka. queue or stack allocator) leads to writes focusing on a small memory areas due to free memory operations follow malloc memory operations in case of memory leak.

In order to limit the allocation frequency of each memory area, NVMalloc[25] timestamps every available free memory block to guarantee that metadata writes once every T seconds. The longer T is, the lower metadata writes get. However, we might frown on that for two reasons: for one thing, tagging every memory area with equal timestamp T will consume more memory spaces, due to there is no available memory for requests in the interval of T; for another, It is possible that the allocation's time pattern of malloc of workloads is similar to NVMalloc's timestamp pattern, the NVMalloc just delays the appears of hot spot memory areas. The easiest way to solve this is hash allocator. However, in Glibc malloc or NVMalloc, every free memory list stores memory areas of some certain size. When allocation request comes, allocator take one free memory areas off the list in FIFO or LIFO. If allocator selects one free memory area randomly, it may flat the allocation times of memory areas in the selected free memory list. While the side effect of hash allocator is generates more fragmentation, the two contiguous memory may break again for one allocated.

In WAlloc, Every memory area allocated contains multi-NVM lines (a NVM line is 64 bytes) and attaches a metadata wear count(MWC) and a data wear count (DWC). MWC indicates the memory area allocation frequency; while DWC indicates memory writes frequency that will be discussed in subsection D. Briefly, WAlloc prefers to allocate the memory area of small MWC, namely, the Less Allocated First Out (LAFO) policy. In detail, WAlloc splits every free memory list into two halves according to the average MWC of the whole memory list; first, WAlloc selects the free memory list according to the allocation size, and hash selection from the candidates of lower MWC; second, if there is no candidate in the appropriate free memory list, WAlloc chooses free memory of large size to allocate in large size lists. At last, if allocation still fails in the former two steps, new memory areas will be allocated.

In LAFO policy, there are some points need to be added. To begin with, the MWC increased by 1 in every memory allocation; furthermore, a few free memory lists become hot lists under skew-size workloads, LAFO sets a global threshold (default state is off)of metadata wear count to limit every free memory list allocation times, and this threshold doubles if the average metadata wear count of the free memory lists is big; last but not the least, data writes are normally more often than metadata writes; the question is that why memory allocation does not take the parameter of data wear count into consideration? As a matter of fact, if data writes overwhelm metadata writes, data writes wear-leveling will do the work.

Generally, the Less the First policy (such as LRU) is in common use [9][37]. However, as far as non-volatile main memory allocator's concerned, it is the first time to employ this to reduce the metadata writes of memory allocation, especially under the workloads of frequent memory allocation

and release. In conclusion, on one hand, the benefits of distinguish metadata with volatility we could reap are reduce small writes of metadata to NVM and keep the performance of memory allocation due to the most frequent accessed volatile metadata stored in DRAM. On the other hand, with LAFO policy, WAlloc spreads metadata writes evenly to NVM memory as much as possible.

D. Data Wear-Leveling

Commonly, user process could access physical memory through virtual memory address allocated by system call brk() or mmap(); and virtual memory addresses map fixedly to physical memory addresses until brk or unmmap to release memory. Usually, General allocator wrapping system call brk() or mmap() provides uniform API and eliminates the complexity management of virtual memory for user process, such as glibc malloc. However, in-place data writes due to fixed address binding wear-out NVM cells very fast. Also, when user program calls free() to release memory, glibc malloc buffers freed virtual memory areas for allocation performance and reusing the mapping between the virtual memory address and physical memory address which increases the possibility of data rewrites.

It is well believed that the solution to most problems in Computer Science is simply a level of indirection. Different from the way RAMCloud log-structures memory with append-on-write instantly [31], WAlloc scatters data to any free memory with lazy-copy-on-write policy. RAMCloud manage memory in pursuit of improving the efficiency of DRAM memory space usage, while WAlloc seeks for data writes wear-leveling of NVM memory. Strictly speaking, WAlloc is belong to manual memory management (non-copy allocator) not rather automatic memory management (copy allocator).

Naturally, data writes scattering associates with non-contiguous memory allocation as a result of a part of data moving around. Nevertheless, the workloads WAlloc targets for is small items (few bytes-hundreds bytes), which is widespread in memcache and normal user program. With this in mind, WAlloc allocates contiguous memory and copies data as a whole. The non-contiguous memory allocation will be the future work and not discussed in this paper.

Basically, there are two ways to apply redirection of memory writes in memory allocator. One is build an virtual address mapping table (VAMT) which stores the source virtual memory address and redirected destination virtual memory address (RVMA), like [13]. Every data access must look up the VAM table for RVMA. The other one is Instant Redirection which always returns the RVMA to user process immediately, in view of that user processes always read or write data through pointers, and do not care about the value of pointer variable. WAlloc prefers the Instant Redirect than VAMT for two simple reasons: VAMT takes more extra space to store mapping table and adds latency for data access.

As mentioned in subsection C, data writes wear-leveling involves a parameter-DMC (data wear count). Different from WMC attaching to the head NVM line of allocated memory area, while DMC presenting the writes frequency attaches to the whole memory area. Every allocated memory area composed by multi-NVM lines, so the simple way to compute the DMC of allocated memory area is to sum the wear contribution factors of every NVM line. Assuming the length of allocated memory area "a" is L_a, NVM line is C (a constant:64), W_m is the wear number of mth NVM line, P_m is the data writes probability of the mth NVM line and f_m is the wear contribution factor of the mth NVM line; so the DMC_a could be computed by the following formula:

$$f_m = W_m * P_m, 1 <= m < \lceil L_a / C \rceil, DWC_a = \sum_{m=1}^{m=\lceil L_a/C \rceil} f_m \qquad (1)$$

The accuracy of DWC in formula (1) depends on the probability distribution of NVM lines' wear number, such as "Long Tail Effect" distribution; instead, each allocated NVM line is treated as equal in WAlloc as data copy in one whole piece, so the default probability distribution is uniform distribution.

Putting memory allocation, redirection policy and wear computation all together, the process of data wear-leveling could be summarized as follows: firstly, WAlloc gets the address of NVM line through the offset from the start address, the length of user data and the address user process accessing; secondly, data makes changeover with item, locating in the corresponding free memory list, whose DWC is lower than the DWC of current memory area accessed; if not, allocated new areas for changeover and free original memory; thirdly, computing the DWC of the new memory area; finally, return the new address to user process.

It is crucial to note some additional remarks. In the first step, WAlloc only provides a different API for user process to write data with the reason of wear-leveling of NVM; and allows user to read data with pointer the way it is used to be. In the second step, you may notice that WAlloc use the free memory areas to bear the data writes, actually, data writes have locality, means that some of the data of user program are read-intensive and their memory area could be used for wear-leveling; currently, WAlloc use free memory for wear-leveling on the basis of the characteristic of high capacity of NVM and minimize the size of data copied. In addition, redirection is lazy in WAlloc, which indicates that data moves around until data wear count exceeds a threshold. Finally, WAlloc uses the sequence of {MFENCE, CLFLUSH, MFENCE} instruction to persist data and non-volatile metadata writes, similar to [32-34]. Based on these, recovery in WAlloc is simple, and rebuilding the volatile metadata by scanning the whole allocated memory (from start address to end address). The overhead of rebuilding is neglectable for the recovery process only happens in process restart.

IV. EVALUATION

In this section, we evaluate our WAlloc by comparing it with glibc malloc and NVMalloc in terms of metadata wear-leveling and data wear-leveling under random and uniform allocation workloads. Then we discuss and analysis thespace overhead of wear-leveling. The data size range is between 8B and 500B just as the description in subsection II-B. All of our experiments are carried on a linux virtual machine (kernel

version 3.16.0-30) with Intel(R) Xeon(R) CPU E5-2620 v3 2.40GHz, 4G DRAM(1G as proxy for NVRAM). We implement WAlloc in C++ about 800 lines of code, and all workloads run in a single thread.

From the metadata's point of view, we compare the wear-leveling deviation of metadata using malloc, NVMalloc and WAlloc. Typically, the data size of uniform distribution and random distribution are the same as depicted in subsection II-B. Figure 6 shows the result of metadata memory writes of random data size distribution. The number of metadata memory writes is between NVMalloc's and malloc's, the most memory writes of metadata is malloc and the least is NVMAlloc; as revealed in Figure 7, the average number of metadata memory writes are 268, 71, and 132 for malloc, NVMalloc and WAlloc; the total NVRAM lines of metadata are 287,801 and 300 for malloc, NVMalloc and WAlloc. The reduction of memory writes in 64B is up to 2 times in WAlloc compared with malloc. Clues can be drawn from Figure 6, the base line for NVMalloc and WAlloc is 100 and 150. The fluctuation of WAlloc is more gentle than that of NVMalloc and malloc; more precisely, comparing the wear-leveling of metadata writes, WAlloc is 30% better than NVMAlloc.

Figure 8 and Figure 9 shows the memory writes of metadata under memory allocation of the uniform distributed data size using WAlloc, malloc and NVMalloc. Figure 10 shows reveal the average memory writes and the total NVRAM lines of metadata. Different from random distributed size of memory allocation, WAlloc is highly equilibrated for the equal metadata writes; while malloc and NVMalloc proposes metadata writes turbulence, especially malloc, because of large free memory is splitted for small size of memory allocation request.

In particular, the average number of metadata memory writes are 594, 80, and 400 for malloc, NVMalloc and WAlloc; the total NVRAM lines of metadata are 130,458 and 101 for malloc, NVMalloc and WAlloc. The more memory used, the little memory writes will be for NVMalloc. The reduction of memory writes in 64B is up to 1.14 times in WAlloc compared with malloc. The base line for NVMalloc and WAlloc is 50 and 400 in Figure 9. To be specific, comparing the wear-leveling of metadata writes, WAlloc is 60% better than NVMAlloc; and with less space to bear metadata wirtes, WAlloc presents higher metadata writes than that of NVMalloc. To begin with, the reasons that an ocean of metadata writes in malloc are mainly because memory allocated/released or merged/split causing the value of pointers changing frequently and resulting in changing state of memory more frequent. Besides, NVMalloc restricts memory use through timestamp reducing memory writes more obvious and clearly leading to wasted more memory. WAlloc curbs metadata writes with distinguish of volatile and non-volatile metadata reducing the memory writes of metadata for space management. It allocates memory evenly according to the current wear status of NVRAM lines. Finally, the space/memory writes ratio for metadata of WAlloc is little higher than that of NVMalloc for better wear-leveling reason. Nonetheless, this small space overhead is neglected in comparison with data space footprint.

From the perspective of data wear-leveling, we compare the memory writes of data under random distributed data size workloads only with NVMalloc and WAlloc, since malloc and NVMalloc both write data in-place. Figure 11 only shows the average memory writes and total NVRAM lines of data using NVMalloc and WAlloc. The average memory writes of data and the total NVRAM lines are 2098/481 and 819/1334 respectively. According to the proportion of NVRAM lines between NVMalloc and WAlloc, the ideal average memory writes of data is 757(almost %8 loss comparing with 819). This little interference is mainly come from metadata writes. The result implicates that WAlloc currently implemented with free memory to spread in-place data writes is plausible.

As mentioned before, considering in-place memory writes shared by metadata of malloc and data of NVMAlloc, banding the wear-leveling performances of WAlloc under random and uniform workloads together, confirming that WAlloc reduces average data memory writes in 64 bytes block by average 1.5X (2X for random data distribution and 1.14X for uniform data distribution) comparing with in-place data writes of malloc and NVMalloc with almost 8% extra space overhead.

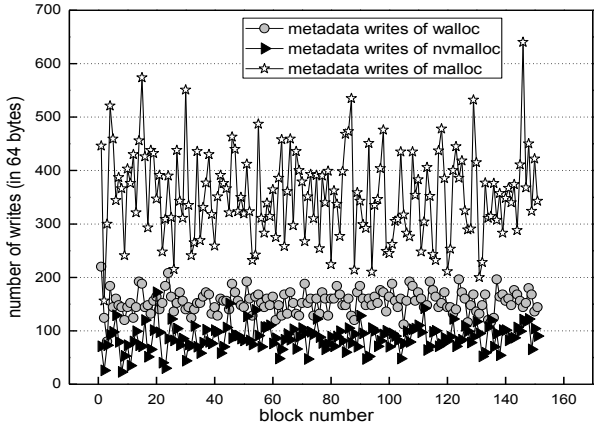

Fig. 6. memory writes pattern of random data distribution using WAlloc, NVMalloc and glibc malloc

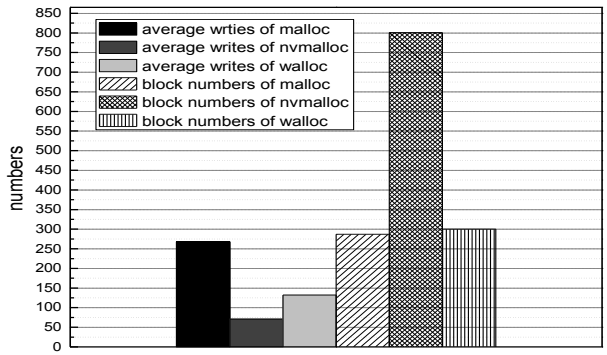

Fig. 7. average memory writes and total NVM lines of metadata with random data distribution using WAlloc, NVMalloc and glibc malloc

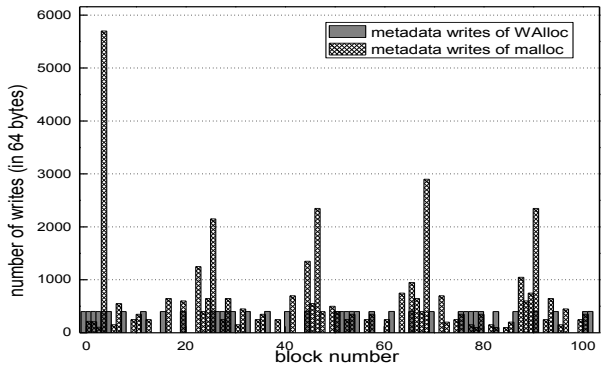

Fig. 8. memory writes of Metadata with uniform data distribution using WAlloc and glibc malloc

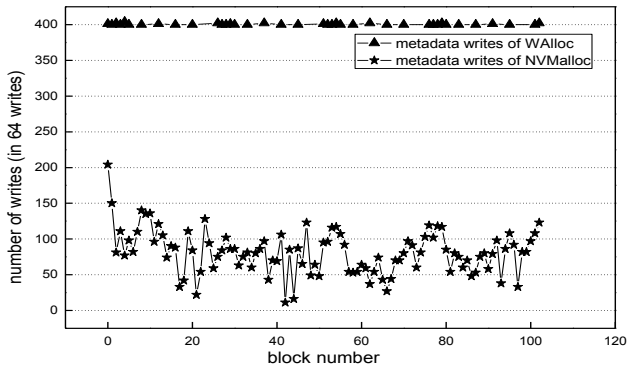

Fig. 9. memory writes of Metadata with uniform data distribution using WAlloc and NVMalloc

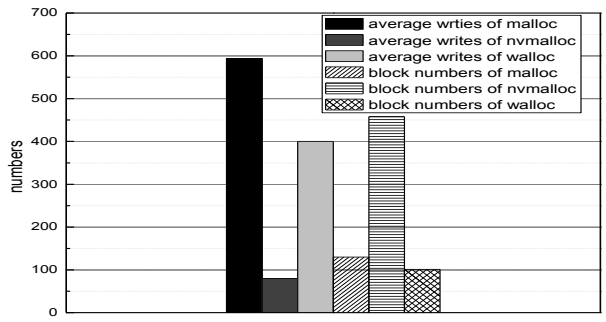

Fig. 10. average memory writes and total NVM lines numbers of metadata with uniform data distribution using malloc, NVMalloc and WAlloc

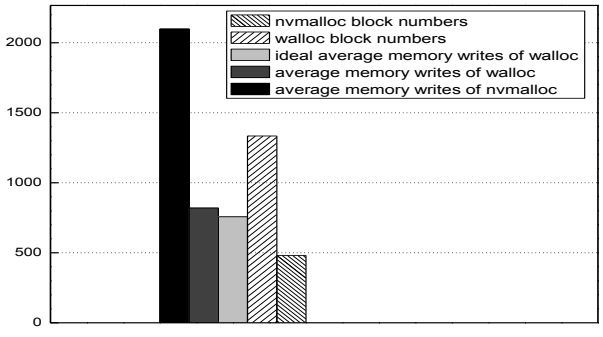

Fig. 11. average memory writes and total NVM lines of data using NVMalloc and WAlloc

V. RELATED WORK

Much of research in managing NVM when integrating it into computer system has examined how to extend its endurance in the past few years. At hardware level, ping zhou et al.[9] proposed to balance write hot spots by reducing redundant writes, fine-grained row shifting and coarse-grained segment swapping; Lei Jiang et al.[10] propose to minimize iteration writes to PCM cells by write truncation; and many ways related to expose translation between the physical address and the logical addresses to the memory controller have also been proposed [11-13]; Alexandre P. Ferreira et al. [27] uses DRAM to "absorb" the writes to PCM; Rodríguez-Rodríguez et al[37] converges on the replacement algorithms for the last-level cache (LLC) coalescing as many modifications to a block as possible in the LLC to reduce the write traffic to memory. In order to do so, a slice of hardware reinforcement should be made. These efforts are orthogonal to our research on writes reduction of non-volatile main memory allocator. At software level, a sea of optimize techniques have been proposed to minimize the total number of memory writes to PCM in terms of program variables, application access patterns[14-19]. Access patterns of embedded systems' applications are fixed such that facilitate optimization at compilation level. Mnemosyne[28] and NV-heaps[29], focusing on an uniform program interface for memory allocation in NVM, while the endurance problem is not considering much. Also, considering that its characteristic of memory-like performance and storage-like capacity, quite a few initial efforts of managing PCM with memory-storage model have been made[20-24]. However, Organizing PCM main memory through file system leads to hardly inevitable overhead of virtual file system during every data accesses.

The most related work to the proposed WAlloc is NVMalloc[25], designing a wear-leveling policy with timestamp. To be specific, firstly, it curbs write frequency of memory block no higher than 1/T. Secondly, NVMalloc decouples metadata and data with a bitmap to track memory block states to reduce metadata writes. Finally, NVMalloc melts metadata writes with DRAM. Nevertheless, it's extremely arduous to make trade-offs between memory footprint and wear-leveling effectiveness and in-place data updates reduce life expectancy of NVM.

VI. CONCLUSION AND FUTURE WORK

In this paper, we quantify the wear problems of memory allocators used for non-volatile main memory such as memory allocation policy (e.g. FIFO, LIFO), unnecessary metadata writes and in-place data writes. Based on our observations, we propose a wear-aware allocator, WAlloc, which decouples metadata and data management, stores volatile metadata in DRAM to reduce small writes, allocates memory with the Less Allocated First Out policy to level non-volatile memory writes and redirects data writes to more free memory to improve lifetime of NVRAM. The experimental results show that metadata writes wear-leveling of WAlloc outperforms NVMalloc by 30% and 60% under workloads of random and uniform data size distribution, respectively; and

average memory writes(in 64B) reduced by average 1.5X comparing with malloc. In our future work, we will continue to optimize the wear-leveling of data writes or use log-structural way [31] to organize non-volatile memory. Reducing the overhead of LAFO policy, and exploring WAlloc in multiple processor environment.

VII. ACKNOWLEDGMENT

We are grateful to our anonymous reviewers for their suggestions to improve this paper. This work is supported by the National High-Tech Research and Development Projects (863) and the National Natural Science Foundation of China under Grant Nos. 2015AA015305, 61232003, 61332003, 61202121.

REFERENCES

[1] Mandelman, Jack A., et al. "Challenges and future directions for the scaling of dynamic random-access memory (DRAM)." IBM Journal of Research and Development 46.2.3 (2002): 187-212.

[2] Mueller, W., et al. "Challenges for the DRAM Cell Scaling to 40nm." IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.. 2005.

[3] Lefurgy, Charles, et al. "Energy management for commercial servers."Computer 36.12 (2003): 39-48.

[4] Barroso, Luiz André, Jimmy Clidaras, and Urs Hölzle. "The datacenter as a computer: An introduction to the design of warehouse-scale machines."Synthesis lectures on computer architecture 8.3 (2013): 1-154.

[5] Raoux, Simone, et al. "Phase-change random access memory: A scalable technology." IBM Journal of Research and Development 52.4.5 (2008): 465-479.

[6] Dieny, B., et al. "Spin-dependent phenomena and their implementation in spintronic devices." VLSI Technology, Systems and Applications, 2008. VLSI-TSA 2008. International Symposium on. IEEE, 2008.

[7] Ho, Yenpo, Garng M. Huang, and Peng Li. "Nonvolatile memristor memory: device characteristics and design implications." Computer-Aided Design-Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on. IEEE, 2009.

[8] Xue, Chun Jason, et al. "Emerging non-volatile memories: opportunities and challenges." Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis. ACM, 2011.

[9] Zhou, Ping, et al. "A durable and energy efficient main memory using phase change memory technology." ACM SIGARCH computer architecture news. Vol. 37. No. 3. ACM, 2009.

[10] Jiang, Lei, et al. "Improving write operations in MLC phase change memory."High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on. IEEE, 2012.

[11] Qureshi, Moinuddin K., et al. "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling." Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture. ACM, 2009.

[12] Qureshi, Moinuddin K., et al. "Practical and secure pcm systems by online detection of malicious write streams." High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on. IEEE, 2011.

[13] Shao, Zili, Naehyuck Chang, and Nikil Dutt. "PTL: PCM translation layer."VLSI (ISVLSI), 2012 IEEE Computer Society Annual Symposium on. IEEE, 2012.

[14] Li, Qingan, et al. "A wear-leveling-aware dynamic stack for pcm memory in embedded systems." Proceedings of the conference on Design, Automation & Test in Europe. European Design and Automation Association, 2014.

[15] Hu, Jingtong, et al. "Software enabled wear-leveling for hybrid PCM main memory on embedded systems." Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013. IEEE, 2013.

[16] Liu, Duo, et al. "Curling-PCM: Application-specific wear leveling for phase change memory based embedded systems." Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific. IEEE, 2013.

[17] Hu, Jingtong, et al. "Reducing write activities on non-volatile memories in embedded CMPs via data migration and recomputation." Design Automation Conference (DAC), 2010 47th ACM/IEEE. IEEE, 2010.

[18] Hu, Jingtong, et al. "Minimizing write activities to non-volatile memory via scheduling and recomputation." Application specific processors (SASP), 2010 IEEE 8th symposium on. IEEE, 2010.

[19] Hu, Jingtong, et al. "Write activity minimization for nonvolatile main memory via scheduling and recomputation." Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 30.4 (2011): 584-592.

[20] Chang, Hung-Sheng, et al. "Marching-based wear-leveling for PCM-based storage systems." ACM Transactions on Design Automation of Electronic Systems (TODAES) 20.2 (2015): 25.

[21] Condit, Jeremy, et al. "Better I/O through byte-addressable, persistent memory." Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. ACM, 2009.

[22] Wu, Xiaojian, and A. L. Reddy. "SCMFS: a file system for storage class memory." Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2011.

[23] Chen, Feng, Michael P. Mesnier, and Seungyong Hahn. "A protected block device for persistent memory." Mass Storage Systems and Technologies (MSST), 2014 30th Symposium on. IEEE, 2014.

[24] Dulloor, Subramanya R., et al. "System software for persistent memory."Proceedings of the Ninth European Conference on Computer Systems. ACM, 2014.

[25] Moraru, Iulian, et al. "Consistent, durable, and safe memory management for byte-addressable non volatile main memory." Proceedings of the First ACM SIGOPS Conference on Timely Results in Operating Systems. ACM, 2013.

[26] Seznec, Andre. "A phase change memory as a secure main memory."Computer Architecture Letters 9.1 (2010): 5-8.

[27] Ferreira, Alexandre P., et al. "Increasing PCM main memory lifetime."Proceedings of the conference on design, automation and test in Europe. European Design and Automation Association, 2010.

[28] Volos, Haris, Andres Jaan Tack, and Michael M. Swift. "Mnemosyne: Lightweight persistent memory." ACM SIGPLAN Notices 46.3 (2011): 91-104.

[29] Coburn, Joel, et al. "NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories." ACM SIGARCH Computer Architecture News. Vol. 39. No. 1. ACM, 2011.

[30] Luk, Chi-Keung, et al. "Pin: building customized program analysis tools with dynamic instrumentation." ACM Sigplan Notices. Vol. 40. No. 6. ACM, 2005.

[31] Rumble, Stephen M., Ankita Kejriwal, and John K. Ousterhout. "Log-structured memory for DRAM-based storage." FAST. Vol. 1. 2014.

[32] Venkataraman, Shivaram, et al. "Consistent and Durable Data Structures for Non-Volatile Byte-Addressable Memory." FAST. 2011.

[33] Yang, Jun, et al. "NV-Tree: reducing consistency cost for NVM-based single level systems." Proceedings of the 13th USENIX Conference on File and Storage Technologies. USENIX Association, 2015.

[34] Chen, Shimin, and Qin Jin. "Persistent b+-trees in non-volatile main memory." Proceedings of the VLDB Endowment 8.7 (2015): 786-797.

[35] Xu-Dong, Fang, Tang Yu-Hua, and Wu Jun-Jie. "SPICE modeling of memristors with multilevel resistance states." Chinese Physics B 21.9 (2012): 098901.

[36] Li, Xu, et al. "NV-process: a fault-tolerance process model based on non-volatile memory." Proceedings of the Asia-Pacific Workshop on Systems. ACM, 2012.

[37] Rodríguez-Rodríguez, R., et al. "Write-Aware Replacement Policies for PCM-Based Systems." The Computer Journal (2014): bxu104.

[38] Zhang, Sen. "Introduction." Electric-Field Control of Magnetization and Electronic Transport in Ferromagnetic/Ferroelectric Heterostructures. Springer Berlin Heidelberg, 2014. 1-48.