Efficient Wear Leveling for Inodes of File Systems on Persistent Memories

Xianzhang Chen*†, Edwin H.-M. Sha*, Yuansong Zeng*, Chaoshu Yang*, Weiwen Jiang*, and Qingfeng Zhuge‡⋆
* College of Computer Science, Chongqing University
† College of Communication Engineering, Chongqing University
‡ School of Computer Science and Software Engineering, East China Normal University
Email: {xzchen109, edwinsha, cquysz, yangchaoshu, jiang.wwen, qfzhuge}@gmail.com

***Abstract*—Existing persistent memory file systems achieve high-performance file accesses by exploiting advanced characteristics of persistent memories (PMs), such as PCM. However, they ignore the limited endurance of PMs. Particularly, the frequently updated inodes are stored on fixed locations throughout their lifetime, which can easily damage PM with common file operations. To address such issues, we propose a new mechanism, Virtualized Inode (VInode), for the wear leveling of inodes of persistent memory file systems. In VInode, we develop an algorithm called *Pages as Communicating Vessels (PCV)* to efficiently find and migrate the heavily written inodes. We implement VInode in SIMFS, a typical persistent memory file system. Experiments are conducted with well-known benchmarks. Compared with original SIMFS, experimental results show that VInode can reduce the maximum value and standard deviation of the write counts of pages to 1800x and 6200x lower, respectively.**

I. Introduction

Emerging persistent memories (PMs), such as phase change memory (PCM)[1] and 3D Xpoint [2], become promising data storages for their advanced features. Existing persistent memory file systems [3], [4], such as PMFS [5] and SIMFS [6], exploit the advantages of PMs to achieve high-performance data accesses and efficient consistency guarantees. Persistent memory storages, on the other hand, have a critical defect, i.e., limited endurance [7], [8], [9]. Persistent memory file systems need wear-leveling technologies to protect the underlying PM storage from being damaged. Particularly, index node (inode) section is one of the most heavily worn section in file system. Our experiments show that if we keep writing one byte to a file of PMFS or SIMFS 10^8 times, the PM storage can be damaged within *6 seconds* for the physical page storing the corresponding inode reaches endurance.

There are two reasons make inodes so fragile to the wear issues. First, inodes are the most frequently updated data in a file system. Despite of the operations updating both file data and inodes, such as *write*(), many file operations directly modify inodes. For example, *link*() and *unlink*() update the link count in the inode of the target file without modifying its file data. Second, inodes are stored on fixed locations throughout their lifetime. Nevertheless, most existing file systems on PMs ignore the design of wear leveling for inodes.

In this paper, we focus on solving the wear leveling problem for inodes of persistent memory file systems. We propose a mechanism called *Virtualized Inode (VInode)* to achieve wear-leveling of physical pages for storing inodes. The proposed VInode mechanism is composed of two techniques. First, *Inode-virtualization* decomposes inodes from “fixed” physical locations by a deflection table. An inode can be re-mapped to different physical locations. Second, *Inode-migration* dynamically moves frequently written inodes (i.e., hot inodes) to less written pages (i.e., cold pages) during runtime. We propose an algorithm, *Pages as Communicating Vessels*, to select candidate hot inodes and target cold pages. VInode can distribute writes of inodes to physical pages evenly by dynamically adjusting the true locations of hot inodes.

To evaluate the proposed VInode mechanism, we implement VInode in SIMFS [6], a state-of-the-art file system for managing persistent memories. We evaluate the effect and efficiency of VInode by the widely used benchmark Filebench[10]. Experimental results show that VInode brings significant wear-leveling improvement for inode section. Compared with the original SIMFS that ignores wear leveling, the version that uses VInode shows 1800× and 6200× lower maximum value and standard deviation of write counts of pages, respectively. Simultaneously, the average performance overhead of VInode is less than 5% for various sizes of file requests.

II. Background and motivation

A. Existing Persistent Memory File Systems

With the development of new persistent memories, a set of persistent memory file systems have been proposed to break the traditional I/O bottleneck. The design of existing persistent memory file systems mainly focus on improving performance or data consistency. For example, BPFS [3] proposes “short-circuit shadow paging”, an improved shadow paging [11] mechanism exploiting the byte-addressability of persistent memory, to provide highly efficient data consistency guarantees. SIMFS [6] proposes “file virtual address space” to boost the file access performance by utilizing virtual address space and hardware MMU in processor.

The limited endurance problem of persistent memory, however, are not well considered in the design of existing persistent memory file systems. The underlying persistent memory can be easily wear out by common file system operations, especially the pages for storing inodes. It is because that inodes are the most frequently updated data in file system. Unfortunately, the wear problem of pages for inodes are ignored in existing persistent memory file systems. Therefore, this paper focuses on the wear leveling problem of inode .

B. Wear Leveling Problem of Inodes

There are two types of designs for inode section in existing persistent memory file systems: tree structures and arrays. For example, PMFS[5], SanGuo[12], and HiNFS[13] use B-tree to organize all the inodes. The inodes are stored in the leaf node of the B-tree. SCMFS[14] and SIMFS[6] organize the inodes by an array. Beyond these data structures, we observe that they have one thing in common: **inodes never move throughout their lifetime**. Therefore, all the

⋆Qingfeng Zhuge is the corresponding author.

This work is supported by the National 863 Program of China (Grant No. 2015AA015304), NSFC (Grant No. 61472052), and China Postdoctoral Science Foundation (Grant No. 2017M620412).

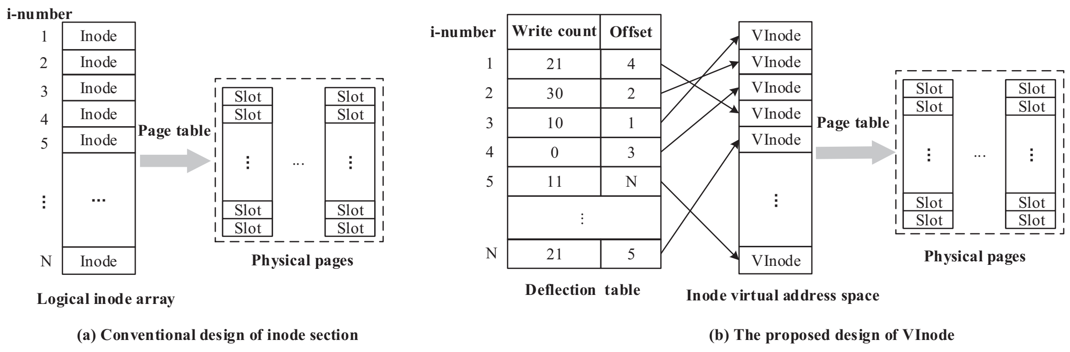

i-number
Inode
Page table
Slot
Physical pages
Logical inode array
(a) Conventional design of inode section
Write count
Offset
Deflection table
VInode
Inode virtual address space
(b) The proposed design of VInode

Fig. 1: Comparison of conventional inode section and the proposed virtualized inode section.

writes to a given inode are performed on a single physical page. This design makes persistent memory fragile to heavy written files.

Moreover, **the write counts of inodes in a file system generally vary a lot**. For example, most system files are read-only and their inodes are seldom updated, whereas some directory files and user's data files are frequently updated. This common feature of systems result in unbalanced write distribution over the pages for storing inodes. Since each page can store multiple inodes, an inappropriate inode combination on a page can expedite the damage of the page. We believe the way to tackle the wear leveling problem of inodes is to dynamically migrate the inodes.

III. VInode Design and Implementation

A. Inode-virtualization

Each inode in a file system has a unique ID called "inode number" (i-number). In existing persistent memory file systems, i-number is used as an index for searching the corresponding inode in the tree or array of inodes. For example, SIMFS organizes logical inodes as an array, as shown in Fig. 1(a). An i-number is the index of an element in the array. The sizes of all the inodes are the same. The array of logical inodes is mapped to multiple physical memory pages. Each physical page is divided into multiple slots. The mapping between logical inodes and physical memory pages are fixed since the file system is mounted. Thus, each slot is one-to-one bound with a logical inode permanently. This permanent mapping restricts the wear leveling of inodes.

We transfer the permanent bind to a convertible bind by a new design of Virtualized inode (VInode). On one hand, VInodes are organized in a contiguous "inode virtual address space", which is mapped to physical pages by page table, as shown in Fig. 1(b). Each VInode is bound with a slot in physical page. Each VInode has a unique *offset* in inode virtual address space. On the other hand, logical inodes are mapped to VInodes via a deflection table. The deflection table is updated only if an inode is migrated. Thus, the write count of the deflection table is lower than that of inode pages. Each logical inode has an entry in the deflection table that stores the corresponding offset and the write count of the logical inode. As a consequence, a logical inode can be mapped to different slots dynamically by changing its offset. Similar to array-structured inode sections, tree-structured inode sections can also be virtualized by constructing a deflection table for inodes in the leaf nodes.

The inode virtual address space is a segment of the kernel virtual space. To access an inode, the file system first gets the offset of the logical inode via i-number. Then, the file system calculates the virtual address of the inode by adding the offset with the beginning virtual address of inode virtual address space. Thus, the overhead for locating inode is negligible.

B. Inode-migration

In Inode-migration, we need to decide how to achieve wear leveling by migrating inodes. The wear leveling problem of inodes can be defined as follows. Given n inodes that each inode has a write count and m physical pages that each page has k slots for storing inodes, how to place n inodes on m pages such that the standard deviation of the write counts of pages is minimized? The key is to find a proper combination of inodes for each page such that the pages can achieve the same write count. The write counts of inodes, however, are dynamically changing in a run-time system. Therefore, we propose to solve the problem by dynamically migrating frequently updated inodes during run-time.

Algorithm 1: The PCV algorithm

```
Input: $WI_i$: write counts of inode $I_i$; $WP(I_i)$: write count of
       the physical page storing $I_i$; $WG_j$: write count of the
       $j$th subset of physical pages;
Output: The relative storing position $o_i$ for $I_i$;
1  if $WI_i \leq \alpha$ then
2      return;
3  else
4      $WL \leftarrow \sum_{j=1}^{n_G} WG_j/m$;
5      if $WP(I_i) - WL > \beta$ then
6          $P_{candidate} \leftarrow$ the least written page of the least written
           page group;
7          if find a free slot $Slot_s$ in $P_{candidate}$ then
8              Copy $I_i$ to $Slot_s$;
9              Reclaim the original slot of $I_i$;
10             $o_i \leftarrow Offset(Slot_s)$;
11         else
12             $I_c \leftarrow$ the least recently accessed inode in
               $P_{candidate}$;
13             Swap $I_i$ with $I_c$;
14             Exchange $o_i$ and $o_c$;
15     $WI_i \leftarrow 0$;
```

A simple way to select the frequently updated inodes for migration is to set up a fixed threshold T. Once the write counts of an inode reaches the threshold, the file system moves the inode to one of the least written page. Actually, each migration brings an additional write (In addition to the file requests of user) to PM rather than cut down the total number of writes to PM. Therefore, the better wear leveling we try to achieve using fixed threshold, the more addition writes act on PM. Fortunately, there are two observations for designing a better migration method. First, it is not necessary to migrate an inode if the write counts of the underlying page is less than the average write counts of pages. Second, only the future number of writes of an inode impacts the page that the inode will be migrated to. Thus, a migration needs to consider the writing counts of pages and the features of inodes simultaneously.

Based on the observations, we propose an algorithm called *Pages as Communicating Vessels (PCV)* for migrating the inodes. We call the average write count of all pages of inodes as "waterline (WL)". The main idea of *PCV* is to achieve wear-leveling by aligning the write count of each page to WL. We use both the write count of an inode and WL to determine whether the inode need to be migrated or not. The detailed algorithm is shown in Algorithm 1.

In the algorithm, we set two thresholds to determine if an inode I_i need to be migrated. First, *PCV* uses threshold α to judge if I_i is highly written. If the write count WI_i of I_i achieves α, I_i becomes a candidate for migration. Suppose the candidate inode I_i is located on a less written page, it is not necessary to migrate I_i. Therefore, the *PCV* algorithm considers the relation between the waterline WL and the write count of the page $P(I_i)$ that stores the candidate inode I_i. For page $P(I_i)$, if its write count $WP(I_i) - WL$ is larger than threshold β, it means that the page is relatively heavy written. In this case, I_i should be migrated. During migration, we first find the least written slot $Slot_s$ from a relatively less written page $P_{candidate}$. Then, if $Slot_s$ is free, *PCV* write the candidate inode I_i to $Slot_s$; otherwise, *PCV* exchange I_i with the least recently accessed inode in $P_{candidate}$.

IV. Evaluation

A. Experimental Setups

We implement two versions of VInode in SIMFS[6], the state-of-the-art persistent memory file system. One version, *VInode-Fixed*, migrates an inode to a less written page once the write count meets a fixed threshold. The other version, *VInode-PCV*, migrates inodes using the proposed PCV algorithm. The original SIMFS has no wear-leveling (denoted by *NoWL*) for inodes. The experiments are conducted on a server equipped with an Intel E5-2630 v3 processor and 128GB DRAM. We configure 64GB DRAM to sand for persistent memory. The system runs with Linux kernel 4.4.4. We use the widely used benchmark Filebench [10] to evaluate the effect of wear leveling and performance.

B. Experimental results

We select three workloads from Filebench [10], including Webproxy, Webserver, and Fileserver. Fig. 2 shows the write distribution of three file systems. VInode-PCV achieves significant wear-leveling improvement over NoWL and VInode-Fixed.

Fig. 2 (c), (f), and (i) show the write distribution of NoWL. The write distributions of NoWL are highly uneven: all the writes are performed on 3-4 pages and the rest of pages are all remain brand new. Fig. 2 (b), (e), and (h) show the write distribution of VInode-Fixed. The threshold is set to 30. Each page is capable of storing 32 inodes. Thus, the write counts of pages scatter in a range around 1950 to 2900. The maximum write count of VInode-Fixed is 2942, 2929, and 2917 for workload Webproxy, Webserver, and Fileserver, respectively. The standard deviation of VInode-Fixed is 307.72, 401.88 and 349.31 for workload Webproxy, Webserver, and Fileserver, respectively. Compared with NoWL, VInode-Fixed improves the standard deviation of write counts of pages 257× on average. It is because VInode-Fixed dynamically migrates hot inodes to less written pages.

TABLE I: Total write counts of inodes.

	NoWL	VInode-Fixed	VInode-PCV
Webproxy	10,763,002	10,895,288	10,875,325
Webserver	13,699,568	14,019,267	13,849,880
Fileserver	10,744,671	11,082,760	10,869,062

On the other side, the dynamic migration brings addition writes to persistent memory for each migration is actually an addition write to persistent memory, as shown in Table I. For example, it brings 3.15% additional writes for Fileserver. If we want to better wear-leveling via VInode-Fixed, we need to set a lower threshold. However, a lower threshold results in more additional writes. Thus, it is difficult to handle the trade-off for VInode-Fixed in a flexible system.

Fig. 2 (a), (d), and (g) show the write distribution of VInode-PCV. The threshold α for selecting candidate inodes is set to 30 and the threshold β for migration is set to two times of α. The maximum write count of VInode-PCV is 2138, 2701, and 2410 for workload Webproxy, Webserver, and Fileserver, respectively. The standard deviation of VInode-PCV is 27.65, 24.1, and 13.18 for workload Webproxy, Webserver, and Fileserver, respectively. Compared with NoWL and VInode-Fixed, VInode-PCV improves the standard deviation of write counts of pages 4574× and 18× on average, respectively. For Fileserver, VInode-PCV achieves 6523× and 1622× improvement over NoWL for standard deviation and maximum write count, respectively. It is because that VInode-PCV not only considers the write count of inodes, it also takes the difference between write count of the corresponding page and the waterline into account.

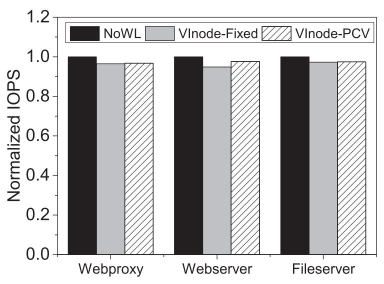
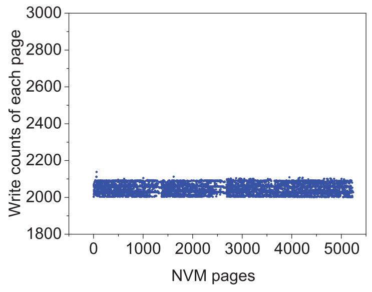


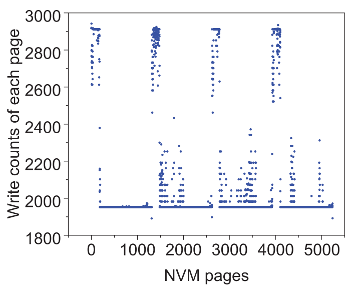
Fig. 3: Comparison of IOPS.

Similarly, VInode-PCV avoids many unnecessary migrations, such as migrate a hot inode from a seldom written page to another seldom written page. As a consequence, the additional writes of VInode-PCV is 1.1% on average, which is 2.03 times less than VInode-Fixed, as shown in Table I. Moreover, the write count of pages can be controlled closely around the waterline. Hereby, the dots in Fig. 2 (a), (d), and (g) seem less than these in Fig. 2 (b), (e), and (h) for more pages have the same write counts in VInode-PCV.

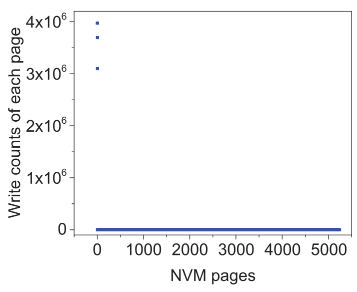
Next, we evaluate the overall performance. We run the workloads with default configurations. The experiments are shown in Fig. 3. The collected IOPS are normalized to these of NoWL. The IOPS of VInode-Fixed is 3.5%, 5.2%, and 2.7% less than that of NoWL for workload Webproxy, Webserver, and Fileserver, respectively. The IOPS of VInode-PCV is 3.2%, 2.4%, and 2.5% less than that of NoWL for workload Webproxy, Webserver, and Fileserver, respectively. It is because that VInode-PCV avoids unnecessary migrations.



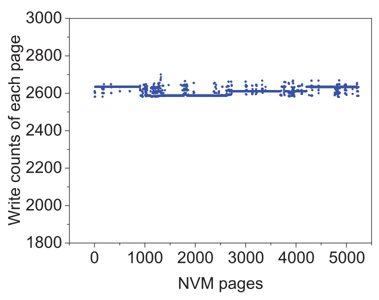
(a) Webproxy on VINode-PCV



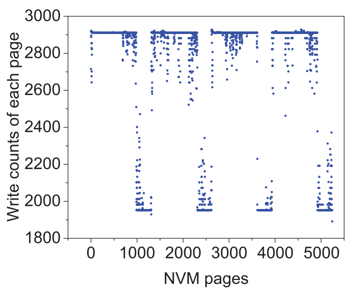
(b) Webproxy on VINode-Fixed



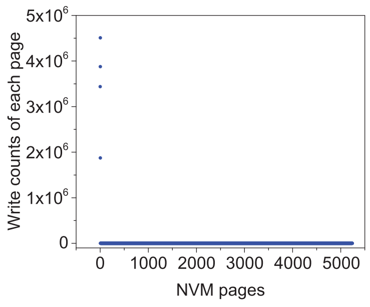
(c) Webproxy on NoWL



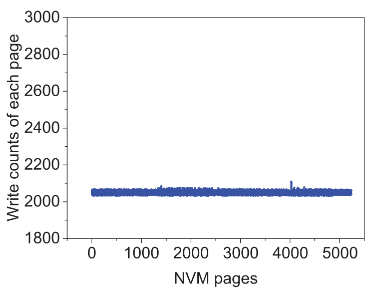
(d) Webserver on VINode-PCV



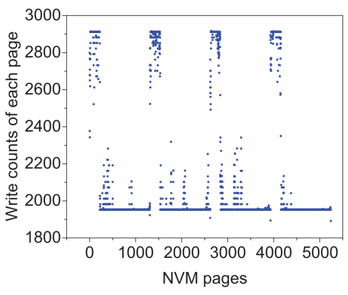
(e) Webserver on VINode-fixed



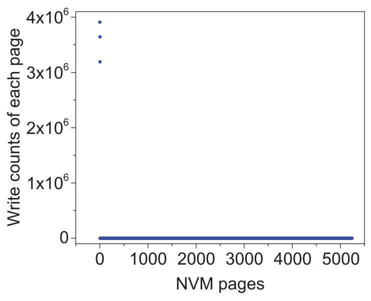
(f) Webserver on NoWL



(g) Fileserver on VINode-PCV



(h) Fileserver on VINode-fixed



(i) Fileserver on NoWL

Fig. 2: Comparison of write distribution in NoWL, VInode-Fixed, and VInode-PCV.

V. Conclusion

In this paper, we studied the wear leveling problem of inodes of file systems on persistent memory, which is generally the most frequently updated data in a file system. We observed that the inodes of existing file systems are fixed on physical locations throughout their lifetime. To this end, we proposed a mechanism called VInode to select and migrate the hot inodes on heavily written pages to the lightly written pages. Experimental results show that the proposed VInode mechanism achieves significant wear-leveling improvement with negligible performance overhead.

References

[1] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ISCA*, 2009, pp. 14–23.

[2] https://newsroom.intel.com/news-releases/intel-and-micron-produce-breakthrough-memory-technology/, 2015.

[3] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee, "Better i/o through byte-addressable, persistent memory," in *ACM SOSP*, 2009, pp. 133–146.

[4] J. Xu and S. Swanson, "Nova: A log-structured file system for hybrid volatile/non-volatile main memories," in *USENIX FAST*, 2016, pp. 323–338.

[5] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System software for persistent memory," in *EuroSys*, 2014, pp. 15:1–15:15.

[6] E. H.-M. Sha, X. Chen, Q. Zhuge, L. Shi, and W. Jiang, "A new design of in-memory file system based on file virtual address framework," *IEEE Transactions on Computers*, vol. 65, no. 10, pp. 2959–2972, Oct 2016.

[7] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *ISCA*, 2009, pp. 24–33.

[8] F. Huang, D. Feng, Y. Hua, and W. Zhou, "A wear-leveling-aware counter mode for data encryption in non-volatile memories," in *DATE*, 2017, pp. 910–913.

[9] H. S. P. Wong, S. Raoux, S. B. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

[10] V. Tarasov, E. Zadok, and S. Shepler, "Filebench: A flexible framework for file system benchmarking," *;login*, vol. 41, no. 1, pp. 6–12, 2016.

[11] C. Mohan, "Repeating history beyond aries," *VLDB*, pp. 1–17, 1999.

[12] K. Zeng, Y. Lu, H. Wan, and J. Shu, "Efficient storage management for aged file systems on persistent memory," in *DATE*, 2017, pp. 1773–1778.

[13] J. Ou, J. Shu, and Y. Lu, "A high performance file system for non-volatile main memory," in *EuroSys*, 2016, pp. 12:1–12:16.

[14] X. Wu, S. Qiu, and A. L. Narasimha Reddy, "Scmfs: A file system for storage class memory and its extensions," *ACM Transactions on Storage*, vol. 9, no. 3, pp. 1 – 11, 2013.