Optimizing Performance of Persistent Memory File Systems using Virtual Superpages

Chaoshu Yang†, Duo Liu†,*, Runyu Zhang†, Xianzhang Chen†, Shun Nie†,
Qingfeng Zhuge‡, and Edwin H.-M. Sha‡
† College of Computer Science, Chongqing University, Chongqing, China
‡ School of Computer Science and Software Engineering, East China Normal University, Shanghai, China.

***Abstract*—Existing persistent memory file systems can significantly improve the performance by utilizing the advantages of emerging Persistent Memories (PMs). Especially, they can employ superpages (e.g., 2MB a page) of PMs to alleviate the overhead of locating file data and reduce TLB misses. Unfortunately, superpage also induces two critical problems. First, the data consistency of file systems using superpages causes severe write amplification during overwrite of file data. Second, existing management of superpages may lead to large waste of PM space. In this paper, we propose a Virtual Superpage Mechanism (VSM) to solve the problems by taking advantages of virtual address space. On one hand, VSM adopts multi-grained copy-on-write mechanism to reduce the write amplification while ensuring data consistency. On the other hand, VSM presents zero-copy file data migration mechanism to eliminate the loss of space utilization efficiency caused by superpages. We implement the proposed VSM mechanism in Linux kernel based on PMFS. Compared with the original PMFS and NOVA, the experimental results show that VSM improves 36% and 14% on average for write and read performance, respectively. Meanwhile, VSM can achieve the same space utilization efficiency of file system that uses the normal 4KB pages to organize files.**

I. Introduction

Persistent Memories (PMs), such as Phase Change Memory (PCM) [1] and 3D-XPoint [2], are promised to revolutionize storage systems by providing non-volatility, byte-addressability, and low latency. The state-of-the-art persistent memory file systems, such as PMFS [3] and NOVA [4], exploit the advanced characteristics of PMs to avoid the overheads of block-oriented I/O stacks. Another key feature is that PMs can be directly linked to the memory bus and accessed by load/store instructions [5]. Hence, PM is able to be managed as 2MB or 1GB *superpages* by utilizing the memory management hardware of processor. In this case, persistent memory file systems can efficiently locate a data page with a shorter file index and reduce the overhead of space management for the large files. Moreover, the file systems using superpages can also benefit from high TLB hit rate than 4KB pages.

Unfortunately, the file systems organizing file data with superpages lead to severe write amplification that largely reduces performance. Generally, existing persistent memory file systems employ the Copy-on-Write (CoW) [6] mechanism to guarantee data consistency, which writes new data and over-written data to newly allocated pages and updates the corresponding entries of file data index. However, CoW may lead to massive data migration when the file system using superpages, especially when the size of over-write data is far less than the size of superpage. For example, the file system needs to migrate 2044KB data into the new allocated superpage for updating 4KB data in a 2MB superpage. Moreover, the file systems using superpages may have huge waste of memory space, since the idle space in the last superpage of each file cannot be used by other files. Assuming that the last superpage of a file has 1MB valid data, there is 1MB or 1023MB waste of memory space for a 2MB or 1GB superpage, respectively.

In this paper, we present an efficient Virtual Superpages Mechanism (VSM) to solve these problems caused by superpages. The main idea of VSM is to employ the virtual superpages to avoid the weakness and exploit the advantages of superpages for file systems via existing virtual-to-physical-address mapping of memory space. VSM consists of two key techniques, Multi-grained Copy-on-Write (MCoW) and Zero-copy File Data Migration (ZFDM) mechanisms, are used to reduce the write amplification while improving the space utilization efficiency. We implement the proposed VSM mechanism in Linux kernel based on PMFS. Compared with superpages, the experimental results show that VSM can significantly reduce the write amplification. Moreover, VSM can achieve the same space utilization efficiency of file system that uses the normal 4KB pages to organize files. Compared with PMFS and NOVA adopt 4KB pages (i.e., 4KB) to organize file data, VSM improves 36% and 14% on average for write and read performance.

The main contributions are as follows:

- We conduct in-depth investigations to reveal the severe write amplification and analyze the space utilization efficiency of superpages.
- The proposed VSM uses virtual superpages to mitigate the write amplification caused by superpages.
- The proposed VSM provides zero-copy file data migration mechanism to improve the space utilization efficiency of superpages.
- Extensive experiments are conducted to evaluate the performance of VSM. The experimental results show that VSM outperforms existing solutions.

The rest of this paper is organized as follows. In Section II,

*Corresponding author: Duo Liu, College of Computer Science, Chongqing University, Chongqing, China. E-mail: liuduo@cqu.edu.cn.

978-3-9819263-4-7/DATE20/©2020 EDAA

we introduce the background and discuss the motivation. We present the design and implementation of VSM in Section III. The experimental results are detailed in Section IV. Finally, we draw a conclusion in Section V.

II. Background And Motivation

A. Write Amplification Problem in Data Consistency

Write-Ahead Logging (WAL) and Copy-on-Write (CoW) are widely adopted data consistency mechanisms for ensuring the reliability of file systems. WAL has the "double-writes problem", where the file system needs to write the new data and the original file data each once for updating file data. Different from WAL, CoW only updates the file data once. First, the file system writes the new data to a newly allocated location. Then, the file system redirects the corresponding entry in the file data index to the new location via small atomic write operation. Therefore, existing persistent memory file systems, such as PMFS[3], SIMFS[7], and NOVA[4], adopt CoW rather than WAL as the data consistency mechanism.

Unfortunately, the over-write operation will lead to severe write amplification due to the relocation of the CoW mechanism when the file data stored on superpages. For example, if there is a 6MB-sized file organized by superpages (i.e., 2MB), the file data occupies three superpages in total, namely P_1, P_2, and P_3. Assuming that the over-write process needs to update 4KB data in P_1, according to the principle of CoW mechanism, the file systems need to combines the original data (511 $\times$ 4KB) with the over-write data, and write them to a new superpage. Then, file systems revise the corresponding entry in the file data index to redirect to the new superpage. Accordingly, the write amplification is 511 of which overhead is far greater than WAL mechanism.

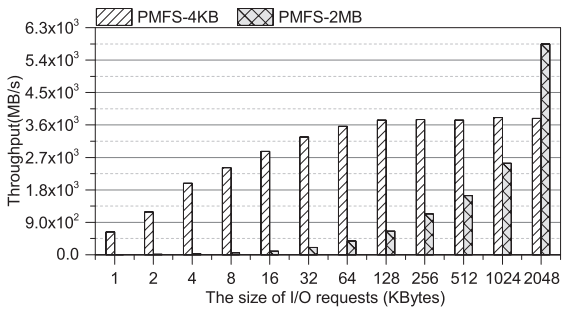


Fig. 1: The throughput of sequential write in PMFS.

To quantify the overhead of the CoW mechanism employed in superpages, we measure the over-write performance of PMFS utilizing superpages and normal pages to organize file data with the widely-used benchmark FIO [8]. We set one thread to write the two existing 8GB-sized files with various I/O sizes. Meanwhile, the data of these files is organized by 4KB pages (denoted by $PMFS{-}4KB$) and 2MB superpages (denoted by $PMFS{-}2MB$), respectively. Then, we calculate the throughput (MB/s, MB per second) of sequential write for each write granularity.

As shown in Fig. 1, the PMFS-2MB suffers dramatic performance degradation when the write granularity decreases. Moreover, PMFS-4KB significantly outperforms PMFS-2MB when the write granularity increases from 1KB to 1024KB. Meanwhile, the gap of throughput largely increases when the write granularity decreases. Especially, the throughput of PMFS-4KB is up to 98$\times$ higher than that of PMFS-2MB when the write granularity is 1KB. This is mainly attributed to the severe write amplification for PMFS-2MB. However, when the write granularity is 2048KB, the write performance of PMFS-2MB is 1.5$\times$ than PMFS-4KB. This is because that there is no write amplification problem in this case for both PMFS-4KB and PMFS-2MB, while PMFS still needs to allocate 512 normal pages to store the over-write data for PMFS-4KB. In contrast, PMFS only need to allocate a superpage for PMFS-2MB. Therefore, existing CoW mechanism of persistent memory file systems can lead to severe write amplification for files organized by superpages, which seriously degrades the over-write performance of file systems.

B. Space Utilization of Superpages

A superpage is a segment of continuous physical memory of PM, whose size is larger than a normal page. For file systems, an in-used data page only belongs to a certain file. Therefore, when the valid data on the last page of a file is less than the size of a page, the space of the last superpage in this file is severely wasted.

$$S_{waste} = S_{page} - F_{totalsize} \ \% \ S_{page} \tag{1}$$

The space waste is calculated by equation 1. The S_{waste}, S_{page}, and $F_{totalsize}$ indicate the size of space waste, the page size, and the total size of a file, respectively. According to equation 1, larger sizes of the page can induce more waste of space. For example, when a file data is 1GB + 2KB, the waste space is 2KB, 2046KB, and 1023MB + 1022KB when file systems employ 4KB, 2MB, and 1GB page size to organize file data, respectively. Based on the above analysis, we reveal that file systems utilizing superpages to organize file data may lead to severe space waste.

C. Motivation

As mentioned above, existing file systems use superpages to organize file data can lead to severe write amplification for over-write operations and reduce the space utilization efficiency. Motivated by these observations, we propose VSM, an efficient virtual superpages mechanism via virtual address space, to solve these problems. The MCoW and ZFDM of VSM are used to reduce the write amplification and the space waste caused by superpages, respectively.

III. Design and Implementation

A. Overview

Existing virtual memory techniques (e.g., x86 architecture) require that the file data in PM can only be accessed by the virtual address. Accordingly, as shown in Fig. 2, VSM maps the entire PM room into a continuous and equivalently-sized amount of kernel virtual address space. Moreover, in order to

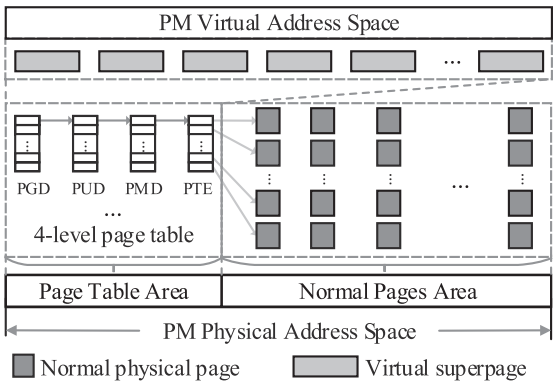


Fig. 2: The architecture of VSM.

store the mapping relationship (i.e., page table) persistently, VSM stores them in PM.

As shown in Fig. 2, the physical address space of PM is divided into *Page Table* and *Normal Pages* areas, which are used to record the mapping relationship for Virtual Address (VA) to Physical Address (PA) of PM and store the data (i.e., metadata, file data) of file systems, respectively. Moreover, a virtual superpage in VSM is a segment of continuous virtual address space, which is mapped to an equivalently-sized amount of physical memory. A virtual superpage consists of multiple logical normal pages, of which size is equal to physical normal page. VSM only requires that the VA of a virtual superpage is continuous, while its PA is not necessarily continuous. This is because that the mapping relationship between logical normal pages of a virtual superpage and physical normal pages can be changed by revising the corresponding entry in page table.

B. Multi-grained Copy-on-Write (MCoW)

Data consistency and crash recovery are the most important parts of the file systems for improving the reliability. However, the over-write operations will cause severe write amplification when the file data is organized by superpages. Therefore, VSM utilizes an efficient Multi-grained Copy-on-Write mechanism, denoted as MCoW, to guarantee data consistency. Meanwhile, VSM provides fast crash recovery after a system crash or power failure by efficiently organizing the log used to record the updated information.

1) Data Consistency: When the size of over-write data is much less than the size of a superpage, over-write operations lead to severe write amplification problem which largely reduces the performance. Hence, the existing CoW mechanism of which the file system only revises the corresponding entries of file data index fails to fit superpages.

Fortunately, the PM space management of VSM provides the opportunity to solve the write amplification problem. The proposed MCoW consists of a fine-grained CoW and a coarse-grained CoW via revising the file data index and page table, respectively. We take an example to illustrate the over-write process of MCoW. In this example, the over-written data occupies two virtual superpages, denoted by VSP_1 and VSP_2, respectively. VSP_i indicates i_{th} virtual superpage, while P_i and LP_i indicate i_{th} normal physical page and virtual normal page, respectively. As shown in Fig. 3a, the VSP_1, LP_1, and LP_2 need to be over-written. LP_1 and LP_2 belong to VSP_2, which is mapped to P_1 and P_2, respectively. As shown in Fig. 3b, the over-write operation will be processed with the following steps.

- **Step 1**. Allocate a virtual superpage (denotes as VSP_3) and two virtual normal pages (denotes as LP_1' and LP_2'), respectively.
- **Step 2**. Get the physical address of LP_1, LP_2, LP_1', and LP_2' from page table (i.e., the physical address of P_1, P_2, P_1', and P_2'), denoted by PA_1, PA_2, PA_1', and PA_2', respectively.
- **Step 3**. Allocate a transactional log to record the information of metadata, such as *start position* and *updated size* of this modification, etc. Meanwhile, this transactional log adopts an array to store the page replacement information of virtual superpages and virtual normal pages, respectively. For the virtual superpages, each entry in the array represents the replace information for a virtual superpage, denoted as $\langle$ Dest_VSP_No, Src_VSP_No $\rangle$. For example, as shown in Fig. 3, two entries should be recorded in array, such as $\langle\langle\ VSP_3,\ VSP_1\ \rangle,\ \langle\ LP_1',\ VSP_2\ \rangle\rangle$. Although LP_1' is not a virtual superpage, but LP_1' and LP_2' is continuous and the recovery process can find LP_2' from LP_1', in this case, LP_1' is considered as a virtual superpage. For the virtual normal pages, each entry in the array represents the replacement information of a swapping physical normal page for a virtual normal page, denoted as $\langle$ Dest_Phy_Addr, Src_Phy_Addr $\rangle$. For example, as shown in Fig. 3, two entries should be recorded in array, such as $\langle\langle\ PA_1',\ PA_1\ \rangle,\ \langle\ PA_2',\ PA_2\ \rangle\rangle$. The order of these entries in the array must be strictly in the order of page modification in the file.
- **Step 4**. Write the updated data to the allocated pages VSP_3, LP_1', and LP_2', respectively.
- **Step 5**. Swap the entries in the page table between LP_1, LP_2 and LP_1', LP_2', meanwhile, replace VSP_1 with VSP_3 in the file data index of this file, respectively.
- **Step 6**. Reclaim the space of VSP_1, LP_1', and LP_2'.

When the over-written data occupies one or more virtual superpages, MCoW employs a coarse-grained CoW mechanism to guarantee data consistency by revising their corresponding entries in file data index. In contrast, MCoW adopts a fine-grained CoW mechanism by revising their corresponding entries in the page table when the data occupies only a portion of a virtual superpage.

2) Crash Recovery: MCoW can recover the file to a consistent state according to the transactional log. When the file systems reboot after a system crash or power failure occurred, the recovery process of MCoW will first check the state of the transactional log. When MCoW finds an over-write process has not been completed. It firstly uses the log information to determine whether the modification is revising page table or file data index to achieve data consistency. Then, MCoW uses the page number and the physical address was stored in the transactional logs to revise the page table or file data index and recover the file to a consistent state.

For example, there is a 8MB-sized file organized by virtual superpages (e.g., 2MB). The size of over-written data is 4MB

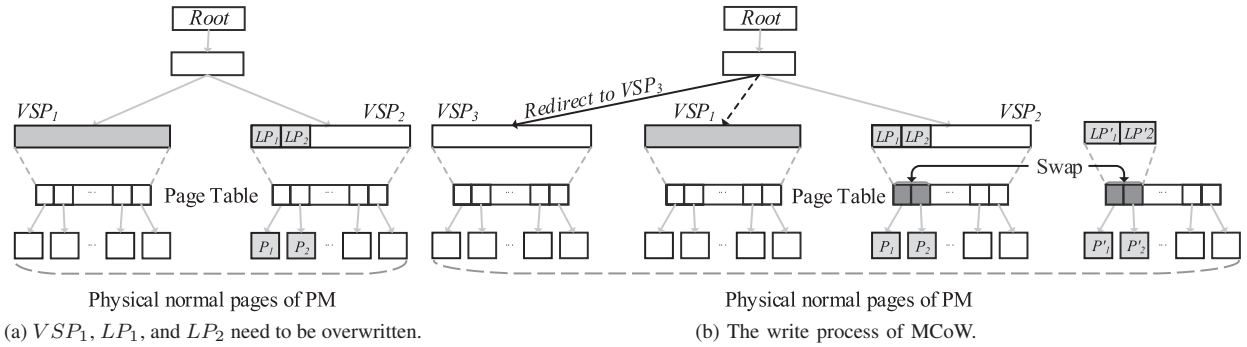


Fig. 3: The over-write example of MCoW for virtual superpages.

and the updated data range from 3MB to 7MB - 1. Firstly, MCoW concludes that the 2nd, 3rd, and 4th virtual superpages of this file were updated in the over-write process. Meanwhile, a portion of data in the 2nd and 4th virtual superpage has been updated and all data in the 3rd virtual superpage has been updated in the over-write process.

In detail, the 3rd virtual superpage has been revised. Besides, the normal virtual pages ranging from 256 to 511 in the 2nd virtual superpage and that from 0 to 255 in the 4th have also been revised. Accordingly, the recovery process of MCoW concludes that the over-write process guarantee data consistency for the data of 2nd/4th virtual superpages by revising page table, and 3rd virtual superpage by revising file data index, respectively.

Then, for the 3rd virtual page, the recovery process of MCoW uses the virtual superpage number stored in the transactional log to modify its corresponding entry in the file data index. Similarly, for the 2nd and 4th virtual superpages, the recovery process of MCoW updates the corresponding entries in the page table according to physical address (mentioned in section III-B1) stored in the transactional log. Finally, the recovery process of MCoW updates the transactional state to complete and check the state of next transactional log.

C. Zero-copy File Data Migration (ZFDM)

Utilizing superpages to organize file data may lead to severe space waste. We propose ZFDM to solve this problem. The proposed virtual superpages is a segment of continuous virtual address space of PM. To eliminate the space waste caused by superpages, we propose Zero-copy File Data Migration via revising page table (ZFDM) mechanism.

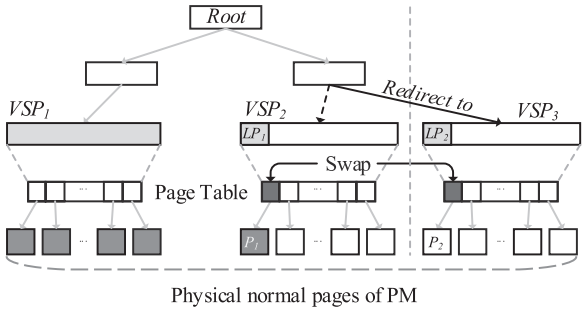


Fig. 4: The example of zero-copy file data migration.

We take an example to explain how ZFDM reduces the space waste. As shown in Fig. 4, the size of valid data in the last virtual superpage (i.e., VSP_2) is only 4KB. In other words, the size of the superpage is 1GB, while 1GB - 4KB is wasted. Then, the process of ZFDM will be conducted as following steps:

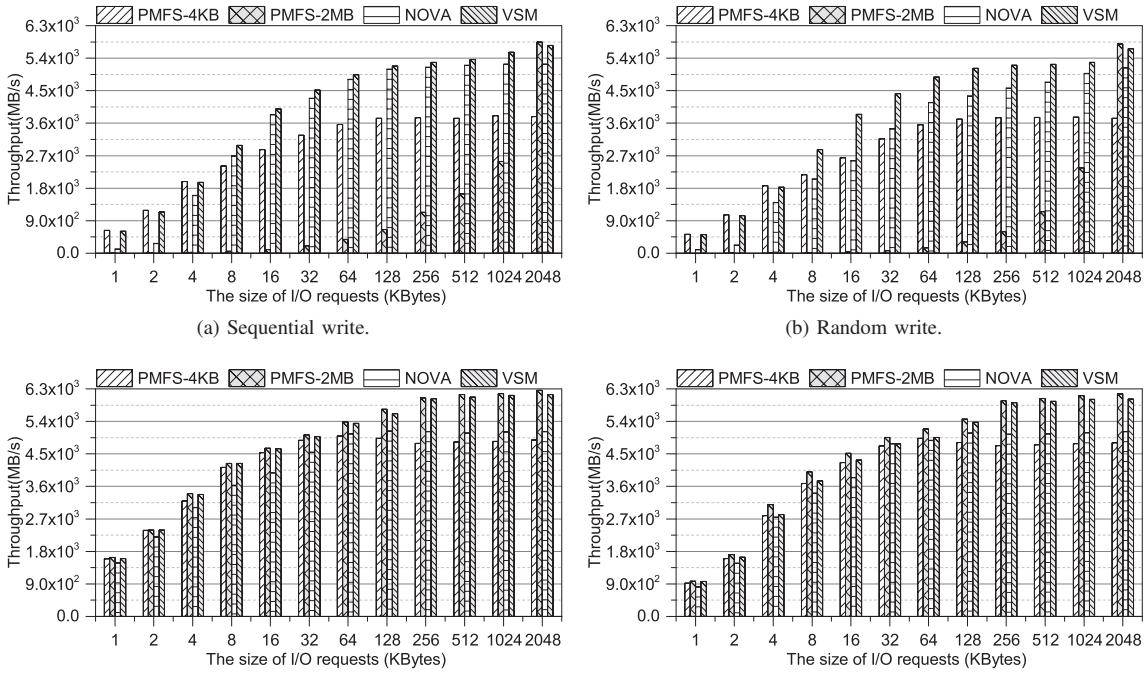
- **Step 1**. ZFDM reclaims the unused space in the last virtual superpage of a file when the file is being closed. Meanwhile, ZFDM sets a flag in *inode* to show that there is no available space within the last page of the file.
- **Step 2**. When this file is reopened to do append-write operations, ZFDM allocates an unused virtual superpage (VSP_3). Meanwhile, ZFDM swaps the entries of LP_1 and LP_2 in page table to achieve LP_1 map to P_2 and LP_2 map to $P1$.
- **Step 3**. ZFDM revises the corresponding entry in the file data index to redirect to VSP_3 and reclaims the space of LP_1 in VSP_2.
- **Step 4**. Append data after LP_2 in VSP_3, completely.

ZFDM achieves zero-copy file data migration to reduce the space waste caused by superpages. To further reduce the overhead of swapping entries in the page table, ZFDM swaps the entries of PMD according to the size of data and the size of a virtual superpage. For example, if ZFDM needs to swap the data occupying 513 normal pages (e.g., 4KB) which occupy a PMD (i.e., 512 PTE entries) and a PTE entry in page table, ZFDM only swaps a PMD entry and a PTE entry (instead of 513 PTE entries) if the size of a virtual superpage is 1GB.

IV. Evaluation

A. Experimental Setups

We implement a prototype of VSM on Linux 4.4.30 and integrate it into PMFS [3]. VSM is compared with the original PMFS and NOVA [4] which are the state-of-the-art persistent memory file systems. The experiments are conducted on a machine equipped with a 2.40GHz Intel Xeon®E5-2640 v4 CPU and 256GB DRAM. **Notes**: the paper of PMFS indicates that it employs the CoW mechanism to guarantee data consistency. However, the existing code of PMFS lacks the implementation of data consistency, which only supports metadata consistency using the journaling technique akin to the *ordered* mode of ext3/ext4 [9]. Hence, we supplement the corresponding code to support data consistency in the experiments.



(a) Sequential write.

(b) Random write.

(c) Sequential read.

(d) Random read.

Fig. 5: The throughput of FIO.

In our tests, we use the Intel Persistent Memory Emulation Platform (PMEP) [3] and configure with 128GB of DRAM to emulate PM. PMFS and NOVA use the default setting of PMEP which maps physical memory to kernel virtual address space by direct mapping (2-level page table). PMFS provides two versions, PMFS-4KB and PMFS-2MB, which indicate PMFS use the 4KB pages and 2MB superpages for file data. NOVA uses 4KB pages because it does not support superpages. Conversely, we revise the code of PMEP for VSM to achieve 4-level mapping for virtual-to-physical-address of PM. Then, VSM can revise the PTE or PMD entries. Moreover, the size of virtual superpage of VSM is 2MB.

In the experiments, we first evaluate the performance of PMFS-4KB, PMFS-2MB, NOVA, and VSM with different I/O request sizes. We use FIO [8] as the micro-benchmarks to evaluate the write and read performance. FIO is a widely-used benchmark for performance evaluation of file systems, which can generate and measure a variety of file operations. In FIO, we set a thread to write to or read from an existing 8GB file.

Finally, we use a typical workload, Postmark [10], as the macro-workload to evaluate the application-level performance. Postmark simulates a server that frequently accesses lots of files. It commits a series of create, delete, read and append operations to access the preset file pool. The type of write in postmark is append write. In postmark, we set the size of files ranging from 2MB to 6MB and the total number of files is 10 thousands, 20 thousands, and 30 thousands, respectively. We calculate the elapsed time.

B. Overall Performance

The throughput (MB/s, MB per second) of write and read with different I/O request sizes in FIO [8] are shown in Fig. 5. For the write performance, the experimental results show that the proposed VSM outperforms PMFS-4KB, PMFS-2MB, and NOVA in all cases when the I/O size is less than the superpage size. For the read performance, the proposed VSM outperforms PMFS-4KB in all cases and NOVA in most cases. Meanwhile, the performance of VSM is slightly less than PMFS-2MB.

The sequential write performance is shown in Fig. 5a. VSM outperforms PMFS-2MB by up to 2.2× (i.e,. 1024KB). Furthermore, this multiple will gradually increases when the write granularity decreases. Especially, the performance of VSM is 94× higher than PMFS-2MB when write granularity is 1KB. Similarly, as shown in Fig. 5b, the random write performance of VSM still outperforms PMFS-2MB by up to 2.3× and 225× when the write granularity is 1024KB and 1KB, respectively. Moreover, compared with PMFS-4KB and NOVA, Fig. 5a and Fig. 5b show that the performance of VSM grows faster than them with the increase of write granularity. In summary, the sequential/random write performance of VSM are 1.36/1.36× and 1.09/1.22× higher than PMFS-4KB and NOVA on average, respectively.

The reason for the performance of VSM outperforms PMFS-2MB is the severe write amplification problem caused by superpage, but VSM can efficiently solve this problem. Moreover, the reason for the performance of VSM outperforms PMFS-4KB and NOVA is that VSM can significantly alleviate the overhead of locating file data. Especially, the virtual superpages mechanism of VSM can fully takes the advantages of Memory Management Unit (MMU) to locate file data when the I/O size increases.

For the read performance, as shown in Fig. 5, the proposed VSM outperforms PMFS-4KB and NOVA in most cases. Meanwhile, the performance of VSM is slightly less than

PMFS-2MB. As shown in Fig. 5c and Fig. 5d, the performance of VSM and PMFS-2MB grow faster than PMFS-4KB and NOVA when the read granularity increases. In summary, the sequential/random read performance of VSM is 1.13/1.15× and 1.14/1.11× higher than PMFS-4KB, NOVA on average, respectively. Meanwhile, the performance of VSM is slightly less than PMFS-2MB.

The reason for the read performance of VSM outperforms PMFS-4KB is like the write operation. Moreover, the reason for the performance of VSM is slightly less than PMFS-2MB is that the TLB miss rate of VSM (i.e., 4-level page table) is higher than PMFS-2MB (i.e., 2-level page table).

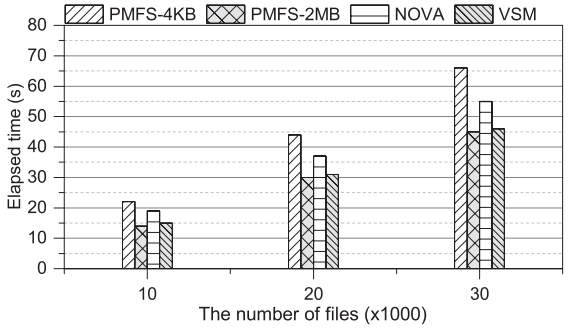


Fig. 6: The elapsed time of Postmark.

Finally, we evaluate the performance of PMFS-4KB, PMFS-2MB, NOVA, and VSM with $Postmark$ workload. The smaller elapsed time indicates the better performance. As shown in Fig 6, the elapsed time of VSM is only 67%, 82% of PMFS-4KB and NOVA, respectively. Like PMFS-2MB, VSM benefits from low overhead of space management and fully takes advantages of MMU. The elapsed time of VSM is sightly greater than PMFS-2MB due to higher TLB miss rate.

C. Comparison of Write Amplification

In order to investigate the reason why VSM can significantly improve the over-write performance compared with PMFS-2MB, we track the write amplification of PMFS-4KB, PMFS-2MB, NOVA, and VSM. As shown in Fig. 7, the experimental results show that PMFS-2MB experiences severe write amplification in FIO tests for over-write operations.

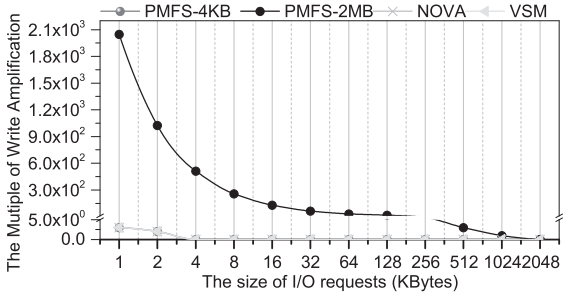


Fig. 7: The comparison of write amplification.

As shown in Fig. 5a and Fig. 5b, we observe that PMFS-2MB experiences dramatic performance degradation when the write granularity decreases. The main reason is that PMFS-2MB encounters serious write amplification problem when the write granularity decreases. Especially, the write amplification of PMFS-2MB is 2047× when the write granularity is 1KB.

However, PMFS-4KB and NOVA employ normal pages (i.e., 4KB) to organize file data. Therefore, they have the write amplification problem only the over-write data for a page is less than 4KB. Similarly, VSM employs virtual superpages to organize file data, it only needs to revise the corresponding entries of page table when the write granularity is less than the size of a virtual superpage. Therefore, VSM can alleviate the write amplification problem like PMFS-4KB and NOVA. Accordingly, VSM can effectively solve the write amplification problem compared with file system using superpages.

V. Conclusion

In this paper, we have studied the write amplification problem and space utilization efficiency when file systems employ superpages to organize file data. We proposed a mechanism, called VSM, to avoid the weakness and exploit the advantages of superpages for file systems. Experimental results show that the proposed VSM can significantly improve the performance of file systems and the space utilization efficiency.

Acknowledgement

This work is partially supported by grants from the National Natural Science Foundation of China (61672116, 61601067 and 61802038), Chongqing High-Tech Research Key Program (cstc2019jscxmbdx0063), the Fundamental Research Funds for the Central Universities under Grant (0214005207005 and 2019CDJGFJSJ001), China Postdoctoral Science Foundation (2017M620412), Chongqing Postdoctoral Special Science Foundation (XmT2018003).

References

[1] P. M. Palangappa, J. Li, and K. Mohanram, "Wom-code solutions for low latency and high endurance in phase change memory," *IEEE Trans. Comput.*, vol. 65, no. 4, pp. 1025–1040, Apr. 2016.

[2] Intel, "3d xpoint unveiled, the next breakthrough in memory technology," in *http://www.intel.com/content/www/us/en/architecture-andtechnology/3d-xpoint-unveiled-video.html*, 2015.

[3] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, R. Sankaran, and J. Jackson, "System software for persistent memory," in *Proceedings of the Ninth European Conference on Computer Systems*, ser. EuroSys '14. New York, NY, USA: ACM, 2014, pp. 15:1–15:15.

[4] J. Xu and S. Swanson, "NOVA: A log-structured file system for hybrid volatile/non-volatile main memories," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*. Santa Clara, CA: USENIX Association, 2016, pp. 323–338.

[5] E. Lee, H. Bahn, and S. H. Noh, "Unioning of the buffer cache and journaling layers with non-volatile memory," in *Usenix Conference on File and Storage Technologies*, 2013, pp. 73–80.

[6] 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 Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, Usa, October*, 2009, pp. 133–146.

[7] E. Sha, X. Chen, Q. Zhuge, S. Liang, 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, 2016.

[8] "Fio: flexible i/o tester," http://freecode.com/projects/fio.

[9] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, "The new ext4 filesystem: Current status and future plans," in *Linux Symposium*, 2007.

[10] I. Network Appliance, "The postmark filesystem benchmark," https://github.com/wolfwood/postmark.