Optimizing CoW-based File Systems on Open-Channel SSDs with Persistent Memory

Runyu Zhang[†],Duo Liu[*†],Chaoshu Yang[†],Xianzhang Chen[†],Lei Qiao[§],and Yujuan Tan[†],
[†]College of Computer Science, Chongqing University, Chongqing, China
[§]Beijing Institute of Control Engineering, Beijing, China
[*]Corresponding Author: Duo Liu (liuduo@cqu.edu.cn)

***Abstract*—Block-based file systems, such as Btrfs, utilize the copy-on-write (CoW) mechanism to guarantee data consistency on solid-state drives (SSDs). Open-channel SSD provides opportunities for in-depth optimization of block-based file systems. However, existing systems fail to co-design the two-layer semantics and cannot take full advantage of the open-channel characteristics. Specifically, synchronizing an overwrite in Btrfs will copy-on-write all pages in the update path and induce severe write amplification. In this paper, we propose a hybrid fine-grained copy-on-write and journaling mechanism (HyFiM) to address these problems. We first utilize persistent memories to preserve the address mapping table of open-channel SSD. Then, we design an intra-FTL copy-on-write mechanism (IFCoW) that eliminates the recursive updates caused by overwrites. Finally, we devise fine-grained metadata journals (FGMJ) to guarantee the consistency of metadata with minimum overhead. We prototype HyFiM based on Btrfs in the Linux kernel. Comprehensive evaluations demonstrate that HyFiM can outperform over Btrfs by 30.77% and 33.82% for sequential and random overwrites, respectively.**

***Index Terms*—Persistent memory, file system, open-channel SSD**

I. Introduction

Block-based file systems, such as Ext3 [1] and F2FS [2], use the indirect block mapping scheme to manage file data, which is fairly efficient for small files but will cause high metadata overhead for large files. To overcome this problem, Ext4 [3] and BtrFS [4] adopt an extent tree to manage file data. An extent is a single descriptor that represents a range of contiguous blocks. Therefore, it can significantly reduce the write operations on metadata. However, Ext4 still needs to employ the Write-Ahead Logging (WAL) [5], [6] mechanism to ensure data consistency. WAL causes a double write problem, which can seriously reduce the write performance. To mitigate the write amplification of WAL, BtrFS adopts the Copy-on-Write (CoW) [4] mechanism to guarantee data consistency. Instead of recording redundant journals, the CoW mechanism duplicates the updated page to a different location and modifies the corresponding indicator in its parent.

Unfortunately, the CoW mechanism still suffers high overhead for ensuring data consistency. First, the CoW mechanism can induce recursive updates, i.e., one write operation propagates the CoWs to the root of the file system. As a consequence, file systems suffer from the high write amplification of metadata. Second, CoW should update file data out-of-place, which causes a serious fragmentation problem for the extent tree. The out-of-place updates will split the extents for each overwrite operation and reduce the efficiency of locating file data. Third, file systems still adopt a lightweight journal mechanism to efficiently synchronize the metadata (e.g., inodes) rather than conducting CoWs through the whole metadata path. Consequently, the write-backs of logs introduce high overhead, akin to the WAL mechanism.

Recently, the Open-Channel SSD (OCSSD) has been proposed to provide in-depth optimization opportunities. Different from traditional SSDs, OCSSDs allow the host to access the storage via a host-side Flash Translation Layer (FTL). Such an architecture provides possibilities of mitigating redundant logical modifications of file systems by manipulating the software-level mapping table in FTL. Meanwhile, emerging Persistent Memories (PM), such as PCM [7]–[9] and STT-RAM [10]–[12], are promised to revolutionize storage systems by providing non-volatility, byte-addressability, and low latency. Hence, PMs can help access and modify the host-side FTL in a fine-grained way. To summarize, the flexibility of OCSSD and the advanced features of PM bring opportunities to resolve the above problems of block-based file systems.

In this paper, we propose a hybrid fine-grained copy-on-write and journaling mechanism, called HyFiM, to mitigate the consistency overhead of CoW-based file systems on OCSSD. The main idea of HyFiM is to fully exploit the flexibility of OCSSD and the advanced features of PM to eliminate the redundant overhead of CoW-based file systems. HyFiM employs two key techniques, including the intra-FTL copy-on-write mechanism (IFCoW) and fine-grained metadata journals (FGMJ), to perform non-recursive physical CoWs and achieve high performance. We implement HyFiM in Linux kernel based on Btrfs, and compare its performance with the original Btrfs. Experimental results show that HyFiM achieves 30.77% and 33.82% improvement for sequential and random overwrites over Btrfs, respectively.

The main contributions are summarised as follows:

- We conduct an in-depth investigation of the problems of current CoW-based file systems.
- We propose a novel intra-FTL copy-on-write mechanism and fine-grained metadata journals to optimize the performance of CoW-based file systems on OCSSD.
- We prototype the proposed HyFiM based on Btrfs in Linux. Comprehensive evaluations show that HyFiM significantly outperforms the original Btrfs.

The rest of this paper is organized as follows. In Section II, we introduce the problems of the conventional copy-on-write

978-3-9819263-6-1/DATE22/©2022 EDAA

mechanism and show a motivational example with experiments. We present the design of HyFiM in Section III. We show evaluation results of HyFiM in Section IV. Section V concludes the paper.

II. Background and Motivation

A. Problems of the Copy-on-Write Mechanism

Existing block-based file systems, such as Ext3/Ext4 [1], [3], BtrFS [4], and F2FS [2] are widely used in storage systems, which adopt the Hard Disk Drive (HDD) or traditional flash-based SSDs as the storage devices. To improve the data reliability, the Write-Ahead Logging (WAL) [5], [6] and Copy-on-Write (CoW) [2], [4] mechanisms are widely used in block-based file systems for guaranteeing data consistency. For example, the WAL mechanism is used in Ext3/Ext4, which requires that all modifications are written to a log before they are applied. Hence, the WAL mechanism can lead to a double write problem, resulting in high write amplification that can seriously degrade the performance of file systems. To mitigate the write amplification of WAL, F2FS and BtrFS adopt the CoW mechanism to guarantee data consistency. CoW mechanism requires the file data only can be updated out-of-place. However, this scheme still brings in three main problems.

First, the CoW mechanism can produce transactional CoW propagation. To CoW a block, its parent descriptor should be modified to index the new address. As a result, the parent block will conduct another CoW. That means, the write operations will trigger propagation of CoWs to the root of the data indexing tree, whether the tree is an extent tree or an indirect mapping tree. Therefore, CoWs lead to serious write amplification when the tree is high enough and the I/O request size is less than the page size.

Second, to improve the performance, BtrFS employs an extent tree instead of the indirect mapping block scheme to manage file data. The extent tree uses one single descriptor to index a range of contiguous blocks. This scheme can significantly shrink the scale of metadata structures. Meanwhile, it can improve the efficiency of space management in terms of both allocation and deallocation, especially for large files. However, CoW will split the extent tree to locate new blocks. As a result, the number of extents will continue to increase if a large file is updated constantly. Accordingly, for the aged file systems, the CoW mechanism can lead to an external fragmentation problem and a high overhead of locating file data, which can severely degrade the performance.

Third, although the high-performance CoW-based file systems, such as F2FS [2] and BtrFS [4], adopt the CoW mechanism to ensure the consistency of data, the light-weight journals are still needed to ensure the consistency of metadata. For example, Btrfs records the modifications of metadata in a log tree. In practical workloads, the structures of metadata are also frequently-updated, inducing massive write operations on the log area. As a consequence, the write-backs of journals can also severely degrade the system performance.

B. Open-channel SSDs

Open-channel SSD (OCSSD) is a new type of SSD. Different from traditional SSD, OCSSD allows the host to access the storage via the physical address of NAND flash. In other words, the Flash Translation Layer (FTL) is implemented on the host side by utilizing the host CPU/memory capabilities. Compared with traditional SSDs, the OCSSD provides flexibility with regard to data placement decisions, over-provisioning, scheduling, garbage collection (GC), and wear-leveling. Although the detailed implementation of an OCSSD is still vendor-specific, the LightNVM Open-Channel Specification [13] is widely used to access the OCSSD.

In recent years, the OCSSD has been widely used in storage systems as the underlying storage devices [14], [15]. By exposing the physical details of the device to the host, the flexibility of OCSSD instigates software designers to overcome the trade-offs in the host storage stack. With OCSSD, the FTL can be directly integrated or merged with storage management in block-based file systems [16]. Accordingly, the emerging OCSSD brings in more opportunities and challenges for the design of block-based file systems to further improve the system performance.

C. Persistent Memories

Emerging Persistent Memories (PMs), such as Spin-Transfer Torque RAM (STT-RAM) [10]–[12] and Phase Change Memory (PCM) [7]–[9], have been actively developed in the past few years. PMs are promised to revolutionize storage systems by providing non-volatility, byte-addressability, low latency, high density, and energy efficiency. These advanced features allow PMs to be linked directly to the memory bus and accessed by load/store instructions [17]–[19] like DRAM.

PMs are non-volatile, like Hard Disk Drive (HDD) and flash memory (flash-based SSD). They also provide high-performance and byte-addressability, similar to the DRAM. Especially, STT-RAM has lower power consumption than DRAM. Therefore, the emerging PMs have the potential to optimize the storage architecture and improve the performance of the block-based file systems.

D. Motivation

We give a motivational example to demonstrate the overhead of CoWs in a typical CoW-based file system, Btrfs. In our example, we vary the size of I/O requests from 1KB to 512KB and evaluate the file system with IOZone [20]. The underlying block device of this evaluation is a true open-channel SSD. As shown in Figure 1, we mount Btrfs with default (CoW) and no-CoW parameters respectively. The y-axis presents the throughput of file random writes. From this figure, we can observe that the performance gap caused by CoWs is 39.06% on average, ranging from 1.85% to 73.04%. Specifically, the smaller I/O sizes introduce larger write amplification and cause significant throughput degradation. This motivational example illustrates the tremendous overhead of the conventional CoW scheme and inspires us with the design philosophy.

Based on the above analysis, it can be concluded that the overhead of CoWs can severely degrade the performance of

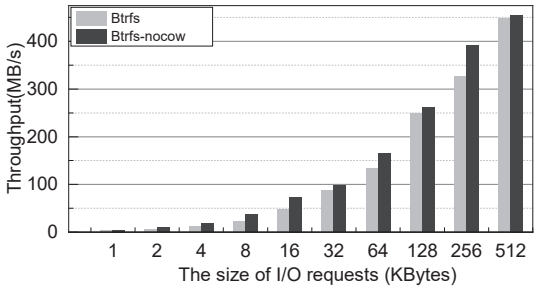


Figure 1. Comparison on random write operations.

cow-based file systems. In the following section, we propose a novel hybrid fine-grained copy-on-write and journaling mechanism, denoted as HyFiM, for cow-based file systems on open-channel SSDs. Our design goals include: (a) the CoWs should not be recursively propagated upward; (b) massive fragmentation should be avoided after CoWs; (c) the overhead for the new scheme should be minimized.

III. HyFiM Design

In this section, we present the design of the hybrid fine-grained copy-on-write and journaling mechanism (HyFiM) to mitigate the copy-on-write overhead of block-based file systems on OCSSD. First, we show the hybrid storage architecture of HyFiM. Then, we present an intra-FTL copy-on-write mechanism and fine-grained metadata journals to eliminate the propagated Copy-on-Write (CoW) of file systems. Finally, we detail the crash consistency strategy of HyFiM.

A. Architecture Overview

CoW [2], [4] is expected to induce less write amplification than Write-Ahead Logging (WAL) [5], [6] in theory. However, this advantage can be dispelled by the coarse I/O granularity of block devices and transactional CoW propagation. To address these problems of CoW, we propose a DRAM-PM hybrid architecture called HyFiM for Open-Channel SSDs (OCSSDs). HyFiM aims to break the semantic barrier between file systems and block devices, and mitigate the CoW overhead by using the DRAM-like performance, byte-addressability, and non-volatility of Persistent Memory (PM).

As Figure 2 shows, the PM in HyFiM is attached directly to the memory bus and shares the unified address space with DRAM. Hence, the PM is accessible with the load/store instructions of the CPU. The underlying block device is an OCSSD, whose FTL is manipulated in the host-end. Block-based file systems, such as Btrfs [4] and Ext4 [3], utilize standard block I/O requests to synchronize files from the host-end to the flash memory. The host-end FTL resolves these I/O requests and conducts physical operations on NAND flash chips. Such logical to physical translations are accomplished via a sector-level address mapping table. HyFiM stores the mapping table and journals in PM for data consistency.

For buffered read/write operations, file systems process regular file management routines with the help of the page cache in DRAM. For synchronization operations, such as opening a file with the synchronized flag (e.g., O_SYNC) or writing back dirty pages (e.g., $fsync$ and $fdatasync$), data will be flushed to the flash chips in the order specified in the transaction routine for consistency requirements. In this case, the proposed HyFiM will record fine-grained journals for the mapping table of FTL and metadata of file systems in the PM. HyFiM employs two key techniques, including Intra-FTL Copy-on-Write Mechanism (IFCoW) and Fine-Grained Metadata Journals (FGMJ), to achieve high performance by avoiding the recursive updates of CoW. The rest of this section will introduce the design and implementation of these two techniques in detail.

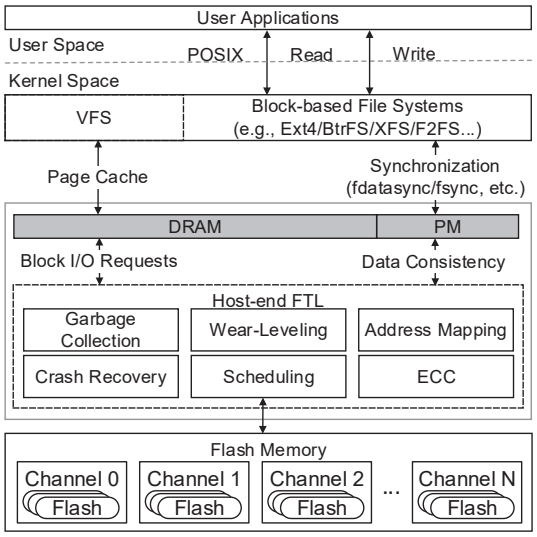


Figure 2. The overview of HyFiM.

B. Intra-FTL Copy-on-Write Mechanism

Based on the observations in Section II, the CoW of a data page will trigger modifications of its parent extent node. This modification will be propagated to the root by the CoW of each layer in the path. Meanwhile, the migration of data pages also splits the corresponding extents into massive fragmentation. Therefore, we propose the intra-FTL copy-on-write mechanism (IFCoW) to eliminate the recursive updates and fragmented extents caused by overwrites.

The main idea of IFCoW is to conduct copy-on-write operations in the FTL layer and avoid redundant logical modifications in the file system layer. Figure 3 demonstrates how IFCoW avoids fragmenting extents and cuts off the transactional propagation of CoWs. When overwriting a data page, as shown in Figure 3(a), IFCoW in-place updates the cached page in DRAM. This step will not split the parent extent node, akin to the routines without using CoW. Then, the synchronization process goes different from existing schemes. As shown in Figure 3(b), IFCoW writes the updated page back to a new physical page while maintaining the pages on the updated path unchanged. After that, IFCoW starts to swap the mapping relationships of the new/stale page in the mapping table. In Figure 3(c), IFCoW first records a small-sized journal on PM for the next steps. The journal is comprised of four addresses,

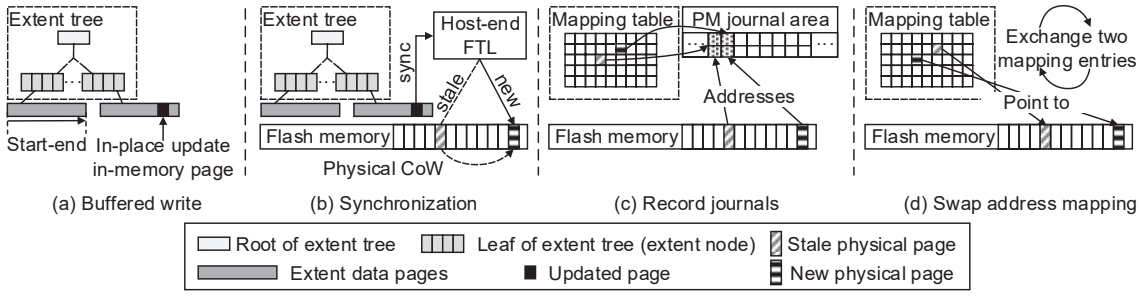


Figure 3. The intra-FTL copy-on-write mechanism.

including the physical and logical addresses of both the new and stale pages, respectively. Finally, an intra-FTL CoW will be performed after persisting in this journal. As illustrated in Figure 3(d), IFCoW exchanges the physical addresses of two corresponding entries in the mapping table.

The proposed IFCoW achieves lightweight CoWs by exploiting the byte-addressability of PM. The updated pages are written and redirected in the FTL layer without revising the logical addresses maintained in the file system. Such intra-FTL modifications break the propagation of CoWs as the metadata keeps unchanged during the whole process. Moreover, this invariability of metadata promises the continuity of existing extents, preventing them from being split into considerable fragmentation. Hence, IFCoW can significantly reduce the overhead of CoW operations while ensuring data consistency.

C. Fine-grained Metadata Journals

The IFCoW mechanism eliminates the recursive updates caused by overwrites on data pages. Although append writes contribute no extent splits, however, they still require updating parents by inserting new extent entries. Furthermore, node splits triggered by insertions will modify their parents. Those modifications will also be propagated to the root in conventional CoW schemes. To address this issue, we propose fine-grained metadata journals (FGMJ) for parent pages to eliminate the recursive updates caused by append writes.

Since inserting an extent entry introduces only small-sized updates on the extent node, we would record a fine-grained journal for the updated node rather than conducting a whole IFCoW process. When writing back a dirty extent node, FGMJ only in-place updates the node as corresponding journals have been recorded in PM. Once the insertion triggers a split operation, the affected node will be split into two via a CoW. Meanwhile, the addresses of the two newly created nodes will substitute the stale one in their parent node. FGMJ treats the updates in internal nodes the same as in extent nodes. It only records journals for substitutions of new/stale entries rather than continuing propagating these CoWs. As a result, benefit from the fine-grained metadata journals in PM, the recursive CoWs caused by append writes are confined in one node.

D. Crash Recovery

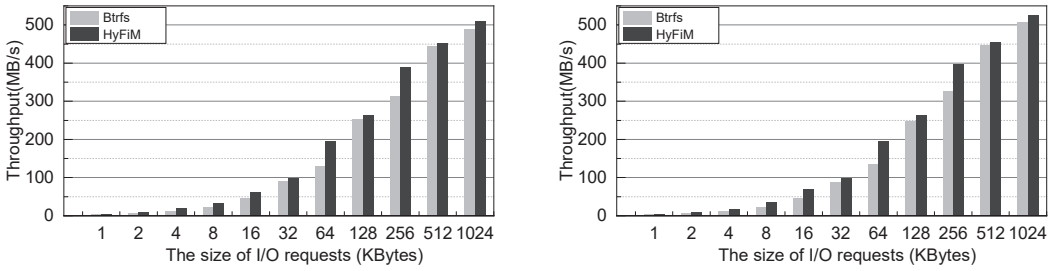
HyFiM stores the journals of two techniques in their dedicated circular queue. Besides regular information, HyFiM also records the state of each journal. When encountering power failures or system crashes, the recovery process of HyFiM will check the recorded journals one by one. HyFiM will start the recovery operation if the state of a journal is not checkpointed. This is because the other three states (i.e., committing, committed, and checkpointing) of a transaction indicate that its corresponding resources have been allocated, the updated information of the synchronized file has been recorded, and the corresponding synchronized file is being updating, respectively. For the journal in committing or committed states, HyFiM only needs to release the corresponding allocated resources. For the journals in the checkpointing state, we recover the system via the following steps. First, we extract information from this journal, such as the address of new/stale pages, the pointer of child nodes, and the size of updated data. Then, for IFCoW journals, we update the corresponding entry of the mapping table according to the information of journals. For FGMJ journals, we retrieve the pointers of child nodes from journals and update entries in the corresponding node. Finally, we set the state of the transaction to checkpointed and release the resources of the transaction. Till now, the recovery process is completed and the file system has been recovered to a consistent state.

IV. Evaluation

A. Experimental setup

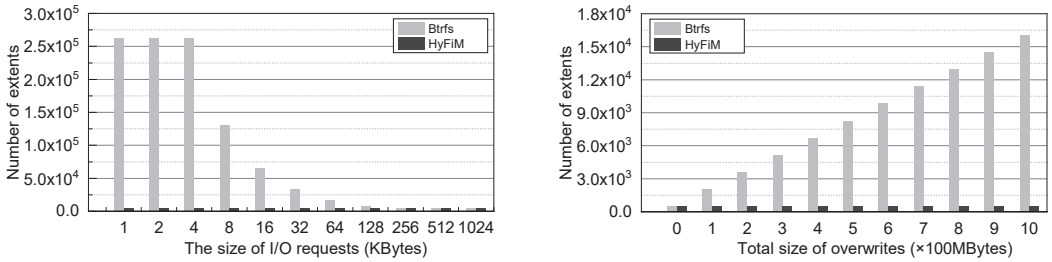
We implement the proposed HyFiM in Linux based on Btrfs. As shown in Table I, evaluation experiments are conducted on a workstation equipped with two Intel(R) Xeon(R) E5-2640 processors and 256GB DRAM. We partition 1G DRAM to simulate the PM area. The operating system is Ubuntu 16.04 with Linux kernel 4.4.238. The underlying block device is an open-channel SSD with our modified lightNVM [13] to support intra-FTL CoWs. There are eight channels in the OCSSD, two parallel units (PU) per channel, two planes per PU, 1478 blocks per plane, and 768 pages per block.

We compare HyFiM with the original Btrfs in terms of performance and extents fragmentation. We use various standard benchmarks to conduct the evaluations. In the experiments, we first evaluate the performance of sequential/random write operations with IOZone [20], a widely-used micro-benchmark. Then, we use the TPCC [21] and Filebench [22] as the macro-benchmarks to evaluate the overall throughput of the two schemes.



(a) *Sequential write* (b) *Random write*

Figure 4. Comparison of overwrites.



(a) *Sequential write* (b) *Random write*

Figure 5. Comparison of extents fragmentation.

Table I
SYSTEM CONFIGURATION

CPU	Intel(R) Xeon(R) E5-2640
Capacity of DRAM	256GB
Kernel Version	Linux 4.4.238
Available Volume of Open-Channel SSD	508 GB
# of Channels	8
Page Size	16 KB

B. Micro-benchmarks

Performance. In this subsection, we measure the performance of HyFiM and original Btrfs. We first evaluate the performance of sequential/random write operations. We use IOZone [20] as the micro-benchmark, which is a widely-used benchmark for performance evaluation on file systems. We set one thread to create a 1GB-sized file and conduct sequential/random overwrites on it with IOZone. Figure 4 reports the throughput of two schemes with I/O sizes ranging from 1KB to 1024KB. Compared with the original Btrfs, HyFiM achieves 30.77% and 33.82% improvement on average for sequential and random overwrites, respectively.

From the results, we can observe that the throughput of the two schemes enlarges with the increase of I/O sizes. HyFiM maintains 8.65% and 3.53% improvement for sequential and random overwrites when the I/O size is 1024KB. Another observation is that smaller I/O sizes bring larger improvements in performance. Specifically, HyFiM gains 62.65% and 67.38% improvement when the I/O size is 1KB. This is because small-sized overwrites severely violate the structure of the extent tree in Btrfs. Massive extent fragmentation introduces considerable metadata updates in Btrfs. Moreover, the fragmented extent tree provides poor performance in locating data pages. Comparatively, benefits from the IFCoW technique, overwrite operations in HyFiM contribute no split of extents.

Fragmentation of extents. To further demonstrate the merits of HyFiM in terms of reducing extent fragmentation, we count the number of extents after overwrites. First, we utilize IOZone to create a 1GB-sized file with 256KB I/O size. Then, we conduct sequential overwrites on this file with different I/O sizes and count the number of extents. As shown in Figure 5(a), HyFiM keeps 4.1×10^3 extents for all I/O sizes, while the total extents of Btrfs range from 4.1×10^3 to 2.6×10^5. Concretely, Btrfs constantly produces 2.6×10^5 extents when I/O sizes vary from 1-4KB. This is because the sector size of the underlying OCSSD is 4KB. When the I/O sizes are larger than 256KB, the numbers of extents become the same in both two schemes. The reason is that the file is created with 256KB I/O size and the broken extent tree is difficult to fix.

To show the side-effects of random writes, we create a 1GB-sized file with 2048KB I/O size. Then, we randomly overwrite this file with 64KB I/O size. We vary the total sizes of random writes and record the number of extents. As shown in Figure 5(b), HyFiM remains 5×10^2 extents for all written sizes. This is because HyFiM utilizes intra-FTL physical CoWs without modifying logical information. On contrary, Btrfs increases the number of extents to 1.6×10^4 after overwriting 1000MB data. These evaluation results highlight the advantages of HyFiM on micro-benchmarks.

C. Macro-benchmarks

In this subsection, we utilize the TPCC [21] and various workloads of the Filebench [22] as the macro-benchmarks to evaluate the overall throughput of two schemes. TPCC is an on-line transaction processing benchmark that issues a large

amount of synchronized random writes. In this experiment, we run TPCC on MySQL [23], a typical and widely-used database system. The numbers of warehouses in TPCC vary from 1 to 16. We record the tpmC (transaction per minute) for each number of warehouses. Figure 6 demonstrates that HyFiM outperforms Btrfs by 13.4%.

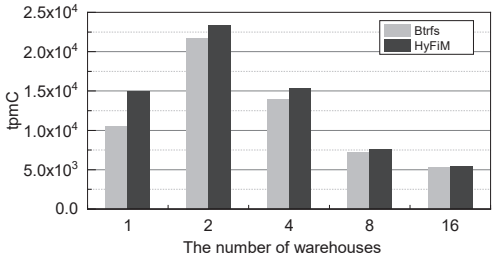


Figure 6. Comparison with the TPCC benchmark.

Besides, we also evaluate the two schemes with typical workloads of Filebench, including oltp and fileserver. The oltp emulates a database that performs file operations using the Oracle 9i I/O model. It evaluates the performance of small random reads and writes, while the fileserver evaluates the performance of append writes. We set the number of writers as 4 in oltp, and as 50 in fileserver, respectively. As shown in Figure 7, HyFiM achieves 43.68% improvement in oltp. Furthermore, HyFiM outperforms Btrfs by 26.82% in fileserver, demonstrating the effectiveness of fine-grained metadata journals in append writes.

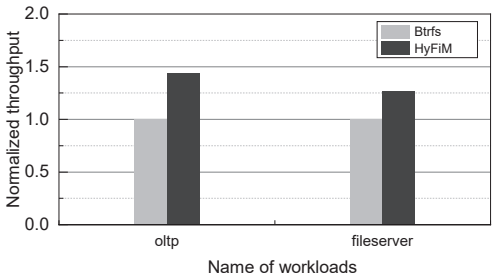


Figure 7. Comparison with the Filebench workloads.

V. Conclusion

This paper investigated the problems of CoW-based file systems. We proposed a hybrid architecture with OCSSD and PM to achieve high performance for CoW-based file systems. Experimental results show that our scheme achieves significant speedup and reduces the fragmentation of extents.

Acknowledgement

We would like to thank the anonymous reviewers for their valuable comments and improvements to this paper. This work is partially supported by grants from the National Natural Science Foundation of China (Grant No. 62072059, 61672116, 61802038, and 62162011), Chongqing High-Tech Research Program (Grant No. cstc2016jcyjA0332), and Fundamental Research Funds for the Central Universities (Grant No. 0214005207005).

References

[1] M. K. Johnson, "Red hat's new journaling file system: ext3," *RedHat Inc*, 2001.
[2] C. M. Lee et al, "F2fs: A new file system for flash storage," in *13th USENIX Conference on File and Storage Technologies (FAST)*, 2015, pp. 273–286.
[3] A. Mathur, M. Cao, and A. Dilger, "Ext4: the next generation of the ext3 file system," *the magazine of USENIX & SAGE*, vol. 32, pp. 25–30, 2007.
[4] O. Rodeh, J. Bacik, and C. Mason, "Btrfs: The linux b-tree filesystem," *ACM Transactions on Storage (TOS)*, vol. 9, no. 3, pp. 1–32, 2013.
[5] W. H. Kim, J. Kim, W. Baek, B. Nam, and Y. Won, "Nvwal: Exploiting nvram in write-ahead logging," in *International Conference on Architectural Support for Programming Languages & Operating Systems*, 2016, pp. 385–398.
[6] K. Han, H. Kim, and D. Shin, "WAL-SSD: address remapping-based write-ahead-logging solid-state disks," *IEEE Trans. Computers*, vol. 69, no. 2, pp. 260–273, 2020.
[7] Y. Hong, Y. Lin, T. Tang, and B. Chen, "Multilevel storage in phase-change memory," *IEICE Trans. Electron.*, vol. 90-C, no. 3, pp. 634–640, 2007.
[8] G. F. Close, U. Frey, J. Morrish, R. Jordan, S. C. Lewis, T. Maffitt, M. J. BrightSky, C. Hagleitner, C. H. Lam, and E. Eleftheriou, "A 256-mcell phase-change memory chip operating at 2+ bit/cell," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 60-I, no. 6, pp. 1521–1533, 2013.
[9] N. S. Kim, C. Song, W. Y. Cho, J. Huang, and M. Jung, "LL-PCM: low-latency phase change memory architecture," in *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*. ACM, 2019, p. 14.
[10] K. Kuan and T. Adegbija, "Lars: Logically adaptable retention time stt-ram cache for embedded systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 461–466.
[11] ——, "Energy-efficient runtime adaptable L1 STT-RAM cache design," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, no. 6, pp. 1328–1339, 2020.
[12] M. A. Qureshi, J. Park, and S. Kim, "SALE: smartly allocating low-cost many-bit ECC for mitigating read and write errors in STT-RAM caches," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 28, no. 6, pp. 1357–1370, 2020.
[13] M. Bjørling et al, "Lightnvm: The linux open-channel ssd subsystem," in *15th USENIX Conference on File and Storage Technologies (FAST)*, 2017, pp. 359–374.
[14] P. Wang, G. Sun, S. Jiang, J. Ouyang, S. Lin, C. Zhang, and J. Cong, "An efficient design and implementation of lsm-tree based key-value store on open-channel SSD," in *Ninth Eurosys Conference 2014, EuroSys 2014, Amsterdam, The Netherlands, April 13-16, 2014*, D. C. A. Bulterman, H. Bos, A. I. T. Rowstron, and P. Druschel, Eds. ACM, 2014, pp. 16:1–16:14.
[15] J. Huang, A. Badam, L. Caulfield, S. Nath, S. Sengupta, B. Sharma, and M. K. Qureshi, "Flashblox: Achieving both performance isolation and uniform lifetime for virtualized ssds," in *15th USENIX Conference on File and Storage Technologies, FAST 2017, Santa Clara, CA, USA, February 27 - March 2, 2017*, G. Kuenning and C. A. Waldspurger, Eds. USENIX Association, 2017, pp. 375–390.
[16] Y. Lu, J. Shu, and W. Zheng, "Extending the lifetime of flash-based storage through reducing write amplification from file systems," in *Proceedings of the 11th USENIX conference on File and Storage Technologies, FAST 2013, San Jose, CA, USA, February 12-15, 2013*, K. A. Smith and Y. Zhou, Eds. USENIX, 2013, pp. 257–270.
[17] 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.
[18] X. Zhang, D. Feng, Y. Hua, and J. Chen, "A cost-efficient nvm-based journaling scheme for file systems," in *IEEE International Conference on Computer Design*, 2017, pp. 57–64.
[19] C. Youmin, L. Youyou, C. Pei, and S. Jiwu, "Efficient and consistent nvmm cache for ssd-based file system," *IEEE Transactions on Computers*.
[20] W. NORCOTT, "Iozone filesystem benchmark," *http://www.iozone.org/*, 2003.
[21] "Tpcc," https://github.com/percona-lab/tpcc-mysql.
[22] V. Tarasov, E. Zadok, and S. Shepler, "Filebench: A flexible framework for file system benchmarking," *The USENIX Magazine*, vol. 41, no. 1, pp. 6–12, 2016.
[23] "Oracle mysql," https://www.mysql.com/.