On Stacking a Persistent Memory File System on Legacy File Systems

21st USENIX Conference on File and Storage Technologies

Hobin Woo; Daegyu Han; Seungjoon Ha; Sam H. Noh; Beomseok Nam

Paper Notes

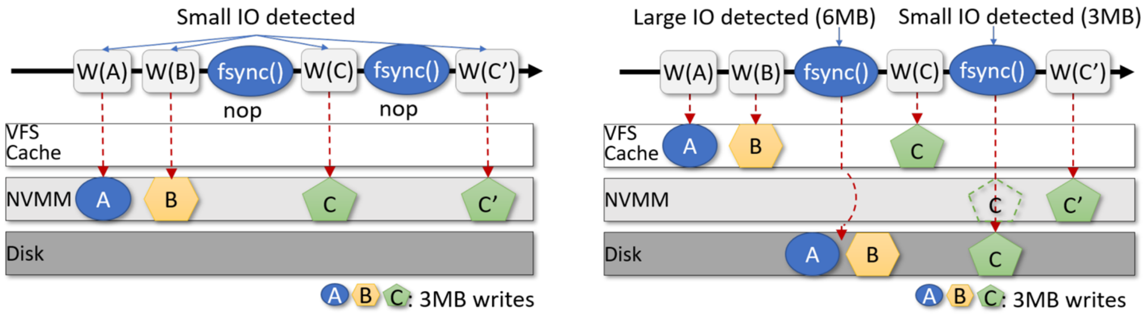
By JeongHa Lee

Abstract

In this work, we design and implement a Stackable Persistent memory File System (SPFS), which serves NVMM as a persistent writeback cache to NVMM-oblivious filesystems. SPFS can be stacked on a disk-optimized file system to improve I/O performance by absorbing frequent order-preserving small synchronous writes in NVMM while also exploiting the VFS cache of the underlying disk-optimized file system for non-synchronous writes. A stackable file system must be lightweight in that it manages only NVMM and not the disk or VFS cache. Therefore, SPFS manages all file system metadata including extents using simple but highly efficient dynamic hash tables. To manage extents using hash tables, we design a novel Extent Hashing algorithm that exhibits fast insertion as well as fast scan performance. Our performance study shows that SPFS effectively improves I/O performance of the lower file system by up to 9.9×.

Problem Statement and Research Objectives

- **Conventional file systems interleave small write requests with expensive** `fsync()` **system calls**, which leads to performance degradation.
- This work advocates a **modular approach through the use of stackable file systems (aka overlay or union file systems)**. Specifically, this study present **SPFS (Stackable PM File System)**, a stackable file system that *can be deployed with only a relatively small amount of NVMM*, whose goal is to *absorb frequent small synchronous writes* required to maintain storage write order.
  - When the second `fsync()` is called, the lower file system flushes C from the cache to disk, but at the same time, SPFS detects small synchronous writes and migrates its block mapping, (not data blocks), to NVMM.
  - When subsequent writes are requested to some of the blocks of C (C'), these writes are steered to NVMM and directly written onto.



(a) Write Point Profiler (Ziggurat) (b) Sync Point Profiler (SPFS)

Figure 3: Write Point Profiler vs. Sync Point Profiler

Proposed Method

Hash-based Block Management : The first 4 KBytes is the superblock. Then comes the cluster bitmap, where each bit indicates whether all blocks in the corresponding cluster are free or not.

1. Free space management (block bitmap table)
   - The cluster bitmap does not indicate which blocks in a cluster are free or in use. Hence, each partially used cluster requires another metadata, the block bitmap.
   - When SPFS allocates some, but not all, blocks in a cluster, it creates and inserts a block bitmap into the block bitmap table.
2. Extent hashing (extent table)
   - SPFS indexes extents in a hash table called extent table. SPFS is the first hash-based file system that indexes extents using a hash table.
   - In contrast to block hashing, the proposed Extent Hashing selects only a few buckets based on the binary representation of cluster numbers, as shown in Figures 5(b).
3. Path-name resolution (name2inode table)
   - The name2inode hash table stores file/directory name and directory entry block number pairs using the hash key generated from the VFS dentry, its parent inode number, and the file name

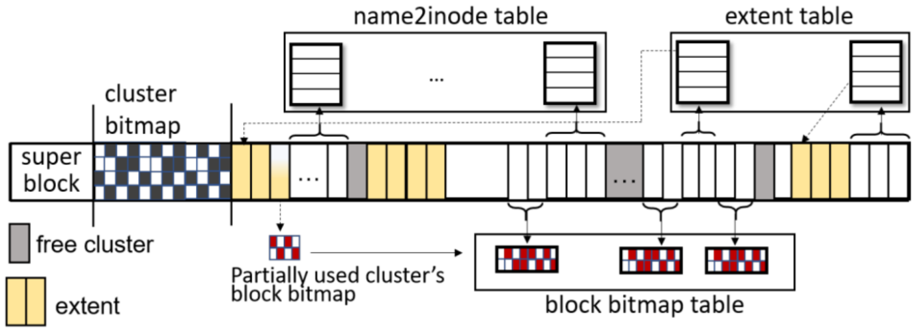
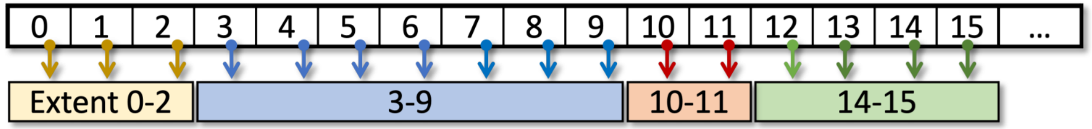
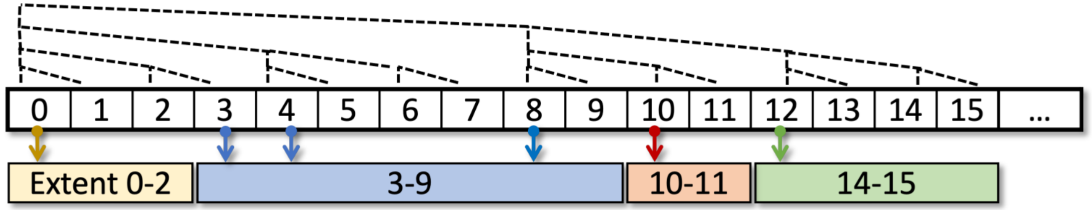


Figure 4: NVMM Space Layout for SPFS



(a) Block Hashing: $O(B)$ Write, $O(1)$ Read

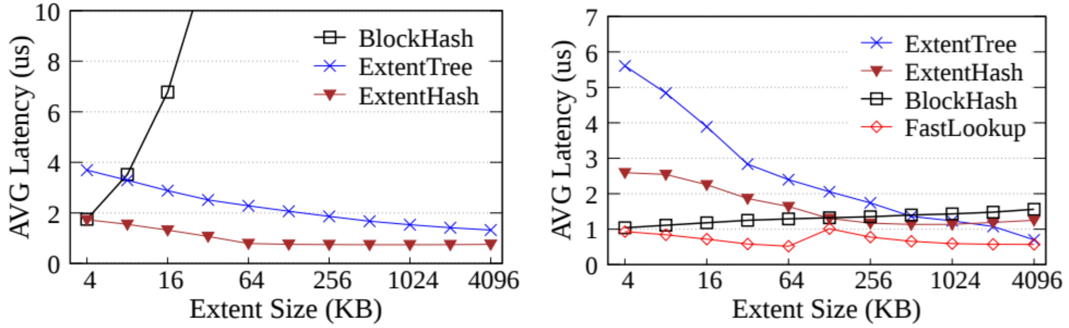


(b) Extent Hashing: $O(logB)$ Write, $O(logB)$ Read

Figure 5: Block vs. Extent Hashing

Evaluation and Results

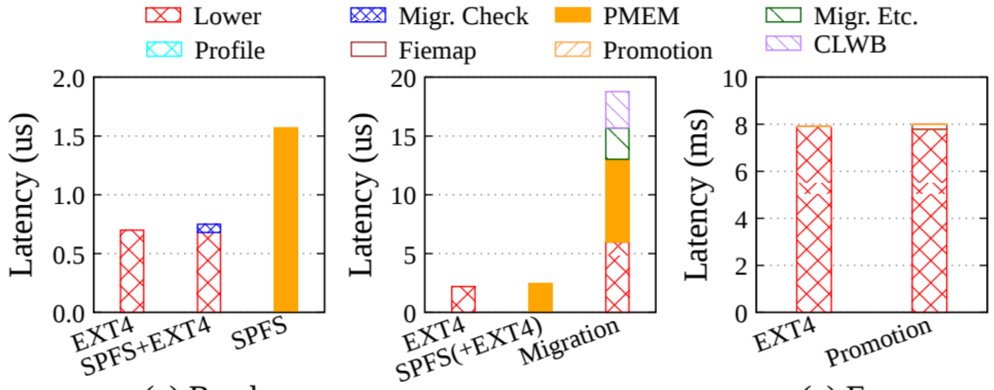
Analysis of extent hashing



(a) Insert (b) Search (Random)

Figure 7: Performance of File Mapping Structures

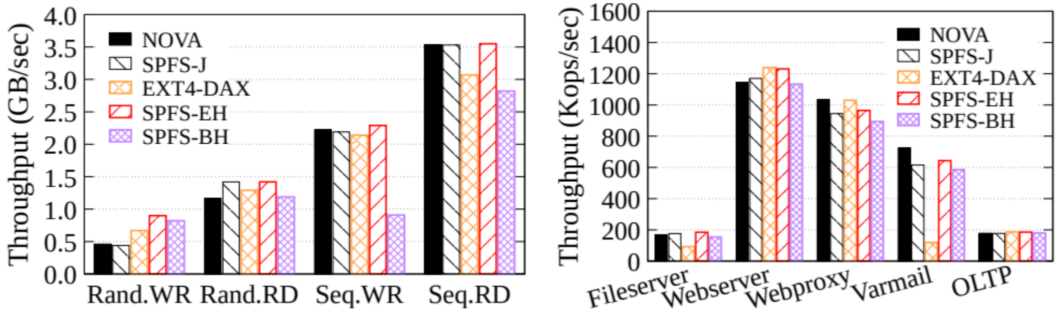
Quantification of stackable design



(a) Read (b) Write (c) Fsync

Figure 11: Latency breakdown of each mode in DCPMM

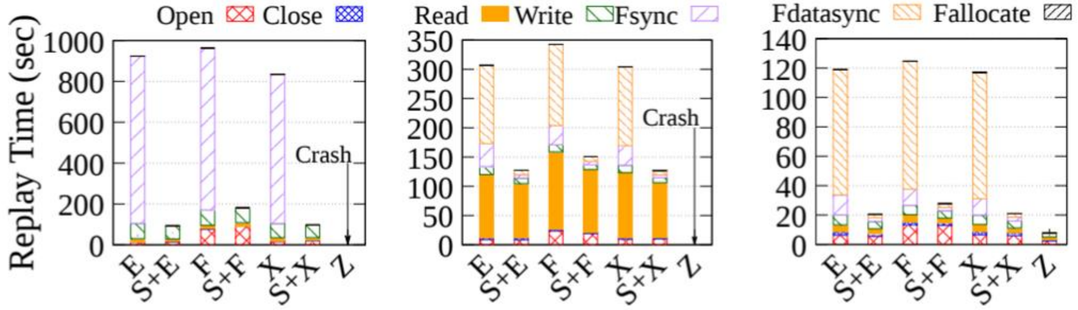
Stnadalone mode with DCPMM



(a) FIO Results (b) Filebench Results

Figure 8: Performance in Standalone Mode (DCPMM)

Stacked mode performance comparison



(d) Moodle (NVDIMM) (e) Usr1 (NVDIMM) (f) Usr2 (NVDIMM)

Figure 12: FIU Trace Replay Time S:SPFS, E:EXT4, X:XFS, F:F2FX, Z:Ziggurat