Transforming Legacy File Systems into Persistent Memory Exploiting File Systems with MeLo@V

Hyunsub Song, Young Je Moon, Se Kwon Lee and Sam H. Noh
School of Electrical and Computer Engineering, UNIST, Ulsan, Korea

I. Introduction

With the advent of Persistent Memory (PM) that is non-volatile and byte-accessible with DRAM-like latency, we are anticipating dramatic changes in computer system organization as well as in the system software that run these systems. File system development has been at the forefront of these changes as the performance characteristics of PM as storage are very different from traditional storage devices. With PM anticipated to be attached to the memory bus through the DIMM interface, file systems are continually being developed [1–3].

A recent study by Kim et al. has shown that modern CPUs employ various optimization techniques such that the long write latency typically found with PMs, which was thought to be a critical limitation for deployment of PM as main memory, actually has very little effect, if any, on the performance of actual, meaningful applications [4]. Combined with the concept of Whole System Persistence (WSP) [5], where volatile data in registers and the CPU cache are safely downloaded to nonvolatile devices upon sudden system failure, we can foresee systems where the entire memory is nonvolatile, whether through support of an external power source or through PM deployment. This paper is presented under this premise.

We present a simple yet general method, which we call MeLo@V (Metadata Logging at the VFS layer), for transforming legacy file systems into PM exploiting file systems. To do so, we find that the key issue is providing file system consistency with minimal overhead, and that this can be achieved by inserting a simple mechanism, comprising roughly 150 lines of code, in the VFS layer of the file system. Through experiments conducted on a PM emulating system, we find that MeLo@V allows legacy file systems to perform in par with state-of-the-art file systems such as NOVA [3], PMFS [1] and Ext4 with DAX extension [6], which is a recent PM aware extension to Ext4.

Development of file systems that better exploit PM must continue. However, we contend that our approach will allow the community to make use of our time-tested legacy file systems as-is in the PM environment, while continuing to make enhancements to legacy file systems without diverting energy on development of separate PM exploiting features.

II. MeLo@V

The key idea in transforming legacy file systems to PM file systems is in removing the page cache flush overhead, while at the same time, maintaining file system consistency. Removing the page cache flush overhead is done based on the WSP assumption, where the CPU cache contents can be safely flushed into memory before shutdown of the system [5], and converting synchronous I/O to asynchronous I/O as the page cache is now nonvolatile. Thus, the system itself supports multi-versioning as a duplicated, dirty copy of clean data in the file system area will safely reside separately in the page cache area upon updates. This results in the page cache being the journal. Thus, page cache flushes simply rely on the page cache daemon to do its job.

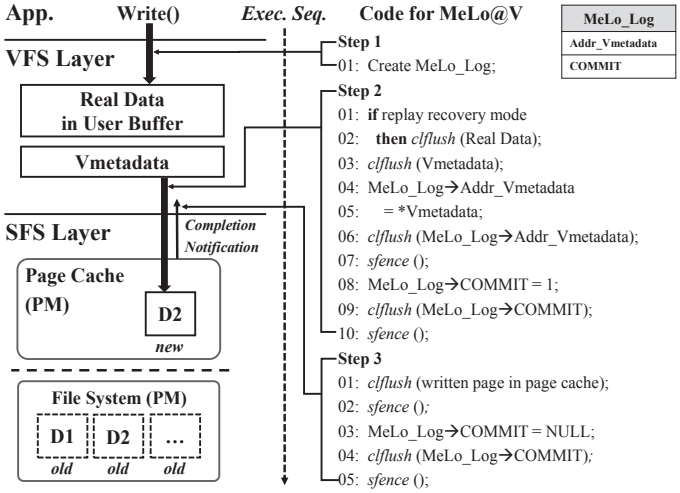

Fig. 1: Left side shows traditional execution sequence and right Steps are MeLo@V code inserted at arrow designated points in the traditional execution sequence

The more interesting and difficult issue is maintaining consistency under such an environment without incurring overhead. For this, we propose MeLo@V, which is based on the key observation that extra care need only be taken for operations that modify existing data, which is done through the page cache. As MeLo@V makes small modifications only at the VFS layer, it is a general solution that transforms any file system into a PM exploiting one.

Note here that we are in essence solving the same problem as UBJ [7] in maintaining a consistent copy in the nonvolatile page cache. While UBJ does this by freezing and having multiple copies of pages, MeLo@V does this by undoing or replaying the actions into the page cache. This allows the system to employ the same page caching mechanism provided by the legacy system.

Overall, as shown in Figure 1, the key role of MeLo@V is logging of information (metadata) in the VFS layer that refers to data that is to be modified (for example, `file`, `kiocb`

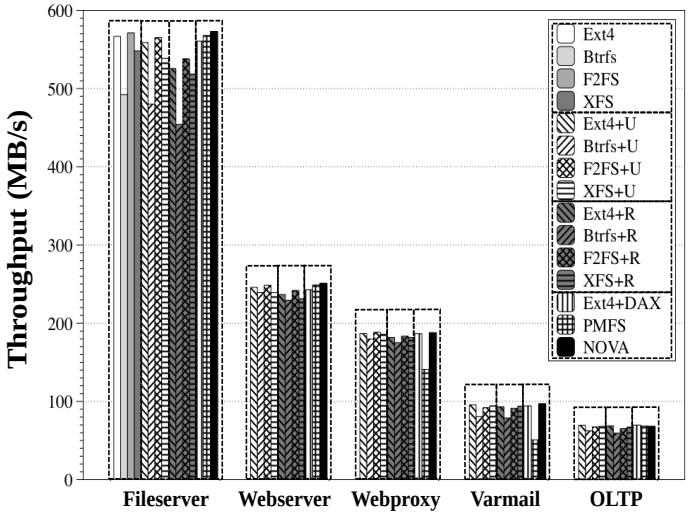

Fig. 2: Performance of Filebench benchmarks

and `iovec` structures in the Linux kernel). Such metadata is created in the VFS layer based on the parameters that are passed through the application system call. The key idea here is that if a system failure occurs while modifying data into the nonvolatile page cache, the system, for recovery, can simply undo or replay (redo) the failed modifying activity using the log information MeLo@V had generated. This guarantees file system consistency without the overhead and extra writes typically involved with consistency in legacy file systems. We emphasize that all of MeLo@V is implemented in roughly 150 lines of code.

III. Performance Evaluation

To evaluate the effect of MeLo@V, we implement MeLo@V in Linux. Experiments are conducted with four legacy file systems Ext4, Btrfs, F2FS, XFS and three PM file systems, namely, Ext4 with DAX (denoted Ext4+DAX), PMFS and NOVA all on kernel version 4.3. Note that 56GB of total 64GB DRAM is used as emulated PM storage, leaving the rest to be used as memory. For the workloads, we use the Filebench benchmarks [8].

Recall MeLo@V can be implemented in two versions, undo and replay, except for the delete operation where it is always replay. To distinguish the two, we use the 'FS name+(U or R)' naming convention, where U and R refers to undo and replay recovery mode, respectively.

Overall Performance: Figure 2 shows the performance comparisons of file systems for the Filebench benchmarks. For comparison, we show the performance of the original file systems, that is, without MeLo@V deployed, just for the Fileserver benchmark in Figure 2 on the leftmost side. This is for asynchronous I/O without any persistency mechanism included, hence are ideal results for legacy file systems.

The key observations from these results are as follows. First, even with MeLo@V deployed, performance is only slightly below those of the ideal legacy file systems. This is consistent throughout all workloads, hence only shown for Fileserver. Second, undo recovery mode provide higher performance benefits than replay mode. Replay mode results in a 2 to 6% performance reduction. The main reason for this is due to the `clflush` instruction needed to guarantee the persistency of real data in user buffer. We see that such small factors can make quite a difference in performance of PM exploiting file systems. Finally, though results differ for particular file systems with particular workloads, legacy file systems transformed into PM exploiting ones through MeLo@V, in particular, for the undo recovery mode, perform comparably to state-of-the-art PM-based file systems.

IV. Conclusions

In this paper, under the premise that Whole System Persistence (WSP) is supported, we present a simple yet general method called MeLo@V (Metadata Logging at the VFS layer) for transforming legacy file systems into PM exploiting file systems. We find that the key issue for such transformation is providing file system consistency with minimal overhead, and that this can be achieved by inserting a simple mechanism in the file system VFS layer. We find that the Ext4, Btrfs, F2FS, and XFS legacy file systems with MeLo@V deployed perform in par with recent PM file systems. Our results show that efforts to improve legacy file systems can continue without diverting effort to incorporate PM specific characteristics into file systems.

References

[1] S. R. Dulloor *et al.*, "System Software for Persistent Memory," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2014.

[2] J. Ou *et al.*, "A High Performance File System for Non-Volatile Main Memory," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2016.

[3] J. Xu and S. Swanson, "NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories," in *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, 2016.

[4] J. H. Kim *et al.*, "An Experimental Study on the Effect of Asymmetric Memory Latency of New Memory on Application Performance," in *Proceedings of the IEEE International Symposium on Modelling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2016.

[5] D. Narayanan and O. Hodson, "Whole-System Persistence," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.

[6] M. Wilcox, "DAX: Page cache bypass for filesystems on memory storage," http://lwn.net/Articles/618064/.

[7] E. Lee *et al.*, "Unioning of the Buffer Cache and Journaling Layers with Non-volatile Memory," in *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, 2013.

[8] Filebench, http://filebench.sourceforge.net/wiki/index.php/Main_Page.