P2FS: supporting atomic writes for reliable file system design in PCM storage

Eunji Lee¹, Kern Koh², and Hyokyung Bahn²a)

¹ Department of Software, Chungbuk National University,
52, Naesudong-ro, Cheongju, Chungbuk, Korea
² Department of Computer Engineering, Ewha University,
52, Ewhayeodae-gil, Seodaemun-Gu, Seoul, Korea
a) bahn@ewha.ac.kr

Abstract: This letter analyzes the characteristics of write operations in file systems for PCM (phase-change memory), and observes that a large proportion of writes only involve a small modification of the previous version. This observation motivates a new file system for PCM called P2FS, which reduces write traffic to PCM by exploiting the similarity of data between versions, while preserving the same reliability level of existing copy-on-write file systems. Experimental results show that P2FS reduces file I/O time by 48%, and energy consumption by 65%, on average, compared to copy-on-write.

Keywords: phase-change memory, non-volatile memory, file system

Classification: Storage technology

References

[1] E. Lee, J. E. Jang, T. Kim and H. Bahn: IEEE Trans. Knowl. Data Eng. 25 (2013) 2841. DOI:10.1109/TKE.2013.35
[2] D. Hitz, J. Lau and M. Malcom: USENIX Winter Technical Conf. (1994) 235.
[3] E. Lee, Y. Kim and H. Bahn: IEEE Trans. Consum. Electron. 59 (2013) 869. DOI:10.1109/TCE.2013.6689701
[4] A. Wilson: The New and Improved FileBench (USENIX FAST, USENIX Association, 2008).
[5] J. Katcher: Postmark: a New File System Benchmark (Network Appliances, 1997).
[6] B. Yang, J. Lee, J. Kim, J. Cho, S. Lee and B. Yu: IEEE Symp. Circuits and Syst. (2007) 3014.
[7] H. Huang, W. Hung and K. Shin: ACM Symp. Operating Syst. Princ. (2005).

1 Introduction

Reliability is one of the most important issues in the design of contemporary file systems [1]. Specifically, as mobile devices such as smartphones are proliferating, a sudden power failure is likely to occur frequently, and prompt recovery after a system crash is increasingly important. To restore a file system to a consistent state, copy-on-write techniques are widely used [2].
This letter presents a development of a copy-on-write technique for PCM (phase-change memory) devices.

PCM is a promising storage medium that has emerged recently, and is likely to be used widely within the next few years [1, 3]. However, there are several challenges in applying existing copy-on-write techniques to PCM, as its characteristics are significantly different to those of a hard disk or flash memory. Copy-on-write does not perform overwrites, but writes all data that has been modified out-of-place. This allows roll-back to the latest image of a file system after a system crash. However, it causes a cascade of out-of-place updates of path nodes up to the file system root. For example, when the data block $D_4$ in Fig. 1 has been updated to $D_4'$, all the nodes on the path to the root, $A_1$, $B_2$, and $C_3$, must have their pointers updated, incurring many additional writes. This recursive updates are necessary as all modified data (including path nodes) should be written out-of-place to support consistency.

This characteristic of copy-on-write does not significantly offset the performance of file systems on hard disks, on which writes can be performed at the nearest available location to the current disk head position without the need to seek the original location of the data. However, as PCM does not have seek time, any location in PCM storage can be accessed in almost the same time, and there is no benefit to be gained from out-of-place updates. Conversely, the cascade of path node updates incurred by copy-on-write seriously degrades the I/O performance of PCM, as write operations are relatively slow [1], as shown in Table I.

To resolve this problem, we design a new file system, which is just as reliable as copy-on-write, but requires significantly less write traffic. This is motivated by observing that a large proportion of file write operations contain data that are very similar to the previous version. This is relevant because PCM allows writes to individual cells, and thus the volume of writes can be reduced significantly if only the modified data are written. However, current copy-on-write file systems do not operate in this way, because blocks are written to new locations.

This problem is addressed in the proposed file system called P2FS (pair-wise PCM file system), because it retains the version of a data block that existed before an update, which can then participate in the next write request. This reduces write traffic dramatically, because only the cells containing modified data are written, whereas copy-on-write writes a whole block. P2FS does not overwrite the current data, and so a consistent file system image is always present, even after a system crash. Experimental results with

![Fig. 1.](image-url)
various file system benchmarks show that P2FS improves file system performance by 48% on average, compared to copy-on-write.

### 2 The P2FS file system for PCM

We analyze file update characteristics, comparing the contents of a block before and after an update resulting from a write request. Fig. 2 shows the results for several file system benchmarks [4, 5] listed in Table II, and 35% of updates are at least 90% similar to the previous version. Benchmark D contains a large number of small writes and metadata updates, and more than a half of its updates are over 90% the same as the previous data.

This high similarity is a consequence of current storage system interfaces. When a write request arrives at a standard storage system, an entire block is written, even if only a few bytes have changed, because the minimum unit of block storage is a logical block of 512 bytes or 4 Kbytes. In contrast, most PCM controllers are able to write just the cells whose current values differ from the existing values, using data-comparison-write (DCW) [6]. This is efficient when a write request modifies only a small fraction of the data, but it is a facility that cannot be used by a copy-on-write file system, which does not update current data in-place.

To resolve this problem, P2FS uses a data block that was previously updated as the new location for write operations, which significantly reduces the write traffic to PCM. Maintaining previous versions of each data block

| Benchmark | # of ops. | # of files | read:write | Avg. op. size | Total data written |
|-----------|-----------|------------|------------|---------------|--------------------|
| A: Varmail | 17449     | 1003       | 15:1       | 8 KB          | 8725 KB            |
| B: Web server | 41320     | 1004       | 37:1       | 9 KB          | 9786 KB            |
| C: Proxy | 53342     | 50897      | 1:2.24     | 16 KB         | 590054 KB         |
| D: Postmark | 175111    | 59106      | 1:1        | 2 KB          | 175111 KB         |

### Table II. Characteristics of file system benchmarks.

The high similarity of current storage system interfaces is a consequence of this. When a write request arrives at a standard storage system, an entire block is written, even if only a few bytes have changed, because the minimum unit of block storage is a logical block of 512 bytes or 4 Kbytes. In contrast, most PCM controllers are able to write just the cells whose current values differ from the existing values, using data-comparison-write (DCW) [6]. This is efficient when a write request modifies only a small fraction of the data, but it is a facility that cannot be used by a copy-on-write file system, which does not update current data in-place.

To resolve this problem, P2FS uses a data block that was previously updated as the new location for write operations, which significantly reduces the write traffic to PCM. Maintaining previous versions of each data block...
might be expected to incur a large space overhead, but P2FS avoids this by using a dual-version tree structure. This consists of current and shadow versions of each node. The current version is up-to-date, whereas the shadow version is the block as it was before the most recent write request. P2FS writes to the shadow version instead of overwriting the current version, which therefore remains consistent, permitting recovery from a crash without loss of data integrity. This dual-versioning structure reduces the write traffic to PCM, as most file write operations involve small modifications, in which case P2FS only updates the modified PCM cells. This results in power saving and improved performance.

Now, let’s move on to the space issue of P2FS. There are three categories of copy-on-write file systems with respect to space requirements. The first category writes the updated data out-of-place and then frees the old version promptly. This category provides the consistency of file systems, but does not support the versioning functionality. The second category maintains the latest history version even after the out-of-place update has been completed. This category provides the roll-back of the file system to the most recent history version as well as the consistency of the file system. The third category maintains multiple versions of old data in order to provide the access to a spectrum of old states. P2FS can be classified into the second category, so its space overhead is more than that of the first category but less than the third category.

Maintaining such old data will exhaust storage space earlier, but we can erase them if free space is not enough. That is, P2FS can be configured to maintain the shadow data only if the system has enough free space. Thus, if the space containing shadow data has been overwritten by other data due to space deficiency, the efficient update function of P2FS cannot be provided as it behaves identical to conventional file systems in this case. Nevertheless, we can expect the effectiveness of P2FS because a large portion of storage space usually remains free. It is reported that most storage in modern computer systems has 30–50% of unused space [7]. Moreover, the space required for shadow data is much smaller than total data volume because P2FS maintains shadows only upon the modification of data, and unmodified parts (including path nodes) are shared with the current version.

Fig. 3 illustrates the operation of P2FS. Fig. 3(a) represents the initial state of the file system. When data block $C_1$ is updated to $C_1'$ (Fig. 3(b)), P2FS first searches for the shadow version of $C_1$ to perform the update operation. In this case, $C_1$ has no shadow, so P2FS allocates a free block and writes new data $C_1'$ to it, just like copy-on-write. Similarly, as path nodes $B_1$ and $A_1$ have no shadow, free blocks are allocated. Then, P2FS switches the current and shadow versions, as shown in Fig. 3(c). When a new data block $C_3$ is created (Fig. 3(d)), P2FS allocates a free block as there is no shadow version of it. A free block is also allocated for updating the parent node $B_2$ as it has no shadow. Then, P2FS finds the shadow block of the root node, and writes the modified cells to it. Finally, P2FS switches the current and shadow versions, as shown in Fig. 3(e). If data block $C_1'$ is now further updated to $C_1''$
(Fig. 3(f)), P2FS finds its shadow block $C_1$, and then writes the modified cells to it. This mechanism is likely to reduce the write traffic to PCM as blocks $C_1$ and $C_{1''}$ will often have similar contents. Path nodes are updated similarly, and finally, the file system becomes a new version as shown in Fig. 3(g).

Although this example shows versioning taking place after each update, real systems generally perform this at checkpoints, which are separated by a time interval chosen to balance performance against reliability. A system crash may occur while updating a series of blocks atomically. In this case, P2FS rolls back to the state before the updates started. This is the current version of the file system tree, which remains consistent while updating is performed on the shadow version.

### 3 Performance evaluations

A set of PCM simulators were developed and experiments were performed on these simulators with four file system benchmarks listed in Table II. The total capacity of PCM used in our experiments is 2 GB and the logical page size between host and storage is set to 4 KB. The latency and energy requirements of PCM operations are configured to the values given in Table I. The DCW overhead is set to 1% and 0.2% of the time and energy required for a write operation, respectively, following the original DCW configurations [6].

When a write request is delivered to storage, DCW writes just the cells whose current values differ from the existing values. To do this, the PCM controller compares the stored data and the data to be written, which accompanies additional read operations. However, this does not incur large overhead as a read operation of PCM needs only 50 ns (time) and 20 pJ/bit (energy), whereas a write needs 100–500 ns (time) and 100–500 pJ/bit (energy) depending on the state change of the cell (1 → 0 or 0 → 1). As DCW eliminates a large portion of writes at the expense of the slight read overhead, P2FS can reduce the latency and energy significantly.

Fig. 4 compares the total I/O time taken by P2FS with the copy-on-write file system, as the checkpointing period is varied. As the checkpointing period...
is extended, file systems perform fewer writes because those that occur between two checkpoints are buffered in a shadow block. As shown in Fig. 4, P2FS performs significantly better than copy-on-write for all the benchmarks, and the total I/O time is reduced by 48% on average. The greatest performance improvements occur with Benchmarks A and D, in which I/O time is reduced by 56% and 52%, respectively. As these two benchmarks contain a large proportion of small writes, there is high similarity between versions. With Benchmark B, P2FS is 41% faster than copy-on-write, even though there is much less similarity between data versions. As shown in Fig. 2, only a small proportion of updates are similar from the previous versions in Benchmark B. P2FS is still effective in this situation because it reduces the traffic involved in writing path nodes, which mostly involve small data. In addition, P2FS does not update a path node when it already points to the old version. In contrast, copy-on-write has to reconstruct all path nodes, as the location of a block changes each time. Fig. 5 compares the energy consumption of P2FS and copy-on-write. As shown in the figure, P2FS reduces the energy consumed in PCM writes by 65% on average.

4 Conclusion

This letter presented a new file system for PCM called P2FS, which performs writes efficiently utilizing the similarity between data blocks before and after write operations. Experiments showed that P2FS reduces I/O times by 48%, and power consumption by 65% compared with copy-on-write.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (2011-0028825 and 2012-0008581).