Design and implementation of a random access file system for NVRAM

Mi Zhou\textsuperscript{1(a)}, Xiaogang Chen\textsuperscript{1b}, Yang Liu\textsuperscript{2}, Shunfen Li\textsuperscript{1}, Gezi Li\textsuperscript{1}, Xiaoyun Li\textsuperscript{1}, and Zhitang Song\textsuperscript{1}

\textsuperscript{1} State Key Laboratory of Functional Materials for Informatics and Nanotechnology Laboratory, Shanghai Institute of Micro-system and Information Technology, Chinese Academy of Sciences, Shanghai 200050, People's Republic of China

\textsuperscript{2} Neusoft, Shenyang 110179, People's Republic of China

\textsuperscript{a)} mizhou@mail.sim.ac.cn
\textsuperscript{b)} chenxg@mail.sim.ac.cn

Abstract: Storage Class Memory (SCM) emerges as a potential technology because of its byte-addressability and non-volatility. However, existing file systems are not efficient for SCM because they are optimized for hard disks devices. In this paper, we propose a new file system called random access file system (RAFS), which is implemented on Linux as a prototype. In RAFS, we simplify system architecture and adopt memory mapping and XIP mechanisms to improve performance. We have analyzed and evaluated its performance with multiple workloads. Results show that RAFS outperforms Ext4 under the same condition, especially for random access scenarios.

Keywords: NVRAM, file system, performance, VFS

Classification: Storage technology

References

[1] C. H. Lam: 2010 10th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT) (2010) 1080. DOI:10.1109/ICSICT.2010.5667551
[2] R. F. Freitas and W. W. Wilcke: IBM J. Res. Develop. 52 (2008) 439. DOI: 10.1147/rd.524.0439
[3] Process integration, devices, and structures. In International Technology Roadmap for Semiconductors (2007).
[4] International Technology Roadmap for Semiconductors, Emerging Research Devices (2007).
[5] S. Qiu and A. L. N. Reddy: 2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST) (2013). DOI:10.1109/MSST.2013.6558434
[6] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger and D. Coetzee: SOSP (2009). DOI:10.1145/1629575.1629589
[7] X. Wu and A. L. Reddy: Proc. of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM (2011) 39.
[8] D. S. Roselli, J. R. Lorch and T. E. Anderson: USENIX Annual Technical Conference, General Track (2000) 41.
[9] Y. Joo, Y. Choi, C. Park, S. W. Chung, E. Chung and N. Chang: Proc. of the 4th International Conference on Hardware/Software Codesign and System
1 Introduction

For decades, the performance gap between main memory and secondary storage has been a bottleneck in computer systems. To relieve this problem, storage class memory (SCM) has been developed to fill up the gap [1]. SCM is depicted to encompass a combination of the attributes of memory and secondary storage [2]. A bunch of non-volatile random access memory (NVRAM) technologies, such as MRAM (Magnetic RAM) and PCRAM (Phase Change RAM) [3] are regarded as potential candidates for SCM. They can be attached to the memory bus directly, blurring the distinction between memory and secondary storage [4].

However, there are several challenging issues in using current file systems directly to NVRAM. Most file systems assume the storage devices are block devices. They access storage devices through generic block layer and the emulated block I/O operations, leading to large amount of data migration [5]. The overhead caused by data migration is unnecessary since a file system designed for NVRAM devices can be built on top of the memory access interface directly [6]. SCMFS is specifically designed for SCM and it eliminates unnecessary overheads in the hierarchy [7]. However, it is implemented on the virtual address space, leading to difficulty in management.

In this paper, we propose a novel file system—Random Access File System (RAFS), which is specifically designed for NVRAM. Compared to SCMFS, RAFS is applied on physical address space and can achieve better efficiency. With consideration of compatibility, RAFS exports the identical interfaces as the regular file systems do so that existing applications can work on it. In this file system, we aim to take full advantage of NVRAM and improve data access efficiency. We implemented RAFS on Linux 3.3.8 and performed experiments with two file system workloads. Measurement results demonstrated that RAFS has much better performance on random access and dealing with small size records than the widely used Ext4.

2 Design of RAFS

2.1 File system architecture

The architectures of regular file system and RAFS are shown in Fig. 1. Most of current file systems are based on block-devices and use complex block allocation algorithms for compatibility via virtual file system (VFS) [8]. They are built on top of the generic block layer and require file transformation from block level to page level, as shown in Fig. 1(a). For NVRAM devices with high access speed, they can be directly attached to CPU. Therefore, the complex page cache module and block management module will cause unnecessary overheads. In order to reduce these
overheads and acquire a more efficient file system, we eliminate block-related operations and layers by modifying page fault interrupt processing, as shown in Fig. 1(b).

![Fig. 1. (a) Architecture of regular file systems (Ext3, Ext4, and tmpfs). (b) Architecture of RAfS.](image)

### 2.2 File system layout

The layout of memory space in RAfS is shown in Fig. 2. The file system space consists of four parts. The first part is the super block (SB), which contains the information about the whole file system. The SB consumes 128 bytes and locates at the very beginning of the file system. The second part is the inode table, which contains the fundamental information of each file. The third part is the block in-use bitmap in which there is a bit signifying whether the target position is available for write. All the content of the files in the file system are stored in the data space.

With the layout, the file system can be easily recovered or restored after rebooting. First, mapping will be rebuilt based on information in metadata. Then the detail information about the file system can be recovered from the super block.

![Fig. 2. The layout of RAfS.](image)

### 2.3 RAfS memory mapping

Memory mapping is an efficient method to access files. Take read operation as an example as shown in Fig. 3. When a process need to access a file, files will not be copied into memory and the relevant data is set up and initialized using mmap() system call instead. Mmap() will return a pointer to an address in logical address space. Processes can manipulate files by the pointer instead of invoking read() or
write() system calls. Regular memory mapping uses interrupt handler function to copy file contents from SCM to user space and there exists one time copy in this process. Considering the memory addressing ability of SCM, we propose a novel memory mapping scheme. In this approach, system can acquire the address of files in SCM devices and access files without copy operation. As a result, the copy overhead can be avoided and RAFS exhibits higher efficiency.

2.4 Execute in place (XIP)
For modern computer system, XIP is a method of executing programs directly from long-term storage rather than copy it into RAM [9]. It is an extension of using shared memory to reduce the total amount of memory required and data migration. Current file systems based on block layer copy programs from memory to pages or vice versa each time reading or writing a file. During this process, the operating system switches from kernel mode to user mode and data has to be copied multiple times. Employing XIP, RAFS doesn’t copy programs to pages, eliminating the overhead of copy operations. File contents don’t need to be copied to memory before executing files and files are executed in NVRAM devices instead. To achieve this, we adopt three schemes in RAFS. The first scheme is that RAFS ignores all block layer operations. This is possible because NVRAM is can be addressed directly. The second scheme is applied in address space operation that RAFS relies on the MMU for mapping logical addresses to physical addresses on NVRAM. Then, read and write operations are simplified to read and write physical addresses directly. With these optimized designs, RAFS can have much shorter data access latency and provide better support for XIP operation.

3 Performance evaluation
To assess the effectiveness of RAFS, we implemented it on Linux 3.3.0. The experimental platform consists of ARM Core A9 MP CPU and 512M DDR3 memory. Since commercially available NVRAM is limited, we use DRAM as storage medium [10]. We also measure the performance of Ext4 mounted on a ram-disk to provide an equivalent hardware platform. A well-known benchmark named IOZONE was used to present the evaluation results. IOZONE is a famous micro benchmark that measures file I/O performance by generating particular types of
operations [11]. Besides, a test program written by ourselves is employed to provide specific measurement. The test program reads and modifies 320 MB data sequentially/randomly, and then writes the modified data back to storage devices.

3.1 Benchmark results

Fig. 4 shows the throughput of Ext4 and RAWS for various configurations of IOZONE 3.429. We perform experiments with four scenarios that are write, read, random write and random read. To investigate the effectiveness of RAWS under different record sizes, we measure the performance varying the record size from 1 KB to 8 MB. The total write amount of test file reaches 200 MB.

We have noticed that RAWS performed much better with all the record lengths compared to Ext4, especially in the random write workload. The reason is that Ext4 is created on the generic block layer and has much higher complexity than RAWS. Simplicity of the hierarchy significantly decreases data migration caused by multi-level cache, resulting in highly efficient data exchange. We also notice that in the random write/read workload, the performance of Ext4 decreases dramatically when the record length is less than 8 KB while RAWS provides consistently the best performance regardless of the record length. This indicates that RAWS achieves large performance enhancement in random access scenarios.

3.2 Program test results

To investigate the effectiveness of RAWS under different write sizes, we perform a read-update-write process with record size of 128 B, 1 KB and 10 KB, respectively. In Fig. 5(a), we can see that the performance of all file systems increases with the record size increases. RAWS spends shorter time with the same record size. There is a performance gap between sequential and random access in Ext4, while RAWS performs well in both modes. This indicates that block based file systems are not suitable for intensive random access and RAWS adaptively handles a various size of write operations efficiently. In the meantime, we can see from Fig. 5(a) that Ext4 is
slower when the cache is cleaned. This phenomenon doesn’t exist on RAFS as RAFS eliminates page cache mechanism.

We also compare the operation time during a test process to further explore the reason for the performance differences. The normalized time ratios for each operation are depicted in Fig. 5(b) and (c). It can be observed that the write time ratio of RAFS is much smaller than that of Ext4 in both cases because NVRAM have higher write speed. Read/Map time of RAFS can be negligible due to memory mapping while it accounts for 23% of total time for Ext4 in small block size case.

4 Conclusions

This paper presents the design of RAFS, a novel random file system especially for SCM. It improves the performance by intelligently exploiting the special features of NVRAM such as byte-accessibility and uniform access time. RAFS simplifies file system hierarchy to avoid overheads between layers. High efficiency mechanisms such as memory mapping and XIP are integrated to make RAFS efficient for NVRAM devices. We evaluate the potential performance improvement in RAFS with comparison between Ext4 based on block devices. Experiments with multiple workloads show that RAFS consistently outperforms Ext4 under the same condition, especially for random access scenarios. We have also observed that the impact of record size is small when using RAFS.

Acknowledgments

This work is supported by National Key Basic Research Program of China (2011CB932804, 2013CBA01900, 2011CBA00607), the “Strategic Priority Research Program” of the Chinese Academy of Sciences (XDA09020402), National Integrate Circuit Research Program of China (2009ZX02023-003), National Natural Science Foundation of China (61176122, 61106001, 61261160500, 61376006), Science and Technology Council of Shanghai (11DZ2261000, 12NM0503701, 12QA1403900, 13ZR1447200, 13DZ2295700).