# In-Memory File System for Non-Volatile Memory

Hyunjun Kim
College of Information and Communication Engineering
Sungkyunkwan University
Suwon 440-746, Korea
hjunkim@skku.edu

Joonwook Ahn
College of Information and Communication Engineering
Sungkyunkwan University
Suwon 440-746, Korea
ajw1507@skku.edu

Sungtae Ryu
College of Information and Communication Engineering
Sungkyunkwan University
Suwon 440-746, Korea
stryu@skku.edu

Jungsik Choi
College of Information and Communication Engineering
Sungkyunkwan University
Suwon 440-746, Korea
chjs@skku.edu

Hwansoo Han
College of Information and Communication Engineering
Sungkyunkwan University
Suwon 440-746, Korea
hhan@skku.edu

## ABSTRACT

Current memory system hierarchy consists of cache memory, main memory, and secondary storage. Each level in this hierarchy has a different access speed to tolerate the long latency of the lower level but supports a large capacity as a whole memory system. As new technologies introduce non-volatile memories powerful enough to close the performance gap of the previous generation non-volatile memories, we can devise a new memory system that replaces DRAM with non-volatile memory for the main memory. To adopt a new memory technology, we discuss a new file system based on *tmpfs* with three aspects that need to be included, and evaluate its performance against popular file systems such as *ext4*.

## Categories and Subject Descriptors

D.4.4 [**Operating Systems**]: File Systems Management—*Access methods*; B.3.2 [**Memory Structures**]: Design Styles—*Primary memory*

## General Terms

Design, Performance, Experimentation.

## Keywords

non-volatile memory, file system, performance, in-memory file system, virtual memory file system.

## 1. INTRODUCTION

Memory hierarchy in current computer architectures is well-established for years. From the register file in the processor, cache, main memory, and secondary storage are organized to provide fast and large memory systems. Recently, new memory technology in circuit level design begins to introduce non-volatile memories, which match up with DRAM technology in access times and excel in power consumption. These prospective memory devices are widely regarded as candidate replacements or supplements for DRAM in the main memory. Compared to DRAM, most non-volatile memories have been still lagging in performance. However, memory devices based on STT-MRAM [3] begin to show equal — at least comparable performance in read/write access times. Furthermore, STT-MRAM can be close performance even to SRAM in read/write access times. Another promising feature of STT-MRAM is its endurance. Compared to DRAM, no non-volatile memories introduced so far fall short in their endurance, but STT-MRAM has virtually equal endurance to DRAM. According to early report [3], STT-MRAM allows lifetime updates of more than $10^{15}$ times, which is very close to the endurance of DRAM — $10^{16}$. Meanwhile, other non-volatile memory technologies such phase-change memory (PCM) or flash memory only show $10^8$ and $10^5$ lifetime endurance, respectively.

Many believe that promising characteristics of STT-MRAM assure its position as a next generation memory technology for the main memory in computer systems. In addition to that, as the device technology advances, many computer architects investigate various memory hierarchy designs which adopt non-volatile memories [8, 6, 2]. To this trend, we believe the file system, which is the last element in memory hierarchy at the secondary storage, should be change to embrace a new non-volatile memory technology. Non-volatile memory such STT-MRAM can replace the main memory composed of volatile DRAM technology. Thus, even after power-off, the contents of the main memory can be kept at the main memory. This is an important change not only in memory system but for the system software which manages the main memory and the secondary storage. Files have been an abstraction for persistent data storage for a long time. In a computer system equipped with non-volatile memory as the main memory, however, files can be located in the main memory. As memory technology improves, the capacity of non-volatile memory will increase. This also help justifies keeping files — the system software abstraction for the persistent data — in the main memory.


*RACS'13* October 1-4, 2013, Montreal, QC, Canada.

In this paper, we will evaluate a file system constructed mainly on the non-volatile main memory. Since operating systems have been developed to keep *page cache* for delayed writes to the second storage, most recently used file contents are kept at the main memory represented with page cache in the kernel space. *Tmpfs* [10] is a file system originally developed to keep temporary files during application runs. Since temporary files can be lost after power-off, *tmpfs* is a page-cache-only file system. No matching file system exists on the secondary storage. Only if the virtual memory demands memory pages, some of its contents in the page cache are swapped out to the backing store at the secondary storage — which is just an extension of memory address space, not file system. We believe file systems similar to *tmpfs* are perfect fit for the file system on non-volatile memory. Since non-volatile memory contains its contents during power-offs, file contents on the main memory are kept as persistent data. Secondary storages only work as backing stores for overflow contents out of the main memory.

# 2. CURRENT TECHNOLOGY REVIEWS

There are various types of non-volatile memories, but those are not yet commercially produced. However, in the near future, it seems to release a non-volatile memory product which is comparable to speed with the conventional main memory. The file system has developed to improve performance and reliability on block-device, such as HDD, Flash, and so on. Most of the file systems are based on block-device, but there are file systems operated on the main memory. These commonly purposes for acceleration of application using data caches.

## 2.1 Non-volatile memory

DRAM is used as a main memory in modern computer architecture and it has a characteristic, which is called volatile, that determines its stored electrons to save data and erase them all when power is off. For this reason, volatile memory needs to refresh its electrons by supplying power periodically. On the other hand, non-volatile memory: resistive memory, phase-change memory, magnetoresistive memory, and ferroelectric memory, can keep its contained information without any power supply. However, file access latency and durability of previously implemented non-volatile was unsuitable for using as a main memory. According to a recent study, the gap in the performance between volatile and non-volatile memory has been blurred and the possibility of using a non-volatile as a computer's main memory has been proposed [3, 8, 6].

## 2.2 Characteristics of existing file systems

In this section, we explain extended file system, RamDisk file system, and virtual memory file system. These file systems are available in the latest Linux kernel distribution. Most file systems are based on block-devices, and use block allocation algorithms and index nodes for compatibility via virtual file system (VFS). These file systems have been developed in order to archive both performance and reliability, but most of them focus on traditional memory system hierarchy that consists of cache memory, main memory, and secondary storage.

### 2.2.1 Extended file system: ext3 & ext4

Extended file system (*extfs*) is a main file system for Linux, and a representative file system for the block-device. Nowadays, most Linux distribution release versions use *ext4*, which is the latest version for the extended file system. Since *ext3* is equipped with the journaling mechanism for reliability, this file system causes additional overheads while writing files, but it is important to maintain a consistency of file systems. To enhance the performance, delayed allocation mechanism is available in *ext4*. However, this mechanism has still a potential risk of data loss in cases of system crash or power failure before all of the data has been written to a real device. In case of in-memory file system on non-volatile memory, it can be an improper mechanism. Because it is possible that both main memory space and file system space are in same device or different devices which have similar I/O speed, it is unnecessary to write a file on non-volatile memory from the main memory or system memory space which is in non-volatile memory. Therefore, it is more important to guarantee reliability mechanism between the cache memory and the main memory than to guarantee it between the main memory and the secondary storage.

### 2.2.2 RamDisk for memory-based file systems

*RamDisk* makes a main memory be a block device like a secondary storage. In fact, RamDisk does not provide file management features and it only implements RamDisk which is viewed as typical block device. Therefore, to store files on RamDisk, we have to make a file system on this RamDisk. Because RAM has faster throughput than second storage devices, all file systems on RamDisk are faster than file systems on block devices. However, the allocated region for RamDisk in memory is operated as a block device. Moreover, because of most existing file systems use a block allocation algorithm or similar one and these can be unnecessary overheads, RamDisk is improper in-memory file system and it needs a new file system with new data-allocation algorithms and reliability guarantee mechanisms.

### 2.2.3 Virtual memory file system: tmpfs

*Tmpfs* [10] is based on page cache. Unlike *RamDisk*, it provides file management and swap mechanism. To store files, *tmpfs* requests the page to VM sub system. This means *tmpfs* is page-cache-only file system, not block-based file system. *tmpfs* stores file data on page and it causes better performance than any other file systems do. *Tmpfs* does not matter where files come from, and deciding where to save the file is the role of VM sub-system. In other words, *tmpfs* uses only VM and demands to load or store the file on the page in VM. If there is a lack of main memory capacity for system use or files required by *tmpfs* are not on VM pages, swapping mechanism operates automatically on VM sub system.

Fundamentally, *tmpfs* is not a file system focused on non-volatile memory. The file system is designed to provide rapid temporary storage space for application cache. Therefore, there is no mechanisms like journaling, shadow-paging or checkpointing for reliability in *tmpfs*. Nevertheless, this can be a better solution for non-volatile memory file system currently than any other file systems, because it is based on VM and has a swapping mechanism. Moreover, *tmpfs* is designed to be fully compatible with VFS layer. This feature makes legacy files be available on in-memory file system.

As above, *tmpfs* can be an currently optimal solution, but

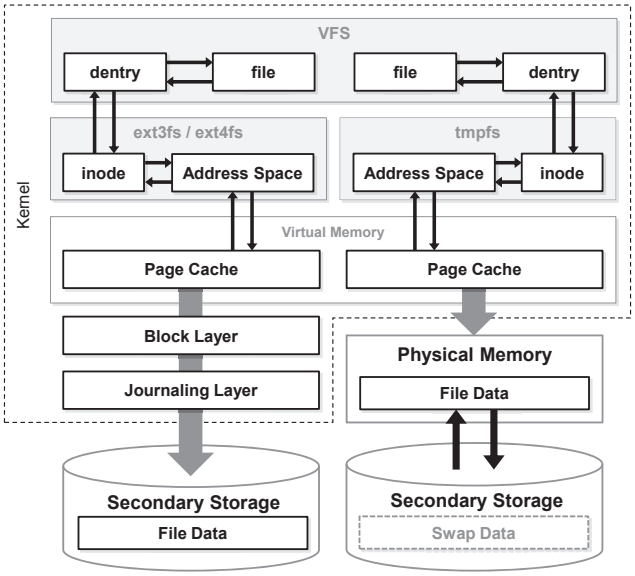


**Figure 1: Architecture of *ext3*, *ext4*, and *tmpfs***

there are still some parts for improvement to be an optimal solution in non-volatile memory file system. Firstly, from the perspectives of CPU and kernel, *tmpfs* is just a file system. It means that *tmpfs* writes page data of a file to the main memory when VFS requires the file data. This is unnecessary transmission because it is highly possible that in-memory file system is inside the main memory or same level on memory hierarchy, and we need to optimize that. Secondly, *tmpfs* has to provide a mechanism to guarantee the reliability of file system. However, this mechanism is different from existing methods, because they mainly consider reliability between main memory and second storage. For in-memory file system, it is more important reliability between CPU cache and in-memory file system. The writing to CPU cache line is guaranteed by hardware, but its size is very limited. Ordinary cache size is only 64 bytes. Therefore next reliability guarantee algorithm has to consider this aspect. Finally, there needs to research novel method to execute application binaries on file system instantly. The file system using non-volatile memory has very rapid I/O speed, and CPU can access to file system directly as fast as main memory. In a similar context of first, it is unnecessary transmission of application data to the main memory space for execution.

# 3. IN-MEMORY FILE SYSTEM

In the Figure 2, it shows proposing memory based file system. This in-memory file system can store its information constantly in the main memory by using non-volatile memory device as the main memory. In other words, the main memory and the file system can share space in a same device. To support the non-volatile main memory, it needs a new design for file system because the formal operating system and file system are implemented to consider the latency between a volatile main memory and a secondary storage. The newly designed memory based file system can expect its improvements with its performance from three aspects below.

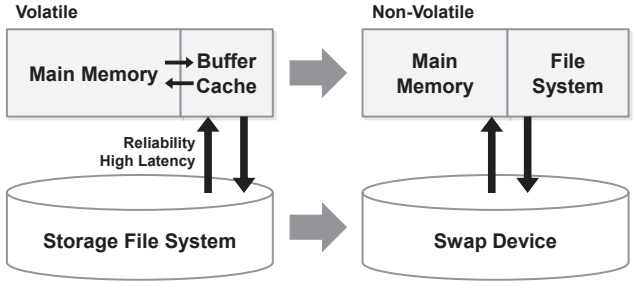


**Figure 2: Design of File System for non-volatile memory**

## 3.1 Page cache as main file system

Since the file system and files are maintained in a page cache in the non-volatile main memory, latency of file access in a secondary storage can be minimized. Our new file system design transfers data in a page-level within the main memory, and it decreases possibilities of accessing secondary storage, which consumes heavy CPU clock cycles for read/write operations. Conventional block-based file system requires file transformation from block level to page level, which is unnecessary data structure for our newly designed memory based file system. Block-level file transfer also copies information from block device into the buffer cache in the main memory that will be used, which is redundant in the in-memory based file system. However, in the in-memory based file system, file is maintained in the page cache and does not require extra caching operation for file access, which saves its storage space from unnecessary file duplication. Therefore, page allocation file system is required. By using page cache only, these unnecessary block-related operations are eliminated and storing the file data in VM pages.

## 3.2 Reliability for CPU cache

Performance of the file system can be improved by reducing modules which are used to guarantee file system reliability. Conventional file system relies on heavy reliability techniques such as journaling, shadow paging and checkpointing to ensure the modified files in the main memory are flushed into the secondary storage without any data loss from system crashes or failures. However, proposing in-memory file system persistently maintains data in its own space, which is non-volatile, and does not need extra operations to sustain consistency between the main memory and the secondary storage. Despite of non-volatile characteristic in the main memory, a reliability between CPU cache and in-memory file system is still important and necessary to concern. To guarantee its reliability, each file operation has to be atomic and ordered.

## 3.3 Keeping clean pages in swap

Since the non-volatile memory device still has its space limits, page swap occurs frequently to manage the small space of the main memory within the in-memory file system. This swap can bring heavy overheads due to frequent disk accesses. Conventional swap is designed to place a page exclusively either on the main memory or on the swap device. Thus, unmodified pages for files can be repeatedly swapped in and out, if these pages are accessed and later selected as

**Table 1: Sysbench/filebench parameters**

| | | #Files | Size of File | Total File Size | Memory Size |
|---|---|---|---|---|---|
| *sysbench* [4] ver. 0.4.12 | sequential read<br>sequential write<br>random read<br>random write | 1 | 5.0 GB | 5.0 GB | 12 GB |
| *filebench* [1] ver. 1.4.9.1 | fileserver | 50,000 | 128 KB | 6.1 GB | 4 GB |
| | webserver | 40,000 | 160 KB | 6.1 GB | |
| | webproxy | 40,000 | 160 KB | 6.1 GB | |
| | random r/w | 1 | 5.0 GB | 5.0 GB | |
| | mongo DB | 4,000 | 1,600 KB | 6.1 GB | |

**Table 2: Experimental Environment**

| | Descriptions |
|---|---|
| CPU | Intel Core i5-2400 3.10GHz |
| Memory | 12 GByte DRAM (sysbench)<br>4 GByte DRAM (filebench) |
| HDD | 500GB 7200RPM HDD |
| Kernel | Linux Kernel 3.9.2 |
| Profiler | Intel VTune Performance Analyzer [9] |

eviction pages. However, these redundant disk write operations can be eliminated by retaining once-swapped file pages in the swap device and allowing only modified pages to be written to the swap device.

# 4. EXPERIMENTAL EVALUATION

To evaluate the performance of in-memory file system on the non-volatile memory, we experimentally run *sysbench* [**?**] on three different file system environments. We measure the performance of *ext4*, one of the representative existing file system, on two different devices such as block-device and RamDisk. Then, we compare the read/write latencies of *tmpfs* [10] against *ext4* on RamDisk. We also run *filebench* [1] to evaluate how the swap mechanism works with *tmpfs*.

## 4.1 Experiment environment

To confirm the three different consideration points explained above, we first run a file system benchmark with *sysbench*. The file system benchmark is composed of four different types of file accesses — sequential read (*seqrd*), sequential write (*seqwr*), random read (*rndrd*), and random write (*rndwr*). We run those four workloads on three different environments. We select a widely used file system, *ext4* [7] and install this file system on two different secondary disks — magnetic hard disk drive (HDD) and RamDisk [5]. In addition, we also experiment with five different types of realistic workloads from *filebench* — file server, web server, web proxy, random read/write, and mongo DB. For these workloads, we ran the first 10 minutes of each workload for swap behavior evaluation. We set up in-memory file system, *tmpfs* [10] on DRAM main memory. To expose swap behavior, we boot up the Linux machine only with 4GB DRAM for *filebench* workloads. Furthermore, since commodity STT-MRAM is not available yet, we assume DRAM simulates the STT-MRAM main memory. In other words, we assume the read/write latencies to STT-MRAM are almost equal to those of DRAM. The summary of benchmark parameters for *sysbench* and *filebench* are shown in Table 1. Our experiments are all performed on a Linux PC described in Table 2.

**Table 3: Execution time comparison (sysbench): *tmpfs*+STT-MRAM vs. *ext4*+HDD**

| | *tmpfs*+STT-MRAM | *ext4*+HDD |
|---|---|---|
| sequential read | 0.76 sec | 53.8 sec |
| sequential write | 1.47 sec | 57.2 sec |
| random read | 0.03 sec | 72.9 sec |
| random write | 0.04 sec | 88.9 sec |

## 4.2 Result analysis

We first compare the performance between *tmpfs* on STT-MRAM and *ext4* on the secondary storage (HDD). The result in Table 3 shows that sequential read and write operation of *ext4* file system operated on hard disk drive is 40–70 times faster than that on secondary storage. Furthermore, random read and write operation of *ext4* file system on RamDisk is at least 2,000 times faster than that on secondary storage. This is how disk I/O can affect the performance of the system as mentioned above.

In Figure 3, we compare the performance between *tmpfs* on STT-MRAM and *ext4* on RamDisk. Particularly, sequential write shows how journaling, a reliability technique in *ext4*, makes performance impact on RamDisk. It causes a heavy overhead, which is presented with the portion labeled *jdb2* in Figure 3. This portion does not exist in the result on *tmpfs*. As *tmpfs* is made for temporary file management, reliability technique is unnecessary. However, for in-memory file system, this result shows the potential performance improvement from reliability technique elimination. Furthermore, by the experimental results of Figure 3, we can compare the performance of the *ext4* and *tmpfs* operating on the memory. This shows that *tmpfs* has better overall performance than conventional file system based on block devices. Particularly, *ext4* is slower in sequential writing speed than that of *tmpfs*, because *ext4* is the file system for block devices. *Ext4* even consumes memory spaces twice than *tmpfs* does, because the operating system recognizes the RamDisk as a block device, and it copies files into a buffer cache then accesses to it by block I/O. Consequently, there are performance discrepancies between *tmpfs* and conventional file systems.

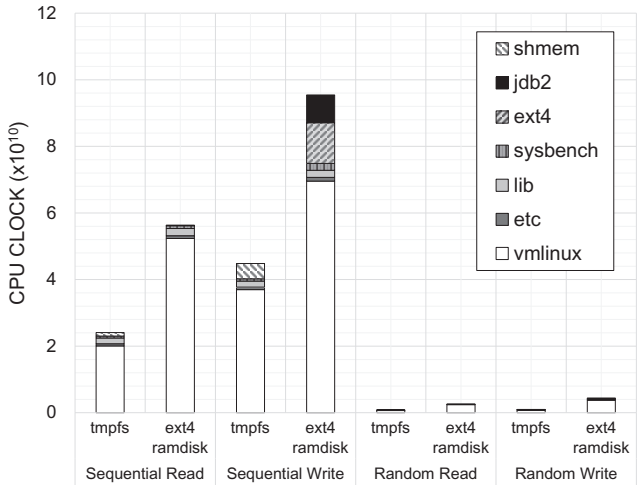


**Figure 3: Execution cycle comparison: *tmpfs*+STT-MRAM vs. *ext4*+RamDisk**

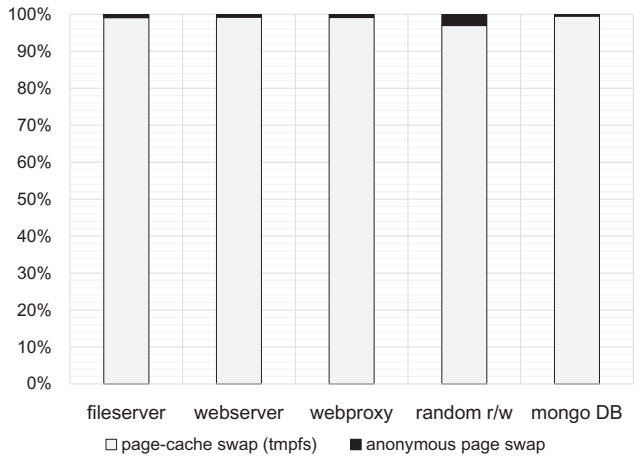


**Figure 4: Percentage of swap from file system: page cache (*tmpfs*) vs. anonymous page**

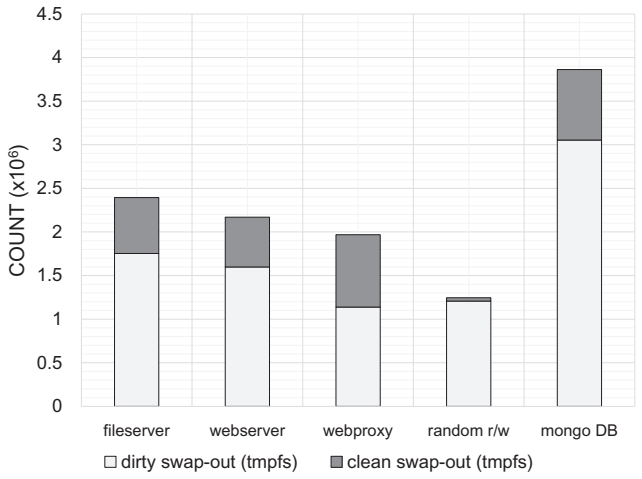


**Figure 5: Number of block I/O: clean swap-out and dirty swap-out (*tmpfs*)**

In Figure 4 and Figure 5, we also verify that swap using in *tmpfs* needs to be modified for its performance improvement. The result shown in Figure 4 concludes that more than 97% of total swaps occurred in the system come from the file system (*tmpfs*), which implies that most of the swap overheads are for page cache swaps for files on *tmpfs*. These overheads actually lead to a downgrade of overall performance. Furthermore, by the experimental result in Figure 5, we can verify that clean-swap-out takes a high portion of *tmpfs* total block I/O operations, except random-read-and-write (*random r/w*). Those unnecessary clean swap out disk I/O operations can be removed. Roughly, 20% of the disk I/O can be reduced for most of the workloads. This result implies that swap in *tmpfs* can be improved by reducing the number of disk accesses.

# 5. CONCLUSION

With the advent of non-volatile memory devices in computer systems, a memory based file system is proposed. *Tmpfs* has matching characteristics for our proposed file system. We experimentally evaluate the potential performance improvement in newly designed system with performance comparisons between the conventional file system based on block device and in-main memory file system based on non-volatile memory. In particular, the large difference in software layer performance is due to the overhead of reliability technique. When *ext4*, a conventional file system is installed on RamDisk, its performance lags behind in-memory file system (*tmpfs*). Since conventional file systems have block-device access layers, which are also equipped with reliability techniques such as journaling, it is inefficient to use conventional file systems for non-volatile memory systems. The contents in page cache on the non-volatile main memory are already persistently written. Thus, writing them to hard disk drives or duplicating them on the main memory via RamDisk is a unnecessary operation. However, reliable transmission of file data from caches in the processor to the main memory still needs additional consideration. Therefore, several techniques are applied between the main memory and the storage disk should be adopted and redesigned to guarantee the reliable file data transfer between the cache and the main memory. In addition, since *tmpfs* is originally used as a temporary storage purposes, this file system needs to be extended to perform optimally as a main file system. Even with the addition of the reliability technique between the cache and the main memory, the performance of the new in-memory file system is expected to be within the range of experimental results. We also can expect the performance improvement with optimizing inefficient swap operations for *tmpfs*. Moreover, in-memory file system requires a lot of memory space to drive large amounts of file system. A huge number of swap operations with limited memory resources needs newly optimized replacement policy. In the future, we will address these issues for the non-volatile memory file systems.

# 6. ACKNOWLEDGMENTS

This research was supported by Korean government under the NRF grant (No. 2012011602) and the Korea Copyright Commission grant in 2013.

# 7. REFERENCES

[1] Filebench. [online] http://filebench.sourceforge.net/.

[2] K. Bailey, L. Ceze, S. D. Gribble, and H. M. Levy. Operating system implications of fast, cheap, non-volatile memory. In *the 13th USENIX conference on Hot topics in operating systems (HotOS-13)*, pages 2–2, April 2011.

[3] A. Driskill-Smith. Latest advances and future prospects of stt-ram. In *Non-Volatile Memories Workshop (NVMWÕ10)*, April 2010.

[4] A. Kopytov. Sysbench: a system performance benchmark. [online] http://sysbench.sourceforge.net/, 2004.

[5] P. Koutoupis. The linux RAM disk. *LiNUX+ magzine*, pages 36–39, April 2009.

[6] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger. Phase-change technology and the future of main memory. *IEEE Micro*, 30(1):143–143, January 2010.

[7] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier. The new ext4 filesystem: current status and future plans. In *Linux Symposium, volume 2*, pages 21–34, June 2007.

[8] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *International Symposium on Computer Architecture (ISCA'09)*, pages 24–33, June 2009.

[9] J. Reinders. *VTune performance analyser essential*. Intel Press, 2005.

[10] P. Snyder. tmpfs: A virtual memory file system. In *the Autumn 1990 European Unix Systems User Group (EUUG) Conference*, pages 241–248, October 1990.