HeRMES: High-Performance Reliable MRAM-Enabled Storage

Ethan L. Miller *elm@cs.ucsc.edu*
Scott A. Brandt *sbrandt@cs.ucsc.edu*
Darrell D. E. Long *darrell@cs.ucsc.edu*
Computer Science Department
University of California, Santa Cruz

Abstract

Magnetic RAM (MRAM) is a new memory technology with access and cost characteristics comparable to those of conventional dynamic RAM (DRAM) and the non-volatility of magnetic media such as disk. Simply replacing DRAM with MRAM will make main memory non-volatile, but it will not improve file system performance. However, effective use of MRAM in a file system has the potential to significantly improve performance over existing file systems. The HeRMES file system will use MRAM to dramatically improve file system performance by using it as a permanent store for both file system data and metadata. In particular, metadata operations, which make up over 50% of all file system requests [14], are nearly free in HeRMES because they do not require any disk accesses. Data requests will also be faster, both because of increased metadata request speed and because using MRAM as a non-volatile cache will allow HeRMES to better optimize data placement on disk. Though MRAM capacity is too small to replace disk entirely, HeRMES will use MRAM to provide high-speed access to relatively small units of data and metadata, leaving most file data stored on disk.

1. Introduction

Current file systems are optimized for the assumption that the only stable storage in the system is a block-oriented, high-latency device such as a disk. As a result, existing file systems use data structures and algorithms that transfer data in large units and take great pains to ensure that the file system's image on disk remains internally consistent. If the file system includes any non-volatile memory (NVRAM), there is usually a limited amount used as a temporary storage area to facilitate staging data to disk.

Magnetic RAM (MRAM) [4] is a new memory technology, currently in development, with the speed, density, and cost of DRAM and the non-volatility of disk. We are investigating the use of MRAM in the HeRMES (**H**igh-performanc**e**, **R**eliable, **M**RAM-**E**nabled **S**torage) file system to dramatically improve file system performance by storing metadata and some data in MRAM. Since MRAM will have cost comparable to that of DRAM, it cannot totally replace disk or other types of secondary storage such as MEMS [9]. Rather, we are researching the most effective ways to use limited amounts of MRAM in a file system.

An MRAM-based file system such as HeRMES has several major advantages over existing file systems in both performance and reliability. As we discuss in this paper, using MRAM in the file system can reduce the cost of metadata operations to nearly zero, leaving them limited solely by CPU speed. It also increases the speed of file reads and writes both by reducing metadata overhead and by allowing the file system to better lay out data on disk by buffering writes longer in safe MRAM. File system reliability is also greatly improved. Simplifying metadata structures results in less complex and more reliable software. Keeping metadata in MRAM also allows HeRMES to run consistency checks on the file system in the background during normal operation, allowing errors to be caught early, before they spread.

2. HeRMES design

The HeRMES file system is built from the ground up using two assumptions that differ from current file systems: metadata accesses need not be in large contiguous blocks, and metadata accesses take microseconds (at most) rather than milliseconds. These assumptions differ from those underlying disk-based file systems, which require milliseconds to access blocks of data.

2.1. Metadata management

HeRMES maintains all of its metadata in MRAM, avoiding the need to access the disk for metadata requests. The ability of MRAM to handle single-word reads and writes further benefits HeRMES by allowing it to use much simpler data structures. For example, the B+-trees used in XFS [16] make efficient use of large blocks at the expense of file system complexity. HeRMES, on the other hand, can use simpler data structures such as binary trees and hash tables with in-memory semantics because it does not need to allocate and reference structures in large blocks.

Keeping all metadata in MRAM could be prohibitive for traditional file systems, which can require up to a 1–2% overhead for metadata; 600 MB of DRAM for a 60 GB disk may be too expensive, with memory costs exceeding those of disk. HeRMES, in contrast, will make extensive use of compression and variable-sized allocations to drastically reduce needed space, avoiding this problem. For example, an inode in Unix might require 128 bytes; there would be little benefit to reducing its size on disk because retrieval time is dominated by access latency which would not be reduced for smaller objects. It might be possible to save small amounts of DRAM at the expense of transforming the inode when transferring it between disk and memory, but using information from other inodes to do the compression would be difficult. HeRMES, however, can use commonalities between inodes to reduce required space. For example, each file's inode can contain a pointer to an access control list; since many of a user's files have identical permissions, their inodes can share a single list. File index pointers can also benefit from compression and variable-sized memory-style allocation. Many file systems use extents to compress index lists; by storing lists of extents in variable-sized blocks of MRAM, HeRMES can eliminate wasted space.

One potential problem with keeping metadata in MRAM is that it may be *too* easy to modify data structures, potentially causing file system inconsistency. Wild references in the file system (or elsewhere in the operating system) could overwrite valid metadata in MRAM, corrupting the file system. HeRMES will avoid this problems using techniques similar to those in Rio [12]. By keeping file system MRAM protected except when explicitly necessary, HeRMES will ensure that only desired changes are made to MRAM. The process of switching a page from read-only to read-write in the page table is fast, and will not significantly slow down HeRMES MRAM operations, particularly since it is only necessary when metadata is modified.

2.2. MRAM write buffer

Like most file systems, HeRMES will buffer writes in memory for several reasons: allowing a process to continue without waiting for a write to go to disk, reordering writes to minimize disk latency, and waiting in the hope that a file will be deleted. Unlike many file systems, however, writes with HeRMES are safe once they are written to MRAM. This allows HeRMES to postpone writes as long as desired without fear of data loss due to a system crash.

The write buffer in HeRMES is similar to that in systems with NVRAM, with two important differences: MRAM is considerably faster than NVRAM, and metadata updates accompanying a write are done immediately in MRAM. Writes to MRAM are considerably faster than writes to flash RAM, which can require more than two milliseconds. MRAM's faster write time reduces the window of vulnerability during which data can be lost from a system failure.

Because MRAM is a long-term stable store, data written there can be kept as long as necessary. This allows HeRMES to optimize data placement on disk, reducing time wasted to disk access latency. Existing file systems do this as well, but they run the risk of data loss if they hold data in the write buffer too long. Many systems with "non-volatile" RAM actually use battery-backed RAM, which can lose data because of dead batteries in addition to the usual dangers of storing data in RAM.

2.3. MRAM file storage

MRAM may also be useful for disk reads, particularly if there is a relatively large amount of MRAM in the system. Disk latencies are currently around 5–10 ms; in that time, a disk can transfer 64–128 KB of data. The file system can keep the first few blocks of each file in MRAM, transferring the data out of MRAM while the disk seek is completed. Combining this technique with file access prediction and clustering on secondary storage [1] will further improve performance by reserving the scarce MRAM resource for "live" data. As probe-based storage [9] becomes available, this technique will become more effective because the latency to data on secondary storage will be lower, reducing the amount of file data that must be buffered in MRAM and increasing the number of files for which such buffering is possible.

As with write buffering, caching file headers (or entire files, if they are small) is not a new technique. However, MRAM makes this technique more attractive because it allows the structures to survive power loss and system reboot, enabling the file system to build such a cache over time without the need to preserve it on disk or reload it after a system restart.

3. Performance

HeRMES can significantly outperform existing file systems for several reasons. First, metadata operations in HeRMES are nearly free because they only require memory-type accesses. Table 1 shows several common file system request types [14], noting the disk operations needed to satisfy each one. Existing file systems cache metadata in DRAM, updating the original on disk when changes occur. Though they can eliminate many (but not all) disk reads by caching, metadata writes must go through to disk to ensure consistency, and writes often have a partial order enforced on them to maintain file system consistency [13]. HeRMES, on the other hand, handles disk requests in the shaded columns entirely in MRAM, leaving only file data reads and writes to use the disk. This results in dramatically faster metadata operations, requiring microseconds rather than milliseconds to complete. Moreover, data writes can be safely buffered in MRAM indefinitely, as described in Section 2.2, further decreasing latency from user write to "safe" commit of the data.

Table 1. Disk I/O needed for file system requests.

Request	**Type of disk requests needed**			
	Global metadata	**File metadata**	**File index**	**File data**
File stat (50%)	–	read	–	–
File read (20%)	–	read write	read	read
File write (5%)	read write	read write	read write	read write

Because HeRMES metadata operations are limited only by CPU speed, the file system can satisfy them in the time it takes to execute the metadata operation in the CPU. For existing file systems, 20,000 – 40,000 operations are sufficient to execute a file system request; this is 40 to 80 μs on a modern processor, allowing a single processor file server to handle about 25,000 metadata operations per second; HeRMES will likely be able to do more operations per second because it can use simpler data structures (and thus fewer instructions to manipulate them) and has no need to spend instructions on managing disk I/O. If a file server provides, on average, one 4 KB file block for every two metadata operations, such a server could sustain 50 MB per second using a single commodity CPU.

The simple MRAM-resident data structures in HeRMES can provide added speed in another way: reduced lock contention. Disk-based file systems must use fine-grained locking to ensure high levels of concurrency in the face of relatively long metadata operations. In particular, operations that require reading data from disk can hold locks for milliseconds, potentially causing contention for locks. HeRMES, in contrast, can complete metadata reads or updates in less than 100 microseconds. This time is shorter than the scheduling quantum on many systems, and is thus less likely to result in high levels of lock contention. The contention problem is exacerbated on symmetric multiprocessor systems; again, HeRMES can use relatively course-grained locking and still maintain low levels of lock contention.

4. Reliability

File system reliability is, for many users, more important than performance: getting the correct data later is better than getting erroneous data now. HeRMES can provide high performance, as seen in Section 3, without sacrificing reliability. Moreover, HeRMES will be more reliable than existing file systems for several reasons, including lower software complexity and the ability to continuously check the system for consistency.

4.1. Reducing software complexity

By using relatively simple structures in MRAM, HeRMES reduces software complexity, making file system software more reliable. Simple data structures are well-understood and less prone to programming errors, reducing the likelihood that a bug will be hidden in thousands of lines of complex code. Because MRAM is so much faster than disk, there will be less temptation for programmers to take shortcuts that save a few microseconds, making it less likely that such a shortcut will malfunction.

The lower number of locks needed in HeRMES also increase software reliability. With metadata operations locking up structures for around 50 μs, there is no need for thousands of locks in the file system. On a uniprocessor system, in fact, a single lock for the entire metadata structure is sufficient because operations are CPU-bound and thus gain minimal benefit from interleaved requests. Even in multiprocessor file servers, a relatively small number of locks—at most one per file (for metadata), one for disk allocation, and one for memory allocation—will be sufficient to guarantee that processors are not waiting on file system locks. The net result is a lower probability of deadlock as well as less chance that data will be improperly modified.

4.2. Metadata checking

HeRMES will also take an active approach to protecting file system consistency by continuously checking the metadata structures while the system is running. A background process checking 2,000 files per second can fully check a system with ten million files in less than 90 minutes, yet it demands less than 10% of the system's resources to do so.

Checking the file system's metadata while the system is operating increases reliability in several ways. First, it is often easier to write a program that *detects* an error than it is to write a file system that doesn't produce errors in the first place. Merely detecting the error may be sufficient to attempt correcting it, or at least to prevent it from spreading to the rest of the file system. Second, most existing file systems *never* have their metadata checked. They rely on logging [10] and other techniques to recover quickly from a crash, but they do not examine metadata written during normal operation. This is necessary because a full check of the metadata on a large file system with ten million files might take hours, if not days, and would consume most of the disk bandwidth during that time. Third, extremely large file systems are now encountering a new problem: disk unreliability due to firmware errors and undetectable bit errors is becoming a concern. A bit error rate of 10^{-12} becomes a problem when file systems store a terabyte of data because bit errors may go unnoticed for days. Rather than do continuous checks, though current file systems must assume that their code does not contain any bugs and that the underlying media is reliable, assumptions that are increasingly less likely as file systems grow larger and more complex.

4.3. Backing up metadata

MRAM, like any other part of a computer, will be subject to component failure. Because MRAM is the only place metadata is stored, HeRMES must guard against MRAM failure. It does so by logging metadata changes to a location other than that holding the MRAM. This can be done in several ways. The first option is to write metadata changes to disk. This is very similar to logging, but does not involve the same ordering issues that metadata updates in conventional systems suffer. The second option is to keep the metadata log in a different bank of MRAM than that holding the original metadata. If MRAM can be removed from a computer, placed in a new one, and its contents read, this solution is sufficient to back up metadata at very little cost.

In either case, metadata update logging requires very little space. The majority of metadata updates are timestamp modifications, which can be recorded in a few bytes. More complex modifications take more space; however, MRAM can buffer changes and flush them to disk several times per minute. Using this mechanism means that total MRAM failure (chip failure) can lose small amounts of data, but that consistency is not affected. It is important to remember that chip failure is not a common source of computer failure, and that chip failure affects *all* file systems that use memory for caching and buffering.

5. Related work

Our work builds on many areas of file system research, but research into non-volatile RAM (NVRAM) systems and schemes to reduce latency for disk accesses, particularly metadata, is most relevant.

Douglis [6] and Wu [17] proposed the use of NVRAM to hold an entire file system. This approach is acceptable for relatively small file systems, but MRAM (like NVRAM) is too expensive to replace disk for general purpose file systems. Additionally, the flash RAM used in these systems does not support single word writes; instead, it requires 1–2 ms (or more) to write a relatively large block of data. This prevents fine-grained modification of data in non-volatile memory. In eNVy [17], copy-on-write and buffering were used to get around the long erase latency of flash RAM; this approach required extensive garbage collection similar to that used in log-structured file systems [3,15].

NVRAM has long been used for recovery and file system reliability [2], again with the restrictions of small size and coarse-grained write access. In such systems, NVRAM is used as a non-volatile cache for disk, but data "lives" on disk. This design improves file system reliability by reducing the window of vulnerability for written data and improves performance by relaxing metadata write constraints. However, it does not allow the rich metadata structures possible when metadata is permanently resident in MRAM, and writes must still be sent to disk, requiring disk seeks and consuming disk bandwidth.

Techniques for reducing disk latency and improving reliability for metadata include writing data to the nearest free disk blocks [7,11], logging [10], and soft updates [13]. All of these techniques reduce access latency for writes, but none reduces the *number* of blocks that must be written. Additionally, these techniques use little beyond caching to speed up metadata read access. Another technique, combining metadata with file data [8], allows data and metadata for small files to be read and written in a single contiguous request. However, this technique was only tried with relatively small files.

6. Current research

Our research into using MRAM for file systems, specifically HeRMES, has just begun. In this paper, we described several ways in which MRAM can be used to improve file system performance, but many questions remain. For example, what happens if MRAM is limited? If insufficient MRAM is available for all of the metadata, how can HeRMES efficiently transform in-memory structures to on-disk structures for infrequently used files? What is the correct tradeoff between using MRAM for metadata, write buffering, and other uses such as caching the first few blocks of a file to reduce access latency?

We are also exploring issues related to using MRAM across a distributed file system. Clearly, some form of sharing, perhaps similar to cooperative caching [5], will be necessary to fully utilize MRAM in such a system. However, there will be differences as well—the access latency across a network, while lower than that of disk, is considerably higher than that of MRAM.

We are just at the beginning of research into using the new technology of MRAM in file systems, and there are many avenues of research that we will pursue.

7. Conclusions

Magnetic RAM will be available commercially within a few years; it is crucially important that file system designers incorporate it into file systems and use it effectively. We have shown how magnetic RAM can be used to dramatically improve file system performance and reliability. Our file system, HeRMES, will keep metadata in MRAM, allowing nearly free metadata operations limited only by CPU speed. Because MRAM is non-volatile, there is never a need to flush metadata to disk, also improving file system data bandwidth by freeing disk from the need to handle frequent metadata accesses.

File system reliability also benefits from the use of MRAM. The simpler metadata structures possible using MRAM will reduce file system complexity, and thus increase software reliability. Background metadata consistency checking, likewise, will increase the chance than an error will be found, increasing file system reliability by snuffing out errors as soon as they happen. It is this combination of performance and reliability that makes MRAM attractive as a technology for incorporation into file systems.

References

[1] A. Amer and D. Long, "Noah: Low-cost file access prediction through pairs," *20th IEEE International Performance, Computing, and Communications Conference (IPCCC 2001)*, pp. 27–33, 2001.

[2] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer, "Non-volatile memory for fast, reliable file systems," *5th Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 10–22, 1992.

[3] T. Blackwell, J. Harris, and M. Seltzer, "Heuristic cleaning algorithms in log-structured file systems," *Proceedings of the 1995 USENIX Technical Conference*, pages 277–288, January 1995.

[4] H. Boeve, C. Bruynseraede, J. Das, K. Dessein, G. Borghs, J. De Boeck, R. Sousa, L. Melo, and P. Freitas, "Technology assessment for the implementation of magnetoresistive elements with semiconductor components in magnetic random access memory (MRAM) architectures," *IEEE Trans. on Magnetics* **35**(5), pp. 2820–2825, 1999.

[5] M. Dahlin, R. Wang, T. Anderson, and D. Patterson, "Cooperative caching: using remote client memory to improve file system performance," *Proceedings of the 1st Symposium on Operating Systems Design and Implementation (OSDI)*, pages 267–280, 1994.

[6] F. Douglis, R. Caceres, F. Kaashoek, K. Li, B. Marsh, and J. A. Tauber, "Storage alternatives for mobile computers," *Proceedings of the 1st Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 25–37, 1994.

[7] R. English and A. Stepanov, "Loge: a self-organizing disk controller," *Winter 1992 USENIX Technical Conference*, pp. 237–252, 1992.

[8] G. Ganger and M. Kaashoek, "Embedded inodes and explicit groupings: exploiting disk bandwidth for small files," *1997 USENIX Technical Conference*, pp. 1–17, 1997.

[9] J. Griffin, S. Schlosser, G. Ganger, and D. Nagle, "Operating system management of MEMS-based storage devices," *4th Symp. on Operating Systems Design and Implementation (OSDI)*, pp. 227–242, 2000.

[10] R. Hagmann, "Reimplementing the Cedar File System using logging and group commit," *11th ACM Symposium on Operating Systems Principles,* pp. 155–162, 1987.

[11] D. Hitz, J. Lau, and M. Malcom, "File system design for an NFS file server appliance," *Winter 1994 USENIX Conference*, 1994.

[12] D. Lowell and P. Chen, "Free transactions with Rio Vista," *16th ACM Symposium on Operating Systems Principles*, pp. 92–101, 1997.

[13] M. McKusick and G. Ganger, "Soft updates: a technique for eliminating most synchronous writes in the Fast Filesystem," *FREENIX Track: 1999 USENIX Technical Conference*, pp. 1–18, 1999.

[14] D. Roselli, J. Lorch, and T. Anderson, "A comparison of file system workloads," *2000 USENIX Technical Conference*, pp. 41–54, 2000.

[15] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Transactions on Computer Systems* **10**(1), pages 26–52, 1992.

[16] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck, "Scalability in the XFS file system," *Proceedings of the 1996 USENIX Conference*, pages 1–14, 1996.

[17] M. Wu and W. Zwaenepoel, "eNVy: a non-volatile, main memory storage system," *6th Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 86–97, 1994.