PRIMS: Making NVRAM Suitable for Extremely Reliable Storage †

Kevin M. Greenan
kmgreen@cs.ucsc.edu

Ethan L. Miller
elm@cs.ucsc.edu

Storage Systems Research Center
University of California, Santa Cruz

Abstract

Non-volatile byte addressable memories are becoming more common, and are increasingly used for critical data that must not be lost. However, existing NVRAM-based file systems do not include features that guard against file system corruption or NVRAM corruption. Furthermore, most file systems check consistency only after the system has already crashed. We are designing PRIMS to address these problems by providing file storage that can survive multiple errors in NVRAM, whether caused by errant operating system writes or by media corruption. PRIMS uses an erasure-encoded log structure to store persistent metadata, making it possible to periodically verify the correctness of file system operations while achieving throughput rates of an order of magnitude higher than page-protection during small writes. It also checks integrity on every operation and performs on-line scans of the entire NVRAM to ensure that the file system is consistent. If errors are found, PRIMS can correct them using file system logs and extensive error correction information. While PRIMS is designed for reliability, we expect it to have excellent performance, thanks to the ability to do word-aligned reads and writes in NVRAM.

1 Introduction

Byte-addressable, non-volatile memory (NVRAM) technologies such as magnetoresistive random access memory (MRAM) and phase-change memory (PRAM) have recently emerged as viable competitors to Flash RAM [1, 2]. These relatively low capacity technologies are perfect for permanent metadata storage, and can greatly improve file system performance, reliability and power consumption. Unfortunately, due to the increased chance of data corruption, storing permanent structures in NVRAM is generally regarded as unsafe, particularly when compared to disk. The simplicity of most memory access interfaces makes erroneous writes more likely, resulting in data corruption—it is far easier to manipulate structures in memory than on disk. Such behavior is common in OS kernels, in which buggy code can issue erroneous *wild writes* that accidentally overwrite memory used by another module or application.

The goal of PRIMS (Persistent, Reliable In-Memory Storage) is to provide reliable storage in NVRAM without hindering the access speed of byte-addressable memory. Given the limitations of current in-memory reliability mechanisms, we believe that a log-based scheme using software erasure codes is the most effective way to ensure the consistency of persistent, memory-resident data. We present a log-based approach that has the ability to detect and correct errors at multiple byte-granularity without using page-based access control or specialized hardware support. PRIMS consists of a single, erasure-encoded log structure that is used to detect and correct hardware errors, software errors and file system inconsistencies.

2 Motivation

Modern operating systems protect critical regions of memory using access control bits in the paging structures. While page-level access control is an effective tool for preventing wild writes in write caches, it is not the best solution for protecting small, persistent structures in byte-addressable memory because every protected write requires a TLB flush and two structure modifications to mark a page as read-write and read-only. During periods of frequent small writes, these permission changes have a dramatic effect on performance.

Disk interfaces also decrease the likelihood of wild writes by requiring access through device drivers containing complex I/O routines. The probability of rogue code accidentally corrupting disk blocks while evading the controlled device driver interface is extremely low. This strict I/O interface greatly improves data reliability with respect to software errors, but hinders performance on low-latency media, such as NVRAM.

In addition to software errors, hardware errors such as random bit flips and cell wear may occur on the media, leading to data corruption. Hardware-based error correction schemes require a specialized controller and resolve errors beyond the correction capability by rebooting the system. Obviously, rebooting is not an option when protecting persistent data in memory; thus, a more robust scheme is necessary. Hardware-based error correction is also computed independent of any software implementation; as a result, wild

†This research was funded in part by NSF-0306650, the Dept. of Energy-funded Petascale Data Storage Institute, and by SSRC industrial partners.

writes will most usually result in values that have consistent hardware-based ECC values. In order to provide a high level of flexibility and resiliency, error correction should be moved into a software module that provides tunable redundancy with respect to the target environment.

As the size of storage systems increases, the probability of corruption and time required to make repairs increase correspondingly. In an effort to prevent permanent damage as a result of corruption and minimize system downtime, PRIMS performs on-line checks that verify the consistency of file system structures and all NVRAM-resident data. We believe such integrity checking should be a requirement for modern file systems.

3 Related Work

A great deal of work has gone into many of the performance implications associated with the use of NVRAM in file systems. In addition, methods to achieve reliability in file systems and data caches have been used for a great deal of time. As we will see, no single method has the ability to effectively handle both software and hardware errors in NVRAM.

Baker, *et al.* [3] observed that the use of NVRAM in a distributed file system can improve write performance and file system reliability. More recently, HeRMES [10] posited that file system performance would improve dramatically if metadata were stored in MRAM. Conquest [16] also used persistent RAM to store small files, metadata, executables and shared libraries. While these systems promise higher performance, none of the systems provided improved reliability. As a result, they are potentially unsafe to use for long-term metadata storage because they are subject to corruption that cannot be fixed by rebooting—the in-memory metadata is the *only* copy.

The Rio file cache [4, 7] utilizes page permission bits to set the access rights for areas of memory, providing a safe non-volatile write cache. In general, page protection does not have a profound effect on write throughput; page protection does heavily degrade write performance in workloads consisting of small I/Os. Since most metadata updates are relatively small, we believe that page protection should not be used to protect persistent metadata.

Earlier work on metadata reliability in NVRAM used a combination of error correcting codes and page-protection via the `mprotect` system call to ensure the integrity of persistent, memory-resident metadata [5]. The cost of page protection significantly hurt performance, while on-line consistency checks of memory-resident metadata proved to be inexpensive.

Mondriaan memory protection [17] provides fine-grained isolation and protection between multiple domains within a single address space. SafeMem [12] deterministically scrambles data in ECC memory to provide protected regions of memory, providing `mprotect`-like protection. After scrambling the data, ECC is re-enabled and the ECC controller faults when the region is accessed. While these approaches provide an interesting alternative to `mprotect`, both require specialized hardware support and prevent data corruption without correcting the corruption when it occurs. Our approach assumes that error *recovery* is just as important is error *detection* in ensuring the reliability of persistent data in NVRAM.

The popular `fsck` program [9] and its background variant [8] attempt to restore file system consistency by scanning all of the file system metadata. Since the running time of `fsck` is a function of the file system size, this operation often takes a great deal of time to complete and does not scale to very large file systems. Chunkfs [6] divides on-disk data into individually repairable chunks promising faster `fsck` and partial on-line consistency checks. These techniques are more similar to ours, and Henson, *et al.* explore tradeoffs between performance and reliability in an effort to speed up the file system repair process.

DualFS [11] physically separates data and metadata onto different devices or different partitions on the same device. The metadata device is treated as a log-based file system and the data device resembles an FFS file system. The separation of metadata and data results in fast crash recovery, simplified log cleaning and improved file system performance. File systems such as XFS [15] and LFS [13] also use log-based recovery to restore file system consistency. Unfortunately, these systems only apply recovery mechanisms after a crash, which may lead to system downtime or data loss. In addition, they do not proactively check file system integrity while the file system is running, potentially leading to latent errors that may go undetected for days or weeks.

4 Design

The performance implications and reliability constraints associated with current memory protection schemes force us to take a new approach when providing reliability for persistent data in memory. The overall structure of PRIMS is based on five key design requirements:

Place metadata in NVRAM and data on disk Moving metadata from disk to NVRAM allows PRIMS to exploit parallelism between the independent devices, enabling better performance, simple structures and the ability to do fast, on-line metadata checks.

Periodically scan file system structures in NVRAM On-line consistency checks are expected to decrease or prevent down-time in the face of media or software failures.

Avoid the need for hardware support An MMU is not necessary and ECC schemes and reliability levels can be changed.

Allow recovery from software and hardware faults Since we plan to store persistent data on easily accessible, low-reliability media, consistency must be maintained by protecting against both software and hardware faults.

Provide reliability with minimal overhead Persistent data is stored in NVRAM in an effort to improve performance, thus any reliability mechanisms should have a small impact on performance.

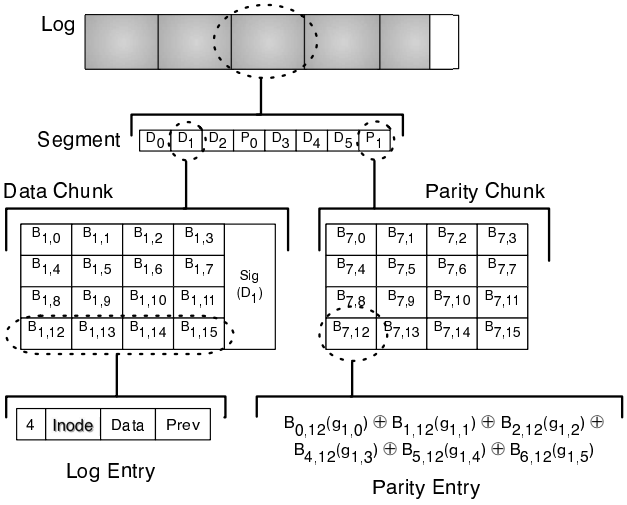

Figure 1: An example of the erasure-encoded log. Each parity block is computed from corresponding data blocks in a segment. Signatures are used in each data chunk to detect integrity violations. Corruption is detected at the chunk level and corrected by decoding the corrupted chunk's segment.

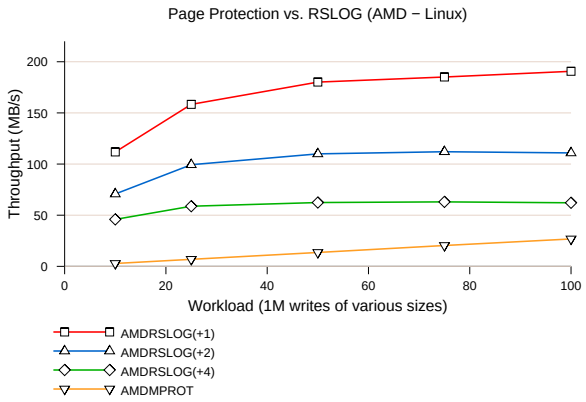

Figure 2: Throughput when writing data using our log-based encoding vs. writes surrounded by mprotect calls. These results were generated from the average throughput over 20 trials of each workload on an 2.4 GHz AMD Opteron machine running Linux. As shown in the graphs, depending on the choice of encoding, small write throughput in the encoded log is much greater than page protection for the same workload.

The most widely-used memory protection technique—page table-based protection—provides isolation and protection through paging and access control using the page protection bits in each page table entry (PTE). Because page-level access control requires modifying PTEs and flushing the TLB each time page permissions are changed, a small but constant penalty is incurred for each write. This penalty is relatively small for large writes, but greatly degrades performance for workloads consisting of small writes.

Access control provides very little in terms of hardware and software error recovery. These mechanisms are designed to prevent different parts of the operating system from performing unauthorized data access. Thus, if a hardware error occurs or a piece of software evades the access control mechanisms—performing an unauthorized write—the original data cannot be recovered. When dealing with persistent data, even a single instance of corruption could lead to loss of file system data.

To address the issues listed above we developed PRIMS. In PRIMS, all file system metadata is written to a log. Maintaining log consistency is extremely crucial because the log holds the *only* copy of the file system metadata—metadata is never written to disk. Thus, NVRAM-resident metadata may be compromised due to wild writes, random bit flips, media wear or, in the case of multiple NVRAM banks, media failure. PRIMS uses the log structure to verify the consistency of file system metadata by replaying log transactions against the live state of the system, and by checking that written data matches the signatures stored for the data. As an additional measure, module identification information may be added to each write, giving PRIMS the ability to determine which part of the OS issued the write.

PRIMS contains a set of logically contiguous, fixed-sized segments, as shown in Figure 1, with each segment containing the same number of fixed-sized chunks. Data is appended to the log by writing to the current data chunk and writing parity updates to the appropriate parity chunks in the current segment. Each chunk is composed of indivisible, fixed-sized blocks; these blocks may be any size, and may be significantly smaller than standard disk blocks. The log-structure gives PRIMS a way to distinguish between *live* and *dead* regions, where a live region contains writable segments and a dead region contains filled segments. If a dead region is changed, PRIMS will detect an unauthorized change and reverse it using the erasure encoding.

PRIMS uses linear erasure codes, such as Reed-Solomon codes, to allow flexible numbers of data chunks and associated redundancy chunks. Linear erasure codes have a reputation for being slow because of the expensive Galois field multiples required for encoding; however, we obtain reasonable encoding speeds by using multiplication tables and incremental parity updates. PRIMS uses algebraic signatures rather than cryptographically secure hash functions such as SHA-1 to perform periodic integrity checks [14]. To check the integrity of a segment, PRIMS must do two things: verify that each chunk is consistent with its algebraic signature, and verify that the algebraic signatures for a segment are consistent with one another. The use of algebraic signatures and linear erasure codes enables this approach, since the computation of algebraic signatures and the use of erasure codes based on the same Galois field commute. These operations can be done together, or they can be decoupled, allowing sets of signatures over a segment to be verified in the background and the correspondence of data and a signature to be verified when the data is used. If an error is found during verification PRIMS treats the chunk as an erasure and recovers the correct data.

Figure 2 shows the performance of our log structure compared to an approach that uses page protection. These

preliminary results are taken from a simple erasure encoded log implementation on two architectures (Intel and AMD) and two operating systems (Linux 2.6.17 and Mac OS 10.4.8) using a Reed-Solomon and algebraic signature library we are developing. Because Intel performance was similar to that on AMD, we only present the AMD results in Figure 2. The experiment performs one million writes, each between 10 and 100 bytes, to the log and to a protected region. Writes to the protected region require two calls to `mprotect` to unprotect and protect a page. Log writes require a single data write and one or more parity updates. In our test, we chose to use 1, 2 and 4 parity chunks per segment. As Figure 2 shows, the use of `mprotect` is not appropriate for small-write workloads that require frequent permission changes. For writes of size 10–50 bytes, our log-based approach outperforms page protection by at least a factor of six, and in most cases more than an order of magnitude. Though not shown in Figure 2, page protection outperforms the encoded-log at write sizes of at least 1 KB, confirming the utility of page protection in write caches for page-based data. We believe that most of the `mprotect` overhead was due to modifying the page tables and flushing regions of the TLB during each protection call.

5 PRIMS Advantages

The goal of PRIMS is to provide a reliable system that permanently places small pieces of data, particularly metadata, in NVRAM. However, our initial design provides a number of additional benefits beyond higher performance.

Storage system capacity is increasing at an exponential rate. As capacity grows, so does the probability of error and time required for recovery. Instead of focusing solely on performance, we are exploring the tradeoffs between performance and reliability. In doing so we are creating a system that will exhibit high performance and maintain firm consistency guarantees. In addition, distributed file systems will benefit from the potential power savings and performance boost associated with storing metadata and indices exclusively in NVRAM. One clear advantage is the ability to search the file system even when a great deal of the disks are spun-down.

PRIMS is being designed to run on any platform or hardware architecture. All error correction is computed in software, so there is no need for hardware-based ECC. Moreover, PRIMS does not even require an MMU because it does not rely upon page tables for protection, reducing hardware costs and resulting in tunable fault tolerance and encoding across independent banks of NVRAM. The software-based erasure-encoded log is expected to tolerate wild writes, file system bugs, media errors and media wear regardless of the underlying platform, making it well-suited for portable devices and single-host file systems as well as distributed systems.

6 Status

We are currently implementing PRIMS using off-the-shelf DRAM as a stand-in for either phase-change RAM or MRAM. We are designing metadata structures that take advantage of the high performance and reliability of PRIMS to provide a significantly faster, more reliable file system than is possible using disk-based metadata. While our preliminary tests do not give any indication of how effective an erasure-encoded log will be at protecting metadata, we expect that our mechanisms provide better protection than page-based access control by catching a wider range of hardware and software errors. We plan to have an in-kernel prototype of PRIMS available in the coming months, and will use this prototype to demonstrate both the improved performance and improved reliability that storing file system metadata in reliable NVRAM can provide.

References

[1] Samsung Introduces the Next Generation of Nonvolatile Memory-PRAM, Sep 2006.

[2] Toshiba and NEC Develop World's Fastest, Highest Density MRAM, Feb 2006.

[3] M. Baker, S. Asami, E. Deprit, J. Ousterhout, and M. Seltzer. Non-volatile memory for fast, reliable file systems. In *Proceedings of ASPLOS-V*, pages 10–22, 1992.

[4] P. M. Chen, W. T. Ng, S. Chandra, C. Aycock, G. Rajamani, and D. Lowell. The Rio file cache: Surviving operating system crashes. In *Proceedings of ASPLOS-VII*, Oct. 1996.

[5] K. M. Greenan and E. L. Miller. Reliability mechanisms for file systems using non-volatile memory as a metadata store. In *Proceedings of ACM/IEEE EMSOFT '06*, Oct. 2006.

[6] V. Henson, A. van de Ven, A. Gud, and Z. Brown. Chunkfs: Using divide-and-conquer to improve file system reliability and repair. In *Proceedings of HotDep '06*, 2006.

[7] D. E. Lowell and P. M. Chen. Free transactions with Rio Vista. In *Proceedings of SOSP '97*, pages 92–101, 1997.

[8] M. K. McKusick. Running fsck in the Background. In *Proceedings of the BSDCon*, pages 55–64, 2002.

[9] M. K. McKusick and T. Kowalski. *4.4 BSD System Manager's Manual*, chapter 3, pages 3:1–3:21. O'Reilley and Associates, Inc., Sebastopol, CA, 1994.

[10] E. L. Miller, S. A. Brandt, and D. D. E. Long. HeRMES: High-performance reliable MRAM-enabled storage. In *Proceedings of HotOS-VIII*, pages 83–87, May 2001.

[11] J. Piernas, T. Cortes, and J. M. Garcìa. DualFS: a new journaling file system without meta-data duplication. In *Proceedings of ICS '02*, pages 137–146, 2002.

[12] F. Qin, S. Lu, and Y. Zhou. SafeMem: Exploiting ECC-memory for detecting memory leaks and memory corruption during production runs. In *Proceedings of HPCA-XI*, 2005.

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

[14] T. Schwarz, S. J. and E. L. Miller. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *Proceedings of ICDCS '06*, July 2006.

[15] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck. Scalability in the XFS file system. In *Proceedings of USENIX Annual Tech. '96*, Jan. 1996.

[16] A.-I. A. Wang, G. H. Kuenning, P. Reiher, and G. J. Popek. Conquest: Better performance through a disk/persistent-RAM hybrid file system. In *Proceedings of USENIX Annual Tech. '02*, Monterey, CA, June 2002.

[17] E. Witchel, J. Cates, and K. Asanović. Mondrian memory protection. In *Proceedings of ASPLOS-X*, Oct 2002.