Heterogeneous-Reliability Memory: 
Exploiting Application-Level Memory Error Tolerance

Yixin Luo\textsuperscript{1} Sriram Govindan\textsuperscript{2} Bikash Sharma\textsuperscript{3,2} Mark Santanil ello\textsuperscript{3,2} Justin Meza\textsuperscript{3,1} 
Aman Kansal\textsuperscript{2} Jie Liu\textsuperscript{2} Badriddine Khessib\textsuperscript{2} Kushagra Vaid\textsuperscript{2} Onur Mutlu\textsuperscript{4,1} 
\textsuperscript{1}Carnegie Mellon University \textsuperscript{2}Microsoft Corporation \textsuperscript{3}Facebook \textsuperscript{4}ETH Zürich

This paper summarizes our work on characterizing application memory error vulnerability to optimize datacenter cost via Heterogeneous-Reliability Memory (HRM), which was published in DSN 2014 \cite{104}, and examines the work’s significance and future potential. Memory devices represent a key component of datacenter total cost of ownership (TCO), and techniques used to reduce errors that occur on these devices increase this cost. Existing approaches to providing reliability for memory devices pessimistically treat all data as equally vulnerable to memory errors. Our key insight is that there exists a diverse spectrum of tolerance to memory errors in new data-intensive applications, and that traditional one-size-fits-all memory reliability techniques are inefficient in terms of cost. For example, we found that while traditional error protection increases memory system cost by 12.5%, some applications can achieve 99.00% availability on a single server with a large number of memory errors without any error protection. This presents an opportunity to greatly reduce server hardware cost by provisioning the right amount of memory reliability for different applications.

Toward this end, in our DSN 2014 paper \cite{104}, we make three main contributions to enable highly-reliable servers at low datacenter cost. First, we develop a new methodology to quantify the tolerance of applications to memory errors. Second, using our methodology, we perform a case study of three new data-intensive workloads (an interactive web search application, an in-memory key-value store, and a graph mining framework) to identify new insights into the nature of application memory error vulnerability. Third, based on our insights, we propose several new hardware/software heterogeneous-reliability memory system designs to lower datacenter cost while achieving high reliability and discuss their trade-offs. We show that our new techniques can reduce server hardware cost by 4.7% while achieving 99.90% single server availability.

We believe the notion of HRM opens up a sea of opportunities in optimizing memory system and overall system cost, reliability, efficiency, and performance in a manner that is aware of applications’ tolerance to memory errors. Thus, our paper just scratches the surface of a large HRM exploration space, which we hope future works will undertake in various novel ways, in a wide variety of systems, ranging from datacenters to mobile and embedded systems.

1. Introduction

A warehouse-scale datacenter consists of many thousands of machines running a diverse set of applications, and comprises the foundation of the modern web \cite{4,151}. While such datacenters are vital to the operation of companies such as Facebook, Google, Microsoft, and Yahoo\textsuperscript{1}, reducing the cost of such large-scale deployments of machines poses a significant challenge to these and other companies. Recently, the need for reduced datacenter cost has driven companies to examine more energy-efficient server designs \cite{38} and build their datacenter installations in cold environments to reduce cooling costs \cite{49,59} or use built-in power plants to reduce electricity supply costs \cite{139}.

There are two main components of the total cost of ownership (TCO) of a datacenter \cite{4}: (1) capital costs (those associated with server hardware) and (2) operational costs (those associated with providing electricity and cooling). Recent studies have shown that capital costs can account for the majority (e.g., around 57% in \cite{4}) of datacenter TCO, and thus represent the main impediment for reducing datacenter TCO. In addition, this component of datacenter TCO is only expected to increase going forward as companies adopt more efficient cooling and power supply techniques.

Of the dominant component of datacenter TCO (capital costs associated with server hardware), the cost of server processors and memory represents the key component—around 60% in modern servers \cite{77}. Furthermore, the cost of the memory in today’s servers is comparable to that of the processors \cite{77}, and is likely to exceed processor cost for data-intensive applications such as web search and social media services, which use in-memory caching to improve response time \cite{54,127,128,129,159} (e.g., a popular key-value store, Memcached, has been used at Google and Facebook \cite{54,127} for this purpose).

Exacerbating the cost of memory in modern servers is the use of memory devices (such as dynamic random access memory, or DRAM) that provide error detection and correction. This cost arises from two components: (1) quality assurance testing performed by memory vendors to ensure devices sold to customers are of a high enough caliber and (2) additional memory capacity for error detection and correction. Device testing has been shown to account for an increasing fraction of the cost of memory for DRAM \cite{2,33}. The cost of additional memory capacity, on the other hand, depends on the technique used to provide error detection and correction.

Table 1 compares several common memory error detection and correction techniques in terms of which types of errors...
that they are able to detect/correct and the additional amount of capacity/logic they require (which, for DRAM devices, whose design is fiercely cost-driven [120,121], is proportional to cost). Techniques range from the relatively low-cost (and widely employed) parity, SEC-DED (single error correction, double error detection), Chipkill [35], and DEC-TED (double error correction, triple error detection), all of which use different error-correcting codes (ECC) to detect and correct a small number of bits or chip errors, to the more expensive RAIM [111] and Mirroring [53] techniques that replicate (or all) of memory to tolerate the failure of an entire DRAM dual in-line memory module (DIMM). The additional cost of memory with high error-tolerance can be significant (e.g., 12.5% of the total memory capacity for SEC-DED and Chipkill, and as high as 125% for Mirroring).

Table 1: Memory error detection and correction techniques. “X/Y Z” means a technique can detect/correct X out of every Y failures of Z. n represents the parity of any odd number of bits between 1 and 63. Adapted from [104].

| Technique   | Error Detection (Correction) | Added Capacity | Added Logic |
|-------------|------------------------------|----------------|-------------|
| Parity      | n/64 bits (None)             | 1.6%           | Low         |
| SEC-DED     | 2/64 bits (2/64 bits)        | 12.5%          | Low         |
| DEC-TED     | 3/64 bits (3/64 bits)        | 23.4%          | Low         |
| Chipkill    | 2/8 chips (3/8 chips)        | 12.5%          | High        |
| RAIM        | 1/5 modules (1/5 modules)    | 40.6%          | High        |
| Mirroring   | 2/8 chips (2/8 modules)      | 125.0%         | Low         |

Yet even with well-tested and error-tolerant memory devices, recent studies from the field have observed a rising rate of memory error occurrences [55,72,116,120,123,141,146]. This trend presents an increasing challenge for ensuring high performance and high reliability in future systems, as memory errors can be detrimental to both. In terms of performance, existing error detection and correction techniques incur a slowdown on each memory access due to their additional circuitry [55,92] and up to an additional 10% slowdown due to techniques that operate DRAM at a slower speed to reduce the chances of random bit flips due to electrical interference in higher-density devices that pack more and more cells per square nanometer [148]. In addition, whenever an error is detected or corrected on modern hardware, the processor raises an interrupt that must be serviced by the system firmware (e.g., BIOS), incurring up to 100 μs latency—roughly 2000× the latency of a typical 50 ns memory access latency [58]—leading to unpredictable slowdowns and sometimes even system hangs [116].

In terms of reliability, memory errors can cause an application to slow down, hang, crash, or produce incorrect results [40]. Software-level techniques such as the retirement of regions of memory with errors [55,76,116,118,150] have been proposed to reduce the occurrence of memory error correction events and prevent correctable errors from turning into uncorrectable errors over time. Hardware-level techniques, such as those listed in Table 1, are used to detect and correct errors without software intervention (but with additional hardware cost). All of these techniques are applied homogeneously to memory systems in a one-size-fits-all manner.

Our goal in our DSN 2014 paper [104] is to (1) understand how tolerant different data-intensive applications and different memory regions of each application are to memory errors, and (2) design a new memory system organization that matches hardware reliability to the error tolerance of the application and the memory region in order to reduce system cost. The main idea of our approach is to classify applications and memory regions based on their memory error tolerance, and map applications and memory regions to heterogeneous-reliability memory (HRM) system designs managed cooperatively between hardware and software to reduce system cost. We make the following contributions:

1. A new methodology to quantify the tolerance of applications and their memory regions to memory errors. Our approach measures the effect of memory errors on application correctness and quantifies an application’s ability to mask or recover from memory errors.
2. A comprehensive characterization of the memory error tolerance of three data-intensive workloads: an interactive web search application [104,138], an in-memory key–value store [34,104], and a graph mining framework [103,104]. We find that there exists an order of magnitude difference in memory error tolerance across these three applications. We also find that there exists an order of magnitude difference in memory error tolerance across different memory regions of each application.
3. An exploration of the design space of a family of new memory system organizations, called heterogeneous-reliability memory, which combines a heterogeneous mix of reliability techniques that leverage application and memory region error tolerance to reduce system cost. We show that an example use of our techniques reduces server hardware cost by 4.7%, while achieving 99.90% single server availability, based on a preliminary evaluation of an example HRM system.

2. Characterizing Memory Error Tolerance

We characterize three commonly-used data-intensive applications to quantify their tolerance to memory errors:

- **WebSearch** [138], an interactive web search application,
- **Memcached** [34], an in-memory key-value store, and
- **GraphLab** [103], a graph mining framework.

We run these three applications in real production systems, and sample hundreds to tens of thousands of unique memory addresses for each application.

2.1. Characterization Methodology

To understand how tolerant different data-intensive applications are to memory errors, our characterization consists of three components: (1) characterizing the outcomes of memory errors on an application based on how they propagate
through an application’s code and data, (2) characterizing how safe or unsafe it is for memory errors to occur in different regions of an application’s data, and (3) determining how amenable an application’s data is to recovery in the event of an error. We describe the implementation of each component in detail in Sections III and IV of our DSN 2014 paper [104].

We characterize an application’s vulnerability to a memory error based on its behavior after a memory error is introduced (we assume for the moment that no error detection or correction is being performed). Figure 1 shows a taxonomy of memory error outcomes. Our taxonomy is mutually exclusive (no two outcomes occur simultaneously) and exhaustive (it captures all possible outcomes). At a high level, a memory error may be either (1) masked by an overwrite, in which case it is never detected and causes no change in application behavior; or (2) consumed by the application. In the case that an error is consumed by the application, it may either (2.1) be masked by application logic, in which case it is never detected and causes no change in application behavior; (2.2) cause the application to generate an incorrect response; or (2.3) cause the application or system to crash.

![Figure 1: Memory error outcomes. Reproduced from [104].](image)

When we refer to the tolerance of an application to memory errors, we mean the likelihood of an error occurring in some data results in outcomes (1) or (2.1). Conversely, when we refer to the vulnerability of an application to memory errors, we mean the likelihood of an error occurring in some data results in outcomes (2.2) or (2.3).

We have three design goals when implementing our methodology for quantifying application memory error tolerance. First, due to the sporadic and inconsistent nature of memory errors in the field [65, 72, 100, 116, 135, 141, 145, 146, 147], we want to design a framework that emulates the occurrence of a memory error in an application’s data in a controlled manner. Second, we want an efficient way to measure how an application accesses its data. Third, we want our framework to be easily adaptable to other workloads or system configurations.

Figure 2 shows a flow diagram illustrating the five steps involved in our error emulation framework. We assume that the application under examination has already been run outside of the framework and its expected output without any memory errors has been recorded. The framework proceeds as follows. (1) We start the application under the error injection framework. Our memory error emulation framework is described in Section IV of our DSN 2014 paper [104]. (2) We use software debuggers\(^1\) to inject the desired number and types of memory errors. (3) We initiate the connection of a client and start executing the desired workload. (4) Throughout the course of the application’s execution, we check to see if the machine has crashed; if it has, we log this outcome and proceed to step (1) to begin testing once again. (5) If the application finishes its workload, we check to see if its output matches the expected results; if the output does not match the expected results, we log this outcome and proceed to step (1) to test again. Each run injects a particular pattern of errors into the application. We can run this framework as many times as needed to test an application with different patterns of injected errors.

![Figure 2: Memory error emulation framework. Reproduced from [104].](image)

There are two main types of memory errors: (1) soft or transient errors and (2) hard or recurring errors.\(^2\) Soft memory errors occur at random due to charged particle emissions from chip packaging or the atmosphere [110]. Hard memory errors may occur from physical device defects or wearout [55, 141, 146], and are influenced by environmental factors such as humidity, temperature, and utilization [141, 144, 147]. Hard errors typically affect multiple bits (for example, large memory regions and entire DRAM chips have been shown to fail [55, 146, 147]). Our characterization covers single-bit soft and hard errors. For a detailed background on DRAM, we refer the reader to prior works [24, 25, 26, 27, 51, 52, 66, 71, 72, 73, 74, 75, 83, 84, 85, 86, 87, 99, 100, 131, 142, 143].

\(^1\)WinDBG [119] in Windows and GDB [42] in Linux.

\(^2\)Recent studies [62, 64, 65, 72, 100, 135] examined the effects of intermittent and access-pattern dependent errors, which are increasingly common as DRAM technology scales down to smaller technology nodes [120].
2.2. Key Findings

We summarize two of the most important findings from our characterization below. We briefly list four other findings in Section 2.3, and describe all six of our findings in detail in Section V-B of our DSN 2014 paper [104].

Finding 1: Error Tolerance Varies Across Applications. Figure 3(a) plots the probability of each of the evaluated three applications crashing due to the occurrence of single-bit soft or hard errors in their memory (we call this application-level memory error vulnerability). For cases where the application does not crash, Figure 3(b) plots the rate of incorrect results per billion application queries under the same conditions. We draw two key observations from these results.

First, there exists a significant variance in vulnerability among the three applications both in terms of crash probability and in terms of incorrect result rate, which varies by up to six orders of magnitude. Second, these characteristics may differ depending on whether errors are soft or hard (for example, the number of incorrect results for WebSearch differs by over two orders of magnitude between soft and hard errors, with hard errors being more problematic). We therefore conclude that memory reliability techniques that treat all applications similarly are inefficient because there exists significant variance in error tolerance among applications.

Finding 2: Error Tolerance Varies Within an Application. Figure 4(a) plots the probability of each of the three applications crashing due to the occurrence of single-bit soft or hard errors in different regions of their memory address space. Figure 4(b) plots the rate of incorrect results per billion queries under the same conditions, for cases where a crash did not occur.

We make two observations from Figure 4. First, for some memory regions, the probability of an error leading to a crash is much lower than for others (for example, in WebSearch, the probability of a hard error leading to a crash in the heap or private memory regions is much lower than in the stack memory region). Second, even in the presence of memory errors, some regions of some applications are still able to tolerate memory errors (perhaps at reduced correctness). This may be acceptable for applications such as WebSearch that aggregate results from several servers before presenting them to the user, in which case the likelihood of the user being exposed to an error is much lower than the reported probabilities. We therefore conclude that memory reliability techniques that treat all memory regions within an application similarly are inefficient because there exists significant variance in the error tolerance among different memory regions.

2.3. Other Findings

In Section V-B of our DSN 2014 paper [104], we discuss four other findings that we make based on our characterization data. These findings focus on the memory error tolerance of WebSearch, which we find to be representative of the behavior of all three of our characterized applications. In particular, we find that:

- More severe failures (i.e., failures that lead to system downtime) due to memory errors tend to crash the application or system quickly, while less severe failures tend to generate incorrect results periodically.
- Some memory regions are safer than others. This indicates that either an application’s access pattern or computational operations on different memory regions can be the dominant factor to mask a majority of memory errors.
- More severe errors mainly decrease correctness, as opposed to increase an application’s probability of crashing.
- Data recoverability varies across memory regions. For data-intensive applications like WebSearch, software-only memory error tolerance techniques are a promising direction for enabling reliable system designs.

3. Heterogeneous-Reliability Memory

Based on the findings from our experimental characterization, we propose heterogeneous-reliability memory (HRM), a software/hardware cooperative framework that employs different levels of memory reliability within a single main memory subsystem to optimize datacenter cost based on the memory error tolerance level of applications and their memory regions. We examine three dimensions, and their benefits and trade-offs in the design space, for systems with heterogeneous reliability memory: (1) hardware techniques to detect and correct errors, (2) software responses to errors, and (3) the granularity at which different techniques are used.
Table 2 lists the techniques we considered in each of the dimensions along with their potential benefits and trade-offs.

Using WebSearch as an example application, we evaluate and compare five example design points (three non-HRM systems, and two HRM systems):

- **Typical Server (non-HRM):** A baseline configuration resembling a typical server deployed in a modern datacenter. All memory is homogeneously protected using SEC-DED ECC.

- **Consumer PC (non-HRM):** Consumer PCs typically have no hardware protection against memory errors, reducing both their cost and reliability.

- **Detect&Recover (HRM):** Based on our observation that some memory regions are safer than others, we consider an HRM system design that, for the private region, uses parity in hardware to detect errors and responds by correcting them with a clean copy of data from disk in software (Par+R, parity and recovery), and uses neither error detection nor correction for the rest of its data.

- **Less-Tested (L; non-HRM):** Testing increases both the cost and average reliability of memory devices [120, 121, 131]. This system examines the implications of using less-thoroughly-tested memory throughout the entire memory system.

- **Detect&Recover/L (HRM):** This system evaluates the Detect&Recover design with less-tested memory. ECC is used in the private region and Par+R in the heap to compensate for the reduced reliability of the less-tested memory.

Section VI-A of our DSN 2014 paper [104] discusses (1) the metrics we use to evaluate the benefits and costs of the designs, and (2) the memory error model we use to examine the effectiveness of the five designs. We refer the reader to Section VI-A in [104] for detail and a full understanding.

Our evaluation illustrates the inefficiencies of traditional homogeneous approaches to memory system reliability, as well as the benefits of heterogeneous-reliability memory system designs. Figure 5 shows the cost savings and single server availability for our five evaluated design points. We observe from the figure that the two highlighted example HRM design points (in orange color), which leverage our heterogeneous-reliability memory system design, both can achieve our target single server availability of 99.90% while reducing server hardware cost by 2.9% and 4.7% respectively.

We therefore conclude that heterogeneous-reliability memory system designs can enable systems to achieve both high cost savings and high single server availability/reliability at the same time.

![Figure 5: Comparison of server hardware cost savings and single server availability for the five design points. Results extracted from [104]. Orange bars indicate HRM designs.](image)

Section VI of our DSN 2014 paper [104] contains a detailed analysis of HRM, including (1) memory cost savings (Section VI-B of [104]), (2) the expected crash and incorrect query frequency for each configuration (Section VI-B of [104]), (3) the maximum number of tolerable errors per month for each application to achieve a reliability target (Section VI-B of [104]), and (4) a discussion of hardware/software support for and feasibility of HRM (Section VI-C of [104]). We summarize the key empirical findings here:

- Our two example HRM designs, Detect&Recover and Detect&Recover/L, reduce memory costs by 9.7% and 15.5%, respectively.

### Table 2: Heterogeneous reliability design dimensions, example techniques, and their potential benefits and trade-offs. Adapted from [104].

| Design dimension | Technique | Benefits | Trade-offs |
|------------------|-----------|----------|------------|
| Example hardware techniques | No detection/correction | No associated overheads (low cost) | Unpredictable crashes and silent data corruption |
| | Parity | Relatively low cost with detection capability | No hardware correction capability |
| | SEC-DED/DEC-TED | Tolerate common single-/double-bit errors | Increased cost and memory access latency |
| | Chipkill [35] | Tolerate single-DRAM-chip errors | Increased cost and memory access latency |
| | Mirroring [53] | Tolerate memory module failure | Increased error rates |
| | Less-Tested DRAM | Saved testing cost during manufacturing | |

| Example software responses | Consume errors in application | Simple, no performance overhead | Unpredictable crashes and data corruption |
| | Automatically restart application | Can prevent unpredictable application behavior | May make little progress if error is frequent |
| | Retire memory pages | Low overhead, effective for repeating errors | Reduces memory space (usually very little) |
| | Conditionally consume errors | Flexible, software vulnerability-aware | Memory management overhead to make decision |
| | Software correction | Tolerates detectable memory errors | Usually has performance overheads |

| Usage granularity | Physical machine | Simple, uniform usage across memory space | Costly depending on technique used |
| | Virtual machine | More fine-grained, flexible management | Host OS is still vulnerable to memory errors |
| | Application | Manageable by the OS | Does not leverage different region tolerance |
| | Memory region | Manageable by the OS | Does not leverage different page tolerance |
| | Memory page | Manageable by the OS | Does not leverage different data object tolerance |
| | Cache line | Most fine-grained management | Large management overhead; software changes |
respectively, compared to the cost of the Typical Server system, which does not use HRM.

- The two example HRM designs limit the number of crashes to 3 and 4 per server per month, respectively, and limit the incorrect query frequency to 9 and 12 per million queries, respectively.
- Without any error detection/correction, two out of our three evaluated applications (WebSearch and Memcached) are able to achieve 99.00% single server availability.

We therefore conclude that heterogeneous-reliability memory system designs can enable systems to achieve both high cost savings and high single server availability/reliability at the same time. We believe that there is significant opportunity in many data-intensive applications for reducing server hardware cost while achieving high single server availability/reliability using our heterogeneous-reliability design methodology.

4. Related Work

To our knowledge, our DSN 2014 paper [104] is the first to (1) perform a comprehensive analysis of memory error vulnerability for data-intensive datacenter applications across a range of different memory error types; (2) propose the idea of heterogeneous reliability memory, which consists of multiple memory types with different levels of reliability and error handling mechanisms; and (3) evaluate the cost-effectiveness of different heterogeneous-reliability memory organizations with hardware/software cooperation. We discuss related research in memory error vulnerability and DRAM architecture below, categorizing the works into six broad classes: (1) memory errors in datacenters, (2) characterizing application error tolerance, (3) hardware-based memory reliability techniques, (4) software-based memory reliability techniques, (5) exploiting application error tolerance, and (6) heterogeneous (hybrid) memory architectures.

Studies of Memory Errors. Various works [92, 116, 141, 145, 146, 147] have conducted studies of DRAM error rates that are deployed in production datacenters, studying failures across a large sample size. These works note that memory errors occur frequently in datacenters, and are induced by a number of error sources. In particular, one of these studies empirically demonstrates the increased memory errors and increased memory cost to tolerate these errors in large-scale datacenters [116]. A recent work [48] examines how various hardware and software techniques to detect and mitigate errors introduce significant performance degradation in production datacenters. This work shows that for WebSearch, software error handling techniques can induce a performance overhead of 3746× [48]. These studies motivate the need for a low-overhead, cost-effective approach to memory reliability, and motivate us to further explore hardware–software cooperative techniques such as HRM.

There are several studies that characterize various sources of errors in DRAM at a fine granularity. Many of these works observe how specific factors affect DRAM errors, analyzing the impact of temperature [37, 86] and hard errors [55]. A large number of works study errors through controlled experiments, usually using FPGA-based DRAM testing infrastructures like SoftMC [51], to investigate errors due to retention time [51, 62, 63, 64, 65, 99, 100, 131, 135], disturbance from neighboring DRAM cells [60, 70, 72, 120], latency variation across/within DRAM chips [21, 23, 25, 82, 83, 86], and supply voltage [23, 27]. None of these works study memory errors in a system with heterogeneous-reliability memory.

Classifying Application Error Tolerance. Error injection techniques based on hardware watchpoints [92, 112], binary instrumentation [89], and architectural simulation [93] have been used to investigate the impact of memory errors on application behavior, including execution times, application/system crashes, and output correctness. These works study a range of applications including SPEC CPU benchmarks, web servers, databases, and scientific applications. In general, these works conclude that not all memory errors cause application/system crashes and many memory errors can be tolerated with minimal difference in the application outputs. We generalize this observation to data-intensive applications, and leverage it to reduce datacenter TCO. Recent work [149] develops a Markov-chain model for the error tolerance of HPC applications. Approximate computing techniques [8, 39, 57], where the precision of program output can be relaxed to achieve better performance or energy efficiency, offer further opportunities for leveraging the error-tolerance of application data, though these typically require very careful changes to the program source code.

Hardware-Based Memory Reliability Techniques.

There are various ECC techniques for memory, and we list the most dominant ones in Table 1. Using eight bits, SEC-DED can correct a single bit flip and detect up to two bit flips out of every 64 bits. DEC-TED is a generalization of SEC-DED that uses fourteen bits to correct two and detect three flipped bits out of every 64 bits. Chipkill [35] improves reliability by interleaving error detection and correction data among multiple DRAM chips. RAIM [111] is able to tolerate entire DIMMs failing by storing detection and correction data across multiple DIMMs. Virtualized ECC [155] maps ECC to software-visible locations in memory so that software can decide what ECC protection to use. While Virtualized ECC can help reduce the DRAM hardware cost of memory reliability, it requires modification to the processor’s memory management unit and cache(s).

Recent works propose new hardware-based techniques to tolerate soft and hard memory errors efficiently. We break these down into four categories: (1) Tolerating soft errors: BambooECC [67] proposes a new single-tier ECC family that enables adaptive graceful downgrade of ECC capabilities. CleanECC [45] provides both high memory reliability and flexible memory access granularity by using fine-grained error detection and coarse-grained error correction. XED [126]
uses in-DRAM ECC to reduce the overhead of double Chipkill. (2) Tolerating hard errors: ArchShield [124] proposes an architectural framework to identify and tolerate hard errors caused by DRAM cell failures. Citadel [125] proposes to tolerate large-granularity failures, such as row/bank failures, by replacing them with spares. Other works propose to identify and mitigate potentially recurring memory errors by page offlining [116], online testing [65, 131], and multi-rate refresh [99, 135]. (3) Reducing memory cost: FrugalECC [68] proposes a new flexible granularity compression to reduce the redundancy and energy consumption of ECC. Morphable ECC [29] proposes to reduce DRAM refresh overhead by reducing ECC strength to 6-bit ECC when the DRAM is in idle mode. (4) End-to-end memory error protection: AIECC [69] provides end-to-end protection for clock, control, command, and address (CCCA) signals in addition to data signals.

**Software-Based Memory Reliability Techniques.** Previous works (e.g., [55, 116, 140, 150]) show that the OS retiring memory pages after a certain number of errors can eliminate up to 96.8% of detected memory errors. While these techniques improve system reliability, they still require costly ECC hardware for detecting and identifying memory pages with errors. Other works attempt to reduce the impact of memory errors on system reliability by writing more reliable software [7], modifying the OS memory allocator [132], or using a compiler to generate a more error-tolerant version of the program [5, 22]. Other algorithmic solutions (e.g., memory bounds checks [88], watchdog timers [88], and checkpoint recovery [30, 31, 32, 90, 91, 95, 96, 97, 153, 154]) can also be used to improve resilience to memory errors.

Li et al. [98] propose to deploy software-based ECC in an in-memory key-value store, and show that it incurs low performance overhead. Recent works [149, 161] improve upon traditional RAIM-3 and use selective replication to reduce unnecessary memory redundancy. SDECC [46, 47] proposes to use strong error detection in the hardware, while tolerating hard memory errors and recovering from soft errors in the software.

**Exploiting Application Error Tolerance.** Flikker [102] proposes a technique to trade off DRAM reliability for energy savings. It relies on the programmer to separate application data into vulnerable or tolerant data. Less reliable mobile DIMMs have been proposed [109, 156] as a replacement for ECC DIMMs in servers to improve energy efficiency. Recent work [128] shows that RAMCloud can recover 35 GB of data from a failed server in 1.6 seconds using a log-structure storage.

**Heterogeneous (Hybrid) Memory Architectures.** Various recent works (e.g., [1, 6, 28, 36, 44, 94, 101, 113, 114, 133, 134, 136, 137, 157, 158, 160]) explore the use of heterogeneous memory architectures, consisting of multiple different types of memories. These works are mainly concerned with either mitigating the overheads of emerging memory technologies or improving performance and power efficiency. They do not investigate the use of multiple devices with different error correction capabilities. CREAM [107] and Odd-ECC [108] develop low-cost techniques to provide flexible provisioning of memory error correction capabilities. Recent works [3, 152] apply our heterogeneous reliability idea to processor caches to achieve better cost-reliability trade-offs.

## 5. Significance and Long-Term Impact

We believe that our DSN 2014 paper [104] will have long-term impact for three major reasons. First, it emphasizes and aims to solve the increasing cost of ensuring memory reliability as the error rates of memory devices continue to grow, which is a major trend as memory technology scales to smaller technology nodes [120, 121]. Second, it tackles memory system cost in datacenters, which is a problem that we expect will be increasingly important in the future. Third, it proposes a novel framework that uses hardware–software co-design to improve memory system reliability as well as cost, thereby hopefully inspiring future works to exploit software characteristics to improve system reliability and reduce system cost (and other important metrics).

**Increasing Memory Error Rate.** As DRAM scales to smaller process technology nodes, the reliability of DRAM continues to degrade [55, 61, 116, 120, 121, 122, 123, 141, 145, 146, 147]. For example, recent works 1) show the existence of disturbance errors in commodity DRAM chips operating in the field [72, 120]; 2) experimentally demonstrate the increasing importance of retention-related failures in modern DRAM devices [51, 62, 63, 64, 65, 99, 100, 131, 131, 135]; 3) examine the trade-off between DRAM reliability and latency [21, 23, 25, 27, 51, 66, 82, 83, 86]; and 4) advocate, including in a paper co-written by the Samsung and Intel memory design teams [61], for the use of in-DRAM error correcting codes to overcome the reliability challenges [61, 135]. As a result of decreasing DRAM reliability, maintaining the effective error rate at the levels we have today can (1) increase DRAM cost due to decreased yield, expensive quality assurance tests, and/or extra capacity for storing stronger error-correcting codes; or (2) reduce DRAM performance due to frequent error correction and logging. All of these solutions might make DRAM technology scaling more difficult and less appealing [120, 121, 122]. Our paper proposes a solution that enables the use of DRAM with higher error rates while still achieving reasonable application reliability, which can enable much more efficient scaling of DRAM to smaller technology nodes in the future.

Other memory technologies such as NAND flash memory [9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 43, 105, 106, 115], phase-change memory (PCM) [79, 80, 81, 114, 117, 136] and STT-MRAM [78, 114] also show a similar decreasing trend in their reliability with process technology scaling and the advent of multi-level cell (MLC) technology [121]. For example, like DRAM, NAND flash memory suffers from retention errors [9, 10, 11, 13, 14, 15], cell-to-cell program in-
terference errors [9, 10, 11, 14, 17, 19], and read disturb errors [9, 10, 11, 14, 20]. Additionally, NAND flash memory suffers from program/erase cycling errors [14, 18], and programming errors [12, 105, 130]. PCM suffers from endurance issues [79, 81, 136] and resistance drift [56]. HRM can be applied to these memory technologies with slight modifications to enable reliable high-density non-volatile devices in the future.

Increasing Datacenter Cost. Recent studies have shown that capital costs can account for the majority (e.g., around 57% in [4]) of datacenter TCO (total cost of ownership). As part of the cost of a server, the cost of the memory is comparable to that of the processors [77], and is likely to exceed processor cost and become the dominant cost for servers running data-intensive applications such as web search and social media services [34, 103, 138]. As future datacenters grow in scale, datacenter TCO will become an increasingly important factor in system design. Our paper demonstrates a way of optimizing datacenter TCO by reducing the cost of the memory system. The cost savings can be significant due to the increasing scale of such datacenters [50], making our proposed technique hopefully more important in the future.

Hardware–Software Co-Design. Our solution, heterogeneous-reliability memory, utilizes hardware–software cooperative design to reduce system cost. Our DSN 2014 paper [104] demonstrates the benefits of exploiting application characteristics to improve overall system design. For example, it shows that a significant number of errors can be corrected in software by reloading a clean copy of the data from storage. This motivates us to rethink the placement of different functionalities (such as error detection and error correction) across different system components and across software versus hardware to improve the cost–reliability trade-off.

Our DSN 2014 paper [104] has started a community discussion [50] on the feasibility of solving the problem of memory reliability by exploiting application memory error tolerance in the future, inspiring reporters to ask the question: “How good does memory need to be?” We hope that our characterization results and mechanisms will hopefully continue to inspire future works that can provide efficient and extensive characterization/estimation of application-level memory error tolerance [41], which can make our proposed technique applicable to a broader set of applications.

Two example works that build on ours include Odd-ECC [108] and CREAM [107]. Odd-ECC provides a mechanism to enable different levels of fault tolerance for the data stored in a commodity DRAM module. Odd-ECC maps the ECC bits to a memory address aligned with the data so that the memory controller can access both the data and the ECC bits efficiently. CREAM provides a mechanism to dynamically adjust the tradeoff between memory capacity/bandwidth used for ECC bits and fault tolerance within an ECC DRAM module. CREAM proposes several data layouts that reduce page faults and improve memory performance significantly when strong fault tolerance is not needed.

6. Conclusion

In our DSN 2014 paper [104], we develop a new methodology to quantify the tolerance of applications to memory errors. Using this methodology, we perform a case study of three new data-intensive workloads that show, among other new insights, that there exists a diverse spectrum of memory error tolerance both within and across these applications. Based on this observation, we introduce the idea of heterogeneous-reliability memory (HRM), which combines multiple different memories that have different reliability characteristics and error correction capabilities. We propose new hardware/software heterogeneous-reliability memory system designs, and evaluate them to show that (1) the one-size-fits-all approach to reliability in modern servers is inefficient in terms of cost, and (2) heterogeneous-reliability systems can achieve the benefits of both low cost and high single server availability/reliability. We hope that our techniques can enable the use of lower-cost memory devices to reduce the server hardware cost of datacenters, and that our analyses will spur future research on heterogeneous-reliability memory systems. As DRAM technology scales into small feature sizes and becomes less reliable and memory cost becomes more important in datacenters in the future, we hope that our findings and ideas will inspire more research to improve the cost–reliability trade-off in memory systems. We believe different HRM designs can be employed to optimize other key trade-offs and target metrics (e.g., performance vs. energy consumption) in modern systems. Our DSN 2014 paper just scratches the surface of a large amount of research and design space to be explored.

Acknowledgments

We thank Saugata Ghose for his dedicated effort in the preparation of this article. We thank the anonymous reviewers and the members of SAFARI research group for feedback. We acknowledge the support of Microsoft and Samsung. This research was partially supported by the Intel Science and Technology Center for Cloud Computing and the NSF (grants 0953246, 1065112, and 1212962).

References

[1] N. Agarwal and T. F. Wenisch, "Thermostat: Application-Transparent Page Management for Two-Tiered Main Memory," in ASPLOS, 2017.
[2] Z. Al-Ars, "DRAM Fault Analysis and Test Generation," Ph.D. dissertation, Delft, 2005.
[3] S. Arslan, H. R. Topcuoglu, M. T. Kandemir, and O. Tosun, "Performance and Energy Efficient Asymmetrically Reliable Caches for Multicore Architectures," in IPDPSW, 2015.
[4] L. A. Barroso et al., The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Morgan & Claypool Publishers, 2009.
[5] A. Benso et al., "A C/C++ Source-to-Source Compiler for Dependable Applications," in DSN, 2000.
[6]  S. Bock, B. R. Childers, R. Melhem, and D. Mossé, "Concurrency Migration of Multiple Pages in Software-Managed Hybrid Main Memory," in ICCD, 2016.
[7] C. Borchert et al., "Generative Software-Based Memory Error Detection and Correction for Operating System Data Structures," in DSN, 2013.
et al. [37] N. El-Sayed, I. A. Stefanovici, G. Amvrosiadis, A. A. Hwang, and B. Schroeder, [38] E. M. Elnozahy [35] T. J. Dell, “A White Paper on the Benefits of Chipkill-Correct ECC for PC Server [33] CST, Inc., “Memory Test Background,” http://tinyurl.com/m7c3wf7. 2000.

et al. [27] K. K. Chang, A. G. Yaglikci, A. Agrawal, N. Chatterjee, S. Ghose, A. Kashyap, [25] K. K. Chang, D. Lee, Z. Chishti, A. Alameldeen, C. Wilkerson, Y. Kim, and [22] J. Chang et al. [20] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu, “Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery,” in ASPLOS, 2012.

et al. [21] K. Chandrasekar, S. Goossens, C. Weis, M. Koedam, B. Akesson, N. Wehn, and [20] Y. Cai, Y. Luo, S. Ghose, and O. Mutlu, “Reducing DRAM Performance by Parallelizing Refreshes With Accesses,” in HPCA, 2014.

et al. [18] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, “Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques,” in HPCA, 2017.

et al. [16] Y. Cai et al., “Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis,” in DATE, 2012.

et al. [15] Y. Cai et al., “Fault Tolerance for I/O Devices,” CASES/TECS, 2011.

et al. [14] Y. Cai et al., “Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis,” in DATE, 2012.

et al. [13] Y. Cai et al., “Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis,” in DATE, 2012.

et al. [12] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, “Improving the Reliability of Chip-Off Forensic Analysis of NAND Flash Memory Devices,” Digital Investigation, Mar 2017.

et al. [11] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, “Improving the Reliability of Chip-Off Forensic Analysis of NAND Flash Memory Devices,” Digital Investigation, Mar 2017.

et al. [10] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, “Improving the Reliability of Chip-Off Forensic Analysis of NAND Flash Memory Devices,” Digital Investigation, Mar 2017.

et al. [9] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, “Improving the Reliability of Chip-Off Forensic Analysis of NAND Flash Memory Devices,” Digital Investigation, Mar 2017.

et al. [8] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, “Improving the Reliability of Chip-Off Forensic Analysis of NAND Flash Memory Devices,” Digital Investigation, Mar 2017.
[154] X. Xu et al., "Understanding Soft Error Propagation Using Efficient Vulnerability-Driven Fault Injection," in DSN, 2012.

[155] D. H. Yoon et al., "Virtualized and Flexible ECC for Main Memory," in ASPLOS, 2010.

[156] D. H. Yoon et al., "BOOM: Enabling Mobile Memory Based Low-power Server DIMMs," in ISCA, 2012.

[157] H. Yoon et al., "Row Buffer Locality Aware Caching Policies for Hybrid Memories," in ICCD, 2012.

[158] X. Yu, C. J. Hughes, N. Satish, O. Mutlu, and S. Devadas, "Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation," in MICRO, 2017.

[159] Y. Yue, "Caching in Datacenters," https://twitter.github.io/pelikan/2016/04/03/caching-in-datacenters.html, 2012.

[160] W. Zhang and T. Li, "Exploring Phase Change Memory and 3D Die-Stacking for Power/Thermal Friendly, Fast and Durable Memory Architectures," in PACT, 2009.

[161] R. Zheng and M. C. Huang, "Redundant Memory Array Architecture for Efficient Selective Protection," in ISCA, 2017.