A Memory Soft Error Measurement on Production Systems*

Xin Li Kai Shen Michael C. Huang
University of Rochester
{xinli@ece, kshen@cs, huang@ece}.rochester.edu

Lingkun Chu
Ask.com
lchu@ask.com

Abstract

Memory state can be corrupted by the impact of particles causing single-event upsets (SEUs). Understanding and dealing with these soft (or transient) errors is important for system reliability. Several earlier studies have provided field test measurement results on memory soft error rate, but no results were available for recent production computer systems. We believe the measurement results on real production systems are uniquely valuable due to various environmental effects. This paper presents methodologies for memory soft error measurement on production systems where performance impact on existing running applications must be negligible and the system administrative control might or might not be available.

We conducted measurements in three distinct system environments: a rack-mounted server farm for a popular Internet service (Ask.com search engine), a set of office desktop computers (Univ. of Rochester), and a geographically distributed network testbed (PlanetLab). Our preliminary measurement on over 300 machines for varying multi-month periods finds 2 suspected soft errors. In particular, our result on the Internet servers indicates that, with high probability, the soft error rate is at least two orders of magnitude lower than those reported previously. We provide discussions that attribute the low error rate to several factors in today's production system environments. As a contrast, our measurement unintentionally discovers permanent (or hard) memory faults on 9 out of 212 Ask.com machines, suggesting the relative commonness of hard memory faults.

1 Introduction

Environmental noises can affect the operation of microelectronics to create soft errors. As opposed to a "hard" error, a soft error does not leave lasting effects once it is corrected or the machine restarts. A primary noise mechanism in today's machines is particle strike. Particles hitting the silicon chip create electron-hole pairs which, through diffusion, can collect at circuit nodes and outweigh the charge stored and create a flip of logical state, resulting in an error. The soft error problem at sea-level was first discovered by Intel in 1978 [9].

Understanding the memory soft error rate is an important part in assessing whole-system reliability. In the presence of inexplicable system failures, software developers and system administrators sometimes point to possible occurrences of soft errors without solid evidence. As another motivating example, recent studies have investigated the influence of soft errors on software systems [10] and parallel applications [5], based on presumably known soft error rate and occurrence patterns. Understanding realistic error occurrences would help quantify the results of such studies.

A number of soft error measurement studies have been performed in the past. Probably the most extensive test results published were from IBM [12, 14–16]. Particularly in a 1992 test, IBM reported 5950 FIT (Failures In Time, specifically, errors in 10^9 hours) of error rate for a vendor 4Mbit DRAM. The most recently published results that we are aware of were based on tests in 2001 at Sony and Osaka University [8]. They tested $0.18\,\mu$m and $0.25\,\mu$m SRAM devices to study the influence of altitude, technology, and different sources of particles on the soft error rate, though the paper does not report any absolute error rate. To the best of our knowledge, Normand's 1996 paper [11] reported the only field test on production systems. In one 4-month test, they found 4 errors out of 4 machines with total 8.8 Gbit memory. In another 30-week test, they found 2 errors out of 1 machine with 1 Gbit memory. Recently, Tezzaron [13] collected error rates reported by various sources and concluded that 1000–5000 FIT per Mbit would be a reasonable error rate for modern memory devices. In summary, these studies all suggest soft error rates in the range of 200–5000 FIT per Mbit.

Most of the earlier measurements (except [8]) were over a decade old and most of them (except [11]) were conducted in artificial computing environments where the target devices are dedicated for the measurement. Given the scaling of technology and the countermeasures deployed at different levels of system design, the trends of error rate in real-world systems are not clear. Less obvious environmental factors may also play a role. For example, the way a machine is assembled and packaged as well as the memory chip layout on the main computer board can affect the

*This work was supported in part by the National Science Foundation (NSF) grants CCR-0306473, ITR/IIS-0312925, CNS-0509270, CNS-0615045, and CCF-0621472. Shen was also supported by an NSF CAREER Award CCF-0448413 and an IBM Faculty Award.

chance of particle strikes and consequently the error rate.

We believe it is desirable to measure memory soft errors in today's representative production system environments. Measurement on production systems poses significant challenges. The infrequent nature of soft errors demands long-term monitoring. As such, our measurement must not introduce any noticeable performance impact on the existing running applications. Additionally, to achieve wide deployment of such measurements, we need to consider the cases where we do not have administrative control on measured machines. In such cases, we cannot perform any task requiring the privileged accesses and our measurement tool can be run only at user level. The rest of this paper describes our measurement methodology, deployed measurements in production systems, our preliminary results and the result analysis.

2 Measurement Methodology and Implementation

We present two soft error measurement approaches targeting different production system environments. The first approach, memory controller direct checking, requires administrative control on the machine and works only with ECC memory. The second approach, non-intrusive user-level monitoring, does not require administrative control and works best with non-ECC memory. For each approach, we describe its methodology, implementation, and analyze its performance impact on existing running applications in the system.

2.1 Memory Controller Direct Checking

An ECC memory module contains extra circuitry storing redundant information. Typically it implements single error correction and double error detection (SEC-DED). When an error is encountered, the memory controller hub (a.k.a. Northbridge) records necessary error information in some special-purpose registers. Meanwhile, if the error involves a single bit, then it is corrected automatically by the controller. The memory controller typically signals the BIOS firmware when an error is discovered. The BIOS error-recording policies vary significantly from machine to machine. In most cases, single-bit errors are ignored and never recorded. The BIOS typically clears the error information in memory controller registers on receiving error signals. Due to the BIOS error handling, the operating system would not be directly informed of memory errors without reconfiguring the memory controller.

Our memory controller direct checking of soft errors includes two components:

- *Hardware configuration*: First, we disable any BIOS manipulation of the memory controller error handling in favor of error handling by the OS software. Particularly, we do not allow BIOS to clear memory controller error information. Second, we enable periodic hardware-level memory scrubbing which walks through the memory space to check errors. This is in addition to error detection triggered by software-initiated memory reading. Errors discovered at the hardware level are recorded in appropriate memory controller registers. Memory scrubbing is typically performed at a low frequency (e.g., 1 GB per 1.5 hours) to minimize its energy consumption and interruption to running applications.
- *Software probing*: We augment the OS to periodically probe appropriate memory controller registers and acquire desired error information. Since the memory controller register space is limited and usually only a few errors are recorded, error statistics can be lost if the registers are not read and cleared in time. Fortunately, soft error is typically a rare event and thus our probing can be quite infrequent — it only needs to be significantly more often than the soft error occurrence frequency.

Both hardware configuration and software probing in this approach require administrative privilege. The implementation involves modifications to the memory controller driver inside the OS kernel. The functionality of our implementation is similar to the Bluesmoke tool [3] for Linux. The main difference concerns exposing additional error information for our monitoring purpose.

In this approach, the potential performance impact on existing running applications includes the software overhead of controller register probing and memory bandwidth consumption due to scrubbing. With low frequency memory scrubbing and software probing, this measurement approach has a negligible impact on running applications.

2.2 Non-intrusive User-level Monitoring

Our second approach employs a user-level tool that transparently recruits memory on the target machine and periodically checks for any unexpected bit flips. Since our monitoring program competes for the memory with running applications, the primary issue in this approach is to determine an appropriate amount of memory for monitoring. Recruiting more memory makes the monitoring more effective. However, we must leave enough memory so that the performance impact on other running applications is limited. This is important since we target production systems hosting real live applications and our monitoring must be long running to be effective.

This approach does not require administrative control on the target machine. At the same time, it works best with non-ECC memory since the common SEC-DED feature in ECC memory would automatically correct single-bit errors and consequently our user-level tool cannot observe them.

Earlier studies like Acharya and Setia [1] and Cipar et al. [4] have proposed techniques to transparently steal idle memory from non-dedicated computing facilities. Unlike many of the earlier studies, we do not have administrative control of the target system and our tool must function completely at user level. The system statistics that we can use are limited to those explicitly exposed by the OS (e.g., the Linux `/proc` file system) and those that can be measured by user-level micro-benchmarks [2].

Design and Implementation Our memory recruitment for error monitoring should not affect the performance of other running applications. We can safely recruit the free memory pages that are not allocated to currently running applications. Furthermore, some already allocated memory may also be recruited as long as we can bound its effects on existing running applications. Specifically, we employ a *stale-memory recruitment policy* in which we only recruit allocated memory pages that have not been recently used (i.e., not used for a certain duration of time D). Under this policy, within any time period of duration D, every physical memory page can be recruited for no more than once (otherwise the second recruitment would have recruited a page that was recently allocated and used). Therefore, within any time period of duration D, the additional application page evictions induced by our monitoring is bounded by the physical memory size S. If we know the page fault I/O throughput R, we can then bound the application slowdown induced by our monitoring to $\frac{S}{D \cdot R}$.

Below we present a detailed design to our approach, which includes three components.

- *Memory recruitment*: We periodically invoke a routine to recruit memory to the monitoring pool. First, it checks the amount of free memory in the system that is not used by any running applications (through existing system interface or a user-level micro-benchmark). In order to further utilize those allocated but rarely used memory, we may also recruit a certain amount of extra memory in the absence of memory contention. Memory contention can be detected by observing recent eviction of pages from the monitoring pool (see below) or a slowdown of memory-intensive tasks (motivated by [6]).
- *Periodic touching*: Since our monitoring tool runs as a normal user-level program, it competes memory with other running applications according to the OS memory management. We assume that the OS employs the Least-Recently Used (LRU) page replacement order or its approximation. To realize the stale-memory recruitment policy, we periodically access the recruited memory pages so that each page is reused at the frequency of once per time duration D. Under the LRU replacement order, application pages that are accessed more often are unlikely to be evicted before the recruited pages in our monitoring tool.

  The touching also serves the purpose of error checking. We read every single word of the page and examines if the pattern written initially still remains. If not, it indicates an error just occurred in the most recent period.
- *Releasing evicted pages*: Recruited memory pages may be swapped out of physical memory during memory contention when all existing application pages are not stale enough (i.e., have been used within the last time duration D). We should detect these evicted pages and release them from the monitoring pool.

We discuss some implementation issues in practice. First, the OS typically attempts to maintain a certain minimum amount of free memory (e.g., to avoid deadlocks when reclaiming pages) and a reclamation is triggered when the free memory amount falls below the threshold (we call `minfree`). We can measure `minfree` of a particular system by running a simple user-level micro-benchmark. At the memory recruitment, we are aware that the practical free memory in the system is the nominal free amount subtract `minfree`.

Second, it may not be straightforward to detect evicted pages from the monitoring pool. Some systems provide direct interface to check the in-core status of memory pages (e.g., the `mincore` system call). Without such direct interface, we can tell the in-core status of a memory page by simply measuring the time of accessing any data on the page. Note that due to OS prefetching, the access to a single page might result in the swap-in of multiple contiguous out-of-core pages. To address this, each time we detect an out-of-core recruited page, we discard several adjacent pages (up to the OS prefetching limit) along with it.

Performance Impact on Running Applications We measure the performance impact of our user-level monitoring tool on existing running applications. Our tests are done in a machine with a 2.8 GHz Pentium4 processor and 1 GB main memory. The machine runs Linux 2.6.18 kernel. We examine three applications in our test: 1) the Apache web server running the static request portion of the SPECweb99 benchmark; 2) MCF from SPEC CPU2000 — a memory-intensive vehicle scheduling program for mass transportation; and 3) compilation and linking of the Linux 2.6.18 kernel. The first is a typical server workload while the other two are representative workstation workloads.

We set the periodic memory touching interval D according to a desired application slowdown bound. An accurate setting requires the knowledge of the page fault

I/O throughput. Here we use a simple estimation of I/O throughput as half the peak sequential disk access throughput (around 57 MB/s for our disk). Therefore a periodic memory touching interval D=30.48 minutes is needed for achieving 2% application slowdown bound. Our program was able to recruit 376.70 MB, 619.24 MB and 722.77 MB on average (out of the total 1 GB) when it runs with Apache, MCF, and Linux compilation respectively. At the same time, the slowdown is 0.52%, 0.26%, and 1.20% for the three applications respectively. The slowdown for Apache is calculated as "$1 - \frac{\text{new throughput}}{\text{original throughput}}$" while the slowdown for the other two applications is calculated as "$1 - \frac{\text{original runtime}}{\text{new runtime}}$". The monitoring-induced slowdown can be reduced by increasing the periodic memory touching interval D. Such adjustment may at the same time reduce the amount of recruited memory.

2.3 Error Discovery in Accelerated Tests

To validate the effectiveness of our measurement approaches and resulted implementation, we carried out a set of accelerated tests with guaranteed error occurrences. To generate soft errors, we heated the memory chip using a heat gun. The machine under test contains 1 GB DDR2 memory with ECC and the memory controller is Intel E7525 with ECC. The ECC feature has a large effect on our two approaches — the controller direct checking requires ECC memory while the user-level monitoring works best with non-ECC memory. To consider both scenarios, we provide results for two tests — one with the ECC hardware enabled and the other with ECC disabled. The results on error discovery are shown in Table 1.

Overall, results suggest that both controller direct probing and user-level monitoring can discover soft errors at respective targeted environments. With ECC enabled, all the single-bit errors are automatically corrected by the ECC hardware and thus the user-level monitoring cannot observe them. We also noticed that when ECC was enabled, the user-level monitoring found less multi-bit errors than the controller direct checking did. This is because the user-level approach was only able to monitor part of the physical memory space (approximately 830 MB out of 1 GB).

Test with ECC enabled

Approach	Single-bit errors	Multi-bit errors
Controller checking	2472	139
User-level monitoring	N/A	106

Test with ECC disabled

Approach	Single-bit errors	Multi-bit errors
User-level monitoring	15	0

Table 1: Errors found in heat-induced accelerated tests. The ECC-disabled test was done at much lower heat intensity compared to the ECC-enabled one. We did this because without the shielding from ECC, all single-bit errors may manifest at the software level, and thus high heat intensity is very likely to crash the OS and disrupt the test.

3 Deployed Measurements and Preliminary Results

We have deployed our measurement in three distinct production system environments: a rack-mounted server farm, a set of office desktop computers, and a geographically distributed network testbed. We believe these measurement targets represent many of today's production computer system environments.

- *Ask.com servers:* These rack-mounted servers are equipped with high-end ECC memory modules and we have administrative control over these machines. We use the memory controller direct checking approach in this measurement. We monitored 212 servers for approximately three months. On average, we were monitoring 3.92 GB memory on each machine.
- *UR desktop computers:* We conducted measurement on a number of desktop computers at the University of Rochester. These machines are provided on the condition of no change to the system and no impact to the running application performance. Therefore we employ the user-level monitoring approach in this measurement. Since this approach works best with non-ECC memory, we identified a set of machines with non-ECC memory by checking vendor-provided machine specification. This measurement has been deployed on 20 desktop (each with 512 MB RAM) computers for around 7 months. On average we recruited 104.23 MB from each machine.
- *PlanetLab machines:* We chose PlanetLab because of its geographically distributed set of machines. Geographic locations (and elevation in particular) may affect soft error occurrence rate. Since we do not have administrative control for these machines, we employ the user-level monitoring approach in this measurement. Again, we search for a set of machines with non-ECC memory to maximize the effect of our measurement. Since we do not know the exact models of most PlanetLab machines, we cannot directly check the vendor-provided machine specification. Our approach is to collect memory controller device ID from the `/proc` file system and then look up its ECC capability. This measurement has been deployed on 70 PlanetLab machines for around 7 months. Since most PlanetLab machines are intensively used and free memory is scarce, we were only able to recruit

Measurement environment	Ask.com	Ask.com (excluding 9 servers with hard errors)	UR Desktops	PlanetLab
Time-memory extent	76,456 GB×day	73,571 GB×day	428 GB×day	23 GB×day
Measurement result	8288 errors, most of which are believed to be hard errors	2 errors, suspected to be soft errors	no error	no error

Table 2: Results of deployed measurements.

approximately 1.54 MB from each machine.

Aside from the respective pre-deployment test periods, we received no complaint on application slowdown for all three measurement environments.

So far we detected no errors on UR desktop computers and PlanetLab machines. At the same time, our measurements on Ask.com servers logged 8288 memory errors concentrating on 11 (out of 212) servers. These errors on the Ask.com servers warrant more explanations:

- Not all logged errors are soft errors. In particular, some are due to permanent (hard) chip faults. One way to distinguish soft and hard errors is that hard errors tend to repeat on the same memory addresses since they are not correctable. On the other hand, soft errors rarely repeat on the same memory addresses with the assumption that soft errors occur on memory addresses in a uniformly random fashion. Using this criterion, we find 9 out of these 11 severs contain hard chip faults.
- We assume that soft errors and hard errors occur independently on host machines. We believe the assumption is reasonable because soft errors are typically due to external environmental factors (e.g., particle strikes) while hard errors are largely due to internal chip properties. With this assumption, we can exclude the 9 machines with known hard errors from our Ask.com measurement pool without affecting the representativeness of soft error statistics on the remaining machines.
- After excluding the 9 servers with known hard errors, there are 2 machines each with a suspected memory soft error.
- Most detected errors on the 11 Ask.com servers are single-bit errors correctable by the ECC, and thus pose no impact on running software. However, the logged memory controller information suggests that at least one machine contains some multi-bit errors that are not correctable.

In Table 2, we list the overall *time-memory extent* (defined as the product of time and the average amount of considered memory over time) and discovered errors for all deployed measurement environments.

Result Analysis Based on the measurement results, below we calculate a probabilistic upper-bound on the Failure-In-Time rate. We assume that the occurrence of soft errors follows a Poisson process. Therefore within a time-memory extent T, the probability that k errors happen is:

$$Pr_{\lambda,T}[N = k] = \frac{e^{-\lambda T}(\lambda T)^k}{k!} \quad (1)$$

where λ is the average error rate (i.e., the error occurs λ times on average for every unit of time-memory extent). And particularly the probability for no error occurrence (k = 0) during a measurement over time-memory extent T is:

$$Pr_{\lambda,T}[\text{no error}] = e^{-\lambda T} \quad (2)$$

For a given T and the number of error occurrences k, let us call Λ a p-probability upper-bound of the average error occurrence rate if:

$$\forall \lambda > \Lambda : \quad Pr_{\lambda,T}[N = k] < 1 - p$$

In other words, if a computing environment has an average error occurrence rate that is more than the p-probability upper-bound, then the chance for k error occurrence during a measurement of time-memory extent T is always less than $1 - p$.

We apply the above analysis and metric definition on the error measurement results of our deployed measurements. We first look at UR desktop measurement in which no error is reported. According to Equation (2), we know that $\frac{1}{T} \ln \frac{1}{1-p}$ is a p-probability upper-bound of the average error occurrence rate. Consequently, since $T = 428$ GB $\times$ day for the UR desktop measurement, we can calculate that 54.73 FIT per Mbit is a 99%-probability upper-bound of the average error occurrence rate for this environment.

We then examine the Ask.com environment excluding 9 servers with hard errors. In this environment, 2 (or fewer) soft errors over $T = 73,571$ GB $\times$ day yields a 99%-probability upper-bound of the average error occurrence rate at 0.56 FIT per Mbit. This is much lower than previously-reported error rate (200–5000 FIT per Mbit) that we summarized in Section 1.

4 Conclusion and Discussions

Our preliminary result suggests that the memory soft error rate in two real production systems (a rack-mounted server environment and a desktop PC environment) is much lower than what the previous studies concluded. Particularly in the server environment, with high probability, the soft error rate is at least two orders of magnitude lower than those reported previously. We discuss several potential causes for this result.

- *Hardware layout*: In the IBM Blue Spruce experiment, O'Gorman et al. [12] suggested that the main source of cosmic ray comes from straight above. The Ask.com machines are arranged in a way such that memory DIMMs are plugged perpendicular to the horizontal plane. This could significantly reduce the area facing the particle bombardment.
- *Chip size reduction*: Given the continuous scaling of the VLSI technology, the size of a memory cell has reduced dramatically over the years. As a result, for an equal-capacity comparison, the probability of a particle hitting any cell in today's memory is much lower than that of a decade ago. We believe that this probability reduction outweighs the increased vulnerability of each cell due to the reduction in device critical charge.
- *DRAM vs. SRAM*: Measurements described in this paper only target the main memory, which is usually DRAM. Previous studies [7, 17] show that DRAM is less sensitive to cosmic rays than SRAM (typically used for fast-access cache today).

An understanding on the memory soft error rate demystifies an important part of whole-system reliability in today's production computer systems. It also provides the basis for evaluating whether software-level countermeasures against memory soft errors are urgently needed. Our results are still preliminary and our measurements are ongoing. We hope to be able to draw more complete conclusions from future measurement results. Additionally, soft errors can occur on components other than memory, which may affect system reliability in different ways. In the future, we also plan to devise methodologies to measure soft errors in other computer system components such as CPU register, SRAM cache, and system bus.

Acknowledgments We would like to thank the two dozen or so people at the University of Rochester CS Department who donated their desktops for our memory error monitoring. We are also grateful to Tao Yang and Alex Wong at Ask.com who helped us in acquiring administrative access to Ask.com Internet servers. Finally, we would also like to thank the USENIX anonymous reviewers for their helpful comments that improved this paper.

References

[1] A. Acharya and S. Setia. Availability and utility of idle memory in workstation clusters. In *SIGMETRICS*, pages 35–46, 1999.

[2] A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau. Information and control in gray-box systems. In *SOSP*, pages 43–56, 2001.

[3] EDAC project. http://bluesmoke.sourceforge.net.

[4] J. Cipar, M. D. Corner, and E. D. Berger. Transparent contribution of memory. In *USENIX*, 2006.

[5] C. da Lu and D. A. Reed. Assessing fault sensitivity in MPI applications. In *Supercomputing*, 2004.

[6] J. Douceur and W. Bolosky. Progress-based Regulation of Low-importance Processes. In *SOSP*, pages 247–260, Kiawah Island, SC, Dec. 1999.

[7] A. H. Johnston. Scaling and technology issues for soft error rates. In *4th Annual Research Conf. on Reliability*, 2000.

[8] H. Kobayashi, K. Shiraishi, H. Tsuchiya, H. Usuki, Y. Nagai, and K. Takahisa. Evaluation of LSI soft errors induced by terrestrial cosmic rays and alpha particles. Technical report, Sony Corporation and RCNP Osaka University, 2001.

[9] T. C. May and M. H. Woods. Alpha-particle-induced soft errors in dynamic memories. *IEEE Trans. on Electron Devices*, 26(1):2–9, 1979.

[10] A. Messer, P. Bernadat, G. Fu, D. Chen, Z. Dimitrijevic, D. J. F. Lie, D. Mannaru, A. Riska, and D. S. Milojicic. Susceptibility of commodity systems and software to memory soft errors. *IEEE Trans. on Computers*, 53(12):1557–1568, 2004.

[11] E. Normand. Single event upset at ground level. *IEEE Trans. on Nuclear Science*, 43(6):2742–2750, 1996.

[12] T. J. O'Gorman, J. M. Ross, A. H. Taber, J. F. Ziegler, H. P. Muhlfeld, C. J. Montrose, H. W. Curtis, and J. L. Walsh. Field testing for cosmic ray soft errors in semiconductor memories. *IBM J. of Research and Development*, 40(1):41–50, 1996.

[13] Tezzaron Semiconductor. Soft errors in electronic memory. *White paper*, 2004. http://www.tezzaron.com/about/papers/papers.html.

[14] J. F. Ziegler. Terrestrial cosmic rays. *IBM J. of Research and Development*, 40(1):19–39, 1996.

[15] J. F. Ziegler et al. IBM experiments in soft fails in computer electronics (1978–1994). *IBM J. of Research and Development*, 40(1):3–18, 1996.

[16] J. F. Ziegler, H. P. Muhlfeld, C. J. Montrose, H. W. Curtis, T. J. O'Gorman, and J. M. Ross. Accelerated testing for cosmic soft-error rate. *IBM J. of Research and Development*, 40(1):51–72, 1996.

[17] J. F. Ziegler, M. E. Nelson, J. D. Shell, R. J. Peterson, C. J. Gelderloos, H. P. Muhlfeld, and C. J. Montrose. Cosmic ray soft error rates of 16-Mb DRAM memory chips. *IEEE J. of Solid-State Circuits*, 33(2), 1998.