Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field

Justin Meza Qiang Wu * Sanjeev Kumar * Onur Mutlu
Carnegie Mellon University * Facebook, Inc.

***Abstract*—Computing systems use dynamic random-access memory (DRAM) as main memory. As prior works have shown, failures in DRAM devices are an important source of errors in modern servers. To reduce the effects of memory errors, error correcting codes (ECC) have been developed to help detect and correct errors when they occur. In order to develop effective techniques, including new ECC mechanisms, to combat memory errors, it is important to understand the memory reliability trends in modern systems.**

In this paper, we analyze the memory errors in the *entire fleet* of servers at Facebook over the course of fourteen months, representing billions of device days. The systems we examine cover a wide range of devices commonly used in modern servers, with DIMMs manufactured by 4 vendors in capacities ranging from 2 GB to 24 GB that use the modern DDR3 communication protocol.

We observe several new reliability trends for memory systems that have not been discussed before in literature. We show that (1) memory errors follow a power-law, specifically, a Pareto distribution with decreasing hazard rate, with average error rate exceeding median error rate by around 55×; (2) non-DRAM memory failures from the memory controller and memory channel cause the majority of errors, and the hardware and software overheads to handle such errors cause a kind of denial of service attack in some servers; (3) using our detailed analysis, we provide the first evidence that more recent DRAM cell fabrication technologies (as indicated by chip density) have substantially higher failure rates, increasing by 1.8× over the previous generation; (4) DIMM architecture decisions affect memory reliability: DIMMs with fewer chips and lower transfer widths have the lowest error rates, likely due to electrical noise reduction; (5) while CPU and memory utilization do not show clear trends with respect to failure rates, *workload type* can influence failure rate by up to 6.5×, suggesting certain memory access patterns may induce more errors; (6) we develop a model for memory reliability and show how system design choices such as using lower density DIMMs and fewer cores per chip can reduce failure rates of a baseline server by up to 57.7%; and (7) we perform the first implementation and real-system analysis of page offlining at scale, showing that it can reduce memory error rate by 67%, and identify several real-world impediments to the technique.

I. Introduction

Computing systems store a variety of data in memory – program variables, operating system and file system structures, program binaries, and so on. The main memory in modern systems is composed of dynamic random-access memory (DRAM), a technology that, from the programmer's perspective, has the following property: a byte written to an address can be read correctly, repeatedly, until it is overwritten or the machine is turned off. All correct programs rely on DRAM to operate in this manner and DRAM manufacturers work hard to design reliable devices that obey this property.

Unfortunately, DRAM does not always obey this property. Various events can change the data stored in DRAM, or even permanently damage DRAM. Some documented events include transient charged particle strikes from the decay of radioactive molecules in chip packaging material, charged alpha particles from the atmosphere [34], and wear-out of the various components that make up DRAM chips (e.g., [7, 6]). Such faults, if left uncorrected, threaten program integrity. To reduce this problem, various error correcting codes (ECC) for DRAM data [12, 11] have been used to detect and correct memory errors. However, these techniques require additional DRAM storage overheads [33] and DRAM controller complexity and cannot detect or correct all errors.

Much past research has been directed toward analyzing the causes and effects of memory errors in the field (e.g., [44, 16, 47, 48, 10, 46, 45, 40, 27, 28]). These past works identified a variety of DRAM failure modes and have formed the basis of the community's understanding of DRAM reliability. Our goal is to strengthen the understanding of DRAM failures in the field by comprehensively studying *new trends* in DRAM errors in a large-scale production datacenter environment using *modern* DRAM devices and workloads. To this end, this paper presents our analysis of memory errors across Facebook's *entire fleet* of servers over the course of fourteen months and billions of device days. Our main contributions are threefold. We: *(1) analyze new DRAM failure trends in modern devices and workloads that have not been identified in prior work, (2) develop a model for examining the memory failure rates of systems with different characteristics, and (3) describe and perform the first analysis of a large-scale implementation of a software technique proposed in prior work to reduce DRAM error rate (page offlining [49]).* Specifically, we observe several new reliability trends for memory systems that have not been discussed before in literature:

(1) The number of memory errors per machine follows a *power-law* distribution, specifically a Pareto distribution, with decreasing hazard rate. While prior work reported the *average* memory error rate per machine, we find that the average exceeds the *median* amount by around 55×, and thus may not be a reliable number to use in various studies.

(2) Non-DRAM memory failures, such as those in the memory controller and the memory channel, are the source of the *majority* of errors that occur. Contrary to popular belief, memory errors are *not* always isolated events and can bombard a server (if not handled appropriately), creating a kind of *denial of service attack*. No prior work that we are aware of that examined DRAM chip-level failures accounted for this effect.

(3) DRAM failure rates increase with newer cell fabrication technologies (as indicated by chip density, which is a good indicator of technology node): 4 Gb chips have 1.8× higher failure rates than 2 Gb chips. Prior work that examined DRAM *capacity*, which is *not* closely related to fabrication technology, observed inconclusive trends. Our empirical finding is that the quadratic rate at which DRAM density increases with each generation has made maintaining or reducing DRAM failure rate untenable, as also indicated by a recent paper by Samsung and Intel [20].

(4) DIMM architecture characteristics, such as the number of data chips per DIMM and the transfer width of each chip, affect memory error rate. The best architecture for device reliability occurs when there are both low chips per DIMM and small transfer width. This is likely due to reductions in the amount of electrical disturbance within the DIMM.

(5) The *type* of work that a server performs (i.e., its workload), and *not* CPU and memory utilization, affects failure rate. We find that the DRAM failure rate of different workloads can vary by up to 6.5×. This large variation in workloads can potentially be due to memory errors that are induced by certain

access patterns, such as accessing the same memory location in rapid succession, as shown in controlled studies in prior work [23].

(6) We develop a model for quantifying DRAM reliability across a wide variety of server configurations and show how it can be used to evaluate the server failure rate trends for different system designs. We show that using systems with lower density DIMMs or fewer CPUs to access memory can reduce DRAM failure rates by 57.7% and 34.6%, respectively. We make this model publicly available at [1].

(7) We describe our implementation of page offlining [49] at scale and evaluate it on a fraction (12,276) of the servers that we examine. We show that it can reduce memory error rate by around 67%. While prior work reported larger error rate reductions in simulation [16], we show that real-world factors such as memory controller and memory channel failures and OS-locked pages that cannot be taken offline can limit the effectiveness of this technique.

II. Background and Methodology

A. Server Memory Organization

Modern servers have one or two processor chips that are connected to DRAM via several memory *channels* that are operated in parallel. Attached to each channel are *dual in-line memory modules (DIMMs)* that provide an interface for accessing data stored across multiple DRAM chips. Processors use the *double data rate (DDR)* protocol to communicate with DIMMs. Chips on a DIMM are logically organized into *ranks* and chips within a rank are operated in lockstep. Chips contain multiple *banks* that are operated in parallel. Each bank is organized into *rows* (typically 16 K to 64 K) and *columns* (typically 2 K to 4 K). At the intersection of a row and a column is a DRAM *cell*, which stores a single bit of data. We refer the reader to [24, 26, 29, 25] for more information on DRAM organization and operation.

B. Memory Errors and Their Handling

As prior works have shown, DRAM errors occur relatively commonly due to a variety of stimuli [34, 44, 16, 47, 48, 10, 46, 6, 27, 45, 30, 23, 21]. To protect against such errors in servers, additional data is stored in the DIMM (in a separate DRAM chip) to maintain *error correcting codes (ECC)* computed over data. These codes can detect and correct a small number of errors. For example, *single error correction, double error detection (SEC-DED)* is a common ECC strategy that can detect any 2 flipped bits and correct 1 flipped bit per 64 bits by storing an additional 12.5% of ECC metadata. An error that can be corrected by ECC is called a correctable error (CE); an error that cannot be corrected by ECC, but which can still be detected by ECC, is called an uncorrectable error (UCE).

The processor's *memory controller* orchestrates access to the DRAM devices and is also responsible for checking the ECC metadata and detecting and correcting errors. While detecting errors does not add overhead when performing memory accesses, correcting errors can delay a memory request and disrupt a system. As an example, on the systems that we examine, when an error is corrected, the CPU raises a hardware exception called a *machine check exception (MCE)* which must be handled by the processor.

When an MCE occurs, the processor stores information about the memory error in special registers that can be read by the operating system. This information includes the physical address of the memory access when the error occurred and what type of memory access (e.g., read or write) was being performed when the error occurred. Note that memory errors do not only occur in DRAM chips: memory errors can occur if the memory controller fails or if logic associated with transmitting data on a memory channel fails.

We distinguish between *errors* and *faults*. A fault refers to the underlying cause of an error, such as a DRAM cell that no longer reliably stores data. An error is the manifestation of a fault. A hard, or permanent, fault causes an error every time the fault is exercised. A soft, or transient/intermittent, fault causes an error only some of the times the fault is exercised.

C. The Systems

We examined all of the DRAM devices in Facebook's server fleet, which have operational lifetimes extending across four years and comprise billions of device days of usage. We analyzed data over a fourteen month period. We examined six different system types with hardware configurations based on the resource requirements of the workloads running on them. Table I lists the workloads and their resource requirements. The workloads perform a diverse set of operations including web serving, caching [41], database management [18], video and image processing/storage [50, 37], and messaging routing/storage. The detailed specifications for the base server platforms that we examine have been published as part of the Open Compute Project [2]. For example, typical servers can support two Intel Xeon CPUs, 16 DIMM slots, and 1 HDD [2]. The resource requirements in Table I refer to the relative number of processor cores, memory capacity, and storage capacity for servers for each type of workload.

Note that each server runs a single type of workload. All the servers configured for a particular workload type have equivalent minimum capabilities, and, in general, a workload can be run on any of them.

TABLE I: The workloads we examine and their resource requirements.

Workload	Resource requirements		
	Processor	Memory	Storage
Web	High	Low	Low
Hadoop [4]	High	Medium	High
Ingest [18]	High	High	Medium
Database [18]	Medium	High	High
Memcache [41]	Low	High	Low
Media [50]	Low	Low	High

The memory in these systems covers a wide range of devices commonly used in servers. The DIMMs are manufactured by 4 vendors in capacities ranging from 2 GB to 24 GB per DIMM. DDR3 is the protocol used to communicate with the DIMMs. The DIMM architecture spans devices with 1, 2, and 4 ranks with 8, 16, 32, and 64 chips. The chip architecture consists of 8 banks with 16 K, 32 K, and 64 K rows and 2 K to 4 K columns, and has chips that transfer both 4 and 8 bits of data per clock cycle. We analyze three different chip densities of 1 Gb, 2 Gb, and 4 Gb, which are closely related to DRAM fabrication technology.

The composition of the modules we examine differs from prior studies (e.g., [44, 16, 47, 48, 10, 45, 40]) in three ways: (1) it consists of a current DRAM access protocol (DDR3, as opposed to older generation protocols with less aggressive memory bus clock frequencies, such as DDR and DDR2 in [44]); (2) it consists of a more diverse range of DRAM device organizations (e.g., DIMMs with a variety of ranks, chips, rows, and columns versus the more homogeneous DIMMs of [44, 16, 47, 48, 10, 45]); and (3) it contains DIMMs with characteristics that have never been analyzed at a large-scale (such as density, number of chips, transfer width, and workload).

Some of the systems we examined had hardware memory scrubbing [36] enabled, which would cause the memory controller to traverse memory, detecting (but not correcting) memory errors in order to help determine faulty locations in memory. The hardware scrubber was enabled only when the machine entered a low enough idle state, so the scrubbing rate of machines varied.

D. Measurement Methodology

We use the mcelog Linux kernel module to log memory errors in a file. We do this for every machine in the fleet. For each correctable memory error, we collect: (1) a time stamp of when the error occurred; (2) the physical address that was being accessed when the error occurred; (3) the server name; (4) the socket, channel, and bank the physical address is located on; and (5) the type of memory access being performed when the error occurred (e.g., read or write). Uncorrectable errors will halt the execution of the processors on the machines we examine, causing a system crash. While we do not have detailed information on uncorrectable errors, we are able to measure their occurrence by examining a separate log of them that is kept in non-volatile memory on the system boards that we examine. We use a collector script to retrieve log data and parse it into a form that can be curated in a Hive [5] table. This process is done in real time every ten minutes.

In addition to information about the correctable errors that occur, we also collect information on systems that had errors (e.g., CPU utilization and system age; see Table II for details). This process is done in a separate step.

The scale of the systems we analyzed and the amount of data being collected posed challenges for analysis. To process billions of device days of information, we used a cluster of machines to perform a parallelized aggregation of the data using MapReduce jobs. This resulted in a set of statistics for each of the devices we analyzed. We then processed this summary data in R [3] to collect our results.

E. Analytical Methodology

When we analyze the reliability trends with respect to a system characteristic (e.g., chip density or CPU utilization), we group systems into buckets based on the particular characteristic and plot the failure rate of the systems in each bucket. When performing bucketing, we round the value of a device's characteristic to the nearest bucket and we eliminate buckets that contain less than 0.1% of the systems analyzed in order to have a statistically significant sample of systems in our measurements. We show the 95th percentile confidence interval for our data when relevant.

Due to the size of the fleet, we could not collect detailed information for all the systems *without* errors (we *do* collect detailed information for every system *with* errors). So, in Sections IV and V, instead of examining the *absolute* failure rate among different types of servers, we examine the *relative* failure rate compared to a more manageable size of servers that we call the *control group*. The servers in the control group are uniformly randomly selected from among all the servers that did not have errors, and we collected detailed information on the servers in this group.

Note that such a selection process preserves the distribution of server types in the underlying fleet, and our analysis in Sections IV and V can be considered as being performed on a "scaled down" version of the fleet. The size of the control group was chosen to be equal to the size of the error group, and the sizes of these groups were sufficiently large to be statistically significant. We bucket servers in each group based on their value for a given characteristic (e.g., age) and plot the fraction of failed servers compared to operational servers in each bucket, which we call the *relative server failure rate*. With this metric, we can perform *relative comparisons* between failure rates that we compute for different factors, but the absolute values for the metric do not have any substantial meaning. We find that the *relative* failure rates we examine are in the range [0, 1], and we plot our data within this range. In Sections IV and V, when we refer to *failure rate* we mean *relative server failure rate* and as a reminder that our data should not be confused with absolute failure rates, we label our graphs with *relative server failure rate*.

III. Baseline Statistics

We will first focus on the overall error rate and error distribution among the systems that we analyze and then examine correlations between different factors and failure rate.

A. Incidence Error Rate and Error Count

Figure 1 shows the monthly incidence error rate for memory over a span of fourteen months. The *incidence error rate* is the fraction of servers in the fleet that have memory errors compared to the total size of the fleet.[1] We observe three trends with respect to incidence error rate.

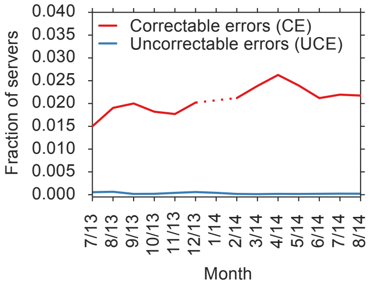

Fig. 1: Timeline of correctable and uncorrectable errors.

First, correctable errors occur relatively commonly each month, affecting 2.08% of servers on average. Though such errors do not corrupt data, they do reduce machine performance due to the hardware required to reconstruct the correct data. While a single correctable error may not be very noticeable, a large number of correctable errors could lead to performance degradation. We examine the distribution of the number of correctable errors among machines at the end of this section.

To compare against prior work, we measured the correctable error incidence rate over the course of twelve months (7/13 up to and including 7/14, excluding 1/14) and found that, cumulatively across all months, around 9.62% of servers experience correctable memory errors. This is much lower than the yearly correctable error incidence rate reported in work from the field seven years ago (32.2% in Table 1 in [44]) and comparable with the 5.48% to 9.10% failure rate reported in more recent work [48] from two years ago. Thus, though the overall correctable error incidence rate may have *decreased* over the better part of a decade of device improvements, our measurements corroborate the trend that *memory errors are still a widespread problem in the field*.

In addition, we find that the correlation between a server having a correctable error in a given month, depending on whether there were correctable errors observed in the previous month is 31.4% on average. In comparison, prior work from the field found around a 75% correlation in correctable errors between two consecutive months [44]. Our lower observed amount of correlation is partially due to how memory errors are *handled* in the servers we evaluate: servers were flagged for memory repair if they had more than 100 correctable errors per week, whereas prior work (e.g., [44]) *only* replaced components with *uncorrectable* errors. Under our more aggressive/proactive repair policy, we find that on average around 46% of servers that have errors end up being repaired each month. As a result, in contrast to prior work, we find that a majority (69.6%) of the machines that report errors each month are *not* repeat offenders from the previous month.

Second, the rate of uncorrectable errors is much smaller than the rate of correctable errors, with uncorrectable errors affecting 0.03% of servers each month on average. Recall

[1] Correctable error data for January 2014 (1/14) is not available. Note that if a server has multiple errors in multiple months, it will be represented in multiple data points.

that uncorrectable errors cause a server to crash, increasing downtime and potentially causing data loss. Therefore, it is desirable to decrease the rate of uncorrectable errors as much as possible.

Schroeder et al. conjectured that repair policies "where a DIMM is replaced once it experiences a significant number of correctable errors, rather than waiting for the first uncorrectable error" could reduce the likelihood of uncorrectable errors [44]. To test this hypothesis in the field on our systems that are repaired with more than 100 correctable errors, we compare the rate of uncorrectable errors relative to the rate of correctable errors, in order to control for the change in rate of correctable errors between the two studies. Interestingly, in Schroeder et al.'s study, uncorrectable error rate was only 25.0× smaller than the correctable error rate, while in our study it is 69.3× smaller. If more aggressive repair policies indeed lead to higher server reliability, then our results suggest that uncorrectable error rate can be lowered by up to 2.8× (i.e., 69.3× / 25.0×). This is achieved by repairing around 46% of the machines with errors (those with more than 100 correctable errors). System designers must decide whether the benefit in reduction of potential data loss is worth the increase in repair rate.

Third, the incidence error rate for correctable errors fluctuates little (its standard deviation is ±0.297%) and is relatively stable over the fourteen months that we examined. Uncorrectable errors also remain low in comparison to correctable errors (with a standard deviation of ±0.018%). We attribute the low standard deviation in error behavior over time to the large population size that we examine.

Figure 2 (left) shows the distribution of correctable errors among servers that had at least one correctable error. The x axis is the normalized device number, with devices sorted based on the number of errors they had during a month. The y axis shows the total number of errors a server had during the month in log scale. Notice that the maximum number of logged errors is in the millions. We observe that a small number of servers have a large number of errors. For example, the top 1% of servers with the most errors have over 97.8% of all observed correctable errors. We also find that the distribution of number of errors among servers is similar to that of a power-law distribution with exponent −2.964. Prior work observed that some failed devices, such as the memory controller or bus, can account for a large number of errors (e.g., [45]), though the *full distribution* of errors has *not* been quantified before.

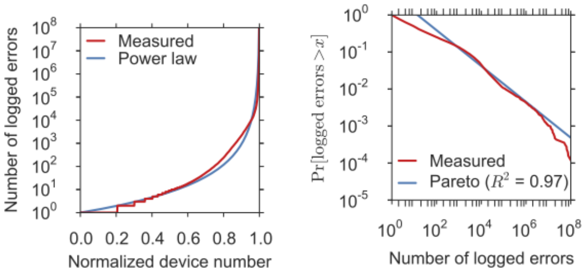

Fig. 2: The distribution of memory errors among servers with errors (left) resembles a power-law distribution. Memory errors also follow a Pareto distribution among servers with errors (right).

Figure 2 (right) shows the probability density distribution of correctable errors. The x axis is the number of errors per month and the y axis is the probability of a server having at least that many errors per month. A Pareto distribution (a special case of the power law) has been fit to the measured data. Similarly to many past works that have found decreasing hazard rates in the behavior of systems (e.g., Unix process lifetimes [14], sizes of files transferred through the Web [8, 9], sizes of files stored in Unix file systems [17], durations of FTP transfers in the Internet [42], CPU requirements for supercomputing jobs [43], and memory access latencies [22]), we find that the distribution of errors across servers follows a Pareto distribution, with a decreasing hazard rate. This means, roughly, that the more errors a server has had so far, the more errors it is expected to have.[2]

Quantifying the skewed distribution of correctable errors is important as it can help diagnose the severity of a memory failure relative to the population. For comparison, Schroeder et al. reported a mean error rate of 22,696 correctable errors per server per year (Table 1 in [44]), or 1,891 correctable errors per server per month. Without knowing the underlying distribution, however, it is not clear whether *all* servers had such a large number of errors each month or whether this average is dominated by a small number of outliers (as we observe here).

If we compute the mean error rate as in prior work, we observe 497 correctable errors per server per month. However, if we examine the error rate for the *majority* of servers (by taking the median errors per server per month), we find that most servers have at most 9 correctable errors per server per month.[3] In this case, using the mean value to estimate the value for the majority overestimates by over 55×. We therefore conclude that, for memory devices, the skewed nature in which errors are distributed among devices call for the full distribution to be examined. Doing so reveals that memory errors follow a power-law distribution, which can be used to accurately assess the severity of machine failures. Therefore, we hope future studies that use error data from the field take into account the new distribution we observe and openly provide.

In addition, we found that hardware scrubbing detected 13.1% of the total number of errors. While we did not monitor how many servers employed scrubbing, we observed that 67.6% of the servers with errors detected at least one error through scrubbing. We do not have detailed memory access information, so the interaction between scrubbing and different workloads is not clear, and requires further examination.

B. *Component Failure Analysis*

Memory errors can occur due to failures in a DRAM device as well as if the memory controller in the processor fails or if logic associated with transmitting data on a memory channel fails. While prior work examined DRAM chip-level failures ([16, 47, 48]) and memory controller/channel failures ([45]) *separately*, no prior work has comprehensively examined failures across the *entire memory system*.

We adopted a methodology for classifying component failures similar to prior work (e.g., [16, 47, 48, 45]). We examined all of the correctable errors across the fleet each month. We began by determining each correctable error's corresponding processor socket, memory channel, bank, row, column, and byte offset. Then, we grouped errors based on the component that failed and caused the error to occur. For grouping errors by components, we used the following criteria:

Socket. If there were > 1 K errors across > 1 memory channel connected to the same processor socket, we classified those errors as being caused by a socket failure. The > 1 K error threshold was chosen so as to ensure that the failures we classify are not due to a small number of independent cell failures. To make sure this was the case, we cross-referenced repair logs of the servers classified with failed sockets and found that 50% of them had a large number of errors that required replacing the

[2] Note that one can take advantage of this property to potentially predict which servers may have errors in the future. We leave this for future work. For more information on the Pareto distribution, decreasing hazard rate, and their properties, we refer the reader to [22, 13].

[3] Concurrent work by Sridharan et al. [46] makes a similar observation, though we quantify and provide a model for the *full distribution* of errors per server.

processor to eliminate the errors and 50% contained intermittent bursts of errors that caused the server to become unresponsive for long periods of time – both of these are characteristics of failed sockets that can generate a large number of machine check exceptions, as observed in prior work [45].

Channel. After excluding the above errors, if there were > 1 K errors across > 1 DRAM banks connected to the same memory channel, we classified the channel as having failed. Similar to sockets, we cross-referenced repair logs for servers classified with failed channels and found that 60% of the servers with failed channels did not have any logged repair action (replacing or reseating the DIMM), suggesting that these failures were transient, potentially caused by temporary misalignment of the transmission signal on the channel. The other 40% of servers required DIMMs to be replaced, suggesting permanent failures related to the channel transmission logic (e.g., the I/O circuitry) within the DIMM.

Bank. After excluding the above errors, we repeated the procedure for banks, classifying a bank as having failed if it had > 1 K errors across > 1 row. Note that our study examines monthly failure trends, and we assume that multiple row failures in the same bank in the same month may be more indicative of a bank failure than multiple independent row failures in the bank.

Row. After excluding the above errors, we classified a row as having failed if > 1 column in the same row had errors.

Column. After excluding the above errors, we classified a column as having failed if > 1 error occurred in the same column.

Cell. After excluding the above errors, we classified a cell as having failed if > 1 error occurred in the same byte within 60 seconds. We chose this amount of time because we found that 98.9% of errors at a particular byte address had another error at the same address within 60 seconds if they ever had an error at the same byte address again in the same day.

Spurious. After excluding the above errors, we are left with what we term *spurious* errors. These errors are isolated to individual cells that do not share a common failed component and do not repeat in a short amount of time. Potential causes of these errors include alpha particle strikes from the atmosphere or chip packaging [34] and cells with weak or variable charge retention times [30, 21, 20].

Figure 3 shows the fraction of logged errors each month that are attributed to different types of failures. Error bars show the standard deviation between months.

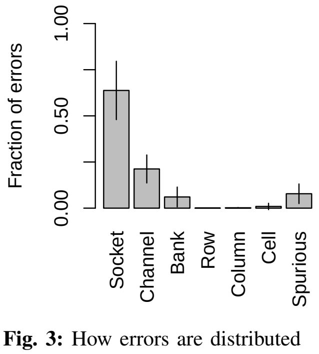

Fig. 3: How errors are distributed among different memory components. Error bars signify the variation in total errors from month to month.

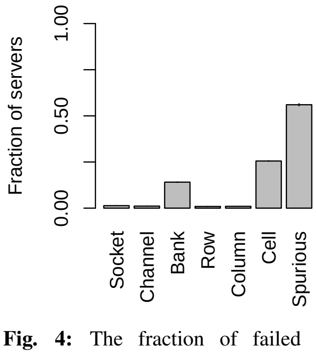

Fig. 4: The fraction of failed servers that had each type of memory component failure.

Sockets and channels generate the most errors when they fail, 63.8% and 21.2% of all errors each month, respectively. This is because when these components fail, they affect a large amount of memory. Compared to a prior work that examined socket (memory controller) and channel failures [45] (but did not examine DRAM chip-level failures), we find that our systems have 2.9× more socket errors and 5.3× more channel errors. This could be due to differences in the server access patterns to memory or how quickly servers crash when experiencing these types of failures.

That sockets and channels cause a large number of errors when they fail helps explain the skew in the distribution of errors among servers (Figure 2, left). For example, servers with socket failures had the highest number of errors in the distribution. This large source of errors, if not controlled for, can confound memory reliability conclusions by artificially inflating the error rates for memory and creating the appearance of more DRAM chip-level failures than in reality. Besides the work that only measured socket and channel failures, but *not* DRAM chip-level failures ([45]), we did not find mention of *controlling for socket and channel errors* in prior work that examined errors in the field (e.g., [44, 16, 47, 48, 10, 27, 28]).

We observe that DRAM chip-level (banks, rows, columns, cells, and spurious) failures contribute a relatively small number of errors compared to sockets and channels: 6.06%, 0.02%, 0.20%, 0.93%, and 7.80%, respectively. This is because when these components fail, they affect only a relatively small amount of memory. Based on these findings, to help with the diagnosis of memory failures, we recommend that memory error classification *should always* include components such as sockets and channels.

So far, we have examined how component failures are related to the number of *errors* reported. We next turn to how component failures *themselves* (the underlying source of errors) are distributed among servers. Figure 4 shows what fraction of servers with correctable errors each month have each type of failure that we examine. We plot error bars for the standard deviation in fraction of servers that report each type of error between months, though we find that the trends are remarkably stable, and the standard deviation is correspondingly very low (barely visible in Figure 4).

Notice that though socket and channel failures account for a large fraction of errors (Figure 3), they occur on only a small fraction of servers with errors each month: 1.34% and 1.10%, respectively (Figure 4). This helps explain why servers that have socket failures often appear unresponsive in the repair logs that we examined. Socket failures bombard a server with a large flood of MCEs that must be handled by the operating system, essentially creating a kind of *denial of service attack* on the server. Systems that have these type of failures have been observed to appear unresponsive for minutes at a time while correcting errors and handling MCEs. We believe that context switching to the operating system kernel to handle the MCE contributes largely to the unresponsiveness.

Thus, memory errors are *not* always isolated events, and correcting errors in hardware and handling MCEs in the system software (as current architectures do) can easily cause a machine to become unresponsive. We suspect that simple hardware changes such as caching error events and having system software poll the contents of the error cache once in a while, instead of *always* invoking the system software on *each* error detection, could greatly reduce the potential availability impact of socket and channel failures. In addition, the DDR4 standard [19] will allow memory accesses to be *retried* by the memory controller (by using a cyclic redundancy check on the command/address bits and asserting an "alert" signal when an error is detected) *without interrupting the operating system*, which can help reduce the system-level unavailability resulting from socket and channel failures.

Bank failures occur relatively frequently, on 14.08% of servers with errors each month. We observe a larger failure rate for banks than prior work that examined DRAM chip-level failures on Google servers, which found 2.02% of banks failed over the course of their study (Table 2 in [16][4]). One reason for this difference could be the different composition of the servers evaluated. For example, while the prior work examined older DDR and DDR2 DIMMs from over five years ago, we examine newer DIMMs that use the DDR3 protocol. The relatively large occurrence of bank failures suggests that devices that support single chip failures (e.g., Chipkill [11]) can provide additional protection to help ensure that such failures do not lead to uncorrectable errors.

We find that row and column failures are relatively infrequent, occurring in 0.92% and 0.99% of servers each month. Prior work on Google servers found much larger rate of row (7.4%) and column (14.5%) failures [16]. We believe that the much larger estimate in prior work could potentially be due to the confounding effects of socket and channel errors. Such errors, if present and unaccounted for, can artificially increase the number of row and column errors (e.g., the socket and channel errors in Figure 3 may end up being misidentified as other types of errors).

We observe that a relatively large fraction of servers experience cell failures, 25.54%. Similar to row and column failures, prior work found a much larger amount of cell failures, 46.1%. As with rows an columns, this could also potentially be due to unaccounted-for socket and channel failures increasing the perceived number of cell failures. The prevalence of this type of failure prompted the prior work to examine the effectiveness of page offlining, where the operating system (OS) removes pages that contain failed cells from the physical address space. While the prior study evaluated page offlining in simulation using the same memory traces from their evaluation, we evaluate page offlining on a fraction (12,276) of the servers we examine in Section VI and find it to be less effective than reported in prior work ([16]).

While prior work, which may not have controlled for socket and channel failures, found repeat cell errors to be the dominant type of failure (e.g., [16, 48, 10]); when controlling for socket and channel failures (by identifying and separately accounting for the errors associated with them), we find *spurious failures* occur the most frequently, across 56.03% of servers with errors. Such errors can be caused by random DRAM-external events such as alpha particle strikes from the atmosphere or chip packaging [34] and DRAM-internal effects such as cells with weak or variable charge retention times [30, 21, 20]. This is significant because, as we mentioned before and as we will show in Section VI, spurious failures can limit the effectiveness of the page-offlining technique. To deal with these type of failures, more effective techniques for detecting and reducing the reliability impact of weak cells are required (some potential options are discussed in [21, 20]).

IV. The Role of System Factors

We next examine how various system factors are correlated with the occurrence of failures in the systems we examine. For this detailed analysis, we examine systems that failed over a span of three months from 7/13 to 9/13. We focus on understanding DRAM failures and excluded systems with socket and channel failures from our study. We examine the effects of DRAM density and DIMM capacity, DIMM vendor, DIMM architecture, age, and workload characteristics on failure rate.

A. DIMM Capacity and DRAM Density

DRAM density is measured in the number of bits per chip and is closely related to the DRAM cell technology and manufacturing process technology [20]. As DRAM cell and fabrication technology improves, devices with higher densities can be manufactured. The most widely-available chip density currently is 4 Gb as of 2014, with 8 Gb chips gaining adoption.

DRAM density is different from DIMM *capacity*. A DIMM of a certain capacity could be composed *in multiple ways* depending on the density and transfer width of its chips. For example, a 4 GB capacity DIMM could have 16 × 2 Gb chips or 8 × 4 Gb chips. Prior work examined DIMM *capacity* when drawing conclusions [44, 40], and observed trends that were, in the authors' own words, either "not consistent" [44] or a "weak correlation" [40] with error rate. This led the prominent Schroeder et al. work to conclude that "unlike commonly feared, we don't observe any indication that newer generations of DIMMs have worse error behavior." Our results with DRAM density stand to refute this claim as we explain below.

Similar to these works, we also find that the error trends with respect to *DIMM capacity* are *not consistent*. Figure 5 shows how the different capacities of DIMMs we examine are related to device failure rate.[5] The large error bars for 16 GB and 24 GB DIMMs are due to the relatively small number of DIMMs of those types. Notice that there is no consistent trend across DIMM capacities.

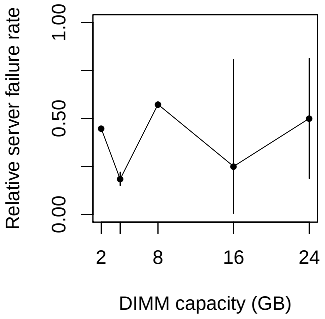

Fig. 5: The relative failure rate for servers with different DIMM capacities. Similar to prior work, we find no consistent reliability trend.

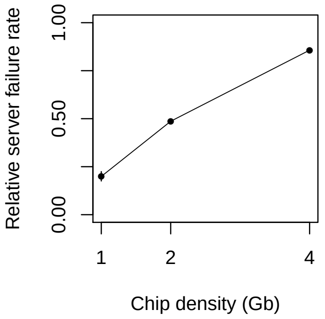

Fig. 6: The relative failure rate for servers with different chip densities. Newer densities (related to newer technology nodes) show a trend of higher failure rates.

In contrast to prior works [44, 40], we *do* observe indication that *newer generations of DRAM chips have worse error behavior* by examining failure rate as a function of DRAM chip density. The servers we analyzed contained three different types of DRAM chip densities: 1 Gb, 2 Gb, and 4 Gb. Figure 6 shows *how different DRAM chip densities are related to device failure rate*. We can see that there is a clear trend of increasing failure rate with increasing chip density, with 2 Gb devices having 2.4× higher failure rates than 1 Gb devices and 4 Gb devices having 1.8× higher failure rates than 2 Gb devices. This is troubling because it indicates that business-as-usual practices in DRAM design will likely lead to increased memory failure rates in the future, as predicted by both industry [20] and academia [21, 23, 38, 39] in recent works. To understand the source of this trend, we next examine the failure rate for DRAM cells.

Figure 7 shows the cell failure rate computed by normalizing the failure rates in Figure 6 by the number of cells in each chip. Interestingly, cell failure rate had a brief increase going from 1 Gb chips to 2 Gb chips but a recent decrease going from 2 Gb chips to 4 Gb chips. This shows that the reliability of *individual DRAM cells* may be improving recently. This is

[4] Other studies (e.g., [47, 48]) have similar findings. For brevity, we compare against [16].

[5] Recall from Section II-E that we examine *relative* server failure rates compared to the sampled control group. Though relative failure rates happen to be in the range $[0, 1]$, they should not be confused with absolute failure rates across the fleet.

likely due to the large amounts of effort that DRAM manufacturers put into designing faster and more reliable DRAM cell architectures. Our insight is that *small improvements in DRAM cell reliability are easily outpaced by the quadratic increase in number of cells per chip*, leading to the trend of net decrease in DRAM reliability as shown by the server failure rate data in Figure 6. Unless more-than–quadratic improvements in DRAM cell reliability are achieved in future devices, maintaining or decreasing DRAM server failure rates in the future (while still increasing DRAM chip capacity) will be untenable without stronger hardware and/or software error correction.

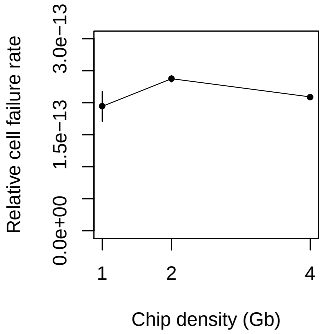

Fig. 7: The relative per-cell failure rate at different technology nodes (chip densities).

B. DIMM Vendor

DIMM vendors purchase chips from DRAM chip manufacturers and assemble them into DIMMs. While we have information on DIMM manufacturer, we do not have information on the DRAM chip manufacturers in our systems.

Figure 8 shows the failure rate for servers with different DIMM vendors.[6] We observe that failure rate varies by over 2× between vendors (e.g., Vendor B and Vendor C). The differences between vendors can arise if vendors use less reliable chips from a particular foundry or build DIMMs with less reliable organization and manufacturing. Prior work [48, 10] also found a large range in the server failure rate among vendors of 3.9×.

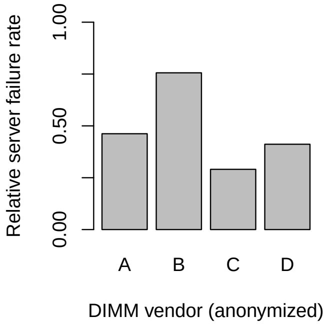

Fig. 8: Relative server failure rate for different vendors varies widely.

C. DIMM Architecture

We next examine how DIMM architecture affects server failure rate. We examine two aspects of DIMM design that have not been studied in published literature before: the number of data chips (not including chips for ECC) per DIMM and the transfer width of each chip.

Figure 9 plots the failure rate for servers with DIMMs with different numbers of data chips for each of the densities that we examine. The DIMMs that we examine have 8, 16, 32, and 48 chips. We make two observations from Figure 9.

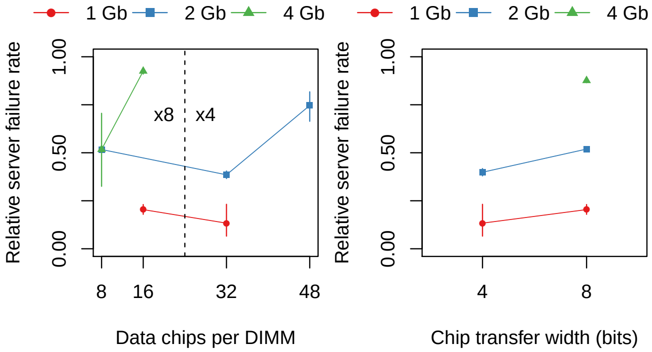

Fig. 9: The relative failure rate of servers with DIMMs with different numbers of data chips separated by chip density.

Fig. 10: The relative failure rate of servers with DIMMs with different chip transfer widths separated by chip density.

First, for a given number of chips per DIMM, servers with higher chip densities generally have higher average failure rates. This illustrates how chip density is a first-order effect when considering memory failure rate (as we showed in Figure 6).

Second, we find that server failure rate trends with respect to chips per DIMM are dependent on the *transfer width* of the chips – the number of data bits each chip can transfer in one clock cycle. In order to transfer data at a similar rate, DIMMs with fewer (8 or 16) chips must compensate by using a larger transfer width of 8 bits per clock cycle (and are called ×8 devices) while DIMMs with more chips (32 or 48) can use a smaller transfer width of 4 bits per clock cycle (and are called ×4 devices). We have annotated the graph to show which chip counts have transfer widths of ×4 bits and ×8 bits.

We observe two trends depending on whether chips on a DIMM have the same or different transfer widths. First, among chips of the *same* transfer width, we find that increasing the number of chips per DIMM increases server failure rate. For example, for 4 Gb devices, increasing the number of chips from 8 to 16 increases failure rate by 40.8% while for 2 Gb devices, increasing the number of chips from 32 to 48 increases failure rate by 36.1%. Second, once the number of chips per DIMM increases beyond 16 and chips start using a *different* transfer width of ×8, there is a decrease in failure rate. For example, for 1 Gb devices, going from 16 chips with a ×8 interface to 32 chips with a ×4 interface decreases failure rate by 7.1%. For 2 Gb devices, going from 8 chips with a ×8 interface to 32 chips with a ×4 interface decreases failure rate by 13.2%.

To confirm the trend related to transfer width, we plotted the failure rates dependent on transfer width alone in Figure 10. We find that, in addition to the first-order effect of chip density increasing failure rate (Effect 1), there is a consistent increase in failure rate going from ×4 to ×8 devices (Effect 2).

We believe that both effects may be partially explained by considering how number of chips and transfer width contribute to the electrical disturbance within a DIMM that may disrupt the integrity of the signal between components. For example, a larger transfer width increases internal data transfer current (e.g., $I_{DD4R/W}$ in Table 19 of [35], which compares the power consumption of ×4 and ×8 DRAM devices), leading to additional power noise across the device. Such power noise could induce additional memory errors if, for example, charge were to get trapped in components. Interestingly, we find that, for a given chip density, *the best architecture for device reliability occurs when there is, first, low transfer width and, second, low chips per DIMM*. This is shown by the 2 Gb devices with 32 chips with a ×4 interface compared to the other 2 Gb devices in Figure 9.

[6] We have made the vendors anonymous.

D. Workload Characteristics

We next examine how workload-related characteristics such as CPU utilization (the average utilization of the CPUs in a system), memory utilization (the fraction of physical memory pages in use), and workload type affect server failure rate. Prior work examined CPU utilization and memory utilization and found that they were correlated positively with failure rate [44].

We measure CPU utilization as the fraction of non-idle cycles versus total cycles across the CPUs in a server. Due to software load balancing, we find that CPU utilization among cores in a server running the same workload are relatively similar, and so the average utilization across the cores is reasonably representative of each core's individual utilization. We measure memory utilization as the fraction of pages allocated by the OS. Note that memory utilization does not describe *how* the cells in pages are accessed. For this reason, we examine workloads as a proxy for how cells are accessed. We plot how CPU utilization and memory utilization are related to server failure rate in Figures 11 and 12.

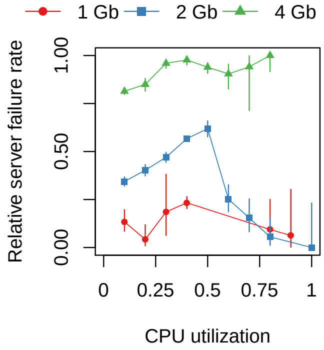

Fig. 11: The relative failure rate of servers with different average CPU utilizations.

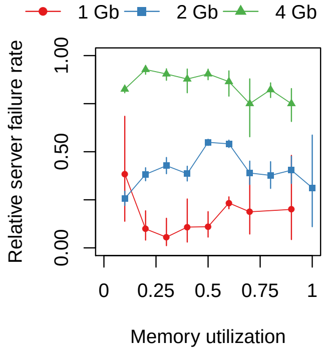

Fig. 12: The relative failure rate of servers with different average memory utilizations.

Contrary to what was observed in prior work, we do *not* find a correlation between either CPU utilization or memory utilization and failure rate. We observe multiple local maxima for failure rate compared to CPU utilization and memory utilization across all the chip densities. We believe that this is due to the more diverse workloads that we examine (Table I) compared to prior work [44, 47, 48, 10], which mainly examined a homogeneous workload. The implications of this are that memory failure rate may depend more on the *type* of work as opposed to the CPU or memory utilization the work causes.

To examine how the *type of work* a server performs affects failure rate, we plotted the server failure rate for the different workload types at Facebook in Figure 13. We observe that, depending on the workload, failure rate can vary by up to 6.5×, as shown by the difference between servers executing a Database-type workload compared to those executing a Hadoop-type workload. While we leave the detailed examination of how workloads affect memory failure rate to future work, we hypothesize that certain types of workload memory access patterns may increase the likelihood of errors. For example, prior work has shown that memory errors can be induced in a controlled environment by accessing the same memory row in rapid succession [23]. Such an access pattern involves modifying data and writing it back to memory using the `clflush` and `mfence` instructions. We believe it would be interesting to examine what types of workloads exhibit this behavior.

E. Server Age

We examine next how age affects server failure rate. The servers we analyzed were between one and four years old, with an average age of between one and two years. Figure 14 shows the monthly failure rate for servers of different ages. We observe that chip density once again plays a large role in determining server failure rate: For a given age, servers with 4 Gb devices have a 15.3% higher failure rate on average than 2 Gb devices, and servers with 2 Gb devices have a 23.9% higher failure rate on average than 1 Gb devices.

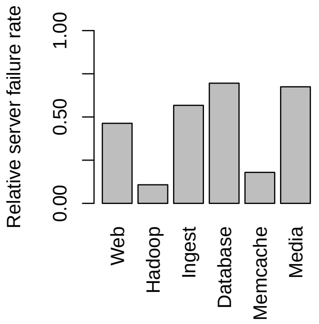

Fig. 13: The relative failure rate of servers that run different types of workloads (Table I) can vary widely.

We do not observe any general *age-dependent* trend in server failure rate when controlling for the effects of density alone. One reason for this is that age is correlated with other server characteristics. For example, we find that in addition to being correlated with chip density (correlation coefficient of −0.69), age is also correlated with the number of CPUs in a system (correlation coefficient of −0.72). Figure 15 shows the trend for age for different combinations of chip density and CPUs (which we will denote as $\langle x, y \rangle$ where x is chip density in Gb and y is number of CPUs). We make two observations from Figure 15.

First, we find that among systems of *the same age*, more cores lead to higher failure rates. For example, consider the $\langle 2, * \rangle$ systems that are two years of age: going from 4 → 12 cores increases failure rate by 21.0% and going from 12 → 16 cores increases failure rate by 22.2%. Figure 16, which plots the server failure rate with respect to different numbers of CPUs confirms this trend, with 2 Gb systems with 16 cores having a 40.0% higher failure rate than 2 Gb systems with 4 cores. This could potentially be due to more cores accessing DRAM more intensely and wearing out DRAM cells at a faster rate, a failure mode that was shown in a prior controlled study [6].

The most related trend observed in prior work was that CPU frequency was shown to be correlated with error rate [40]. The trend we observe with respect to CPU count is significant because the number of CPU cores per processor is increasing at a much faster rate than CPU frequency and so our results allow us to predict that future processor generations will likely continue to induce higher rates of errors in DRAM devices.

Second, among systems with *the same number of cores*, older machines generally have higher failure rates than younger machines. For example, for the $\langle 2, 12 \rangle$ system, average failure rate increases by 2.8% going from 2 → 3 years of age, and average failure rate increases by 7.8% going from 3 → 4 years of age. This is consistent with prior observations from the field that showed that failure rates can increase with age [44], though we observe a much clearer trend compared to prior work (e.g., Figure 10 in [44] shows large fluctuations in failure rate over time) because we have controlled for correlated factors such as chip density and CPU count. Not all systems exhibit this trend, however: the $\langle 1, 12 \rangle$ system shows a small decrease in failure rate going from 3 → 4 years of age, which could be due to second-order effects on failure rate from other factors that may be correlated with age, such as transfer width.

V. Modeling DRAM Failures

We next develop a model for DRAM failures using the data collected from our study. We use a statistical regression analysis

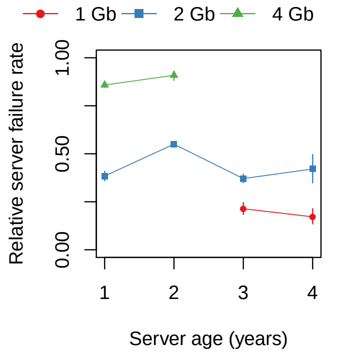

Fig. 14: The relative failure rate of servers of different ages. There is no clear trend when controlling *only* for chip density.

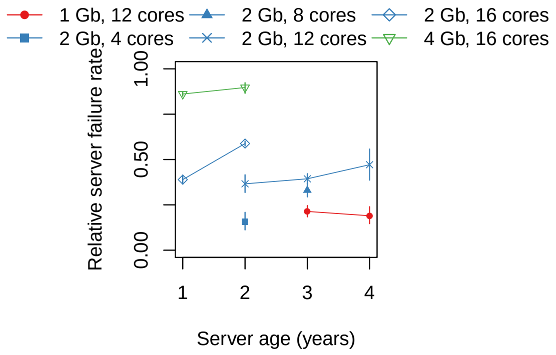

Fig. 15: The relative failure rate of servers of different ⟨chip density, CPU count⟩ configurations. When controlling for density and CPUs together, older devices usually have higher failure rates.

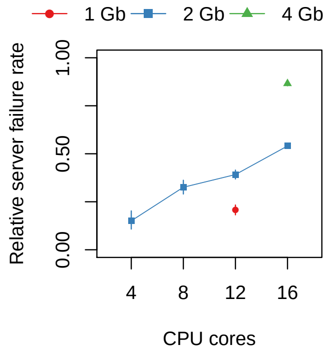

Fig. 16: The relative failure rate of servers with different numbers of CPU cores. Servers with more CPUs have higher failure rates.

TABLE II: The factors in our regression analysis and the resulting error model. p-value is the likelihood that a characteristic is inaccurately modeled: lower p-values indicate more accurate modeling. Significant? is whether the p-value is $< \mathbf{0.01}$, corresponding a $< 1\%$ chance that the characteristic is inaccurately modeled. β-coefficient is the characteristic's contribution to error rate and standard error is how much the model differs from the measured values for a given characteristic. The model is publicly available at [1].

Characteristic	Description	p-value	Significant?	β-coefficient	Standard error
Intercept	A baseline server with 1 Gb chips with a ×4 interface and 0 for all other factors.	$<2.000 \times 10^{-16}$	Yes	-5.511	3.011×10^{-1}
Capacity	DIMM capacity (GB).	$<2.000 \times 10^{-16}$	Yes	9.012×10^{-2}	2.168×10^{-2}
Density2Gb	1 if the server has 2 Gb density chips; 0 otherwise.	$<2.000 \times 10^{-16}$	Yes	1.018	1.039×10^{-1}
Density4Gb	1 if the server has 4 Gb density chips; 0 otherwise.	$<2.000 \times 10^{-16}$	Yes	2.585	1.907×10^{-1}
Chips	Number of chips per DIMM.	$<2.000 \times 10^{-16}$	Yes	-4.035×10^{-2}	1.294×10^{-2}
Width8	1 if the server has ×8 DRAM chips; 0 otherwise.	0.071	No	2.310×10^{-1}	1.277×10^{-1}
CPU%	Average CPU utilization (%).	$<2.000 \times 10^{-16}$	Yes	1.731×10^{-2}	1.633×10^{-3}
Memory%	Average fraction of allocated physical pages (%).	0.962	No	5.905×10^{-5}	1.224×10^{-3}
Age	Server age (years).	$<2.000 \times 10^{-16}$	Yes	2.296×10^{-1}	3.956×10^{-2}
CPUs	Number of physical CPU cores in the server.	$<2.000 \times 10^{-16}$	Yes	2.126×10^{-1}	1.449×10^{-2}
Failure model	$\ln[\mathcal{F}/(1-\mathcal{F})] = \beta_{Intercept} + (Capacity \cdot \beta_{Capacity}) + (Density2Gb \cdot \beta_{Density2Gb}) + (Density4Gb \cdot \beta_{Density4Gb}) + (Chips \cdot \beta_{Chips}) + (CPU\% \cdot \beta_{CPU\%}) + (Age \cdot \beta_{Age}) + (CPUs \cdot \beta_{CPUs})$				

to determine which server characteristics have a statistically significant effect on failure rate and how much they contribute to failure rate. The resulting model can then be used to examine how relative server failure rate changes for servers with different characteristics, which can be used to reason about the relative reliability of different server configurations.

We used R [3] for our statistical analysis. We performed a logistic regression [15, 32] on a binary characteristic that represented whether a server was part of the error group or control group of servers (see Section II-E for our error and control group classification/formation). We include most of the characteristics we analyzed in Section IV in our regression with the exception of DIMM vendor because it is anonymized and workload type because it is difficult to apply outside the context of our fleet.[7] One limitation of the logistic regression model is that it is able to identify only *linear relationships* between characteristics and failure rates. On the other hand, using a logistic regression made analyzing our large data set of errors across many variables tractable.

Table II shows the parameters and output of the regression and the resulting model (in the last row). The first two columns describe the factors included in the regression. The third column lists the resulting p-value for each factor after performing the logistic regression. The p-value is the likelihood that a characteristic is inaccurately modeled: lower p-values indicate more accurate modeling. The fourth column describes whether the p-value is < 0.01, corresponding to a $< 1\%$ chance that the characteristic is inaccurately modeled. The fifth column, β-coefficient, is the characteristic's contribution to error rate and the last column, standard error, is how much the model differs from the measured values for a given characteristic.

The *Intercept* is a byproduct of the regression and helps the model fit the measured data better. It represents a server with a certain set of baseline characteristics (listed in Table II) and 0 for all other factors (0 CPUs, 0 years old, and so on). The factors *Density2Gb* and *Density4Gb* take on the value 0 or 1 depending on whether the server has the characteristic (in which case the value is 1) or does not (0). Note that the regression analysis computes the β-coefficients for these variables in such a way that when they are added to the model, they effectively replace the default values represented in $\beta_{Intercept}$ (e.g., though $\beta_{Intercept}$ represents a server with 1 Gb chips, when *Density2Gb* is set to 1, the model computes the failure rate of servers with 2 Gb chips).

Note that a characteristic that is included in the model does not necessarily mean that it affects failure rate in the real world. It may mean that it is only correlated with other characteristics that *do* affect failure rate. The opposite is true as well: A characteristic that is *not* included in the model may in fact contribute to failure rate but its effects are captured by other characteristics present in the model. For example, Figure 17 shows a heatmap representing the correlation between the different measured factors: darker colors correspond to a stronger correlation while lighter colors correspond to a weaker correlation. While some factors that are independent of one another have weak or no correlation (i.e., close to 0, such as CPUs and Chips), others show strong correlations (i.e., more/less than ±0.8, such as Capacity and Chips). We have discussed and attempted to control for these correlations in Section IV.

[7] This results in these contributions to the model being expressed *indirectly* though other factors, whose values will be computed, in part, based on how they are correlated with different vendors/workloads.

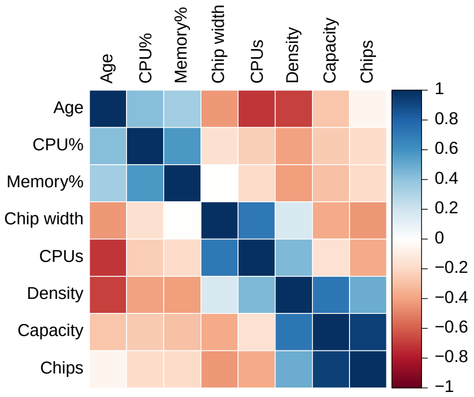

Fig. 17: The correlation between different measured factors.

Using the equation in Table II, we can solve for $\mathcal{F}$, the rate of memory failure for a server with a given set of characteristics. For example, Table III compares the failure rates predicted by the model for four different server types: (1) a *low-end server* with low density DIMMs and few CPUs, (2) a *high-end (HE) server* with high density DIMMs and twice as many CPUs as the low-end server, (3) a high-end server that uses *lower-density DIMMs (HE/↓density)*, and (4) a high-end server that uses *half as many CPUs (HE/↓CPUs)*. So that the workload is kept roughly similar across the configurations, we double the CPU utilization for servers with half as many CPUs.

TABLE III: Predicted relative failure rates among different server types.

Factor	Low-end	High-end (HE)	HE/↓density	HE/↓CPUs
Capacity	4 GB	16 GB	4 GB	16 GB
Density2Gb	1	0	1	0
Density4Gb	0	1	0	1
Chips	16	32	16	32
CPU%	50%	25%	25%	50%
Age	1	1	1	1
CPUs	8	16	16	8
Predicted relative failure rate	**0.12**	**0.78**	**0.33**	**0.51**

We can see that the model-predicted failure rate of the high-end server is 6.5× that of the low-end server. This agrees with the trends that we observed in Section IV, which showed, for example, increasing failure rates with increasing chip density and number of CPUs. Interestingly, the model can be used to provide insight into the relative change in error rate for *different system design choices*. For example, the failure rate of the high-end server can be reduced by 57.7% by using lower density DIMMs and by 34.6% by using half as many cores. This indicates that designing systems with lower density DIMMs can provide a larger DRAM reliability benefit than designing systems with fewer CPUs. In this way, the model that we develop allows system architects to easily and quickly explore a large design space for memory reliability. We hope that, by using the model, system architects can evaluate the reliability trade-offs of their own system configurations in order to achieve better memory reliability. We make the model also available online at [1].

VI. Effect of Page Offlining at Scale

We next discuss the results of a study performed to examine ways to reduce memory errors using page offlining [49, 16]. Page offlining removes a physical page of memory that contains a memory error from the set of physical pages of memory that the operating system can allocate. This reduces the chance of a more severe uncorrectable error occurring on that page compared to leaving the faulty page in the physical address space. While prior work evaluated page offlining using simulations on memory traces [16], we deployed page offlining on a fraction of the machines we examined (12,276 servers) and observed the results. We next describe the system design decisions required to make page-offlining work well at a large scale, and analyze its effectiveness.

A. Design Decisions and Implementation

The three main design decisions we explore with respect to utilizing page offlining in practice are: (1) *when* to take a page offline, (2) *for how long* to take a page offline, and (3) *how many* pages to take offline (the first and last of which were also identified in [16]).

(1) When to take a page offline? ECC DIMMs provide flexibility for tolerating correctable errors for a certain amount of time. In some settings, it may make sense to wait until a certain number of memory errors occur on a page in a certain amount of time before taking the page offline. We examine a conservative approach and take any page that had a memory error offline immediately (the same as the most aggressive policy examined in prior work [16]). The rationale is that leaving a page with an error in use increases the risk of an uncorrectable error occurring on that page. Another option could be to leave pages with errors in use for longer and, for example, design applications that are not as adversely affected by memory errors. Such an approach is taken by Flikker [31], which developed a programming model for reasoning about the reliability of data, and by heterogeneous-reliability memory systems where parts of memory can be less reliable and application data that is less vulnerable to errors can be allocated there [33].

(2) For how long to take a page offline? One question that arose when designing page offlining at a large scale was how to make an offlined page persist across machine reboots (both planned and unplanned) and hardware changes (e.g., disk replacement). Neither of these cases are handled by existing techniques. Allowing an offlined page with a permanent error to come back online can defeat the purpose of page offlining by increasing the window of vulnerability for uncorrectable errors. We examine a policy that takes pages offline *permanently*. To keep track of offlined pages across machine reboots, we store offlined pages by host name in a distributed database that is queried when the OS kernel loads. This allows known-bad pages to be taken offline before the kernel allocates them to applications. Entries in this database need to be updated as DRAM parts are replaced in a system.

(3) How many pages to take offline? Taking a page offline reduces the size of physical memory in a system and could cause increased swapping of pages to storage. To limit the negative performance impact of this, we place a cap on the number of physical pages that may be taken offline. Unlike prior work, as we showed in Section III-B, socket and channel failures can potentially cause page offlining to remove large portions of the physical address space, potentially causing large amounts of swapping to storage and degrading performance. To check how many pages have been taken offline, logs are routinely inspected on each machine. When the amount of physical memory taken offline is greater than 5% of a server's physical memory capacity, a repair ticket is generated for the server.

B. Effectiveness

Figure 18 shows a timeline of the normalized number of errors in the 12,276 servers that we examine (unlike the rest of this study, we only examine a small number of servers for this technique). The experiment was performed for 86 days and we measure the number of errors as a moving average over 30 days. As it was a production environment, page offlining was deployed on all of the machines over the course of several days. We divide the graph into three regions based on the different phases of our experiment. Region a shows the state of the servers before page offlining was deployed. Region b shows the state of the servers while page offlining was deployed gradually to 100% of the servers (so that any malfunctions of the deployment could be detected in a small number of machines and not all of them). Region c shows the state of the servers after page offlining was fully deployed.

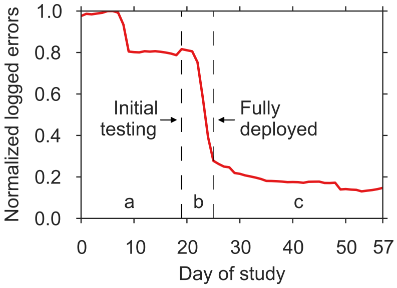

Fig. 18: The effect of page offlining on error rate.

The initial hump in Region a from days 0 to 7 was due to a bank failure on one server that generated a large number of errors. By day 8 its effects were no longer noticeable in the moving average and we compare the effectiveness of page offlining to the error rate after day 8.

There are three things to note from Figure 18. First, after deploying page offlining to 100% of the fleet at day 25, error rate continues to decrease until day 50. We believe that this is because some pages that contained errors, but which were not accessed immediately after deploying page offlining, were eventually accessed, triggering an error and taking the page offline. In addition, some pages cannot be taken offline immediately due to restrictions in the OS, which we will describe in the next section. Second, comparing the error rate at day 18 (right before initial testing) to the error rate at day 50 (after deploying page offlining and letting the servers run for a couple of weeks), the error rate decreased by around 67%. This is smaller than the 86% to 94% error rate reduction reported in Hwang et al.'s study [16]. One reason for this could be that the prior study may have included socket and channel errors in their simulation – increasing the number of errors that could be avoided due to page offlining. Third, we observe a relatively large rate of error occurrence (e.g., at day 57 the error rate is still around 18% of the maximum amount), even after page offlining. This suggests that it is important to design devices and other techniques that help reduce the error rate that does not seem to be affected by aggressive page offlining.

C. Limitations

While page offlining is relatively effective at reducing the number of reported errors, we find that it has two main limitations that were not addressed in prior work. First, it reduces memory capacity, which requires repairing a machine after a certain fraction of its pages have been taken offline. Second, it may not always succeed in real systems. We additionally logged the failure rate of page offlining and found that around 6% of the attempts to offline a page initially failed. One example we found of why a page may fail to be offlined in the Linux kernel is if its contents cannot be locked for exclusive access. For example, if its data is being prefetched into the page cache at the time when it is to be offlined, locking the page could result in a deadlock, and so the Linux kernel does *not* allow this. This, however, could be easily fixed by retrying page-offlining at a later time, at the expense of added complexity to system software.

Despite these limitations, however, we find that page offlining – when adapted to function at scale – provides reasonable memory error tolerance benefits, as we have demonstrated. We look forward to future works that analyze the interaction of page offlining with other error correction methods.

VII. Related Work

To the best of our knowledge, we have performed the first analysis of DRAM failure trends (on modern DRAM devices using modern data-intensive workloads) that have not been identified in prior work (e.g., chip density, transfer width, workload type), presented the first regression-based model for examining the memory failure rate of systems, and performed the first analysis of page offlining in the field. Prior large scale empirical studies of memory errors analyzed various aspects of memory errors in different systems. We have already presented extensive comparisons to the most prevalent of them [44, 16, 47, 48, 10] throughout the paper. We will discuss these and others here briefly.

Schroeder et al. performed the first study of memory errors in the field on a majority of Google's servers in 2009 [44]. The authors' study showed the relatively high rate of memory errors across Google's server population, provided evidence that errors are dominated by device failures (versus alpha particles), and noted that they did not observe any indication that newer generations of DRAM devices have worse error behavior, that CPU and memory utilization are correlated with error rate, and that average server error rate is very high – findings clarified by our study in this paper, five years later, as we explained in Section III. Their work formed the basis for what is known of DRAM errors in the field.

Hwang et al. performed an analysis on a trace of memory errors from a sample of Google servers and IBM supercomputers, showing how errors are distributed across various DRAM components [16], but *without* controlling for the effect of socket and channel failures. The high number of repeat address errors led them to simulate the effectiveness of page offlining (proposed in [49]) on the memory error traces, which they found to reduce error rate by 86% to 94%. Note that their study of page offlining, unlike ours (presented in Section VI), was done purely in simulation, not in a large scale system.

Sridharan et al. examined memory errors in a supercomputing environment [47, 48, 10]. Similar to Hwang et al., they found that most memory errors are permanent and additionally identified occurrences of multi-DIMM errors, and speculated as to their origin. They also found that DRAM vendor and age are correlated with error rate. Concurrent to our work, Sridharan et al. also observe that average server errors are much larger than median server errors [46], though we quantify and provide a model for the *full distribution* of errors per server. Siddiqua et al. provided an error classification methodology for the memory controller and memory bus, but did not classify memory errors at a finer DRAM chip-level granularity as we do [45]. They found that a small number of faults generate a large number of errors and that faults are predominantly permanent.

Nightingale et al. examined the failure rate of consumer PCs and showed that increased CPU frequency is correlated with increased DRAM error rates [40]. A pair of works by Li et al. analyzed memory errors on 212 Ask.com servers and evaluated their application-level impact [27, 28]. They found that most memory errors are permanent and that memory errors affected applications in noticeable ways, and proposed a technique to monitor memory for errors to reduce application impact.

VIII. Conclusions

We performed a comprehensive analysis of the memory errors across all of Facebook's servers over fourteen months. We analyzed a variety of factors and how they affect server failure rate and observed several new reliability trends for memory systems that have not been discussed before in literature. We find that (1) memory errors follow a power-law distribution, specifically, a Pareto distribution with decreasing hazard rate, with average error rate exceeding median error rate by around 55×; (2) non-DRAM memory failures from the memory controller and memory channel contribute the majority of errors and create a kind of *denial of service attack* in servers; (3) more recent DRAM cell fabrication technologies (as indicated by chip density) show higher failure rates (prior work that measured DRAM *capacity*, which is not closely related to fabrication technology, observed inconclusive trends); (4) DIMM architecture decisions affect memory reliability: DIMMs with fewer chips and lower transfer widths have the lowest error rates, likely due to electrical noise reduction; and (5) while CPU and memory utilization do not show clear trends with respect to failure rates, workload type can influence server failure rate by up to 6.5×.

We developed a model for memory failures and showed how system design choices such as using lower density DIMMs and fewer processors can reduce failure rates of baseline servers by up to 57.7%. We also performed the first analysis of page offlining in a real-world environment, showing that error rate can be reduced by around 67% identifying and fixing several real-world challenges to the technique.

We hope that the data and analyses presented in our work can aid in (1) clearing up potential inaccuracies and limitations in past studies' conclusions, (2) understanding the effects of different factors on memory reliability, (3) the design of more reliable DIMM and memory system architectures, and (4) improving evaluation methodologies for future memory reliability studies.

Acknowledgments

We thank Konrad Lai, Manish Modi, and Jon Brauer for their contributions to the work. We also thank the students who attended the Fall 2014 lectures of 18-742 at CMU, who provided feedback on earlier drafts. We thank the anonymous reviewers from ISCA 2014, OSDI 2015, and DSN 2015 and the members of the SAFARI research group for their comments and suggestions. This work is supported in part by Samsung and the Intel Science and Technology Center for Cloud Computing, as well as NSF grants 0953246, 1065112, 1212962, and 1320531.

References

[1] "DRAM Failure Model," http://www.ece.cmu.edu/~safari/tools.html.

[2] "Open Compute Project," http://www.opencompute.org/.

[3] "The R Project for Statistical Computing," http://www.r-project.org/.

[4] D. Borthakur *et al.*, "Apache Hadoop goes realtime at Facebook," SIGMOD, 2011.

[5] P. Chakka *et al.*, "Hive: A Warehousing Solution Over a Map-Reduce Framework," VLDB, 2009.

[6] P.-F. Chia, S.-J. Wen, and S. Baeg, "New DRAM HCI Qualification Method Emphasizing on Repeated Memory Access," IRW, 2010.

[7] C. Constantinescu, "Trends and Challenges in VLSI Circuit Reliability," *IEEE Micro*, 2003.

[8] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: Evidence and possible causes," *IEEE/ACM TON*, 1997.

[9] M. E. Crovella, M. S. Taqqu, and A. Bestavros, *A Practical Guide to Heavy Tails*. Chapman & Hall, 1998.

[10] N. DeBardeleben *et al.*, "Extra Bits on SRAM and DRAM Errors – More Data from the Field," SELSE, 2014.

[11] T. J. Dell, "A White Paper on the Benefits of Chipkill-Correct ECC for PC Server Main Memory," *IBM Microelectronics Division*, Nov. 1997.

[12] R. W. Hamming, "Error Correcting and Error Detecting Codes," *Bell System Technical Journal*, Apr. 1950.

[13] M. Harchol-Balter, "Task assignment with unknown duration," *J. ACM*, 2002.

[14] M. Harchol-Balter and A. Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing," SIGMETRICS, 1996.

[15] D. Hosmer and S. Lemeshow, *Applied Logistic Regression (Second Edition)*. John Wiley and Sons, Inc., 2000.

[16] A. Hwang, I. Stefanovici, and B. Schroeder, "Cosmic Rays Don't Strike Twice: Understanding the Characteristics of DRAM Errors and the Implications for System Design," ASPLOS, 2012.

[17] G. Irlam, "Unis File Size Survey – 1993," http://www.base.com/gordoni/ufs93.html.

[18] N. Jain *et al.*, "Data Warehousing and Analytics Infrastructure at Facebook," SIGMOD, 2010.

[19] JEDEC Solid State Technology Association, "JEDEC Standard: DDR4 SDRAM, JESD49-4A," Nov. 2013.

[20] U. Kang *et al.*, "Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling," The Memory Forum, 2014.

[21] S. Khan *et al.*, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," SIGMETRICS, 2014.

[22] Y. Kim *et al.*, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," HPCA, 2010.

[23] ——, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA, 2014.

[24] ——, "A Case for Subarray-Level Parallelism (SALP) in DRAM," ISCA, 2012.

[25] D. Lee *et al.*, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," HPCA, 2015.

[26] ——, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA, 2013.

[27] X. Li *et al.*, "A Memory Soft Error Measurement on Production Systems," USENIX ATC, 2007.

[28] ——, "A Realistic Evaluation of Memory Hardware Errors and Software System Susceptibility," USENIX ATC, 2010.

[29] J. Liu *et al.*, "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA, 2012.

[30] ——, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," ISCA, 2013.

[31] S. Liu *et al.*, "Flikker: Saving DRAM Refresh-power through Critical Data Partitioning," ASPLOS, 2011.

[32] J. S. Long, *Regression Models for Categorical and Limited Dependent Variables*. Sage Publications, 1997.

[33] Y. Luo *et al.*, "Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost via Heterogeneous-Reliability Memory," DSN, 2014.

[34] T. C. May and M. H. Woods, "Alpha-Particle-Induced Soft Errors in Dynamic Memories," *IEEE Transactions on Electron Devices*, 1979.

[35] Micron Technology, Inc., "4 Gb: ×4, ×8, ×16 DDR3 SDRAM," 2009.

[36] S. S. Mukherjee *et al.*, "Cache Scrubbing in Microprocessors: Myth or Necessity?" PRDC, 2004.

[37] S. Muralidhar *et al.*, "f4: Facebook's warm blob storage system," OSDI, 2014.

[38] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," MEMCON, 2013.

[39] O. Mutlu, J. Meza, and L. Subramanian, "The Main Memory System: Challenges and Opportunities," *Comm. of the Korean Institute of Information Scientists and Engineers*, 2015.

[40] E. B. Nightingale, J. R. Douceur, and V. Orgovan, "Cycles, Cells and Platters: An Empirical Analysis of Hardware Failures on a Million Consumer PCs," EuroSys, 2011.

[41] R. Nishtala *et al.*, "Scaling Memcache at Facebook," NSDI, 2013.

[42] V. Paxson and S. Floyd, "Wide-area traffic: The failure of poisson modeling," *IEEE/ACM TON*, 1995.

[43] B. Schroeder and M. Harchol-Balter, "Evaluation of task assignment for supercomputing servers: The case for load unbalancing and fairness," *Cluster Computing*, 2004.

[44] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM Errors in the Wild: A Large-Scale Field Study," SIGMETRICS/Performance, 2009.

[45] T. Siddiqua *et al.*, "Analysis and Modeling of Memory Errors from Large-scale Field Data Collection," SELSE, 2013.

[46] V. Sridharan *et al.*, "Memory Errors in Modern Systems: The Good, The Bad, and The Ugly," ASPLOS, 2015.

[47] V. Sridharan and D. Liberty, "A Study of DRAM Failures in the Field," SC, 2012.

[48] V. Sridharan *et al.*, "Feng Shui of Supercomputer Memory: Positional Effects in DRAM and SRAM Faults," SC, 2013.

[49] D. Tang *et al.*, "Assessment of the Effect of Memory Page Retirement on System RAS Against Hardware Faults," DSN, 2006.

[50] P. Vajgel *et al.*, "Finding a Needle in Haystack: Facebook's Photo Storage," OSDI, 2010.