

AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems

Moinuddin K. Qureshi[†]Dae-Hyun Kim[†]Samira Khan[‡]Prashant J. Nair[†]Onur Mutlu[‡][†]Georgia Institute of Technology

{moin, dhkim, pnair6}@ece.gatech.edu

[‡]Carnegie Mellon University

{samirakhan, onur}@cmu.edu

Abstract—Multirate refresh techniques exploit the non-uniformity in retention times of DRAM cells to reduce the DRAM refresh overheads. Such techniques rely on accurate profiling of retention times of cells, and perform faster refresh only for a few rows which have cells with low retention times. Unfortunately, retention times of some cells can change at runtime due to *Variable Retention Time (VRT)*, which makes it impractical to reliably deploy multirate refresh.

Based on experimental data from 24 DRAM chips, we develop architecture-level models for analyzing the impact of VRT. We show that simply relying on ECC DIMMs to correct VRT failures is unusable as it causes a data error once every few months. We propose AVATAR, a VRT-aware multirate refresh scheme that adaptively changes the refresh rate for different rows at runtime based on current VRT failures. AVATAR provides a time to failure in the regime of several tens of years while reducing refresh operations by 62%-72%.

Keywords—Dynamic Random Access Memory, Refresh Rate, Variable Retention Time, Error Correcting Codes, Performance, Memory Scrubbing

I. INTRODUCTION

Dynamic Random Access Memory (DRAM) has been the basic building block of computer memory systems. A DRAM cell stores data as charge in a capacitor. Since this capacitor leaks over time, DRAM cells must be periodically refreshed to ensure data integrity. The *Retention Time* of a single DRAM cell refers to the amount time during which it can reliably hold data. Similarly, the retention time of a DRAM device (consisting of many cells) refers to the time that it can reliably hold data in *all* of its constituent cells. To guarantee that all cells retain their contents, DRAM uses the worst-case refresh rate determined by the cell with the minimum retention time as a whole. JEDEC standards specify that DRAM manufacturers ensure that all cells in a DRAM have a retention time of at least 64ms, which means each cell should be refreshed every 64ms for reliable operation.

Despite ensuring reliable operation, using such high refresh rates introduce two problems: 1) refresh operations block memory, preventing it from performing read and write requests. 2) refresh operations consume significant energy [6, 28, 35]. In fact, as technology continues to scale and the capacity of DRAM chips increases, the number of refresh operations also increases. While the refresh overheads have been quite small (less than a few percent) in previous generations of DRAM chips, these overheads have become significant for current generation (8Gb) DRAM chips, and they are projected to increase substantially for future DRAM technologies [18, 28, 34, 35]. Figure 1 illustrates the trend, showing the throughput loss (the percentage of time for which the DRAM chip is unavailable due to refresh) for different generations of DRAM. As the memory capacity increases, memory throughput reduces

and refresh power grows. In fact, at the 32Gb-64Gb densities, the overheads of performance and power reach up to 25-50% and 30-50% respectively. Such overheads represent a *Refresh Wall*, and we need scalable mechanisms to overcome them.

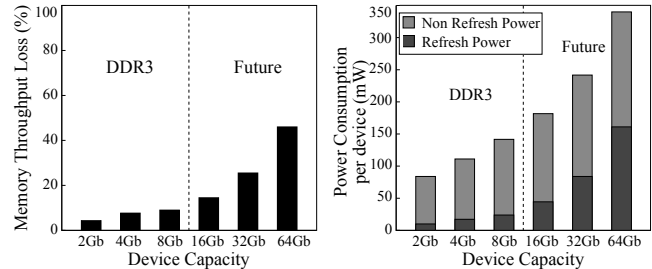


Fig. 1. Refresh Wall for scaling DRAM memory systems. (a) Memory throughput loss and (b) Power overheads of refresh (source [28]). The refresh overheads are significant and unmanageable for high-density chips.

To ensure that DRAM cells retain data reliably, DRAM conservatively employs the refresh interval of 64ms based on the DRAM cell with the shortest retention time. In fact, the vast majority of DRAM cells in a typical DRAM device can operate reliably with much longer refresh intervals [19, 29]. Multirate refresh mechanisms (e.g., [4, 21, 28, 36, 38, 41, 44]) exploit this discrepancy by identifying the few cells that require high refresh rates and refreshing only those portions of memory at the nominal refresh rate of 64ms. The rest of memory has a much lower refresh rate (4-8x less than the nominal rate). Multirate refresh schemes rely on an accurate retention time profile of DRAM cells. However, accurately identifying cells with short retention times remains a critical obstacle due to *Variable Retention Time (VRT)*. VRT refers to the tendency of some DRAM cells to shift between a low (leaky) and a high (less leaky) retention state, which is shown to be ubiquitous in modern DRAMs [29]. Since the retention time of a DRAM cell may change due to VRT, DRAM cells may have long retention times during testing but shift to short retention times at runtime, introducing failures¹ during system operation. A recent paper [18] from Samsung and Intel identifies VRT as one of the biggest impediments in scaling DRAM to smaller technology nodes.

This paper has two goals: 1) To analyze the impact of VRT on multirate refresh by developing experiment-driven models. 2) To develop a practical scheme to enable multirate refresh in the presence of VRT. To understand how VRT impacts multirate refresh, we use an FPGA-based testing framework [19, 24, 25, 29] to evaluate the impact of a reduced refresh rate on DRAMs in a temperature-controlled environment.

Prior works indicate that even after several rounds of testing performed for several days, new (previously unidentified)

¹We use terms of failure and error interchangeably in this paper.

bit errors continue to occur [19, 29]. However, we observe two important properties that provide us insights for developing an effective solution. First, after the initial testing, the number of active (failing) VRT cells during a given time period stabilizes close to an average value and follows a lognormal distribution. We refer to this constantly changing pool of active VRT cells as the *Active-VRT Pool (AVP)*. Second, although new bit errors, previously unseen, continue to surface even after several hours, the rate at which these new bit errors emerge stabilizes at a relatively low rate that we refer to as the *Active-VRT Injection (AVI)* rate. In our studies of 24 modern DRAM chips, we find that 1) 2GB memory has an Active-VRT pool of 350 to 500 cells on average within a 15-minute period; 2) AVI rate stabilizes at approximately one new cell within a 15-minute period.

The AVP and AVI metrics motivate much of the remaining analysis in this paper. The continual discovery of new bit errors even after hours of tests precludes the possibility of relying solely on memory tests to identify and eliminate bit errors. We can potentially use error correction code (ECC) DIMMs to correct VRT-related data errors: for example, we can use either in-DRAM ECC or SECDED DIMMs to correct VRT-related errors, as suggested by a recent study [18]. We refer to the approach of using SECDED for treating VRT-related errors the same way as soft errors as a *VRT-Agnostic* multirate refresh scheme. Our analysis shows that simply relying on ECC DIMMs still causes an uncorrectable error once every six to eight months (even in the absence of any soft errors). Such a high rate of data loss is *unacceptable in practice*, making multirate refresh *impractical* to reliably deploy even for a memory system employing DIMMs with ECC capability.

This paper introduces the first practical, effective, and reliable multirate refresh scheme called AVATAR (**A** Variable-Retention-Time **A**ware multirate **R**efresh), which is a system-level approach that combines ECC and multirate refresh to compensate for VRT bit errors. The key insight in AVATAR is to adaptively change the refresh rate *for rows that have encountered VRT failures* at runtime. AVATAR uses ECC and scrubbing to detect and correct VRT failures and *upgrade* rows with such failures for faster refresh. This protects such rows from further vulnerability to retention failures. We show that the pool of *upgraded* rows increases very slowly (depending on AVI), which enables us to retain the benefits of reduced refresh rate (i.e. slower refresh) for most of the rows. AVATAR performs infrequent (yearly) testing of the upgraded rows so that rows not exhibiting VRT anymore can be *downgraded* to slower refresh.

We show that AVATAR improves the reliability of a traditional multirate refresh scheme by 100 times, increasing the time to failure from a few months to several tens of years (even in the presence of high soft-error rates, as discussed in Section VI-C). AVATAR provides this high resilience while retaining most of the refresh savings of VRT-Agnostic multirate refresh and incurring no additional storage compared to VRT-Agnostic multirate refresh. AVATAR is especially beneficial for future high-density chips that will be severely limited by refresh. For example, our evaluations show that for a 64Gb DRAM chip, AVATAR improves performance by 35% and reduces the Energy Delay Product (EDP) by 55%.

II. BACKGROUND AND MOTIVATION

A. DRAM Organization and DRAM Refresh

A DRAM cell consists of one transistor and one capacitor (1T-1C), as shown in Figure 2. DRAM cells are organized as banks, a two-dimensional array consisting of rows and columns. The charge stored in the capacitor tends to leak over time. To maintain data integrity, DRAM systems periodically perform a refresh operation, which simply brings the data from a given row into the sense amplifiers and restores it back to the cells in the row. Thus, refresh operations are performed at the granularity of a DRAM row.²

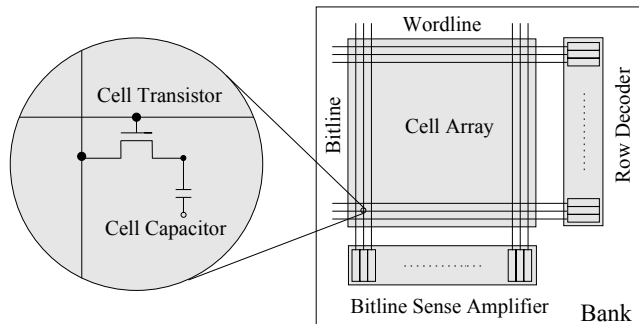


Fig. 2. DRAM Organization (source [14]).

B. Refresh Wall for Scaling DRAM

As the capacity of DRAM increases, the time spent in performing refresh also increases. The performance and power of future high-density DRAMs are expected to be severely constrained by overheads of refresh operations (Figure 1). As the increased variability of DRAM cells with smaller geometries might reduce the DRAM refresh period from 64ms to 32ms even for operation at normal temperature [16, 17], the refresh problem is likely to become worse for future DRAMs [18, 28, 34, 35]. Thus, techniques that can eliminate or reduce refresh operations can be greatly effective in overcoming the *Refresh Wall*.

C. Multirate Refresh

The retention time of different DRAM cells is known to vary, due to the variation in cell capacitance and leakage current of different cells. The distribution of the retention time tends to follow a log-normal distribution [10, 22], with typical DRAM cells having a retention time that is several times higher than the minimum specified retention time. Multirate refresh techniques exploit this non-uniformity in retention time of DRAM cells to reduce the frequency of DRAM refresh. Multirate refresh schemes (e.g., [21, 28, 36, 38, 41, 44]) group rows into different bins based on the retention time profiling and apply a higher refresh rate only for rows belonging to the lower retention time bin.

1) *Implementation*: Figure 3(a) shows a generic implementation of multirate refresh scheme using two rates: a *Fast Refresh* that operates at the nominal rate (64ms) and a *Slow Refresh* that is several times slower than the nominal rate. Multirate refresh relies on retention testing to identify rows that must be refreshed using Fast Refresh, and populates the

²For more detail on DRAM operation and refresh, we refer the reader to [6, 23, 25, 26, 28, 29, 35].

Refresh Rate Table (RRT) with this information. At runtime, RRT is used to determine the refresh rate for different rows. For an 8GB DIMM with an 8KB row buffer, the size of RRT is 128KB.³ For our studies, we assume that the RRT information is available at the memory controller, similar to RAIDR [28].

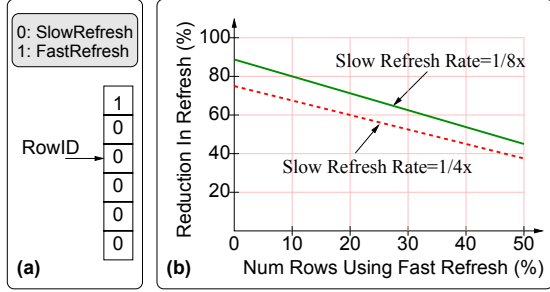


Fig. 3. Multirate Refresh (a) Implementation with an RRT (b) Effectiveness at reducing refresh.

2) *Effectiveness*: The effectiveness of multirate refresh at saving refresh operations depends on the rate of Fast and Slow Refresh. For a slow refresh rate that is 4x-8x lower than a fast refresh rate, only a small fraction of DRAM rows end up using fast refresh rates. For example, for our studies with 8GB DIMMs and a slow refresh rate that is five times slower than a fast refresh rate, 10% of the rows get classified to use Fast Refresh. Figure 3(b) shows the reduction in refresh operations compared to always using Fast Refresh, when a given percentage of memory rows use Fast Refresh. We analyze two different rates of Slow Refresh, 4X and 8X lower than that of Fast Refresh. Even with 10% of the rows using Fast Refresh, the total refresh savings with multirate refresh range from 67% to 78%. Thus, multirate refresh is highly effective at reducing refresh operations.

D. The Problem: Retention Time Varies Dynamically

The key assumption in multirate refresh is that the retention time profile of DRAM cells does not change at runtime. Therefore, a row classified to use Slow Refresh continue to have all the cells at higher retention time than the period of the Fast Refresh. Unfortunately, the retention time of DRAM cells can change randomly at runtime due to a phenomenon called *Variable Retention Time (VRT)* [45]. VRT can cause a cell to randomly flip from a high retention state to a low retention state, thus causing data errors with multirate refresh. The existence of VRT makes it challenging to use multirate refresh schemes reliably. The next section provides insights into how VRT impacts multirate refresh.

³The storage for tracking the refresh rate can be reduced if the number of rows that need Fast Refresh is very small. For example, RAIDR [28] employs Bloom filters for tracking 1000 weak rows for a memory with one million rows (i.e., 0.1% of total rows). It can be shown that Bloom filters become ineffective at reducing storage when the number of weak rows become a few percent of total rows. For our target refresh rate, 10% or more rows get classified for using Fast Refresh, therefore we use an RRT with one bit per row. The SRAM overhead of RRT can be avoided by storing the RRT in a reserved area of DRAM (128KB for 8GB is 0.0015% of memory space). While refresh decisions for the current RRT line (512 rows) get used, the next RRT line can be prefetched from DRAM to hide latency of RRT lookup. The RRT in DRAM can be replicated three times (while incurring a total storage overhead of only 0.005%) for tolerating VRT related errors in the RRT.

III. VARIABLE RETENTION TIME

VRT causes a DRAM cell to change its retention characteristics. A cell with VRT exhibits multiple retention states and transitions to these states at different points of time in an unpredictable fashion [29,45]. As a result, the same cell can fail or pass at a given refresh rate, depending on its current retention time. Although VRT only affects a very small fraction of cells at any given time, the retention time change of even a single cell can be sufficient to cause data errors in a memory system that employs multirate refresh. We explain the reasons behind VRT and then characterize the behavior of VRT cells.

A. Causes of VRT

VRT phenomenon in DRAM was reported in 1987 [45]. The physical phenomenon behind the VRT cells is attributed to the fluctuations in the *gate induced drain leakage (GIDL)* current in the DRAM cells. Prior works suggest that presence of traps near the gate region causes these fluctuations. A trap can get occupied randomly, causing an increase in the leakage current. As a result, the cell leaks faster and exhibits lower retention time. However, when the trap becomes empty again, the leakage current reduces, resulting in a higher retention time [7,20]. Depending on the amount of the leakage current, VRT cells exhibit different retention times. VRT can also occur due to external influences such as high temperature during the packaging process or mechanical or electrical stress. It is hard for manufacturers to profile or screen such bits since VRT can occur beyond post-packaging testing process [7,33]. Recent experimental studies [19,29] showed that the VRT phenomenon is ubiquitous in modern DRAM cells. Future memory systems are expected to suffer even more severe VRT problems [18]. They are likely to apply higher electrical field intensity between the gate and the drain, which increases the possibility of charge traps that may cause VRT bits. A recent paper [18] from Samsung and Intel identifies VRT as one of the biggest challenge in scaling DRAM to smaller technology nodes.

B. Not All VRT is Harmful

Not all changes in retention time due to VRT cause a data error under multirate refresh. For example, VRT can also cause the retention time of a cell to increase, which makes the cell more robust against retention failures. Figure 4 shows the relationship between the refresh interval and variable retention times.

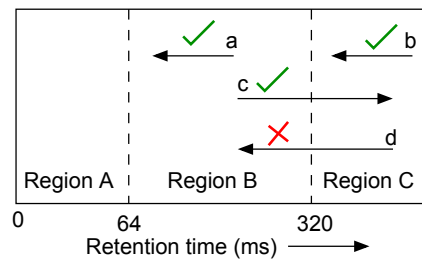


Fig. 4. VRT can cause a data error only when a cell moves from a high-retention region to a low-retention region.

We assume that the system performs refresh at two rates: 64ms (Fast Refresh) and 320ms (Slow Refresh). The vertical lines at 64ms and 320ms divide the figure into three regions. Transitions within a region (exemplified by cells *a* and *b*),

and transitions from region B to region C (exemplified by cell *c*) cause no data failure. A multirate refresh mechanism is vulnerable to random VRT failures only when a cell moves from region C to region B (exemplified by cell *d*). In our studies, we identify only such cells as exhibiting VRT.

C. Experimental Setup

To understand the impact of VRT, we test the effect of the extended refresh interval on cells in commercial DRAM chips. We use an FPGA-based infrastructure [19, 24, 25, 29], consisting of an ML605 FPGA development board and modify the RTL to increase the refresh interval at a temperature controlled environment. Our experiments are done at a refresh interval of 4 seconds at 45 °C, which corresponds to 328ms at 85 °C (similar assumptions on temperature based scaling of retention time were made in prior DRAM studies [19,29]). Our experimental temperature closely matches to typical operating temperature as prior works show that even with 100% utilization, temperature in server and desktop systems remain in the range of 40–60 °C [9,25]. We study a multirate refresh scheme that employs a Slow Refresh at a refresh period of 320ms which is very close to our tested retention time of 328ms.

We conduct our experiments with three 2GB DIMMs (A, B, and C), each from a different DRAM vendor. Each module consists of 8 DRAM chips. To locate the VRT failures, we write specific test patterns in the entire module, increase the refresh interval, and read the contents after all rows have been refreshed at the extended interval. Any mismatch in the content implies a retention error at that location during the given time period. We log the statistics of retention failures once every one minute and perform the experiment for a period of 7 days. To keep the analysis tractable, we present statistics for an interval of every 15 minutes (a total of 4 x 24 hours x 7 days = 672 periods, of 15 minutes each).

D. Observation 1: Population of Weak Cells Increases

We first study the impact of VRT on multirate refresh. In our studies, a cell that has never caused failure with the Slow Refresh rate is deemed as a *strong cell*, whereas a cell that encountered at least one failure due to VRT is deemed as a *weak cell*. Figure 5 shows the number of unique weak cells and weak rows. A row is classified as a weak row if it contains at least one weak cell. There are three important implications derived from Figure 5, consistent with prior studies [19,39]. The first is that the number of weak cells in the first time period is quite large (27841 for A, 24503 for B, and 22414 for C) [19]. Thus, fortunately the initial testing that multirate refresh deploys identifies a majority of the weak cells. Multirate refresh enforces the rows containing these weak cells to always use Fast Refresh. The 2GB DIMMs in our experiments have 256K rows (each 8KB). Thus, multirate refresh assigns approximately 9%-10% of the total memory rows to Fast Refresh. For the remainder of our studies, we assume that the weak rows identified during the initial testing are always refreshed with Fast Refresh. So, we exclude these rows from the rest of our analysis. The second implication of Figure 5 is that the number of weak rows is very close to the number of weak cells, which implies that the weak

cells are randomly scattered throughout the memory⁴ [39]. This observation can help us assume a random distribution for VRT cells and develop models for analyzing their behavior on longer time scales than possible with experiments. The third implication is that initial testing (or testing alone) is not sufficient to identify all weak cells [19]. Even after several days, VRT causes new bits to have retention failures. For example, for module A, the number of weak cells increases from 27841 in the first time period to 31798 in the last time period. The consistency of our results with prior works [19,39] attests to the soundness of our infrastructure, validating the new observations we make in our studies.

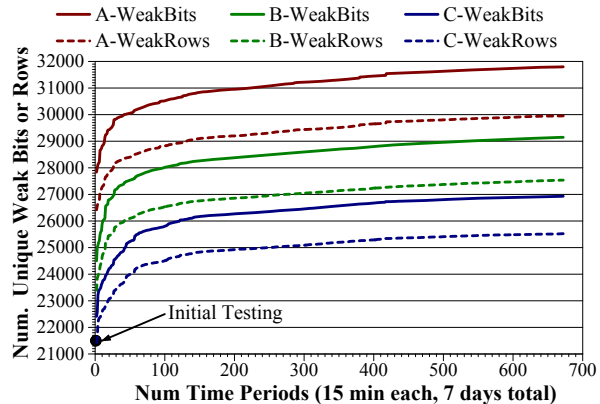


Fig. 5. The cumulative number of unique weak cells and weak rows for the three modules (A, B, and C). Even after several hours of testing, VRT causes new (previously unidentified) bits to cause retention failures.

E. Observation 2: VRT Cells Can Switch Randomly

The fact that a cell enters a low retention state due to VRT does not mean that the cell continues to be in the low retention state indefinitely. Figure 6 shows the behavior of two typical VRT cells for every tested time period. We deem the cell to pass if it has a retention time greater than 328ms and to fail otherwise. Cell X transitions randomly and frequently between high and low retention states. However, some other cell affected by VRT (say cell Y) may continue to stay in the same retention state for several hours or days, before moving to another retention state. In general, any cell in the DRAM

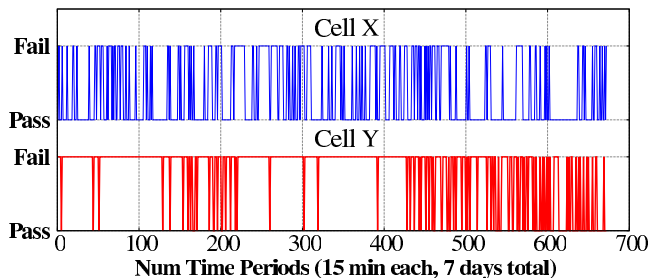


Fig. 6. A VRT cell can randomly and frequently transition between high and low retention states.

array can experience VRT. However, in practice, only a very small fraction of DRAM cells change their retention time at

⁴Multiple weak cells may still map to the same row, albeit with a small probability. The number of weak rows obtained experimentally closely follows that of a statistical random mapping of weak bits to rows, indicating VRT cells are randomly scattered in memory.

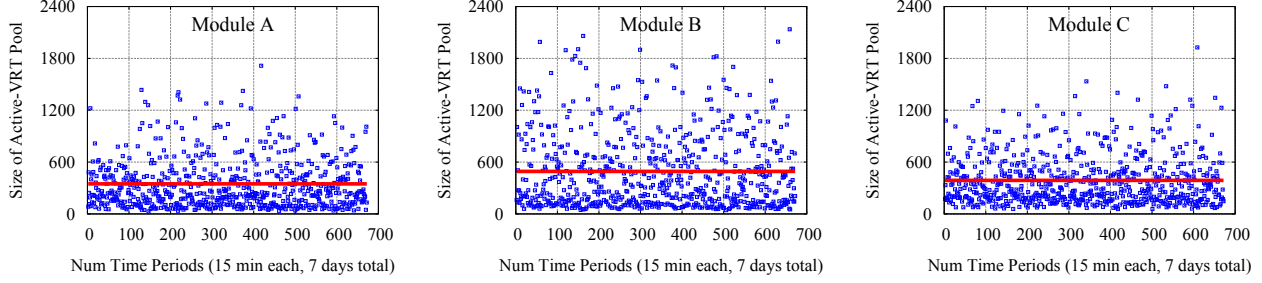


Fig. 7. The size of the Active-VRT Pool for each time period for each of the three modules.

any given time period. We define the cell that causes an error in a given time period as an *Active-VRT* cell. If a cell has caused a VRT-related error in any of the previous time periods but not the current time period, it is deemed to be a *Dormant-VRT* cell.

F. Observation 3: Size of the Active-VRT Pool Varies

Given that a cell affected by VRT can switch between being an Active-VRT cell and being a Dormant-VRT cell, we would expect that the total number of Active-VRT cells within a given time period to be smaller than the number of unique weak cells encountered since initial testing. We call the group of all cells that are Active-VRT cells within a given time period as forming an *Active-VRT Pool (AVP)*. Figure 7 shows the size of the AVP for each of the 15-minute time periods in our experiments.

The size of the AVP varies dynamically for all modules across the time periods. The average size of the AVP for module A is 347 (standard deviation, or σ , of 288), for module B is 492 (σ of 433), and for module C is 388 (σ of 287). Since predicting the exact size of the AVP is difficult, our experimental data can help us develop models for capturing the size of the AVP as a means of analyzing the behavior of VRT cells.

G. Modeling the Dynamic Size of Active-VRT Pool

We observe that the size of the AVP tends to follow a lognormal distribution. Figure 8, obtained experimentally from the three modules, shows the histogram of the log of the size of the AVP across the 672 time periods. It also shows the lognormal fit (the thick line) for the AVP size based simply on the mean and the standard deviation obtained from the measurements. We observe that the frequencies of occurrence under the lognormal fit and the experimental data match well. Therefore, we can model the size of the AVP at any time period as a random variable originating from a lognormal distribution whose parameters are derived from the experimental data. We use such a model for the AVP size in our analysis.

H. Observation 4: Rate of New VRT Cells Steadies

Another important parameter for analyzing the impact of VRT is the rate at which new (previously undiscovered) cells become Active-VRT cells. Any scheme for mitigating VRT-related errors is likely to be influenced by this rate, given that these newly-vulnerable cells can appear anywhere in the memory array and cause VRT-related errors. We call the rate at which new cells become Active-VRT cells as the *Active-VRT*

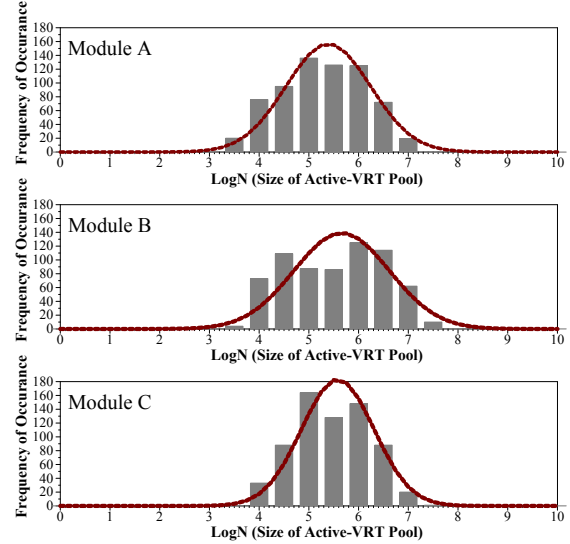


Fig. 8. The size of the Active-VRT Pool closely follows a lognormal distribution (the bars represent histogram from experimental data and the line represents a lognormal fit based on the mean and the standard deviation of the Active-VRT Pool).

Injection (AVI) Rate. Figure 9 shows the AVI rate (moving average, measured over a six-hour window) for each time period in our experiments. After the initial few hours of observation, the AVI rate tends to become steady and stabilizes at a small value. The average AVI rate measured in the second half of the experiments is close to 1 for all modules. For our studies, we use a default AVI rate of 1 (for 2GB module) and perform a sensitivity analysis.

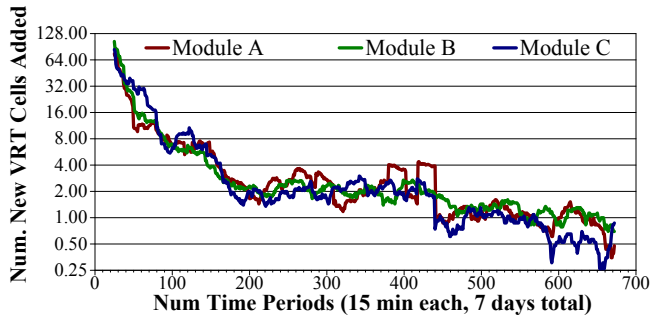


Fig. 9. The rate of new cells becoming Active-VRT cells.

IV. ARCHITECTURE MODEL FOR ANALYZING VRT

If the system is not provisioned to tolerate VRT-related errors, then the first cell affected by VRT will cause data

loss. However, a system provisioned with some means of mitigating a VRT-related data error may suffer data loss over many months or years. Conducting experimental studies over such a long period of time can be prohibitive. We use the key observations made in the previous section to develop a simple and practical analytical model for analyzing long-term impact of VRT. We first present the cell-level model before presenting the architecture-level model.

A. Cell Model under VRT

We classify each cell as either a strong cell or a weak cell. A weak cell that causes a failure in the current time period is deemed an *Active-VRT* cell. Otherwise, the cell is deemed as a *Dormant-VRT* cell. The AVI determines the rate at which strong cells get converted into weak cells, and the AVP determines the number of weak cells that are currently in the Active-VRT state. Figure 10 captures our cell-level model.

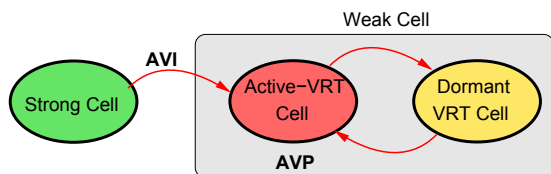


Fig. 10. In our cell model, AVI controls the conversion of strong cells to weak cells and AVP regulates the population of Active-VRT cells.

B. Architecture Model for VRT

Our architecture-level model is driven by two parameters: The size of the Active-VRT Pool (AVP size, determined by the lognormal fit based on experimental parameters of mean, Mu , and standard deviation, $Sdev$) and the rate of discovering new cells that become Active-VRT cells (AVI rate, denoted by K). For each time quantum, the number of cells that are Active-VRT cells are obtained from a random variable seeded with the lognormal fit, and these cells are chosen from random locations in memory. Then, K new VRT cells join the Active-VRT pool, selected again from random locations in memory, and an equal number of random cells leave the Active-VRT pool. In each time period, we compute the probability that the system encounters an uncorrectable failure and repeat this process until the desired duration of simulation. As the size of the pool is determined by statistical methods, we repeat the model-based simulation a large number of times (100K times) and report the average system failure probability. Our analytical model is shown in Figure 11.

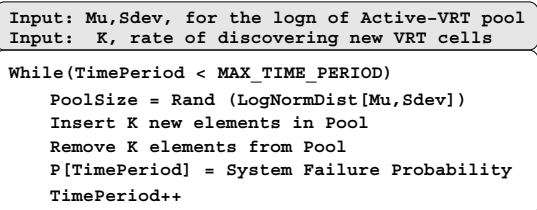


Fig. 11. Architecture model for VRT based on AVP and AVI.

C. Parameter Scaling for Larger Systems

To study larger systems, we scale the values of AVP and AVI derived from our experiments in proportion to the DIMM capacity. For example, we will analyze a 32GB memory system

containing four ECC DIMMs, each with data capacity of 8GB. Each 8GB ECC-DIMM has 4.5x more bits (8GB data + 1GB ECC) compared to the 2GB non-ECC DIMM we experimentally analyzed. To accommodate this, we scale our measured AVP sizes by 4.5x and use the natural logarithm of these scaled values to determine the mean and the standard deviation for the lognormal fit for AVP. We also scale the AVI rate by 4.5x, and use AVI=4.5 for an 8GB ECC-DIMM instead of a default value of AVI=1 for a 2GB module. This means that each of the four DIMMs in our 32GB system encounters 4.5 new VRT cells every 15 minutes.

V. IMPACT OF VRT IN THE PRESENCE OF ECC DIMM

Given the prevalence of VRT, a system employing multirate refresh can be expected to encounter a data error within a few minutes after the initial testing. Therefore, even though multirate refresh is quite effective at reducing the number of refresh operations, it cannot operate reliably.

A. Agnos: A VRT-Agnostic Approach

We can tolerate the VRT-related errors by using ECC [18]. Conventional DRAM DIMMs are also available in ECC versions, and such DIMMs support a SECCED code at an eight-byte granularity. While these ECC DIMMs are typically used to tolerate soft errors, we can also use ECC to correct the errors that happen due to VRT.⁵ We call such an approach, i.e., treating VRT-related errors similarly to soft errors, as *VRT-Agnostic (Agnos) multirate refresh*.

B. Failure Rate Analysis of Agnos

We assume that Agnos performs initial testing to profile the weak cells and ensures the rows containing weak cells always use Fast Refresh. Agnos corrects any error that happens after initial testing using ECC. As long as no eight-byte word has two-bit errors, Agnos will be able to correct an error. A two-bit error in a word gets detected but results in data loss.

We use the analytical model for VRT to estimate the time to failure for a system with Agnos. We first perform the failure analysis for a single DIMM and then use the assumption that DIMM failures are independent to estimate the system failure rate. We also assume that the weak cells identified during initial testing always use Fast Refresh and have no VRT-related errors so we ignore such cells in our analysis.

Let there be W words in an ECC DIMM and each word be protected by SECCED (W excludes the words corresponding to the rows identified by initial testing as weak rows). At steady state, a pool of Active-VRT cells, the size of which is deemed to follow a lognormal distribution, is formed. Let the size of the AVP be $\mathcal{P} = Random(LnN(Mu, Sdev))$. We assume that the Active VRT cells are randomly selected from the memory. Given that \mathcal{P} is much smaller than W , we assume that \mathcal{P} words in memory have a one-bit error, each of

⁵The soft-error rate (SER) is reported to be between 200-5000 FIT per Mbit [1,5,27], which corresponds to one soft error every 3 hours to 75 hours for an 8GB DIMM. For our study, we assume 4.5 errors every 15 minutes due to VRT. Thus soft errors happen at a 54x-2700x lower rate than VRT and have a negligible effect on our reliability calculations. We ignore SER for the analysis of Agnos and pessimistically assume the highest SER while analyzing our proposal in the next section. Note that if the rate of SER is even lower (or zero), it will only enhance the efficacy of our scheme.

which can be corrected with ECC. However, the system will face an uncorrectable error if there is one more error in any of these \mathcal{P} words. We divide time into equal-length periods. In each time period, let K unique errors enter the pool and K errors leave the pool. We assume that the new errors come from random locations in memory. In each time period, the module will have no uncorrectable error if all K errors come from locations other than the ones which are in the Active-VRT pool (\mathcal{P}). The probability of no uncorrectable error at each time period is given by Equation (1).

$$P(\text{DIMM has no uncorrectable error}) = \left(1 - \frac{\mathcal{P}}{W}\right)^K \quad (1)$$

After T time periods, there will be $K \cdot T$ new errors entering the VRT pool. So, the probability that the DIMM does not have an uncorrectable error is given by Equation (2).

$$P(\text{DIMM has no uncorrectable error}) = \left(1 - \frac{\mathcal{P}}{W}\right)^{K \cdot T} \quad (2)$$

For a system with D DIMMs, the probability that the system does not have an uncorrectable error after T time periods is given by Equation (3).

$$P(\text{System has no uncorrectable error}) = \left(1 - \frac{\mathcal{P}}{W}\right)^{K \cdot T \cdot D} \quad (3)$$

C. Failure Rate Results for Agnos

We use Equation (3) to assess the failure rate for our system with 32GB memory consisting of four ECC DIMMs, each of 8GB. For each DIMM, the AVP size is determined by the scaled parameters (mean and standard deviation of log of AVP values) of our modules A, B, and C. The rate of new VRT cells (AVI) per 15-minute period is 4.5 per DIMM. We repeat the experiments 100K times and report the average value of system failure probability (the spread around the average value is extremely narrow). Figure 12 shows the probability that the system with Agnos does not have an uncorrectable error for systems based on the AVP parameters derived from Module A (Sys-A), Module B (Sys-B), and Module C (Sys-C). The Agnos system is expected to encounter an uncorrectable error once every few months. For example, there is a 50% probability that an error will happen within 6 months for Sys-B and 8 months for Sys-A. Such a high rate of data loss is unacceptable in practice.

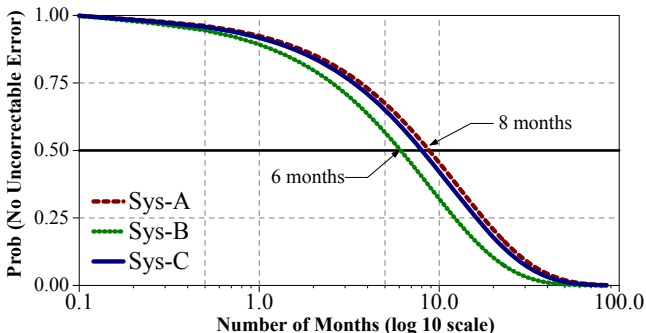


Fig. 12. Failure rate of Agnos, VRT-Agnostic multirate refresh. Even with ECC, Agnos encounters an uncorrectable error once every few months.

D. Conclusion from Agnos

The failure analysis of Agnos shows that simply relying on SECDED alone is not enough to mitigate VRT-related errors for a system with multirate refresh. One can potentially consider using higher levels of ECC to mitigate VRT-related errors, however this would increase cost significantly since ECC DIMMs of higher strength are not typically available as commodity parts. The next section shows how a “VRT-Aware” approach can effectively mitigate VRT-related errors.

VI. AVATAR: A VRT-AWARE MULTIRATE REFRESH

The problem with Agnos is that it uses ECC to correct VRT-related errors in a way similar to how ECC is used to correct soft errors. Unfortunately, while a soft error goes away once it is corrected, a VRT-cell can continue to remain vulnerable for several hours. At any time, several hundreds of cells can exhibit VRT, so using ECC to continuously correct these persisting errors means that the ECC for the word can no longer correct any new error in those words. Therefore, instead of the VRT-agnostic approach of treating VRT similarly to soft errors, we propose AVATAR, a **V**ariable-**R**etention-**T**ime **A**ware multirate **R**efresh.

A. Design

Figure 13 shows the design of AVATAR. Similar to multirate refresh, AVATAR performs an initial retention time testing to populate the Row Refresh Table (RRT) leveraging the profiling mechanisms described in [19, 29]. Like Agnos, AVATAR also employs ECC DIMMs to detect and correct errors due to VRT. Detection and correction happens only when the line in memory gets accessed (either due to a read or a write operation). The key insight of AVATAR is to break the formation of a large pool of Active-VRT cells. AVATAR does so by upgrading a row to use Fast Refresh, as soon as any word within the row encounters an ECC error (2). This means the vulnerable row gets protected from any further retention failures and no longer participates in the Active-VRT pool.⁶

Typically ECC check happens only when the data item is accessed from memory, which means parts of the memory region that have low activity can still be vulnerable due to accumulation of VRT-related errors. AVATAR addresses this problem by proactively applying a memory scrub (3), which periodically checks all the memory for potential data errors. We employ a scrub once every time period (15 minutes in our studies). We assume that the scrub operation identifies all the VRT-related data errors that happen during the scrub interval.

AVATAR upgrades the row to use Fast Refresh on every ECC correction, regardless of whether such correction happens due to a regular data access or due to a scrub operation. Thus, the total number of rows using Fast Refresh tends to grow over time, albeit very slowly. The effectiveness of AVATAR can be

⁶ECC correction can also get invoked due to a soft error strike. However, the frequency of soft errors is very small compared to that of VRT errors (e.g., the soft error rate is 54-2700x lower than the VRT error rate for an 8GB DIMM). Thus, we can conservatively assume that an error happens mainly due to VRT, and the row containing the error is upgraded to use Fast Refresh although less than 0.05%-2% of the total upgrades may still occur due to a soft error. Note that an even lower rate of soft errors would improve the efficacy of our mechanism.

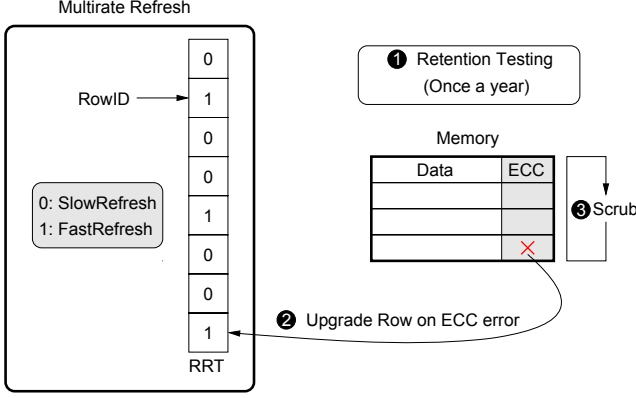


Fig. 13. Design of AVATAR.

improved by infrequent retention testing (1) (e.g., once a year; see analysis in Section VI-E).

B. Failure Rate Analysis

As AVATAR instantly upgrades any row with an ECC error to use Fast Refresh, the number of Active-VRT cells in any time period is equal to (or less than) the number of newly injected errors since the last scrub operation. Thus, the pool of Active-VRT cells is equal to the rate at which new (previously undiscovered) cells become VRT cells. To avoid data errors, we want all the newly injected cells to map to different words.

Let there be W words in an ECC DIMM, where each word is protected by SECDED (we ignore the words belonging to the rows identified as weak rows during the initial testing of multirate refresh). In each time period, let K unique cells in memory become Active-VRT cells. We assume that the new errors come from random locations in memory. In each time period, the probability that all K errors map to different locations in memory is given by Equation (4).

$$\text{Prob}(\text{DIMM has no uncorrectable error}) = \left(1 - \frac{1}{W}\right) \times \left(1 - \frac{2}{W}\right) \times \dots \times \left(1 - \frac{K-1}{W}\right) \quad (4)$$

Given that K is very small compared to the number of words in the DIMM (few tens versus millions), we simplify the above equation using the approximation $e^{-x} \approx (1-x)$, which degenerates Equation (4) into Equation (5).

$$\text{Prob}(\text{DIMM has no uncorrectable error}) = e^{-\frac{K^2}{2W}} \quad (5)$$

After T time intervals, the probability that the DIMM does not have any uncorrectable error is given by Equation (6).

$$\text{Prob}(\text{DIMM has no uncorrectable error}) = e^{-\frac{TK^2}{2W}} \quad (6)$$

For a system with D DIMMs, the probability that the system does not have an uncorrectable error after T time periods is given by Equation (7).

$$\text{Prob}(\text{System has no uncorrectable error}) = e^{-\frac{DTK^2}{2W}} \quad (7)$$

C. Incorporating Failures Due to Soft Errors

Thus far, we have assumed that the failure in memory happens only due to a VRT-related retention error. However, failures can also happen due to a soft error. We note that the rate of soft errors is much lower than the rate of VRT-related errors we consider, so the relative contribution of soft errors to the overall failure rate is negligible. For example, the reported soft-error rate from prior studies [1, 5, 27] ranges from 200 to 5000 FIT per Mbit, which is equivalent to one failure every 3-75 hours for an 8GB DIMM. In our analysis, we assume that the error rate related to VRT is 4.5 per 15 minutes, which is equivalently 54-2700x higher than the error rate of soft errors. Nonetheless, our model can simply be extended to accommodate for soft errors by modulating the rate at which new errors are encountered in memory (i.e., the AVI rate). For example, to accommodate a soft error rate of 5000 FIT per Mbit, we would increase the AVI from 4.5 to 4.6 per 15 minutes.

D. Failure Rate Results

We use Equation (7) to assess the failure rate of AVATAR for our system with 32GB memory consisting of four 8GB ECC DIMMs. The reliability of AVATAR is dependent only on the rate of discovering new VRT cells (the AVI rate). Figure 14 shows the probability that a system with AVATAR has no uncorrectable error for three rates of AVI, a nominal rate derived from our experiments (4.6 errors per 15 minutes per DIMM, including 0.1 errors per 15 minutes for soft errors), a doubled AVI rate, and a quadrupled AVI rate.⁷

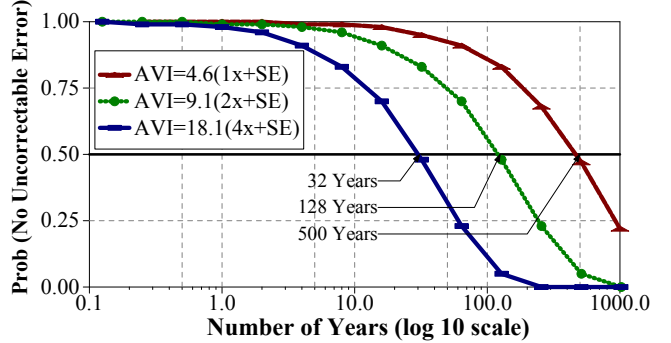


Fig. 14. The time to uncorrectable error with AVATAR is 500 years with nominal AVI, and 32 years with a 4x higher AVI Rate. We include a 0.1 AVI for soft error (SE).

For the nominal rate, even in the presence of a high soft-error rate, the time to failure with AVATAR is 500 years. When the AVI rate is doubled, the time to failure is more than 128 years. Even for a quadrupled AVI rate, the time to failure is 32 years. Thus, AVATAR enhances the reliability of multirate refresh by approximately a thousand times compared to Agnos, making it feasible to deploy multirate refresh reliably. Except for the negligible hardware for scrubbing, AVATAR requires no extra hardware than what is required for multirate refresh (Section II-C) and ECC DIMM.

⁷We analyze doubled and quadrupled AVI rates to account for potential increases in AVI rate, e.g., due to changes in environmental conditions during system operation.

E. Effective Refresh Savings

AVATAR conservatively upgrades any row that encounters an ECC error so that the number of rows refreshed with Fast Refresh grows over time. Figure 15 compares the reduction in refresh operations of AVATAR with that of an ideal multirate refresh scheme that does *not* suffer from VRT (Agnos, without VRT errors). For AVATAR, we assume an AVI rate of 4.6. AVATAR saves 72% of refreshes on a day after testing for the retention time profile. This is similar to other multirate refresh schemes such as RAIDR [28]. However, the prior multirate refresh schemes provide this refresh savings while compromising data integrity. Even after 12 months of continuous operation, the refresh savings with AVATAR are still high (62.4%). Thus, AVATAR improves reliability of multirate refresh while maintaining most of the refresh savings. To make AVATAR even more effective, we recommend that the system invoke retention testing once a year, which retains the refresh savings of AVATAR back at 72%.⁸

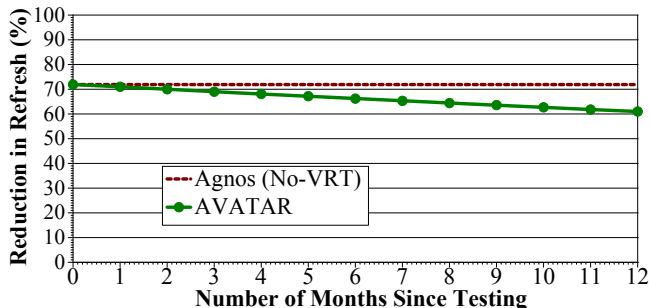


Fig. 15. Refresh savings of Agnos and AVATAR. AVATAR maintains most of the refresh savings of a VRT-Agnostic scheme while avoiding data errors.

F. Overheads of Scrubbing

We assume a default scrub interval of 15 minutes. Frequent scrubbing enhances the reliability of AVATAR as it allows fewer undetected Active-VRT cells to stay in memory. If a system desires even higher reliability than we report, the system can perform scrubbing at a shorter interval. However, scrubbing has the overheads of performance (as memory becomes unavailable) and energy (to perform memory reads).

1) *Performance Overhead of Scrubbing*: To estimate the performance overhead, we assume that scrubbing is done at a row granularity. The time to scrub one row would be equal to the sum of the time to open a row (t_{RC}, 14ns), the time to stream out the row buffer (256ns for 8KB), and the time to close the row (t_{RP}, 14ns), a total time of 284ns [15]. We estimate the total time required to scrub the 8GB DIMM (284ns × 2²⁰ rows) to be 298ms. This time is spread over several minutes. Hence, the memory unavailability due to scrub is negligible. Table I shows the memory throughput loss as the scrub interval varies. For the 15-minute scrub interval we use, the memory throughput loss is negligible (0.03%).

⁸Periodic testing once every few days or months does not have a significant effect on the reliability of the Agnos scheme. Shortly after testing, a pool of Active-VRT cells would get formed, reverting the system to the same vulnerable state. Similarly, periodic scrubbing does not have a significant impact on the reliability of Agnos as the VRT cells fail again soon.

TABLE I. PERFORMANCE AND ENERGY OVERHEAD OF SCRUB

Scrub Interval (minutes)	Mem Throughput Loss	Scrub Energy / Refresh Energy
4	0.130%	4.00%
8	0.065%	2.00%
15	0.035%	1.06%
30	0.018%	0.53%
60	0.009%	0.27%

2) *Energy Overhead of Scrubbing*: Scrubbing is more expensive in terms of energy than refresh as it needs to stream the data on the memory bus. However, scrubbing is performed much less frequently than refresh, so its overall contribution to the system energy is quite small. The total energy to refresh an 8GB DIMM once is approximately 1.1mJ, whereas the energy for one scrub operation is approximately 161mJ (150x more). However, the scrub energy is spent four orders of magnitude less frequently (15 minutes vs. 64ms) than the refresh energy. Table I shows the total energy consumed by scrubbing compared to that consumed by refresh, as the scrub interval is varied from four minutes to one hour. Scrubbing with a 15-minute interval, as we assumed in our evaluations, causes only a 1% increase in energy compared to refresh energy.

VII. PERFORMANCE AND ENERGY ANALYSIS

We use a detailed memory system simulator, USIMM [8]. We model a quad-core system operating at 3.2GHz connected to a DDR3-1600 (800MHz) memory system. As refresh overheads increase with technology scaling, we analyze DRAM chips with density varying from 8Gb to 64Gb. The memory system consists of four DIMMs, so the total size of the memory system ranges from 32GB (for 8Gb chips) to 256GB (for 64Gb chips). The baseline system employs JEDEC-specified 64ms refresh. We increase the refresh cycle time linearly with density (T_{RFC} varies from 350ns to 2800ns).

We evaluate all the workloads provided by USIMM for the Memory Scheduling Championship [2]. These workloads are memory intensive and exercise a large number of transactions between the memory system and the core [35]. These 18 workloads come from various suites, including SPEC(2), PARSEC(9), BioBench(2), and Commercial(5). We report an average over all the 18 workloads.

As the effectiveness of refresh savings with AVATAR depends on time elapsed after retention time testing is performed, we evaluate three designs: AVATAR-1, AVATAR-120, and AVATAR-360 representing one day, 120 days, and 360 days after a retention time test, respectively. We also show results of a theoretical scheme that does not perform refresh.

A. Speedup

Figure 16 shows the speedup for AVATAR-1, AVATAR-120, AVATAR-360, and *No Refresh* over the JEDEC specified refresh scheme. The performance benefit of eliminating refreshes increases with chip density, going from 4% at the 8Gb node to 54% at the 64Gb node (as denoted by the *No Refresh* bars). AVATAR provides about two-thirds of the performance benefit of *No Refresh*. Even after a year of continuous operation, AVATAR maintains most of the performance benefits close to that of the first day after retention testing. For instance,

AVATAR improves performance by 35%, even a year after retention time testing.

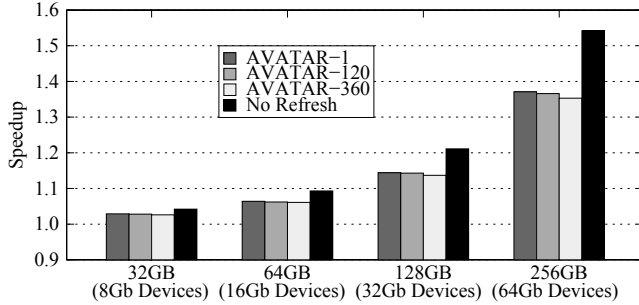


Fig. 16. Speedup from refresh savings. The performance of AVATAR improves with technology node.

B. Energy-Delay Product

Refresh operations not only cost performance but also consume energy. Figure 17 shows the Energy Delay Product (EDP) of AVATAR-1, AVATAR-120, AVATAR-360, and *No Refresh* compared to the JEDEC specified refresh scheme. The energy benefits of eliminating refreshes also increase with high density. *No Refresh* potentially reduces the EDP by 68% at the 64Gb node. AVATAR-1 reduces EDP by 8%, 16%, 31%, and 55% for the 8Gb, 16Gb, 32Gb, and 64Gb nodes, respectively. AVATAR-360 has EDP savings close to those of AVATAR-1.

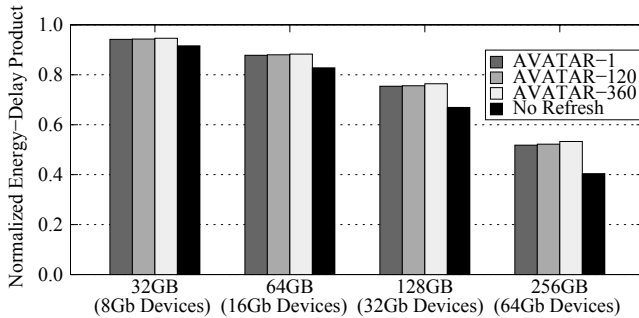


Fig. 17. Energy Delay Product. The savings in Energy Delay Product increases with technology node.

Overall, our analysis shows that AVATAR is not only effective at improving reliability by orders of magnitude, but it also obtains most of the performance and energy benefits of an otherwise-unreliable multirate refresh scheme.

VIII. RELATED WORK

To the best of our knowledge, this is the first work to comprehensively study and model the effect of VRT cells on multirate refresh mechanisms. We provide a new analytical model showing that relying only on ECC to correct VRT failures can result in an unacceptable rate of data loss. Our VRT-aware multirate refresh mechanism can guarantee reliable operations in the presence of VRT failures. In this section, we discuss prior works that have proposed different mechanisms to mitigate the negative effects of DRAM refresh operations. We also discuss prior works on profiling VRT failures.

A. Lowering Refresh Rate

Prior works on minimizing refresh overhead by extending the refresh interval can be categorized into three classes:

Profiling Based: Profiling based-multirate refresh works exploit the non-uniformity in retention time of DRAM cells to mitigate the problem of DRAM refresh (e.g., [4, 21, 28, 36, 38, 41, 44]). These multirate refresh schemes group rows into different bins based on an initial retention time profiling and apply a higher refresh rate only for rows belonging to the lower retention time bin. They depend on a simple profiling mechanism at the initial stage to detect the rows with retention failures and place them into the lower retention time bin. Their key assumption is that the retention time profile of DRAM cells does not change at runtime. However, all these mechanisms will result in unacceptable data loss because of the VRT failures [29]. These mechanisms can potentially use ECC to mitigate the VRT failures. However, we show that simply relying on SECDED ECC cannot provide an acceptable reliability guarantee. To mitigate the VRT failures with ECC, we may need stronger ECC codes, which significantly increase system cost.

ECC Based: Prior work proposed to minimize the refresh overhead by extending the refresh interval and using higher strength ECC (5EC6ED) to correct the retention failures [42]. However, to reduce the cost of ECC, this work proposes to amortize the ECC cost by protecting larger chunks of data (1KB). Thus, this mechanism has significant bandwidth and performance overheads as it reads the entire 1KB chunk of data at every access to verify/update ECC.

Software Hint Based: Software-hint based refresh mechanisms rely on software/OS hints on the criticality or error-vulnerability of program data. They lower the refresh rate or reliability of DRAM for non-critical or invalid regions [11, 30, 31]. These mechanisms cannot fully exploit the non-uniformity of the retention time across the chip as only a restricted fraction of memory can benefit from reduced refreshes.

B. Refresh Scheduling

Prior works proposed to reduce performance overhead of refreshes by scheduling refresh operations in a flexible way that reduces their interference with program accesses [6, 12, 35, 40]. Our work is complementary to these works as these mechanisms propose to minimize refresh overhead at the nominal refresh rate. All these techniques are applicable to our mechanism that reduces refresh overhead by extending the refresh interval for most memory rows.

C. Profiling for VRT

Although the VRT phenomenon has been widely studied in the literature [7, 10, 13, 20, 32, 33, 37, 43, 45], only recent works discuss issues in retention time profiling in the presence of VRT cells [19, 29]. Khan et al. studied the effectiveness of multi-round testing, guard-banding and different-strength ECC codes at tolerating VRT failures [19]. Their work does not propose any analytical models or mechanisms to enable realistic multirate refresh in the presence of VRT. Another prior work uses profiling to detect retention failures whenever the module enters the self-refresh mode [3]. This work cannot guarantee data integrity as VRT failures can occur after testing [19, 29].

IX. CONCLUSIONS

Multirate refresh schemes exploit the non-uniformity in retention times of DRAM cells to reduce refresh operations. While multirate refresh schemes are highly effective at reducing refresh operations, they are plagued by the fact that the retention time of a DRAM cell tends to vary at runtime due to the Variable Retention Time (VRT) phenomenon. VRT causes multirate refresh to suffer from data errors, even with the use of ECC DIMMs, which makes it a challenge to deploy multirate refresh mechanisms reliably in practice. This paper introduces AVATAR, the first practical, effective, and reliable multirate refresh scheme. This paper makes the following contributions:

- 1) We characterize the behavior of VRT cells and develop an architecture-level model to analyze multirate refresh in the presence of VRT.
- 2) Using our model, we show that a VRT-agnostic approach that relies on the use of ECC DIMMs to correct VRT failures leads to an unacceptable rate of a data loss.
- 3) We propose AVATAR, a VRT-aware multirate refresh mechanism that adaptively changes the refresh rate to handle VRT failures at runtime. It improves reliability of multirate refresh to tens of years while maintaining most of the refresh savings and performance and energy benefits of multirate refresh.

We show that AVATAR reduces refresh operations by 62%-72% for a DRAM system without incurring any additional hardware changes than what is required for multirate refresh and ECC modules. This refresh reduction leads to an approximately 35% performance improvement and 55% energy-delay product reduction with 64Gb DRAM chips. We conclude that AVATAR is a highly-effective and simple multirate refresh mechanism that provides correct DRAM operation even in the presence of VRT failures.

ACKNOWLEDGMENTS

Chris Wilkerson contributed substantially to this work. We thank him for his insightful feedback and discussions that have helped shape this work significantly. We also thank the anonymous reviewers for providing useful feedback on our work. This work was supported in part by NSF grants 1319587, 1212962, 0953246, 1320531, 1065112, the Intel Science and Technology Center on Cloud Computing, and the Center for Future Architecture Research (C-FAR), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

REFERENCES

- [1] "Soft-errors in electronic memory - A white paper," *Tezzaron Semiconductor*, 2004.
- [2] "Memory scheduling championship (MSC)," 2012.
- [3] J.-H. Ahn *et al.*, "Adaptive self refresh scheme for battery operated high-density mobile DRAM applications," in *ASSCC*, 2006.
- [4] S. Baek *et al.*, "Refresh now and then," *IEEE TC*, 2013.
- [5] L. Borucki *et al.*, "Comparison of accelerated DRAM soft error rates measured at component and system level," in *IRPS*, 2008.
- [6] K. K.-W. Chang *et al.*, "Improving DRAM performance by parallelizing refreshes with accesses," in *HPCA*, 2014.
- [7] M. Chang *et al.*, "Si-H bond breaking induced retention degradation during packaging process of 256 Mbit DRAMs with negative wordline bias," *IEEE TED*, 2005.
- [8] N. Chatterjee *et al.*, "USIMM: the Utah Simulated Memory Module," *Tech. Rep.*, 2012.
- [9] N. El-Sayed *et al.*, "Temperature management in data centers: Why some (might) like it hot," in *SIGMETRICS*, 2012.
- [10] T. Hamamoto *et al.*, "On the retention time distribution of dynamic random access memory (DRAM)," *IEEE TED*, 1998.
- [11] C. Isen and L. John, "ESKIMO - Energy savings using semantic knowledge of inconsequential memory occupancy for DRAM subsystem," in *ISCA*, 2009.
- [12] Y. Ishii *et al.*, "High performance memory access scheduling using compute-phase prediction and writeback-refresh overlap," in *JILP Workshop on Memory Scheduling Championship*, 2012.
- [13] Y. Ito and H. Iwai, "Data storing method of dynamic ram and semiconductor memory device," US Patent No. 6697992 B2, 2004.
- [14] B. Jacob *et al.*, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers Inc., 2007.
- [15] JEDEC, *DDR3 SDRAM Specification*, 2012.
- [16] —, *High Bandwidth Memory (HBM) DRAM Specification*, 2013.
- [17] —, *WIDE-IO DRAM Specification*, 2013.
- [18] U. Kang *et al.*, "Co-architecting controllers and DRAM to enhance DRAM process scaling," in *The Memory Forum, ISCA*, 2014.
- [19] S. Khan *et al.*, "The efficacy of error mitigation techniques for DRAM retention failures: A comparative experimental study," in *SIGMETRICS*, 2014.
- [20] H. Kim *et al.*, "Characterization of the variable retention time in dynamic random access memory," *IEEE TED*, 2011.
- [21] J. Kim and M. Papaefthymiou, "Block-based multi-period refresh for energy efficient dynamic memory," in *ASIC*, 2001.
- [22] K. Kim and J. Lee, "A new investigation of data retention time in truly nanoscaled DRAMs," *IEEE EDL*, 2009.
- [23] Y. Kim *et al.*, "A case for exploiting subarray-level parallelism (SALP) in DRAM," in *ISCA*, 2012.
- [24] —, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *ISCA*, 2014.
- [25] D. Lee *et al.*, "Adaptive-latency DRAM: Optimizing DRAM timing for the common-case."
- [26] —, "Tiered-latency DRAM: A low latency and low cost DRAM architecture," in *HPCA*, 2013.
- [27] X. Li *et al.*, "A memory soft error measurement on production systems," in *USENIX ATC*, 2007.
- [28] J. Liu *et al.*, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.
- [29] —, "An experimental study of data retention behavior in modern DRAM devices: Implications for retention time profiling mechanisms," in *ISCA*, 2013.
- [30] S. Liu *et al.*, "Flicker: Saving DRAM refresh-power through critical data partitioning," in *ASPLOS*, 2011.
- [31] Y. Luo *et al.*, "Characterizing application memory error vulnerability to optimize datacenter cost via heterogeneous-reliability memory," in *DSN*, 2014.
- [32] R. L. Meyer and R. Beffa, "Method of reducing variable retention characteristics in DRAM cells," US Patent No. 6898138 B2, 2005.
- [33] Y. Mori *et al.*, "The origin of variable retention time in DRAM," in *IEDM*, 2005.
- [34] O. Mutlu *et al.*, "The main memory system: Challenges and opportunities," *KHISE*, 2015.
- [35] P. Nair *et al.*, "A case for refresh pausing in DRAM memory systems," in *HPCA*, 2013.
- [36] P. J. Nair *et al.*, "ArchShield: Architectural framework for assisting DRAM scaling by tolerating high error rates," in *ISCA*, 2013.
- [37] P. J. Restle *et al.*, "DRAM variable retention time," in *IEDM*, 1992.
- [38] D.-Y. Shen *et al.*, "SECRET: Selective error correction for refresh energy reduction in DRAMs," in *ICCD*, 2012.
- [39] C. Shirley and W. Daasch, "Copula models of correlation: A DRAM case study," *IEEE TC*, 2013.
- [40] J. Stuecheli *et al.*, "Elastic refresh: Techniques to mitigate refresh penalties in high density memory," in *ISCA*, 2010.
- [41] R. K. Venkatesan *et al.*, "Retention-aware placement in DRAM (RAPID): Software methods for quasi-non-volatile DRAM," in *HPCA*, 2006.
- [42] C. Wilkerson *et al.*, "Reducing cache power with low-cost, multi-bit error-correcting codes," in *ISCA*, 2010.
- [43] R. Yamada *et al.*, "Dynamic random access memories and method for testing performance of the same," US Patent No. 7450458 B2, 2008.
- [44] K. Yanagisawa, "Semiconductor memory," US Patent No. 4736344 A, 1988.
- [45] D. Yaney, *et al.*, "A meta-stable leakage phenomenon in DRAM charge storage-variable hold time," in *IEDM*, 1987.