IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 19, 2019, accepted January 6, 2020, date of publication January 17, 2020, date of current version January 28, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2967217

CAT-TWO: Counter-Based Adaptive Tree, Time Window Optimized for DRAM Row-Hammer Prevention

INGAB KANG[1], EOJIN LEE[1], AND JUNG HO AHN[1,2], (Senior Member, IEEE)
[1]Department of Transdisciplinary Studies, Seoul National University, Seoul 08826, South Korea
[2]Inter-University Semiconductor Research Center, Seoul National University, Seoul 08826, South Korea

Corresponding author: Jung Ho Ahn (gajh@snu.ac.kr)

This work was supported in part by the IDEC (EDA Tool), in part by the NRF of Korea Grant (NRF2017R1A2B2005416), and in part by the Research and Development Program of MOTIE/KEIT (10077609).

ABSTRACT Row-hammering flips bits in a victim DRAM row by frequently activating its adjacent rows, compromising DRAM integrity. Several studies propose to prevent row-hammering by counting the number of activates to a DRAM row and refreshing the corresponding victim rows before the count surpasses a row-hammer threshold. However, these approaches either incur a significant area overhead or a large number of additional activations (ACT) that could degrade the system performance. In this paper, we propose CAT-TWO, a time-window-optimized version of the existing Counter-based Adaptive Tree (CAT) scheme for row-hammer prevention. We first ensure that the victim rows are always refreshed at the last level of the tree without counter overflow by configuring the threshold and the number of CAT-TWO counters based on the fact that the maximum number of ACTs is limited within the refresh window. We further reduce the size and latency of CAT-TWO by applying high-radix rank-level CAT-TWO with multiple tree roots. CAT-TWO incurs less than 0.7% energy overhead on a baseline DDR4 DRAM device, and generates less than 0.03% additional ACTs to refresh victim rows in the worst case, which hardly affects system performance.

INDEX TERMS DRAM chips, DRAM reliability, row-hammering.

I. INTRODUCTION

In modern computer systems, DRAM has been the de facto standard for main memory for decades because it offers a large capacity at a small cost compared to its alternatives. DRAM stores data in rows of DRAM cells and a cell uses a single capacitor to store a single bit of data. If a cell is fully charged, the data would be one; if the cell is not charged, the data would be zero. However, the use of capacitors makes DRAM dynamic; charge leaks out of DRAM cells, and if they are not refreshed within a specific time period, the data would be lost [2]. To prevent the loss of data, DRAM standards [11], [12] define tREFW, the time interval within which a DRAM row must be refreshed to retain data.

Recently, due to process scaling, it has been found that activating a DRAM row (aggressor row) creates a disturbance to its neighboring rows (victim rows), which expedites the charge loss. If the disturbance accumulates enough due to repeated activation (ACT), the neighboring rows eventually lose charge earlier than tREFW, and hence data is flipped. This phenomenon is called row-hammering [17], [22], and since its discovery, the exploitation and protection of row-hammering have been an active area of research. Row-hammering was initially a DRAM reliability problem, but because an attacker could choose to target a specific row to flip bits, it was soon shown that row-hammering is also a system security problem as it could be leveraged to attack systems [3], [5], [18], [24], [25], [32].

Numerous solutions have been proposed to mitigate row-hammering [7], [15]–[17], [19], [20], [26], [27], [29], [33], [34], where these previous row-hammer mitigation schemes can be classified as probabilistic or counter-based (deterministic). Probabilistic solutions prevent row-hammering with very high probability and with minimum hardware overhead, but counter-based solutions provide better protection as they can guarantee that a row under attack is refreshed before its neighbor rows are activated by a certain number of times (called a row-hammer threshold).

The associate editor coordinating the review of this manuscript and approving it for publication was Stavros Souravlas.

Furthermore, because counter-based solutions can carry out targeted refreshes, the number of additional row ACTs incurred by the deterministic solutions can be smaller. However, deterministic solutions often require a large storage table because they have to count the number of ACTs to a specific row.

Among the schemes which have been proposed to mitigate this storage overhead problem [15], [19], [20], [26], [27], Counter-based Row Activation (CRA) [15] suggests to store the counters in DRAM and bring the data to an on-chip cache whenever a DRAM ACT command is sent. This scheme relies on memory access locality; when DRAM accesses concentrate on a small number of DRAM rows at a given time, a small cache would be sufficient to catch the majority of ACT commands. However, it incurs additional DRAM accesses to manage CRA when it misses the cache and hence performs poorly for applications with the limited locality.

Other proposals target to reduce the total number of counters and reduce the size of tables, where Counter-based Adaptive Tree (CAT) [26], [27] and Time Window Counters (TWiCe) [19], [20] are representative. CAT proposes to reduce the number of counters by counting ACTs per group of DRAM rows. It further divides the group into subgroups when the number of ACTs passes a threshold. In this way, groups that are activated more frequently are split into smaller groups of rows, thereby tracking rows that are more likely to cause row-hammering in a fine-grained manner, and saving counters on groups that are not frequently activated and less likely to cause row-hammering. However, if CAT runs out of counters or if it cannot divide the row groups into small enough groups, it must refresh a handful of rows at once because it cannot discern which exact row in the row group is a row-hammer victim.

TWiCe takes a different approach to reduce counters. TWiCe maintains a one-counter-per-row approach but reduces the total number of necessary counters by periodically pruning rows that are not activated frequently enough and therefore deemed innocuous. This intuition stems from the observation that the maximum frequency of ACTs to a DRAM bank is limited by tRC, which limits the number of ACTs within tREFW. Therefore, if a row is not activated frequently enough, it cannot be activated enough to trigger row-hammering within tREFW. However, while this approach cuts down the total number of counters to a few hundred counters, it requires a larger table compared to CAT.

To maintain a small table size while detecting row-hammer aggressors with high accuracy, we propose CAT-TWO, a Time Window Optimized version of CAT. CAT requires a smaller table compared to TWiCe, but it cannot pinpoint row-hammer aggressor rows. By contrast, CAT-TWO increases the number of tree levels such that the last level groups (leaves) correspond to individual DRAM rows. It also ensures that all the refreshes for row-hammer prevention occur at the last level of the tree by provisioning CAT-TWO with the exact number of counters required to prevent counters from overflowing. The number of counters needed is deduced by borrowing the intuition from TWiCe that the number of ACTs within tREFW is limited, which limits the number of tree splits and so the number of counters. As the number of counters differs by how the tree level thresholds are set, we conduct a sensitivity study to explore when the number of counters is minimized as we vary the threshold values of the tree. Through the study, we configure CAT to have equally spaced thresholds throughout all levels.

We optimize CAT-TWO to require fewer counters and to reduce the table size by populating multiple roots, deploying CAT-TWO per rank (not per bank), and changing the CAT table structure. Because the spacing (number of counts) between level thresholds are inversely proportional to the number of levels, we configure CAT-TWO to have multiple roots. Initializing a tree to have multiple roots reduces the levels needed for the last level to have only one associated row. This increases the spacing between level thresholds and reduces the maximum number of splits and hence the maximum number of counters. We further reduce the size of CAT-TWO by deploying CAT-TWO per rank. Because the maximum frequency of ACTs to a DRAM rank is smaller than the sum of maximum frequency of ACTs to all banks in a rank, by consolidating all CAT-TWO tables that were deployed per bank into one large CAT-TWO table per rank, we can reduce the maximum number of ACTs by 46%, reducing the table size. Lastly, we propose to unify the counter table and the search table to reduce the size of the tables and eliminate one surplus table access.

Deploying CAT-TWO per rank reduces the interval between two ACTs, which increases CAT-TWO clock speed to around 4 GHz. As this speed is too high for a memory controller or DRAM devices, it must be reduced to match DRAM clock speed. We reduce the clock speed of CAT-TWO by implementing the CAT-TWO tree as a high-radix tree. Because a high-radix tree split a counter into more than two children every time a level threshold is reached, fewer levels are needed to divide row groups to a single row. Therefore, high-radix trees require fewer table accesses between ACTs, reducing the clock speed of CAT-TWO.

Our analysis shows that CAT-TWO requires less than half the size of TWiCe, only requiring a 1.2 KB table per 1 GB DRAM bank. When running multi-programmed workloads and synthetic workloads emulating row-hammer attack scenarios, CAT-TWO performs on par with TWiCe, incurring little to no additional ACTs. The cost of maintaining CAT tables was negligible, requiring only 0.7% more energy on table updates and 0.2% more energy per table reset.

In this paper, we make the following key contributions:

- We propose CAT-TWO, a time window optimized version of CAT that minimizes the number of additional ACTs while retaining a small table size, a key benefit of CAT. CAT-TWO performs on par with TWiCe, whereas requiring half the table size.
- We show that the size of CAT-TWO can be reduced by employing multiple roots, deploying CAT-TWO per

rank, not bank, and fusing its search and counter tables without an increase in the number of additional ACTs.
- We reduce the clock speed of CAT-TWO by employing a high-radix tree.

II. BACKGROUND

A. DRAM BASICS

A DRAM rank is a collection of DRAM chips that operate in tandem; A DRAM bank is an array of DRAM cells within a rank that can load and store data independent of other banks [9], [14]. A single DRAM cell is connected to a local wordline (WL) and a bitline (BL), and the local WLs are connected to the global WL. The DRAM cells connected to the same WL form a DRAM row. When the data in a row is accessed, the memory controller must first send a row activate command to raise the WL voltage to be high and connect the corresponding cells to the BLs. Before the cells are connected, the BLs are at a precharged state, with its voltage at VDD/2, but when the cells are connected, the charge in the cells flows to the BLs and causes a disturbance, which is then amplified by the BL sense amplifiers (BLSA) at the bottom of the BLs. Because the charge in a cell is lost in the course of distribution when the WL voltage is raised, it must be restored when BLSAs amplify the charge. After a row is activated, to access a different row, the BLs are first precharged (PRE) before a different row is activated. The minimum time that two consecutive ACT commands to the same bank can be issued is tRC, which limits the maximum frequency of ACTs on a bank. However, ACT frequency on a rank is limited by row-to-row delay (tRRD) and four activate window (tFAW). tRRD restricts the minimum time interval between two ACT commands to a rank, and tFAW is the time window in which no more than four ACT commands could be sent to a rank.

Another characteristic of DRAM is that it must be refreshed in a specific time period. DRAM cells leak charge over time, and therefore the charge must be refreshed before the data is lost. This time period within which a DRAM cell must be refreshed is called a refresh window (tREFW). As there are many rows in a DRAM bank, it is impossible to refresh all rows of a bank at the same time. Therefore, DRAM refreshes are distributed across tREFW where a subset of the rows in a bank is refreshed every DRAM refresh interval (tREFI) for the duration of a row refresh cycle time (tRFC). However, with the discovery of row-hammering [17], it has been found that activating rows causes disturbances to neighboring rows, expediting the loss of charge in DRAM cells. If an aggressor row is activated frequently enough within tREFW, the charge in neighboring rows can have its data flipped before being refreshed. This phenomenon poses a severe breach to DRAM integrity as it means data in rows can be tainted even without being directly accessed. In order to mitigate row-hammering, rows adjacent to a row that is activated frequently must be refreshed by activating the rows before bit flips.

B. PREVIOUS ROW-HAMMER SOLUTIONS

PARA: Probabilistic Adjacent Row Activation (PARA) is a probability-based row-hammer protection scheme [17]. PARA protects against row-hammering by activating adjacent rows to a row that has been activated with a low probability. Even though the adjacent rows are activated with a low probability, as the number of times that a row is activated increases, the probability that the adjacent rows are refreshed becomes higher. Therefore, by the time it is likely that a row would cause row-hammering to adjacent rows, it is highly likely that the adjacent rows would have been refreshed. However, this approach often causes more refreshes than counter-based solutions because it must also refresh rows that are not likely to have their bits flipped (false positives).

TWiCe: TWiCe is a per-row counter-based row-hammer prevention solution based on the intuition that the number of ACTs within tREFW is limited [19]. In order to cause row-hammering, a row must be activated frequently enough such that it must surpass a row-hammer threshold (RH_{th}), a threshold beyond which adjacent rows are deemed unsafe, and therefore must be refreshed. However, because all rows of a bank are refreshed within tREFW, if RH_{th} is not reached within tREFW, row-hammering could not occur. Therefore, if we monitor all ACTs to individual rows, but periodically prune out rows that are not being activated frequently enough to reach RH_{th} within tREFW, we can count ACTs per row while bounding the total size of counter tables down to the number of potential rows that could reach RH_{th}. A model TWiCe suggested in [19] requires only 553 counters per 1 GB of DRAM bank, which is orders of magnitude fewer than the total number of rows per bank (65,536 rows). However, TWiCe requires a larger table than CAT, therefore in certain situations where reducing die size is crucial, CAT could be a better row-hammer mitigation solution.

CAT: CAT is a counter-based solution where row-hammering is mitigated by counting the ACTs to a group of DRAM rows and refreshing the group of rows and their adjacent rows when the count reaches RH_{th} [27]. Especially, CAT grows a tree of counters to determine how many rows are counted by a counter according to the frequency of ACTs. For example, CAT begins by allocating a single counter at the root of the tree for the entire rows in a DRAM bank. Each counter holds an ACT_{CNT} value, and each time that an ACT command is sent to a row associated with a counter, the corresponding ACT_{CNT} is increased by one. If the ACT_{CNT} of a counter reaches a level threshold, the group is split into two child counters initialized with the ACT_{CNT} of the parent counter. The number of splits from the root counter to the currently active counter is called the level of the counter, with the root counter having the lowest level 0 and the last level having the highest level. Whenever a counter splits, the counter's level is increased by 1. We denote the level threshold of a counter at level n as Lv_{th}[n]. As the splitting continues, the counters span a tree where counters that are frequently activated are at the higher levels, covering fewer rows per counter. By contrast, less-frequently activated counters are at

the lower levels, covering more rows per counter. If a counter is at the last level L, and Lv_{th}[L] is reached, all the rows in that counter and the two rows adjacent to the row group is refreshed, and ACT_{CNT} is reset to zero. Lv_{th}[L] is named the refresh threshold as it is unique in that this threshold does not cause the counter to split, but refreshes the rows that belong to the counter. Counters that are not at level L continues to split every time Lv_{th}[n] is reached until there are no free counters left in CAT, in which case Lv_{th}[n] of all counters are set to Lv_{th}[L], and they are refreshed once they reach Lv_{th}[L]. The CAT tree is reset back to a single counter every tREFW as all the rows within a bank would have been refreshed once.

The rationale behind this scheme is that the more frequently activated rows are more likely to cause row-hammering; therefore, they are allocated more counters and counted at a more fine-grained level. Thus, only a small number of rows would be refreshed when a group reaches RH_{th}. However, if not provisioned with enough levels and counters, CAT must refresh rows in groups too, whereas TWiCe counts and targets potential aggressors per row. Therefore, CAT incurs a high additional ACT overhead when compared to TWiCe.

III. EXPLORING CAT

Although CAT requires a smaller table size than TWiCe, it incurs too many additional ACTs compared to TWiCe on adversarial memory access patterns [19]. However, CAT could be improved to significantly reduce the number of additional ACTs by slightly increasing its table size. Prior to showing how CAT could be optimized, we first show why CAT incurs excessive ACTs and describe the structure of CAT in detail.

A. PROFUSE REFRESHES

[27] sets the number of levels, the level thresholds, and the maximum number of counters without considering the number of additional ACTs CAT incurs. Having a sufficient number of levels is crucial because it affects how many rows are refreshed together when a counter is at the last CAT level. For example, in [27] the number of levels is set to 11 for a DRAM bank with 65,536 ($= 2^{16}$) rows, which means 32 ($= 2^{16-11}$) rows that belong to a group and two more adjacent rows are refreshed at the same time when $Lv_{th}[L = 10]$ is reached for a counter at the last level. Moreover, when DRAM rows are remapped to correct faulty rows [4], [8], [30], the group of rows may not be physically contiguous in DRAM, in which case additional rows must be refreshed to account for the remapped rows.

If an attacker attempts to compromise DRAM integrity, a simple row-hammer attack would be to repeatedly activate one specific row as often as possible. Figure 1 shows the number of ACTs each row-hammer mitigation scheme incurs to refresh rows under this single-row attack scenario. Although CAT is a counter-based solution, it incurs 17 times more ACTs than TWiCe, even higher than PARA, to prevent row-hammering. Because such a high number of additional ACTs defeats the purpose of a counter-based solution, CAT must be improved to incur fewer additional refreshes.

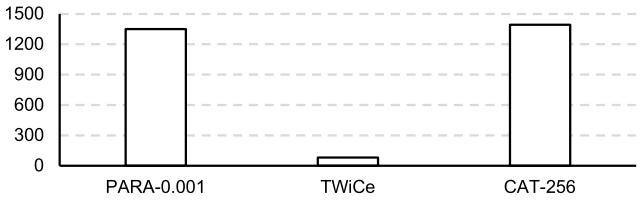

FIGURE 1. The number of additional ACTs per DRAM bank of PARA, TWiCe, and the original CAT [27] when a single row is repeatedly activated. PARA refreshes adjacent rows with 0.001 probability upon each ACT (PARA-0.001) and CAT employs 256 counters with 11 levels (CAT-256).

Another crucial element of CAT is that it must have a sufficient number of counters to prevent profuse refreshes. If CAT is implemented with a small number of counters, it could experience a case that a counter reaches its level threshold when all the available counters are occupied, and hence CAT cannot further split the counter. In this case, the counters that cannot split could reach RH_{th} while maintaining a large row group, requiring CAT to refresh the entire row group and two adjacent rows to the group to prevent row-hammering, which incurs a large number of additional ACTs. In the worst case, if an attacker recognizes that DRAM is protected against row-hammering using CAT, the attacker could split all the rows in one side of a tree to deplete CAT's free counters and start activating the rows in the other side, in which case, half the rows in a DRAM bank ($= 2^{16-1}$ rows) would have to be refreshed when $Lv_{th}[L]$ is reached, as shown in [19].

B. CAT IMPLEMENTATION

[27] proposes to implement CAT as a linked list; to access a counter, you must iterate through a table that records the tree structure (search table) to access the address of the counter, which is stored in a separate table (counter table). As CAT begins with one counter and for every split, one counter and one split entry is added to the counter table and the search table, respectively. The number of counters is always one more than the number of splits, so for a CAT with a maximum of M counters, a search table with a size of M-1 entries are needed. An exemplar CAT table is shown in Figure 2b.

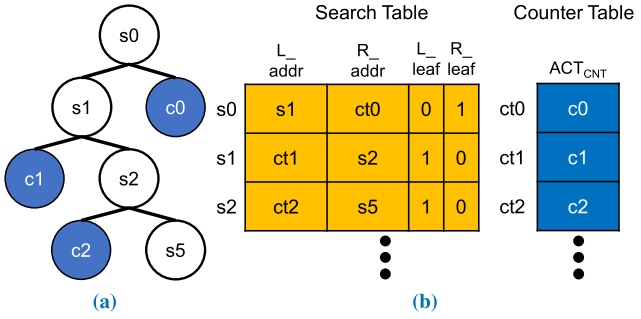

FIGURE 2. An exemplar CAT tree (left) and its corresponding CAT tables (right).

A single search table entry is comprised of two fields and two flags; (L_addr, R_addr) fields and (L_leaf, R_leaf) flags. L_addr and R_addr point to the next table entries whereas L_leaf and R_leaf represent whether the next entry could be found in the search table or the counter table. A single counter table entry only holds ACT_{CNT}, which records the number of ACTs to a row group.

An exemplar CAT tree and the corresponding CAT table is shown in Figure 2a and 2b. Once the target row address A ($0 \leq A <$ (number of rows in a bank)) of a row ACT command is received by CAT, CAT must increment the ACT_{CNT} of $c2$ in the counter table entry $ct2$, which counts the ACTs to the row group of row A. To access $c2$, search table entry $s0$ is first accessed, and the first bit of A is used to determine whether $c2$ is in the left node or the right node. As $c2$ is in the left node, L_leaf flag is checked. Seeing that L_leaf is 0, CAT uses L_addr value to access the next entry in the search table ($s1$), and the next bit in address A is used to determine which node $c2$ belongs to in this entry. CAT iterates through this process until it finds the entry in which $\_leaf$ flag is 1 ($s2$), indicating that $\_addr$ field points to a counter in the counter table. Therefore, $\_addr$ is used to access the counter table, finally accessing $c2$ and incrementing the ACT_{CNT}. By implementing CAT as a linked list, the maximum time to access a counter entry in CAT becomes directly proportional to the maximum level of the CAT tree because one additional search tree access is required per counter level increase. This makes CAT scalable in size, as the latency to access a counter is proportional to the number of accesses to CAT, which is determined by the levels of CAT rather than the number of entries in the CAT table. In Section IV-D, we show how this property can be exploited to reduce the table size of CAT.

C. EVALUATING CAT WITH RESPECT TO TIME WINDOW

The reason why CAT incurs the most additional ACTs in Figure 1 and in [19] is that CAT has to refresh rows in groups, whereas TWiCe can pinpoint which exact rows have the potential to cause row-hammering and refresh just two adjacent rows whenever the refresh threshold is reached. If CAT is constructed with 17 levels (levels 0-16) for a DRAM bank with 65,536 ($= 2^{16}$) rows, a counter at the 17th (level 16) level would be associated with a single DRAM row as the number of rows per counter is halved every level. Therefore, we only need to refresh the two adjacent rows adjacent to the row when $Lv_{th}[L]$ is exceeded for a counter at the 17th level. However, increasing the number of levels also decreases the number of ACTs between level thresholds, which increases the number of splits. Therefore, increasing the number of levels makes it more likely that all counters in CAT would be active, aggravating the problem outlined in Section III-A where a very large row group is allocated to a single counter because counters cannot be split further.

To prevent CAT counters from being overflowed as the number of levels increase, CAT must be provisioned to the maximum possible number of counters within tREFW. To evaluate precisely how many counters are needed, we leverage the intuition from TWiCe that the number of ACTs within a refresh window is bounded. Because one additional counter is needed every time an existing counter reaches $Lv_{th}[n]$ and splits, the maximum number of counters that could be used in tREFW is (maximum number of splits in tREFW) + (initial counters). If we can calculate how many splits could occur using the maximum number of ACTs in tREFW, we can deduce the number of counters CAT needs to stop its counters from being overflowed.

We name this time window optimized version of CAT as CAT-TWO, which differs from CAT in that it refreshes rows on a per-aggressor-row basis, without running out of counters to allocate whenever a counter needs to be split. To calculate the exact number of counters that would stop CAT-TWO from overflowing, we should first decide how to set $Lv_{th}[n]$.

D. CAT-TWO COUNTERS VERSUS $LV_{TH}[n]$

We conduct a $Lv_{th}[n]$ sensitivity analysis, evaluating the maximum number of counters needed depending on how $Lv_{th}[n]$ is set. We configure the difference between consecutive $Lv_{th}[n]$ ($\Delta[n] = Lv_{th}[n] - Lv_{th}[n-1]$) to be an arithmetic sequence and set the common difference to three different cases; when the common difference is smaller than zero, when the common difference is zero (when $\Delta[n] = \Delta[n-1]$), and when the common difference is larger than zero. Figure 3a shows the change in $Lv_{th}[n]$ when $\Delta[n]$ is decreasing. Setting $\Delta[n]$ to be $a + n \times d$, when d (common difference of arithmetic sequence) is less than 0, $\Delta[n]$ decreases as the level goes higher; therefore counters in the higher levels require fewer ACTs to split. This trend flips when d is greater than 0, where $\Delta[n]$ increases as the level gets higher, and when d is 0, $\Delta[n]$ is the same regardless of the counter's level. With $\Delta[n]$ set as an arithmetic series, the maximum number of splits in tREFW can be calculated by using a greedy algorithm: a counter with the least ACTs to $Lv_{th}[n]$ is chosen to be split repeatedly until the maximum number of ACTs within tREFW is reached. When $d < 0$, a counter at the highest level is chosen to be split over other counters, and when $d > 0$, a counter at the lowest level is chosen to be split. Because $\Delta[n]$ is the same across all levels when $d = 0$, any counter can be chosen in this case.

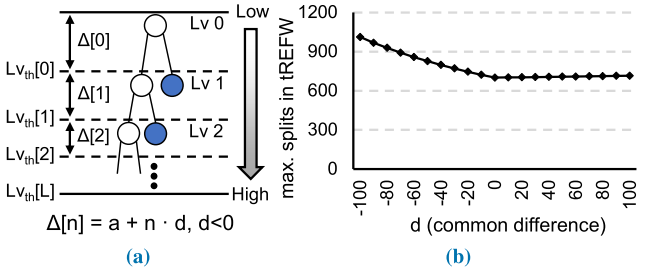

FIGURE 3. The Figure of a CAT-TWO with L levels when $\Delta[n]$ is an arithmetic sequence with a negative common difference (d) (left). Graph of the maximum number of splits in tREFW versus d (right). The maximum number of splits steadily decreases as d is increased from -100 to 0 and steadily increases as d is increased from 0 to 100.

Figure 3b depicts the maximum number of splits within tREFW when d is increased from -100 to 100. The number of splits steadily decreases as d gets close to 0 and starts rising as d passes zero because if there are levels that have lower $\Delta[n]$ than other levels, ACTs can be focused on those counters to incur more splits with fewer ACTs. As the capacity to hold counters is doubled with each level, this phenomenon is more prevalent when the higher levels need fewer ACTs to split. Therefore, more counters can be split with fewer ACTs, and less prevalent when the higher levels need fewer ACTs to split. This is why the number of splits increases more steeply when d is decreased from 0 rather than when it is increased from 0.

Therefore, the maximum number of splits is minimized when the $\Delta[n]$ is the same throughout all levels. Setting the equal difference between $Lv_{th}[n]$ brings another benefit; it is easy to analyze how many splits could occur in tREFW. When the $\Delta[n]$ is the same throughout all levels, it does not matter which counter is chosen to split as all counters would require the same amount of ACTs to split. Therefore, the maximum number of splits would be (# of ACTs in tREFW)/a and the number of counters would be:

$$\#\ \text{counters} = \frac{\#\ \text{ACTs in tREFW}}{\Delta} + \#\ \text{of roots},$$
$$\Delta = \Delta[n] = Lv_{th}[n] - Lv_{th}[n-1] = \frac{RH_{th}}{\#\ \text{Lvs}} \quad (1)$$

IV. OPTIMIZING CAT-TWO

Given the aforementioned new restrictions, there are more optimization opportunities to reduce the number of counters or reduce the size of CAT-TWO.

A. MULTIPLE TREE ROOTS

From eqnarray (1), if we reduce the number of levels, we increase the difference between $Lv_{th}[n]$ (Δ), decreasing the number of counters. Previously, it was assumed that CAT-TWO stems from a single root. If we start from multiple roots, we can reduce the number of levels needed for the last level to count only one row. For example, if we start from 4 tree roots, only 15 levels are needed to split the counters to individual rows. $Lv_{th}[n]$ becomes $Lv_{th}[L]/15$, which is larger than $Lv_{th}[L]/17$. However, increasing the number of roots means that counters that do not receive ACT commands and, therefore, would not have split are initially split, increasing the total number of counters. Therefore, the number of roots must balance the decrease in the maximum number of splits and the increase in initial counters. Figure 4 depicts the number of maximum counters versus the number of roots. Initially, the maximum number of counters decreases as the number of levels is decreased. When the number of roots surpasses 64, the number of counters increases as the number of roots becomes a dominant source of counters. Therefore, we implement CAT-TWO with 64 roots and 11 levels.

The idea of using multiple roots is also discussed in [27]. However, multiple roots in our scheme reduce the number of levels, whereas [27] maintains the number of levels, choosing

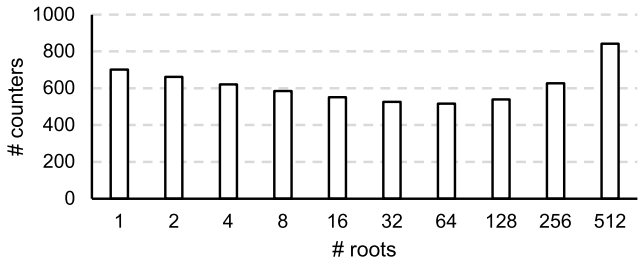

FIGURE 4. Number of roots vs. maximum number of counters.

to pre-split all nodes of a tree to a certain level M and not to alter the tree structure. If the tree structure is maintained, the number of ACTs for the first split is increased from $Lv_{th}[0]$ to $Lv_{th}[M]$. By contrast, in CAT-TWO, as the number of counters must be provisioned to accommodate the worst-case memory access pattern, this approach increases the number of counters rather than decreasing it. It is because after reaching $Lv_{th}[M]$, the number of ACTs needed for the next splits is Δ regardless which level CAT-TWO is pre-split to, and the number of initial counters that are pre-split (2^M) is greater than the number of counters saved by preventing splits until $Lv_{th}[M]$ ACTs (M). Therefore, the maximum number of counters is increased from when the tree is grown from one initial node, which does not benefit our effort to reduce the size of CAT-TWO.

B. RANK-LEVEL CAT-TWO

All previous row-hammer mitigation solutions were designed to be deployed on a per-bank basis. However, the frequency of DRAM ACTs are limited by tRC for a DRAM bank, but the frequency of DRAM ACTs for a DRAM rank is limited by tRRD and tFAW. As an example, a DDR4 DRAM whose tRC is set to be 44.5 ns amounts to 1,351,680 ACTs per bank in tREFW; however, tFAW is set to 21 ns, which amounts to 11,632,640 ACTs per rank in tREFW. For a DRAM rank consisting of 16 banks, this means the sum of ACTs per bank, $1{,}351{,}680 \times 16 = 21{,}626{,}880$, is about 1.85 times more than the maximum ACTs per rank. Therefore, rather than distributing a smaller CAT-TWO table to each bank, if we implement CAT-TWO at a per-rank basis and aggregate the tables to create one large CAT-TWO table to track all ACTs to a rank, we can cut down the maximum number of ACTs in tREFW, reducing the aggregate number of splits and counters. While TWiCe is also affected by the maximum number of ACTs per tREFW, this idea does not apply to TWiCe as content addressable memories are needed to manage TWiCe tables, and the total latency would increase too much when implementing one big table for the entire DRAM rank.

Although rank-level CAT-TWO reduces the total number of counters of a rank, it comes at the cost of increasing the size of a single search table entry. When CAT-TWO is modified from being deployed per bank to per rank, the number of entries of a single CAT-TWO table (search and counter) increases from hundreds to several thousand as a single

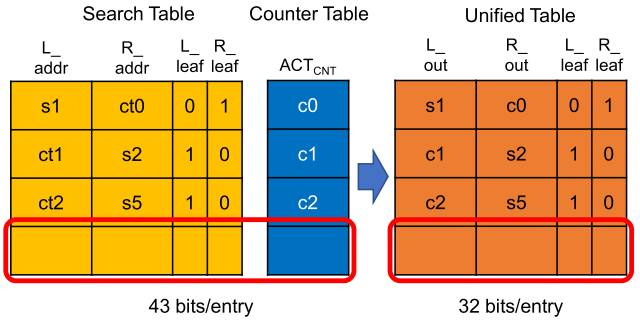

FIGURE 5. Original CAT tables (left) and unified tables (right).

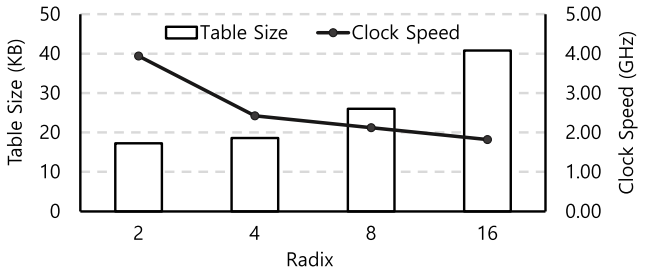

FIGURE 6. CAT-TWO radix vs. table size (bar) and clock speed (line).

CAT-TWO table covers the entire rank. In this case, the width of *_addr* entries would have to be increased to be over 10 bits, increasing the size of a single entry. In the next section, we propose unifying the search and counter tables together to reduce the size of CAT-TWO despite the increase in the width of an entry.

Another challenge to rank-level CAT-TWO is the increase in CAT-TWO clock speed. The clock speed of CAT-TWO was not a problem when it was deployed per bank as the maximum frequency of ACTs for a bank is limited by tRC (44.5 ns for DDR4). A CAT-TWO with 11 levels would require a maximum of 13 table accesses per ACT (When a counter split, the total number of accesses would be 13; 11 accesses to read ACT_{CNT}, plus one access to write-back the location of the split counter, plus one access to record the new counter), which requires CAT-TWO to operate at a minimum clock speed of 13 accesses/44.5 ns = 0.292 GHz. If CAT-TWO is deployed per rank, the throughput of ACTs to a rank is limited by tFAW/4, which translates to 13 accesses/(21 ns / 4) = 2.48 GHz in clock speed. However, this frequency is too high to be implemented in a wide range of systems, and while the average time between ACTs is limited by tFAW/4, the minimum time between two ACTs is limited by tRRD (3.3 ns). Therefore, if CAT-TWO operates below 13 accesses/3.3 ns = 3.93 GHz, CAT-TWO needs buffers to store the ACT commands and a more complicated control logic. To deploy CAT-TWO on a wide range of systems, CAT-TWO must operate at lower speeds while satisfying both tFAW and tRRD timing constraints, for which the number of levels of CAT-TWO must be decreased to reduce the number of table accesses. Employing multiple tree roots cuts down the number of levels of CAT-TWO, but using the idea to reduce the levels beyond 10 levels is impractical as the number of counters grows exponentially with each level reduction. To solve this problem, we introduce high-radix CAT-TWO, which will be discussed in Section IV-D.

C. UNIFIED TABLES

Instead of separating the search and counter tables as [27] suggests, we propose to unify the two tables together to reduce CAT-TWO size and table accesses. Rank-level CAT-TWO requires over 10 bits for *_addr*, which is comparable to the 15 bits that the counter table requires to record ACT_{CNT}. Therefore, instead of separating the search table and counter table, it makes sense to reduce the total size of CAT-TWO by increasing the width of *_addr* to 15 bits to be used as both *_addr* and ACT_{CNT}.

A rank-level CAT-TWO with 64 roots requires 4,416 counters, which means *_addr* would have to be 13 bits wide to cover all counters in CAT-TWO. When the search and counter tables are separated, the search table would have M-1 entries with 28 bits/entry ($R\_addr+L\_addr+L\_leaf+R\_leaf$) and a counter table with M entries with 15 bits/entry, which equates to a consolidated size of 43 bits per entry. However, if we were to increase the size of the *_addr* fields to 15 bits and use the field to point to the next table entry when *_leaf* entry is 0 and use the field as ACT_{CNT} when *_leaf* is 1, this reduces the bits per counter to 32 bits per entry. Therefore, unifying the search table and the counter table reduces the total table size by about 26%. Moreover, the unified table reduces one table access as there is no need to check the last search table entry in checking the address of the counter in the counter table.

D. HIGH-RADIX CAT-TWO

CAT-TWO has been assumed to have a binary tree. However, if CAT-TWO has a higher-radix tree that splits to more than 2 counters every time a counter splits, the number of levels required to count a single row per counter at the last level would decrease. For example, a CAT-TWO with radix-4 with a single root would require only 8 levels to divide the initial rows to a single counter, while a binary CAT-TWO required 16 levels. Therefore, high-radix CAT-TWO could be used to reduce the number of table accesses to access a counter, reducing CAT-TWO clock speed. However, while doubling the radix roughly halves the number of levels, it also increases the number of new counters per split, increasing the total number of counters.

Figure 6 depicts the change in CAT-TWO table size and clock speed, as the radix of CAT-TWO is increased from 2 to 16. CAT-TWO is assumed to be deployed per-rank, with a unified table and 64 roots, and finishes counter updates within tRRD. Table size is used to compare the tables because the number of counters does not offer a fair comparison when the radix is changed, as higher radix CAT-TWO requires fewer *_out* fields to record the structure of the tree. When the radix increases from 2 to 4, there is only an 8% increase in size,

whereas the size increases 39% and 56% when the radix incremented from 4 to 8 and 8 to 16, respectively. The reduction in clock speed is the highest when the radix is increased from 2 to 4, and the clock speed reduction diminishes when the radix is increased to 8 and 16. We choose to implement CAT-TWO with a radix-4 tree as it can be operated at a low enough speed. When reducing the clock speed is crucial, further increasing the radix at the cost of an increase in size is a viable option.

V. EVALUATION

A. EXPERIMENTAL SETUP

We analyzed the energy and timing of CAT-TWO, and compared the table size and performance, in terms of an increase in ACTs, against other row-hammer solutions. We assessed the increase in ACTs due to the row-hammer solutions on a system simulated with McSimA+ [1]. The row-hammer prevention schemes were assumed to be implemented on a multi-core system with DDR4-2400 DRAM. Other simulation parameters are summarized in Table 1.

From SPEC CPU2006 benchmarks [6], we created 29 SPECrate workloads and 2 mixed multi-programmed workloads to run in the simulations. The most representative 100M instructions were extracted from each application using Simpoint [28], and 16 copies of the same application were used as a multi-programmed workload. Mix-high workload was created using 9 applications with the most memory access per kilo-instructions (MAPKI) (mcf, milc, leslie3d, soplex, GemsFDTD, libquantum, lbm, sphinx3, and omnetpp), and mix-blend workload was created using 16 random SPEC CPU2006 applications regardless of MAPKI. We also evaluated the protection schemes against targeted row-hammer attack scenarios, where a single row in a bank (single-bank attack) or rows distributed across multiple banks (9 banks of the same rank, multi-bank attack) are repeatedly activated at maximum frequency. Because ACTs to a bank are limited by tRC and ACTs to a rank by tFAW, only 8 banks can be activated at the maximum frequency of ACTs to a bank and 1 bank at 60% of the maximum frequency of ACTs to a bank.

Table size and increase in ACTs were evaluated between PARA, original CAT with 64 counters per bank (CAT-64), original CAT with 256 counters per bank (CAT-256), CAT-TWO, and TWiCe. CAT-TWO was designed to be radix-4, 6 levels, and 64 roots with 2,386 table entries, where each entry is 64 bits wide, holding a maximum of 4 counters. Parameters for the DRAM that the row-hammer prevention schemes were deployed to are summarized in Table 2.

TABLE 1. Parameters of simulated system.

Resource	Value
Number of cores, MCs	16, 2
Per core:	
Freq, issue/commit width	3.6 GHz, 4/4 slots
Issue policy	Out-of-Order
L1 I/D $, L2 $	16 KB, 128 KB private
L1, L2, L3 $ line size	64 B
Hardware (linear) prefetch	On
L3 $	16 MB shared
Per memory controller (MC):	
# of channels, Req Q	2 Ch, 64 entries
Module type	DDR4-2400
Capacity per rank, bandwidth	8 GB, 19.2 GB/s
Scheduling	PAR-BS [23]
DRAM page policy	Minimalist open [13]

TABLE 2. DRAM parameters [10].

Parameter	Value
tREFW	64 ms
tREFI	7.8 μs
tRFC	350 ns
tRC	44.5 ns
tFAW	21 ns
RH_{th}	65,536
Max ACTs per tREFW to a bank	1,351,680
Max ACTs per tREFW to a rank	11,632,640
Pages per bank	65,536

TABLE 3. Timing/energy of CAT-TWO/DRAM operations.

		# of Accesses	Timing (ns)	Energy (nJ)
CAT-TWO	ACT_{CNT} increment	min 2	0.82	0.0137
		max 6	2.46	0.0412
	Split	min 3	1.23	0.0206
		max 6	2.46	0.0412
	Row Refresh	6	2.46	0.0412
	Table Reset	256	104.80	1.7588
DRAM	ACT to ACT ($tRRD$)		3.30	5.74 [1]
	Refresh/Rank ($tRFC$)		350	1057.92

[1] Energy here is the energy that is taken to activate and precharge a row, rather than the energy consumption between two ACTs.

B. RESULTS

1) TIMING AND ENERGY

We analyzed the energy overhead of CAT-TWO tables using CACTI-6.5 [31] with 32 nm process and derived DRAM energy usage from DDR4 SDRAM system-power calculator [21]. While the logic layer of CAT-TWO contributes to the energy, it incurs negligible energy overhead compared to the CAT-TWO tables. Therefore, we focus on the CAT-TWO table accesses in this paper.

We designed CAT-TWO as one bank of SRAM with 8 bytes of data per line with 2,386 lines, amounting to 19 KB per 8 GB DRAM rank, and CAT-TWO is assumed to operate at 2.44 GHz. As the number of accesses to increment ACT_{CNT} differs by the level that the counter is in, we show the minimum and maximum energy consumption and timing. The number of accesses to increment ACT_{CNT} equals to the tree traversal to locate the ACT_{CNT} and another write access to increment the ACT_{CNT}. Hence, the number of accesses at

the bottom level is 2 while the number of access at level 5, the last level, is 6. Splits require one additional access than normal ACT_{CNT} increment as the newly created entry must be allocated. However, as splits only occur until the second to last level, the maximum number of accesses remains the same. Refreshes require finding a counter at the last level and resetting the ACT_{CNT} back to zero, incurring the same number of accesses as when the ACT_{CNT} of a counter at the last level is incremented. Periodic resetting requires CAT-TWO to reset 256 entries (entries per bank $\times$ number of banks $= 16 \times 16$), requiring 256 writes.

The timing requirements of CAT-TWO are within DRAM timing parameters, and the energy overhead is negligible compared to DRAM operations. The longest time to update the table is 2.46 ns, which is less than tRRD of 3.3 ns, and the time it takes to reset the table is 104.8 ns while tRFC is 350 ns. The CAT-TWO table operations take less time than DRAM operations, causing no performance overhead. The energy overhead is negligible compared to DRAM operations, with 0.7% maximum energy overhead per table update, and 0.2% energy overhead per reset.

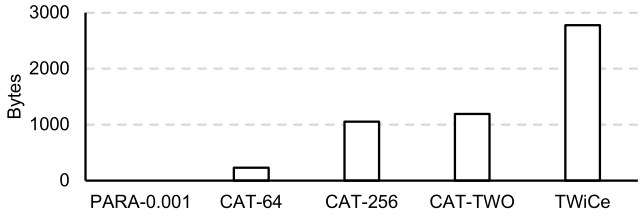

FIGURE 7. Row-hammering prevention scheme table sizes.

2) TABLE SIZE

Figure 7 shows the table size of each row-hammer prevention scheme in bytes per DRAM bank. The table size of CAT-TWO is the amortized size per bank; the table size per rank divided by the number of banks. PARA, which does not use a table, incurs zero table size overhead. CAT-64 has the next smallest table as it uses a small number of counters. The size of CAT-256 and CAT-TWO is comparable to each other, with CAT-TWO being 13% larger than CAT-256. TWiCe requires the largest table, 2.32 times the size of CAT-TWO.

3) ADDITIONAL ACTs

Figure 8 shows the results from the simulation. Benchmarks with the highest increase in ACTs were selected to be shown in the figure. On average of SPECrate, PARA incurred the most increase in ACTs, 0.1%, followed by CAT-64 and CAT-256, 0.0487% and 0.0318%, respectively. However, CAT-64 incurred a higher increase in ACTs than PARA in soplex, GemsFDTD, libquantum, and mix-high, reaching as high as 0.25% in mix-high. CAT-256 also incurred a higher increase in ACTs than PARA in soplex and libquantum, reaching as high as 0.21% for libquantum. This is due to the effects of limited number of counters and levels, and the effect of duplicated ACTs. CAT-TWO incurred a 0.000006% increase in ACTs on average, which is less than 0.02% of the increase in ACTs incurred by CAT-256. In detail, CAT-TWO incurred no increase in ACTs except for the omnetpp application, where the number of ACTs is increased merely by 0.00018%. TWiCe incurred no increase in ACTs across all tested applications.

In the targeted row-hammer attacks, when attacking a single row of one bank (single-bank attack), PARA, CAT-64, and CAT-256 all incurred around 0.1% increase in ACTs, while CAT-TWO and TWiCe both incurred 0.006% increase in ACTs. In the multi-bank attack, CAT-64 and CAT-256 incur a slight decrease in ACTs, incurring slightly less than 0.1% increase in ACTs. This is because not all the rows that are being attacked can be activated at the maximum frequency as it is being limited by tFAW. CAT-TWO and TWiCe again incur the same increase in ACTs at 0.006%.

All in all, CAT-TWO incurs the same rate of additional ACTs to TWiCe in all the applications but omnetpp, where CAT-TWO has slightly more ACTs. PARA, CAT-64, and CAT-256 all incur significantly more ACTs as they incur over 5,151 times and 16 times the additional ACTs of CAT-TWO in the SPECrate benchmarks and the targeted attack scenarios, respectively.

4) WORST-CASE ANALYSIS (MAXIMUM NUMBER OF ADDITIONAL ACTs)

Although CAT-64, CAT-256, CAT-TWO, and TWiCe all have the same *RH_th*, in some applications, CAT-64 and CAT-256 incurred additional ACTs while CAT-TWO and TWiCe did not. It is because, at all levels except the last level, CAT based row-hammer mitigation solutions track ACT_{CNT} in groups of rows and duplicate the ACT_{CNT} whenever the counters are split. When ACTs are sent to rows of the same group, ACT_{CNT} increases indiscriminately of the row that receives the ACT. Therefore, the difference between the ACT_{CNT} of a row and the actual number of ACTs the row receives is large when ACTs are concentrated to the rows of the same group. This phenomenon becomes problematic if the traffic pattern is such that the duplicated ACT_{CNT} is used to trigger refreshes to rows that are not vulnerable to row-hammering, and hence the additional row refreshes incur a large overhead. Therefore, we analyzed the maximum number of ACTs caused by the duplicated ACTs to verify that CAT-TWO would not hamper system performance under the worst traffic pattern.

A greedy algorithm was used to assess the maximum number of increased ACTs when duplicated ACT_{CNT} is exploited. Figure 9 depicts the algorithm that was used to maximize refreshes. First, the CAT tree is grown by continuously sending ACTs to a single row (mapped to counter *A*) until it is split to the last level, then the row is activated again until it is refreshed. Once refreshed, the ACT_{CNT} of counter A resets to 0. Therefore, the subsequent ACTs are sent to the row of counter *B*, which requires the least number of ACTs to be refreshed. While no ACTs were sent to the row of counter *B*, because ACT_{CNT}s were duplicated by CAT while the row of counter A was activated, counter *B* is at the last level with

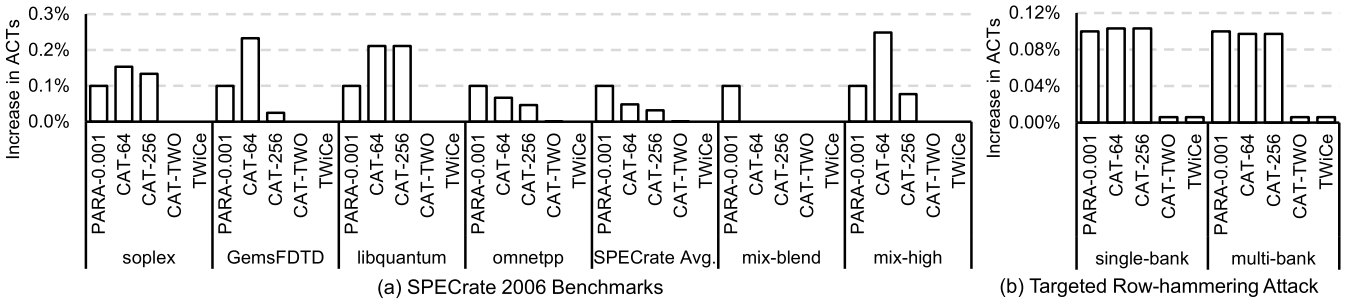

FIGURE 8. Increase in the number of ACTs according to workload. Applications with the highest increase are ACTs is shown.

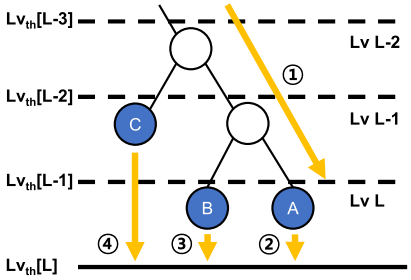

FIGURE 9. The attack algorithm to maximize ACTs from ACT_{CNT} duplicates. The tree is grown by concentrating ACTs on one row, then counters with the least ACT_{CNT} to $Lv_{th}[L]$ are selected to be refreshed.

TABLE 4. The number of worst-case additional ACTs within tREFW.

	CAT-64	CAT-256	CAT-TWO
# of Additional ACTs	2,166,964	14,784	368

$Lv_{th}[L-1]$ as ACT_{CNT}, requiring only $\Delta[L]$ ACTs to be refreshed. When counter B is refreshed, counter C is then selected to be refreshed as it requires the least number of ACTs. The algorithm continues until all ACTs in tREFW is used to refresh rows or until all counters are used at the original CAT. When all counters are being used, the algorithm targets the lowest level counters that can reach RH_th with the remaining ACTs to maximize the number of refreshes.

Figure 4 shows the number of refreshes in a bank within tREFW when the duplicate ACTs are exploited on the original CAT-64, CAT-256, and CAT-TWO. CAT-64 incurs 2,166,964 additional ACTs and the most refreshes out of all schemes as 64 counters are used up within tREFW, and the counter at level 1 is targeted to maximize the number of additional ACTs. CAT-256 does not use up all counters, but the number of additional ACTs is 14,784 and still large as every time a counter reaches RH_th, 34 rows are refreshed. CAT-TWO incurs only 368 additional ACTs, the least number of additional ACTs, and less than 2.5% of either CAT configurations. This result concurs with the simulation results in figure 8 where CAT-64 and CAT-256 incurs a large number of refreshes because of the duplicated ACTs, whereas CAT-TWO incurs negligible to zero additional ACTs.

Additionally, this result also highlights that even in the worst case, the additional ACTs are negligible for CAT-TWO because CAT-TWO only incurs 368 maximum additional ACTs in tREFW, which is only 0.03% of the maximum number of ACTs in tREFW to a bank.

C. DECREASING $\Delta[L]$

While $\Delta[n]$ affects the number of splits in tREFW, $\Delta[L]$ does not, because a counter that reaches $Lv_{th}[L]$ is refreshed. Therefore, if we were to decrease $\Delta[L]$ and use the surplus ACTs to increase other $\Delta[n]$, we can reduce the number of ACTs in tREFW and decrease the total number of counters. However, this optimization comes at the expense of increasing the effect of duplicated ACT_{CNT}s. As more ACTs are accumulated before counters are split, more ACT_{CNT} is duplicated and increases the maximum number of additional ACTs.

Figure 10 shows the number of additional ACTs and the number of entries per bank against the reduction in $\Delta[L]$. When the reduction in $\Delta[L]$ goes from 0 to 5,400, the number of entries per bank decreases from 149 to 127, and the additional ACTs increases from 368 to 1,192. While the additional ACTs increases steeply with the reduction in $\Delta[L]$, if the additional ACTs are within a tolerable envelope or if it is crucial to reduce the size of CAT-TWO, this optimization could be used to trade CAT-TWO table size with an increase in the maximum number of additional ACTs.

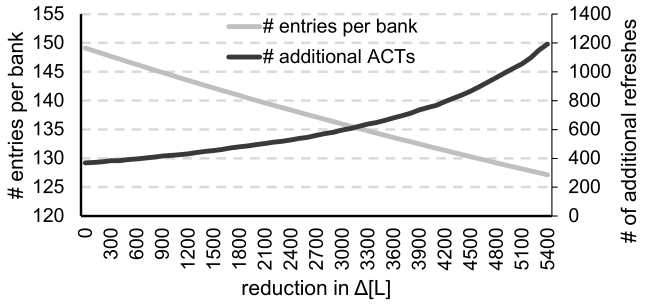

FIGURE 10. The number of additional refreshes and number of entries per bank against reduction in $\Delta[L]$.

VI. CONCLUSION

We have proposed CAT-TWO, a time window optimized counter-based adaptive tree that counts the ACTs to a group of rows, but always refreshes rows on a per-aggressor-row basis. CAT-TWO is guaranteed to refresh rows adjacent to a single aggressor row by provisioning it with enough counters such that a counter that passes $Lv_{th}[n]$ can always split. We further optimize CAT-TWO to shrink its table size by analyzing the effect of $Lv_{th}[n]$ to the number of counters, utilizing multiple tree roots, deploying CAT-TWO per rank, and unifying CAT-TWO tables. We also reduce the number of accesses to a CAT-TWO table to reduce the clock speed by utilizing a high-radix tree. Our analysis shows CAT-TWO incurs only 0.7% more energy on table updates and 0.2% more energy on table resets. CAT-TWO performs on par with TWiCe with regard to the increase of ACTs and 16× to 5,151× better than CAT-256. The table size of CAT-TWO is less than half of that of TWiCe and only 13% larger than that of CAT-256. Furthermore, even in the worst-case CAT-TWO only requires 0.03% of the maximum number of ACTs within tREFW to ensure row-hammer prevention.

REFERENCES

[1] J. H. Ahn, S. Li, S. O, and N. P. Jouppi, "McSimA+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Austin, TX, USA, Apr. 2013, pp. 74–85.
[2] I. Bhati, M.-T. Chang, Z. Chishti, S.-L. Lu, and B. Jacob, "DRAM refresh mechanisms, penalties, and trade-offs," *IEEE Trans. Comput.*, vol. 65, no. 1, pp. 108–121, Jan. 2016.
[3] S. Bhattacharya and D. Mukhopadhyay, "Advanced fault attacks in software: Exploiting the rowhammer bug," in *Fault Tolerant Architectures for Cryptography and Hardware Security*. Singapore: Springer, 2018, pp. 111–135.
[4] S. Cha, S. O, H. Shin, S. Hwang, K. Park, S. J. Jang, J. S. Choi, G. Y. Jin, Y. H. Son, H. Cho, J. H. Ahn, and N. S. Kim, "Defect analysis and cost-effective resilience architecture for future DRAM devices," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Austin, TX, USA, Feb. 2017, pp. 61–72.
[5] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of Rowhammer defenses," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 245–261.
[6] J. L. Henning, "SPEC CPU2006 memory footprint," *SIGARCH Comput. Archit. News*, vol. 35, no. 1, p. 84, Mar. 2007.
[7] N. Herath and A. Fogh, "These are not your grand Daddys CPU performance counters—CPU hardware performance counters for security," in *Proc. Black Hat Briefings*, 2015.
[8] M. Horiguchi and K. Itoh, *Nanoscale Memory Repair*. New York, NY, USA: Springer, 2013.
[9] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. San Mateo, CA, USA: Morgan Kaufmann, 2007.
[10] *DDR4 SDRAM Standard*, Standard JESD79-4B, JEDEC, 2012.
[11] *Low Power Double Data Rate 3 (LPDDR3)*, Standard JESD209-3C, JEDEC, 2013.
[12] *Low Power Double Data Rate 4 (LPDDR4)*, Standard JESD209-4B, JEDEC, 2014.
[13] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist open-page: A DRAM page-mode scheduling policy for the many-core era," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Porto Alegre, Brazil, 2011, pp. 24–35.
[14] B. Keeth, R. J. Baker, B. Johnson, and F. Lin, *DRAM Circuit Design*, 2nd ed. Piscataway, NJ, USA: IEEE, 2008.
[15] D.-H. Kim, P. J. Nair, and M. K. Qureshi, "Architectural support for mitigating row hammering in DRAM memories," *IEEE Comput. Archit. Lett.*, vol. 14, no. 1, pp. 9–12, Jan. 2015.
[16] M. Kim, J. Choi, H. Kim, and H.-J. Lee, "An effective DRAM address remapping for mitigating Rowhammer errors," *IEEE Trans. Comput.*, vol. 68, no. 10, pp. 1428–1441, Oct. 2019.
[17] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit. (ISCA)*, Jun. 2014.
[18] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "RAMBleed: Reading bits in memory without accessing them," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2020.
[19] E. Lee, I. Kang, S. Lee, G. E. Suh, and J. H. Ahn, "TWiCe: Preventing row-hammering by exploiting time window counters," in *Proc. 46th Int. Symp. Comput. Archit. (ISCA)*, Phoenix, AZ, USA, 2019, pp. 385–396.
[20] E. Lee, S. Lee, G. E. Suh, and J. H. Ahn, "TWiCe: Time window counter based row refresh to prevent row-hammering," *IEEE Comput. Arch. Lett.*, vol. 17, no. 1, pp. 96–99, Jan. 2018.
[21] *DDR4 SDRAM System-Power Calculator*, Micron Technol., Boise, ID, USA, 2016.
[22] O. Mutlu and J. S. Kim, "RowHammer: A retrospective," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, to be published.
[23] O. Mutlu and T. Moscibroda, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems," in *Proc. Int. Symp. Comput. Archit.*, Beijing, China, Jun. 2008, pp. 63–74.
[24] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip Feng Shui: Hammering a needle in the software stack," in *Proc. USENIX Secur. Symp.*, Austin, TX, USA, 2016, pp. 1–18.
[25] M. Seaborn and H. Flake, "Exploiting the DRAM Rowhammer bug to gain kernel privileges," in *Proc. Black Hat Briefings*, 2015.
[26] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Counter-based tree structure for row hammering mitigation in DRAM," *IEEE Comput. Archit. Lett.*, vol. 16, no. 1, pp. 18–21, Jan. 2017.
[27] S. M. Seyedzadeh, A. K. Jones, and R. Melhem, "Mitigating wordline crosstalk using adaptive trees of counters," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Los Angeles, CA, USA, Jun. 2018, pp. 612–623.
[28] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proc. ASPLOS*, San Jose, CA, USA, 2002, pp. 45–57.
[29] M. Son, H. Park, J. Ahn, and S. Yoo, "Making DRAM stronger against row hammering," in *Proc. 54th Annu. Design Autom. Conf. (DAC)*, Austin, TX, USA, 2017, pp. 1–6.
[30] Y. H. Son, S. Lee, O. Seongil, S. Kwon, N. S. Kim, and J. H. Ahn, "CiDRA: A cache-inspired DRAM resilience architecture," in *Proc. IEEE 21st Int. Symp. High Perform. Comput. Archit. (HPCA)*, Burlingame, CA, USA, Feb. 2015, pp. 502–513.
[31] S. Thoziyoor, J. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies," in *Proc. ISCA*, Beijing, China, 2008, pp. 51–62.
[32] V. Van Der Veen, Y. Fratantonio, M. Lindorfer, D. Gruss, C. Maurice, G. Vigna, H. Bos, K. Razavi, and C. Giuffrida, "Drammer: Deterministic Rowhammer attacks on mobile platforms," in *ACM CCS*, Vienna, Austria, 2016, pp. 1675–1689.
[33] Y. Wang, Y. Liu, P. Wu, and Z. Zhang, "Detect DRAM disturbance error by using disturbance bin counters," *IEEE Comput. Archit. Lett.*, vol. 18, no. 1, pp. 35–38, Jan. 2019.
[34] J. M. You and J.-S. Yang, "MRLoc: Mitigating Row-hammering based on memory locality," in *Proc. 56th Annu. Design Autom. Conf. (DAC)*, Las Vegas, NV, USA, 2019, pp. 1–6.

INGAB KANG received the B.S. degree in electrical and computer engineering from Seoul National University, where he is currently pursuing the M.S. degree with the Graduate School of Convergence Science and Technology. His research interests include secure computing and secure memory systems.

EOJIN LEE received the B.S. degree in electrical and computer engineering from Seoul National University, in 2013, where he is currently pursuing the Ph.D. degree with the Graduate School of Convergence Science and Technology. His research interest includes memory system optimization and computer architecture for accelerating emerging applications.

JUNG HO AHN (Senior Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA.

He is currently a Professor with the Graduate School of Convergence Science and Technology, Seoul National University. He is interested in bridging the gap between the performance demand of emerging applications and the performance potential of modern and future massively parallel systems.