Revealing DRAM Operating GuardBands through Workload-Aware Error Predictive Modeling


Published in:
IEEE Transactions on Computers

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

Publisher rights
© 2020 The Authors.
This work is made available online in accordance with the publisher's policies. Please refer to any applicable terms of use of the publisher.

General rights
Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.
Revealing DRAM Operating GuardBands through Workload-Aware Error Predictive Modeling

Lev Mukhanov, Konstantinos Tovletoglou, Hans Vandierendonck, Senior Member, IEEE, Dimitrios S. Nikolopoulos, Senior Member, IEEE, and Georgios Karakonstantis, Senior Member, IEEE.

Abstract—Improving the energy efficiency of DRAMs becomes very challenging due to the growing demand for storage capacity and failures induced by the manufacturing process. To protect against failures, vendors adopt conservative margins in the refresh period and supply voltage. Previously, it was shown that these margins are too pessimistic and will become impractical due to high power costs, especially in future DRAM technologies.

In this paper, we present a new technique for automatic scaling the DRAM refresh period under reduced supply voltage that minimizes the probability of failures. The main idea behind the proposed approach is that DRAM error behavior is workload-dependent and can be predicted based on particular program inherent features. We use a Machine Learning (ML) method to build a workload-aware DRAM error behavior model based on the program features which we extract from real workloads during our DRAM characterization campaign. With such a model, we identify the marginal value of the DRAM refresh period under relaxed voltage for each DRAM module of a server that enable us to reduce the DRAM power.

We implement a temperature-driven OS governor which automatically sets the module-specific marginal DRAM parameters discovered by the ML model. Our governor reduces the DRAM power by 24% on average while minimizing the probability of failures. Unlike previous studies, our technique: i) does not require intrusive changes to hardware; ii) is implemented on a real server; iii) uses a mechanism that prevents any abnormal DRAM error behavior; iv) can be easily deployed in data centers.

Index Terms—DRAM, GuardBands, reliability, low-power electronics, energy consumption.

1 INTRODUCTION

The explosive growth of Internet-connected devices and subsequently generated data drive the need for increasing the memory capacity and density within high-end systems. Such a surge is projected to turn DRAMs into one of the main power consumers on servers in the near future [1]. This challenge has turned attention to improving the energy efficiency of the memory subsystem in servers by scaling down essential circuit parameters. These parameters are pessimistically set based on the assumed worst-case conditions [1], [2], [3], [4], [5]. Unfortunately, a relaxation of such circuit parameters comes at the cost of increased DRAM errors, even though it provides better energy and performance efficiency in memory systems [2], [3], [6].

Recent studies in this area have minimized the refresh power by identifying weak cells for memory in operation (in the field) and by setting a shorter refresh period for such cells [1], [7]. By contrast, other works have suggested using a supervised data mapping to avoid the errors induced by these weak cells [2], [8], [9], [10], [11], [12]. Yet other schemes have attempted to exploit the inherent error resilience of applications [13], [14]. Typically, since many of the proposed ideas cannot be realized on real machines due to the limitations of the existing memory controllers, the effectiveness of the majority of the existing schemes is evaluated using random error injection methods. Such methods are based on the failure probabilities discovered by a few thorough studies using FPGAs and worst-case data pattern micro-

benchmarks [10], [13], [14]. Moreover, the same failure probabilities are being considered for any workload, even if the failure rates depend strongly on the data patterns, as shown in [2], [10], and on the access patterns [4], [15].

Although all these studies provide valuable insights, they have been conducted either with simulators, because they required intrusive changes to memory controllers, or with experimental setups based on custom FPGAs, which lack a realistic operating system and system software stack. These limitations have hindered the understanding of the implications of the entire application-level and system software stack on DRAM reliability.

The goal of the work presented in this paper is to efficiently and effectively reveal and exploit the margins of DRAM circuit parameters that minimize the risk of data integrity violations on real servers when running full software stacks. We aim at exploiting extended margins of DRAM circuit parameters by considering the varying effect of running programs on DRAM reliability, in order to minimize the DRAM power. We strive to understand which program features may implicitly affect DRAM error behavior and how to build a model for the prediction of this effect. Such a model will enable us to understand the impact of a particular workload on DRAM reliability without long-running characterization campaigns that may take months. Once implemented, the model will make it possible to predict the marginal values of the circuit parameters that allows us to reduce the DRAM power with minimal risk of failures regardless of executed programs.
In our previous study [16], we demonstrated that DRAM error behavior is workload-dependent and it can be predicted using a set of program inherent features that have a strong impact on DRAM reliability. We predict DRAM errors for a running workload by extracting the program features and applying Machine Learning (ML) to model DRAM error behavior for these features. To enable this study, we developed a novel experimental framework for characterizing DRAM reliability under various operating parameters and temperatures and applied this framework to a state-of-the-art 64-bit ARM server.

In this paper, we extend our previous work and present a new study on predicting the marginal values of the DRAM refresh period under lowered supply voltage using an ML-based workload-aware DRAM error behavior model. To the best of our knowledge, this is the first study that attempts to reveal the DRAM operating guardbands (or the marginal DRAM operating parameters) for a specific device through workload-aware error predictive modeling. Our contributions are summarized as follows:

- We present a new technique to identify the marginal refresh period under lowered supply voltage for each DRAM device using an ML-based memory error behavioral model. This technique implements a search engine that finds the program inherent features that trigger the highest number of DRAM errors. To this end, the engine varies program features, which have the strongest impact on DRAM error behavior, and predicts the number of DRAM errors for each set of program features using the ML model. Once the values of program features that trigger the worst-case DRAM error behavior are detected, the search engine finds the maximum refresh period under lowered supply voltage that reduces the DRAM power with minimal risk of failures.
- We present a new temperature-driven OS governor that dynamically sets the discovered marginal circuit parameters for each DRAM module of the server. To prevent any abnormal DRAM error behavior, we introduce an additional mechanism that ensures that scaling the DRAM parameters will not lead to a system crash with a probability of $1 - 4 \times 10^{-45}$.
- We evaluate the efficacy of the developed governor. Our results show that the governor reduces the DRAM power by 24% on average without triggering any errors. We demonstrate that our technique enables us to achieve almost the maximum power savings that are possible on DDR3 memories used in this study for temperatures below 50 °C. The proposed technique requires minimal hardware support, and can be deployed in Edge and Cloud data centers.

The rest of the paper is organized as follows. Section 2 presents the background. Section 3 discusses our experimental framework, while Section 4 summarizes the results of our study on workload-dependent DRAM error behavior. Section 5 introduces the proposed technique for scaling DRAM parameters. Section 6 presents the results of evaluating the developed governor. Section 7 discusses the related work. Finally, conclusions are drawn in Section 8.

2 BACKGROUND

2.1 DRAM Organization

The main memory subsystem based on DRAMs is organized hierarchically into channels supporting a number of DRAM modules. Each Dual In-Line Memory Module (DIMM) usually has two ranks, each consisting of DRAM chips. Within each chip, DRAM cells are organized into banks that consist of two-dimensional arrays of rows and columns. Each cell contains a capacitor and an access transistor, and all access transistors are in a row connected by a wire called wordline, controlling access to the capacitors. When DRAM is accessed, an entire row is read, and sense-amplifiers measure the charge on each capacitor through bitlines, which are connected to access transistors [2], [6].

Refresh Period. The main drawback of the DRAM technology is the limited retention time [10] of a cell’s charge. To avoid any errors induced by the charge leakage over time, DRAMs employ an Auto-Refresh mechanism that recharges the cells in the array periodically [17]. Conventionally, all DDR technologies adopt a refresh period, $T_{REFP}$, of 64 ms (or 32 ms) for refreshing each cell of DIMMs; although, many cells may have a much higher retention time than the $T_{REFP}$, and conditions in the field may not be as adverse as assumed [1].

Supply Voltage. Another critical parameter for the operation of DRAMs is supply voltage. The nominal supply voltage, $V_{DD}$, of DRAM chips is chosen conservatively by vendors to ensure that each chip operates correctly under a wide range of conditions. However, the minimum voltage at which the DIMM chips stop functioning varies greatly [3]. In DDR3 DRAM chips, $V_{DD}$ is set by default to 1.5 $V$ [16], based on the worst-case operating conditions.

Temperature. Apart from the above circuit parameters, one of the main environmental conditions affecting the reliability of DRAM is temperature. It has been shown that the retention time of DRAM cells decreases exponentially as temperature increases [10], [18].

2.2 Workload-dependent DRAM behavior

Each program defines a unique data content (data pattern) and the order in which the program refers to this data (access pattern). The DRAM data and access patterns depend on the organization of data caches, since the majority of accesses to the memory subsystem are handled by caches in modern processors. Figure 1 illustrates the interaction between several workloads and the memory subsystem. Particularly,
in this figure, the first instruction of the workload \( t \) copies data from the register \( r_{19} \) to the \( c2 \) cache row (for simplicity, we combined L1, L2 and L3 caches in this figure) and then to the \( m2 \) DRAM row; \( 3.load \) and \( n.load \) instructions fetch the data from the \( m4 \) DRAM row, whereas \( k.load \) fetches the cached data from the \( c6 \) cache line without accessing DRAM.

It was shown that the number of DRAM errors strongly depends on the data pattern [2], [10]. At the same time, the memory access pattern also affects DRAM error behavior. The number of errors may reduce when data are accessed frequently due to each read/write naturally refreshes memory [4], [15]. For example, in Figure 1, \( 3.load \) and \( n.load \) refresh data in the \( m4 \) DRAM row. On the other hand, a high memory access rate to specific DRAM rows may increase the charge leakage in the neighbouring rows due to the DRAM cell-to-cell interference [12]. We illustrate this in Figure 1 where data in the \( m3 \) and \( m5 \) DRAM rows may be compromised when the \( m4 \) row is accessed frequently. Thus, the program inherent features that define memory data and access patterns may have an implicit effect on DRAM reliability.

Figure 2 demonstrates how DRAM error behavior may vary when running different workloads. This figure shows the spatial and density distribution of the detected single-bit errors across 4 DIMMs (8 ranks) when running \( kmeans \) (a compute-intensive benchmark) and \( memcached \) (a data caching benchmark) for DRAM operating under 2.283s \( T_{REFP} \) and lowered \( V_{DD} \) at 50 °C. Both benchmarks use 8Gb of memory to allocate data. We see that the number of errors manifested by the benchmarks in DIMM3/rank0 differs by up to 1000x.

### 2.3 DIMM-to-DIMM variation

The reliability of DRAM chips may vary even for chips manufactured by the same vendor [10], [16]. This variation is attributed to the manufacturing process [19] and the internal design of DRAM modules, such as true-anti cell organization [10], address scrambling [16], [20] and the remapping of faulty cells [11]. For example, Figure 2 shows that \( memcached \) incurs 633 errors in DIMM2/rank0, but only one error was observed in DIMM3/rank0.

Thus, there are cross-layer parameters, such as circuit parameters (e.g. \( V_{DD}, T_{REFP} \)), micro-architecture parameters (i.e. cache organization and DRAM architecture), workload parameters (data and DRAM access patterns) and environmental parameters (the DRAM temperature) that have an impact on DRAM reliability. All these parameters may manifest errors for memory operating under relaxed or even nominal parameters. The extent to which DRAM errors depend on workloads indicates that the marginal DRAM circuit parameters which do not induce errors cannot be found using a certain set of benchmarks. This holds because DRAM may behave differently when running benchmarks which are not in the set of benchmarks used to derive the marginal parameters. To find these parameters, it is essential to build a behavioral error model which can predict the effect of any workload on DRAM reliability. Moreover, due to the DIMM-to-DIMM variation, such a model should be developed for each DRAM device.

In the following sections, we summarize the results of our previous study on workload-dependent DRAM error behavior under varying parameters and reliability variation across DIMMs. The next section discusses the experimental framework used in our studies.

### 3 EXPERIMENTAL SETUP

To characterize DRAM error behavior, we use an experimental framework based on a state-of-the-art commodity 64-bit ARMv8-based server, the X-Gene2 Server-on-a-Chip. The X-Gene2 processor contains eight 64-bit ARMv8 cores running at 2.4GHz and four Memory Controller Units (MCUs), each of which supports up to two DIMMs. In our campaign, we make experiments with 4 Micron DDR3 8GB DIMMs (72 chips in total) operating at 1866 MHz [21], with one DIMM per MCU.

**DRAM power measurements and settings.** The X-Gene2 has a special management processor (SLIMpro) that provides access to the on-board sensors to measure the temperature and power of the SoC and DRAM. SLIMpro allows the configuration of the parameters of the MCUs, such as \( T_{REFP} \) and \( V_{DD} \). Specifically, \( T_{REFP} \) may be changed from the nominal 64 ms to 2283 ms.

**Error Accounting.** There are several types of errors that may manifest in DRAM chips due to the DRAM cell-to-cell interference or charge leakages [22]. It is known that the vast majority of these errors are single-bit, i.e. errors where only one bit is corrupted per a 64-bit word (or a 72-bit ECC word) [22]. To handle such errors, vendors implement a special hardware (ECC) for automatic correction of single-bit errors. Note that there are different types of ECC schemes and some of them may correct more than 1 bit per word.

The X-Gene2 server implements ECC SECDED (Single Error Correction Double Error Detection) and reports all the errors to the Linux kernel, providing information about the DIMM, bank, rank, row, and column in which an error occurred. In Table 1, we present three types of memory errors that may occur when ECC SECDED is enabled: single-bit errors (or correctable errors, CE); detected errors where
more than one bit is corrupted (or uncorrectable errors, UE); and errors where more than 2 bits are corrupted, which are not corrected and not detected by ECC. The last type of errors manifest so-called Silent Data Corruption (SDC), since such errors are invisible for hardware. Note that if the Linux kernel detects an UE, then it halts.

**DRAM Thermal Testbed.** We implement a special temperature-controlled testbed that uses heating elements [16] to change the DRAM temperature. Figure 3 shows the X-Gene2 board with four DIMMs fitted with our custom adapters which contain resistive elements heating DIMMs in a uniform way. Each adapter also implements a thermocouple to measure the temperature. The temperature of each element is controlled by a controller board, as shown in Figure 4, which contains a Raspberry Pi 3 and four closed-loop PID controllers, and eight solid-state relays controlling the resistive elements of each DIMM and rank independently. By measuring the temperature on the DIMMs with both the thermocouple and the embedded sensor on the SPD (Serial Presence Detect) chip, the controllers can regulate the temperature with a maximum deviation of ±1 °C.

**4 CHARACTERIZATION OF WORKLOAD-DEPENDENT DRAM ERROR BEHAVIOR**

In this section, we summarize the results of our previous study on characterization of workload-dependent DRAM error behavior and investigating program inherent features that have a strong impact on DRAM errors [16].

### 4.1 DRAM error metrics

Below, we introduce DRAM error metrics which we use to measure DRAM reliability in our study. We also demonstrate how to extract program inherent features, including those that reflect data and memory access patterns. By correlating these features with DRAM error metrics, we identify those features that have a strong impact on DRAM reliability.

**Correctable errors (CEs):** We estimate the probability to obtain a CE by measuring the rate of single-bit errors per a 64-bit word (or a 72-bit word), \( WER \) for the amount of memory allocated by an application as:

\[
WER = \frac{N_{CE}}{MEM_{SIZE}}
\]

(1)

where \( N_{CE} \) is the number of error-prone words that trigger CEs and \( MEM_{SIZE} \) is the size (in words) of memory allocated by the application.

**Uncorrectable errors (UEs):** We estimate the probability of an UE triggered by a running application as:

\[
P_{UE} = \frac{N_{UE}}{N_{EXP}}
\]

(2)

where \( N_{UE} \) is the number of runs of the application that resulted in an UE, and \( N_{EXP} \) is the total number of the application runs.

### 4.2 Program inherent features

To investigate software-level factors that may affect DRAM reliability, we extract the following program features.

**DRAM Reuse Time.** In our study, we measure \( T_{reuse} \) to estimate the average period between accesses to the DRAM cells. \( T_{reuse} \) affects DRAM reliability since memory accesses inherently refresh the stored charge [4, 15] when \( T_{reuse} < T_{REFP} \) and thus decrease the number of DRAM errors induced by the charge leakage.

To define \( T_{reuse} \) formally, let us assume that a running application triggers \( N \) Memory Accesses (\( \bar{A} \)) to DRAM so that:

\[
\bar{A} = A_0, ..., A_i, ..., A_N
\]

Then we can define \( \bar{D} \)

\[
\bar{D} = D_0, ..., D_i, ..., D_N
\]

(4)

where \( D_i \) is the number of instructions executed since the last access to the DRAM address referenced by \( A_i \). We estimate \( T_{reuse} \) as:

\[
T_{reuse} = CPI \times \frac{\sum_{i=0}^{N} D_i}{N}
\]

(5)

In this equation, \( CPI \) is the average number of clock cycles per instruction measured for an entire program. We extract \( \bar{D} \) using a dynamic binary instrumentation tool, DynamoRIO [23]. Specifically, we run each benchmark with DynamoRIO using the following options: -t drcachesim -simulator_type reuse_time -page_size 64K. We validated \( T_{reuse} \) estimates using micro-benchmarks where we can control and measure \( T_{reuse} \) for specific memory accesses. For example, Figure 5 shows the percentage error of \( T_{reuse} \) estimates compared to the real measurements for a micro-benchmark where \( T_{reuse} \) varies from \( 7\mu S \) up to \( 4000\mu S \). In this micro-benchmark, we insert a loop with different instructions between two memory accesses to the same address, and measure \( T_{reuse} \) between these accesses using standard glibc interfaces. We vary \( T_{reuse} \) by changing the latency of the loop with inserted instructions, and compare \( T_{reuse} \) measurements with the estimates provided by DynamoRIO. We see that the percentage error of the estimates does not exceed 15 %, while the average error is about 8%.

**The Data Entropy.** Earlier research indicated that the pattern of data stored in the memory might affect DRAM reliability significantly [10]. We quantify the varying data patterns (DPs) stored in DRAM using a specific metric, the DP entropy, \( H_{DP} \). We estimate \( H_{DP} \) by sampling the data stored in DRAM as:

\[
H_{DP} = -\sum_{i=0}^{2^{32}-1} P(x_i) \times \log_2(P(x_i)); P(x_i) = \frac{N_{WR}(x_i)}{N_{WR}^2}
\]

(6)

where \( N_{WR}(x_i) \) is the number of writes operations with data \( x_i \) in a word, while \( N_{WR} \) is the total number of writes (Figure 6 illustrates an example of the data entropy...
measured for 4-bit words). We obtain $H_{DD}$ by instrumenting and profiling applications using DynamoRIO, which samples the data stored by each DRAM write instruction.

**Hardware Performance Counters.** DRAM reliability is significantly affected by the number of memory accesses executed per cycle, since a high memory access rate may increase the number of errors due to the-cell-to-cell interference. To evaluate and model such an effect along with the potential effect of other system-level parameters on DRAM error behavior, we collect 247 program metrics, including L1/L2/DRAM accesses (writes and reads) per cycle, and IPC and the SoC utilization, for each application using the perf hardware performance counters tool.

### 4.3 Characterization Results

In our previous research study [16], we characterized 72 DRAM chips using the described framework under lowered $V_{DD}$, different levels of $T_{REFP}$ and a range of DRAM temperatures. In particular, we run the benchmarks from Rodinia, Parsec and Ligra suites [24], [25], [26] under relaxed temperatures. In particular, we run the benchmarks from Rodinia, Parsec and Ligra suites [24], [25], [26] under relaxed $T_{REFP}$ and $V_{DD}$ at different temperature levels, i.e. 50 °C, 60 °C and 70 °C.

**Voltage:** We discovered that $V_{DD}$ scaling has a negligible effect on DRAM reliability, since we observed only a few CEs when reducing $V_{DD}$ down to 1.428V (the nominal $V_{DD}$ is 1.5V). Due to such a negligible effect, in the rest of the paper, we use the minimum $V_{DD}$ (1.428V) and investigate the effect of $T_{REFP}$ scaling on DRAM behavior. Note that if we lower $V_{DD}$ less than 1.428V, then the server immediately crashes. We assume the DRAM circuit is to stop working when we set $V_{DD}$ below this margin.

**Correctable errors.** In our experiments with all the benchmarks for DRAM operating under scaled $T_{REFP}$ and $V_{DD}$, we encounter only CEs at 50 °C and 60 °C, and no UEs or SDCs. Our study indicated that the rate of CEs (or WER) averaged over all DIMMs may vary by 3.4× across benchmarks when DRAM operates under relaxed $T_{REFP}$ and $V_{DD}$. We also discovered that WER varies across different DRAM chips by up to 188x [16]. Note that $V_{DD}$ is set to the minimum value (1.428V).

**Uncorrectable errors.** In our experiments with DRAM operating at 50 °C and 60 °C, we have witnessed no Silent Data Corruptions (SDCs) or uncorrectable errors (UEs). However, we encounter UEs and system crashes when raising the DRAM temperature to 70 °C and scaling $T_{REFP}$ up to 1.45 s under lowered $V_{DD}$. Note that any UE, once detected by ECC, will result in a system crash.

We found that $P_{UE}$ varies across benchmarks significantly, while the $P_{UE}$ averaged over benchmarks grows with $T_{REFP}$ [16]. Furthermore, similar to WER, we discovered that $P_{UE}$ varies across DRAM chips. For example, we found that 67% of DRAM errors were obtained on one DIMM, while less than 1% of UE were observed on another DIMM [16].

### 4.4 The impact of program features

To identify the program inherent features that are likely to have a strong impact on DRAM reliability, we extract 249 program features from workloads used in our characterization study and correlate them with WER and $P_{UE}$. To quantify the correlation, we use the Spearman’s rank correlation coefficient ($r_S$), which enables us to detect different types of linear and nonlinear relationships [27]. Coefficient values lie in a range $[-1, +1]$ in which −1 or +1 occurs when there is a strong correlation between variables (−1 is a negative correlation, +1 is a positive correlation) [16].

We build the correlation of WER and program features using the combined measurements taken under different refresh periods ($T_{REFP}$) and DRAM temperatures. Figure 7 shows the correlation coefficients for 249 program features and WER on the Y-axis, whereas the correlation coefficients for these features and $P_{UE}$ are shown on the X-axis. We see that the data entropy($H_{DD}$) and wait cycles (the ratio of the number of cycles spent on waiting for data) are correlated with WER ($r_S \geq 0.39$), while the highest correlation is obtained for the memory access rate ($r_S \geq 0.57$). Note that even though the $r_S$ for the DRAM reuse time ($T_{reuse}$) is only 0.23, it also affects WER, as we show in our previous study [16]. Finally, we found that all these program features are correlated with $P_{UE}$, however the correlation coefficients ($r_S$) for these features are lower than the same coefficients obtained for WER [16]. Thus, we may conclude that the memory access rate, wait cycles, the data entropy and the reuse time affect DRAM errors. In addition to the Spearman’s correlation analysis, we applied Analysis of Variance (ANOVA) based on F-test and mutual information regression to correlate the collected program inherent features and the DRAM error metrics. Both analyses confirmed that the chosen program features have a strong impact on DRAM error behavior [27].

### 5 Revealing the DRAM GuardBands

The goal of this work is to develop a DRAM error prediction model, which takes into account all the factors that may affect DRAM reliability, and use this model to find the marginal DRAM operating parameters that are likely to induce no errors, regardless of running workloads. In fact, if we were to mathematically formulate the prediction problem, then we could see that this results in a problem with a huge intractable parameter space. In particular, let us assume that a target DRAM error metric $M_{ERR}$ (for example, the number of single-bit errors) should be predicted for a workload having a specific set of program features ($F_{TRS} = (f_1, f_2, ..., f_K)$, where $f_i$ is the i-th feature) which allocates data on a DRAM device ($Dev$), when the device operates under $T_{REFP}$ and $V_{DD}$ at a certain temperature ($TEMP_{DRAM}$). Then, all possible combinations of these parameters can be expressed mathematically as the Cartesian product of the parameters sets given by:

$$\Theta = S_{DEV} \times S_{FTRS} \times S_{TREFP} \times S_{VDD} \times S_{TEMP}$$

where $S_{DEV}$, $S_{FTRS}$, $S_{TREFP}$, $S_{VDD}$, $S_{TEMP}$ define all possible values for $Dev$, $Ftrs$, $TREFP$, $V_{DD}$, the DRAM
temperature, respectively. To predict $M_{err}$ for a specific set of parameters $\theta$, we need to model a prediction function $(M)$ such that:

$$\text{M}_{err} = M(\theta \in \Theta) \quad (8)$$

However, it is hard, if not impossible, to find an analytical model $M$ that predicts DRAM error behavior accurately considering the DIMM-to-DIMM variation and all the factors discussed in the previous sections. To address this challenge, we propose to use a supervised Machine Learning (ML) method.

The ML model: We use the K-nearest neighbours (KNN) algorithm, since our previous study showed that it has the highest accuracy of predicting DRAM errors among other popular ML algorithms, such as Support Vector Machines (SVM) and Random Decision Forests [16]. We implement the algorithm using the scikit-library [28]. Note that the KNN algorithm does not have a traditional explicit training step. Instead of the traditional training, the algorithm stores the original training data (i.e. the the program features extracted from workloads and error metrics) and uses this data for predicting DRAM error behavior. Additional experiments revealed that the highest accuracy for the KNN algorithm in our study is obtained when the mean values averaged over 5 nearest values.

Training: To collect data for training of KNN, we run workloads used in our characterization campaign under varying DRAM circuits parameters and temperature, measuring $WER$ and $P_{UE}$ (see Figure 8). We additionally run each workload to collect all their inherent program features using DynamoRIO and the perf tool. Then, we combine collected program inherent features with the measured $WER$ or $P_{UE}$.

We train the model and predict the DRAM error metrics ($WER$ and $P_{UE}$) on the basis of the DRAM parameters $(T_{REFP}, V_{DD}, TEMP_{DRAM})$ and program inherent features that have the strongest impact on DRAM reliability, i.e. the memory access rate, wait cycles, $H_{DP}$, $T_{use}$ (see Section 4.4).

5.1 Exploring DRAM operating margins

**Algorithm 1** The search of the marginal $T_{REFP}$

**Input:** $\text{MARIN}, \text{Dev}, \text{TEMP}_{DRAM}, V_{DD}$

**Result:** $\text{MAX}_{T_{REFP}}$

$T_{REFP} = 0$

for $T_{REFP} \in [0.064, 2.283]$ do

for $Ftrs \in S_{FTRS}$ do

if $M(\text{Dev}, Ftrs, T_{REFP}, V_{DD}, TEMP_{DRAM}) \leq \text{MARIN}$ then

if $T_{REFP} > \text{MAX}_{T_{REFP}}$ then

$\text{MAX}_{T_{REFP}} = T_{REFP}$

end if

end if

end for

end for

Once the model is trained, it allows us to investigate the effect of various program inherent features on DRAM reliability without long-running characterization campaigns. By applying this model, we can identify the set of program inherent features (the worst-case set of program features) that are likely to incur the highest $WER$ or $P_{UE}$ for a specific $T_{REFP}, V_{DD}$ and DRAM temperature. Furthermore, through an exhaustive search we can find the set of DRAM circuit parameters for which the maximum possible $WER$ or $P_{UE}$ does not exceed a user-defined margin, according to the ML model. Specifically, to achieve maximum power savings, we need to find a pair of $T_{REFP}$ and $V_{DD}$ which minimizes the DRAM power. However, provided that the scaling of $V_{DD}$ has a negligible effect on DRAM reliability, to minimize the power, it is sufficient to find the maximum $T_{REFP}$ under the lowered $V_{DD}$. For which $WER$ or $P_{UE}$ does not exceed the margin. The logic of such a search is presented in Algorithm 1, where $Dev$ is the DRAM device (see also Equation 8); $S_{FTRS}$ defines possible values of program inherent features that have been used for training the model; $M$ is a predicted DRAM error metric ($WER$ or $P_{UE}$); $T_{MAX}$ is the maximum $T_{REFP}$ that matches the search criteria. Note that the program features used in our study are continuous and thus it is impossible to check all possible values of the program inherent features. As a result, to find the parameters that trigger the worst-case DRAM error behavior, we vary values of each program feature with the increments specified in Section 6.2.

By applying this search algorithm, we can find the marginal $T_{REFP}$ for which the maximum possible $WER$ obtained in the worst-case scenario is 0 (according to the ML model) when DRAM operates at a specific temperature. In other words, this model enables us to find the $T_{REFP}$ for which it is likely that none of workloads will trigger errors, including single-bit errors (or CEs). This algorithm can be also applied to find the marginal $T_{REFP}$ for which the maximum possible $P_{UE}$ obtained in the worst-case scenario is likely to be 0. Although this $T_{REFP}$ will be higher than the marginal $T_{REFP}$ obtained for $WER = 0$ and theoretically, will provide more power savings, DRAM operating under such a $T_{REFP}$ may manifest correctable errors. However, in the vast majority of data centers even CEs are not desirable [22], [29], especially in case of hundreds or thousands of CEs, since detected CEs indicate an abnormal hardware behavior. To follow this, in our study, we search for the marginal $T_{REFP}$ that doesn’t manifest any errors ($WER = 0$). It is important to note that it is extremely challenging to find the marginal $T_{REFP}$ ($WER = 0$) without applying the ML model, since the model enables us to vary program inherent features and detect the worst-case set of program features that is likely to incur the highest $WER$. It is hardly possible to detect such features by characterizing DRAMs, even with hundreds of benchmarks where the set of program inherent features is fixed.

5.2 Governor: Proactive policy

In our scheme, we characterize DRAM error behavior and train the ML model for each DIMM of a server, as shown in Figure 8. Using the ML model, we estimate the marginal $T_{REFP}$ for each temperature level which is possible in a particular data center. To enable dynamic control of $T_{REFP}$ and $V_{DD}$, we implement an OS governor and two different policies. According to our first policy (a proactive policy), the governor sets the marginal $T_{REFP}$ for each DIMM based on the measured temperature and estimates provided by the ML model (see Figure 8). A second, reactive policy is explained later in the text.
5.3 Incorrect predictions and side effects

Any incorrect prediction from the model may theoretically result in an UE and thus a system crash when we scale $T_{REFP}$ and $V_{DD}$. Moreover, the ML model does not take into account the effect of OS on DRAM reliability. Our study indicated that the effect of Linux on DRAM errors is insignificant, since we discover only a few CEs for DRAM operating under the maximum $T_{REFP}$ at 70 °C when there is no running applications. On the contrary, the memory intensive applications used in this study induce millions of errors when DRAM operates under the same parameters and conditions. Nonetheless, a particular event may start an OS service which will affect DRAM reliability. Beside this, DRAM reliability can be affected by alpha-particles [30] and degrade over time due to aging [31]. Finally, the integrity of data in DRAM can be compromised by row hammer attacks [12], especially when we relax $T_{REFP}$ and $V_{DD}$.

Thus, to address all these issues, we need to implement an additional mechanism that will detect any abnormal behavior for DRAMs operating under scaled circuit parameters and prevent system crashes.

5.4 Governor: Reactive policy

To implement a mechanism that automatically detects any abnormal DRAM error behavior, we extend the governor with a second policy (a reactive policy). The main idea behind this policy is that no errors should be manifested when DRAM operates under the marginal $T_{REFP}$ and $V_{DD}$ ($W_E R=0$) according to our model. Therefore, manifested CEs can be an indicator that DRAM operates abnormally. It is essential to note that before obtaining UEs, we always observe hundreds of CEs. Importantly, we observe this in our experiments with 36 DRAM chips, which contain over 137 billion cells in total. We thus consider our observation statistically significant for the specific DIMMs from one vendor. Moreover, our observation follows the previous studies that demonstrated that the probability of CEs is $10^{12}$ higher than the probability of UEs [2]. Thus, the governor needs to monitor the occurrence of CEs, and if CEs are detected, then the governor should set the nominal DRAM operating parameters to minimize the risk of a system crash.

However, the question arises what number of CEs (a threshold) should be considered as an indicator of an abnormal behavior? What is the probability of such a policy to miss an UE for a specific threshold?

To investigate this, we designed a framework that generates micro-benchmarks with randomized memory access/data patterns and collect the distribution of the number of CEs obtained before UEs appear. Note that we use benchmarks with randomly generated patterns to correctly estimate the probability to obtain a specific number of CEs before UEs. Figure 9a shows the probability density function (PDF) for the number of CEs obtained before an UE on DIMM2, which has manifested the vast majority of UEs in our study.

We discovered that the density function combines several Gaussian functions with different means and standard deviations. An additional analysis of the collected data revealed that these functions correspond to different erroneous memory locations that manifested UEs. For example, the number of CEs reported before obtaining an UE in memory location 1 varies from 1300 up to 1650 (see Figure 9a), while the same number for location 2 is in the range between 1650 and 1800. After filtering the data, we found that the density function for each memory location, that manifested UEs, follows the normal distribution, since $p$-values of the D’Agostino-Pearson test [32] for all functions are higher than 0.39.
Interestingly, we also found that the mean number of CEs manifested before obtaining an UE in a memory location decreases with the growing probability to obtain an UE in this particular location. For example, the majority of UEs has been manifested in location 1, while the mean number of CEs reported before observing an UE in this location is the lowest (1531) among means measured for other locations. Based on these findings, we estimate the number of reported CEs (the threshold) after which the governor should set the nominal DRAM parameters using the density function discovered for location 1. Figure 9b depicts the cumulative density function for the number of CEs observed before an UE in location 1. According to this function, if we use a threshold of 100 CEs, then the probability to obtain an UE before the governor will set the nominal DRAM parameters is $4 \times 10^{-45}$ (for a threshold of 1 CE this probability is $3 \times 10^{-50}$). This probability is negligible in comparison with the probability of a server crash due to memory failures ($9 - 15 \times 10^{-5}$) obtained in modern data centers [33].

In this study, we use a threshold of 100 CEs, since it allows us to avoid a false activation of the reactive governor policy and, at the same time, provides enough time for the governor to change the DRAM parameters. In case of detecting an abnormal DRAM behavior in a server, the governor does not relax the circuit parameters until the model is retrained and new marginal $T_{REFP}$ are estimated (see Figure 8). Importantly, the specific actions in case of detecting an abnormal DRAM behavior should be defined by an operator of a data center, which is beyond the scope of our study.

One may argue that we could apply the developed reactive policy to scale the refresh rate without applying the proactive policy. However, if we use only the reactive policy to adjust the refresh rate, then we increase the probability to miss an UE and thus a system crash, which will be higher than $1 - 4^{-45}$. Meanwhile, by combining the proactive and reactive policies, we reduce this probability, which is essential for reliable and continuous operation of data centers.

6 Evaluation

In this section, we present the evaluation results of the developed governor.

6.1 Accuracy of the ML model
We validate the accuracy of the applied ML model using the cross-validation technique [16] by partitioning all the collected data into a test set and a training set, as discussed in our previous study [16].

It is known that the accuracy of an ML model depends on the input features that are chosen for training [16]. We trained the KNN model using the DRAM temperature, $T_{REFP}$ and various sets of program inherent features and investigate the accuracy of the model for each set. We found that KNN achieves the highest prediction accuracy in predicting of WER (89.5%) when the following program inherent features are used: the memory access rate, wait cycles, $H_D$, $T_{reuse}$. While the accuracy of $P_{UE}$ estimates is about 95 % for the same set of program features. To achieve the highest prediction accuracy, we use this set of program features to train the KNN model implemented in our governor. Note that this accuracy was averaged over different DRAM chips from the same vendor used in our study.

To further investigate how the accuracy of the model varies over chips from other vendors, we project the variation of DRAM error behavior across chips from several vendors. To this end, we build the distribution of biases of the number of manifested errors from the average number of errors across DRAM devices from different vendors, following the results presented in a previous study [2]. Figure 10 shows such a distribution where the X-axis presents the bias of the number of DRAM errors from the average number of errors detected in DIMMs from 4 different vendors. Based on previously reported results [2], we assume that this distribution follows the normal distribution with the standard deviation 4.0. We use this distribution to vary WER and $P_{UE}$ in the data used to train and test the ML model, simulating the variation of DRAM error behavior across chips from many vendors. Similar to our experiments with the Micron devices, we found that the accuracy of the ML model is about 90% and 95 % for WER and $P_{UE}$, respectively.

6.2 Evaluation of the governor
After training the model, we apply Algorithm 1 to find the marginal $T_{REFP}$ for WER = 0. In this search, we vary the values of program features in the following ranges: $T_{reuse} \in [0.05, 10]$; $H_D \in [0.31]$; the memory access rate $\in [0.001, 0.5]$; wait cycles $\in [0.001, 0.5]$. We change $T_{reuse}$, $H_D$, wait cycles and the memory access rate in increment of 0.05, 1, 0.01 and 0.01, respectively.

Figure 11a depicts the discovered marginal $T_{REFP}$ when WER = 0 for each DIMM/rank predicted by the model for DRAM operating at 50 °C. We see that the predicted $T_{REFP}$ varies across DIMMs by almost 9x; for example, the marginal $T_{REFP}$ detected by our model for DIMM1/rank1 is 1447 ms, while for DIMM2/rank1 this $T_{REFP}$ is 135 ms. In other words, according to our model, there exists a workload that may trigger single-bit errors on DIMM2/rank1 when it operates under $T_{REFP}$ which is higher than 135 ms at 50 °C. Thereby, according to our model, we almost cannot relax $T_{REFP}$ for DIMM2/rank1 without a risk to obtain CEs. Thus, a guardband $T_{REFP}$ of 64 ms used by the vendor is almost optimal for this specific DIMM. However, in the case of DIMM1/rank1 such a guardband for $T_{REFP}$ is too pessimistic, since the model shows that $T_{REFP}$ can be increased by 9x without observing errors. Notably, in the worst-case sets of program features, discovered by the search algorithm for different DIMMs and temperatures, $T_{reuse}$ is always slightly higher than $T_{REFP}$. Accordingly, a workload can degrade DRAM reliability significantly only when $T_{reuse} > T_{REFP}$, regardless of values of other program features, such as $H_D$.

By setting the marginal $T_{REFP}$, our governor enables power and energy savings without compromising DRAM
reliability. The governor sets the $T_{REFP}$ for a DIMM which is the minimum of the marginal $T_{REFPs}$ discovered for both ranks of the DIMM. Figure 12b shows the power savings averaged over all DIMMs that can be achieved by scaling $T_{REFP}$ and $V_{DD}$. In this figure, we provide the DRAM power measurements taken for the pagerank benchmark, which incurs the highest DRAM power in our experiments, and power measurements averaged over all benchmarks. We see that in both cases the DRAM power does not change if we set $T_{REFP}$ higher than 872 ms. We explain this by the fact that the DRAM power is almost logarithmically proportional (the base $< 1$) to $T_{REFP}$, and it does not change a lot with increasing $T_{REFP}$ after some point. Interestingly, the marginal $T_{REFP}$ discovered by the ML model for DIMM0, DIMM1 and DIMM3 operating at 50 °C are exactly 872 ms or even higher (1447 ms for DIMM1/rank1 and DIMM3/rank0). Thus, our governor achieves the maximum possible power savings, which is 24% on average, for these DIMMs by setting the marginal $T_{REFP}$ and $V_{DD}$ at 50 °C. However, if we raise the DRAM temperature to 60 °C, then the governor can scale $T_{REFP}$ only for DIMM1 and DIMM3 without compromising reliability. Moreover, according to the ML model, $T_{REFP}$ cannot be relaxed for any of the DIMMs when they operate at 70 °C without a risk to obtain an error (see Figure 11c). As a result, the governor will not provide any savings at this temperature.

We validated the predicted marginal $T_{REFP}$ by running different benchmarks and Virtual Machines (VMs) for DRAM operating at various temperatures for three weeks, and we have not discovered any CEs or UEs. Figure 11d shows the DRAM power measurements taken during one of our validation experiments. In this particular experiment, we run radix (Parsc), fft(Parsc) and lulesh benchmarks, which have not been used for training of the model. We also executed blackscholes, x264 and fluidanimate benchmarks running on QEMU/KVM VMs. We see that the DRAM power varies across benchmarks: fft incurs the highest DRAM power (more than 11W), while the benchmarks running on VMs consume less than 5W. We explain this by the fact that the memory access intensity drops when we use VMs, and as a result, the dynamic DRAM power also decreases. Nonetheless, the governor enables us to reduce the average power and the peak DRAM power by 24 % and 38% (fft), respectively, when DRAM operates at 50 °C. When we run benchmarks at 60 °C the DRAM power savings reduce slightly, since the governor reduces $T_{REFP}$ for DIMM0 to avoid memory errors following the model.

To validate the reactive policy of the governor in this experiment, we injected 100 CEs into the system log file monitored by the governor when running blackscholes and x264 (VMs) at 60 °C. We see that the governor sets the nominal values for $T_{REFP}$ and $V_{DD}$ when 100 CEs have been detected, and as a result, there is no power savings for these benchmarks at 60 °C. Finally, when we run our experiment at 70 °C, the governor uses the nominal DRAM settings according to the predictions provided by model.

**Workload-Aware Prediction:** Note that by default the governor sets the marginal $T_{REFP}$ discovered by the search algorithm (Algorithm 1) for a workload with the worst-case set of program features. To enable power savings at high temperatures, higher than 70 °C, we extend our governor to predict the marginal $T_{REFP}$ specifically for a running workload. In such a scheme, the workload-aware marginal $T_{REFP}$ is estimated by the ML model based on the program inherent features that are collected for each workload statically or at runtime. Figure 12a shows the workload-aware marginal $T_{REFP}$ (545 ms on average) predicted for the lulesh benchmark running at 70 °C, which is 9x greater than the nominal $T_{REFP}$ on average. Hence, by applying the workload-aware prediction scheme, we can achieve power savings even at temperatures above 70 °C.

Note that the extended version of the governor implements only static prediction of the DRAM guardbands for a particular workload. In other words, if the user would like to set the marginal DRAM refresh rate under lowered supply voltage for a workload, then this workload should be
run before to extract the most significant program inherent features and find the marginal refresh rate for this workload using the ML model. The discovered refresh rate is stored in a specifically designed hash table of the governor. Once the user indicates that he starts a workload (i.e., write the name of the workload in a specific file which is monitored by the governor), the governor extracts from the hash table the DRAM parameters for this workload and use them during the workload run. We also implemented an experimental version of the governor where the DRAM refresh rate under lowered supply voltage is dynamically adjusted based on the program features which are extracted at runtime. In particular, the governor monitors the performance counters for 10 seconds and predicts the safe refresh rate based on the collected counters using the ML model. We found experimentally that a 10 second period strikes a good balance between overhead, responsiveness of the governor and the coverage of different program phases. Note that in the case of workloads where the program phases take less than 10 seconds, the prediction can be inaccurate. However, in this case the reactive policy of our governor should set the nominal DRAM parameters when DRAM failures are obtained.

**Energy savings:** As opposed to common servers, the DRAM power on our platform is comparable with the processor power. As a result, our governor saves 10% of the total system energy, which is the sum of the processor energy and DRAM energy, on average. Moreover, for single-threaded versions of some applications, such as *lulesh*, the DRAM power is 1.6x more than the processor power, and the overall system energy savings for these applications may reach 17%. Notably, if we use 8 DIMMs, then the energy savings will increase up to 23%. Thus, the energy savings enabled by our governor scales up with the number of DRAM devices and can be significant, especially in the dedicated servers built for data analytics applications that use a lot of memory. Last but not least, it is predicted that the energy losses due to refresh operations will increase significantly for future DRAM technologies [1], which implies that the proposed design will provide higher power and energy savings for future DRAMs.

Importantly, the demonstrated power and energy savings are higher than the savings reported in the majority of previous studies which have introduced hardware modifications to reduce the number of refresh operations [1], [34], [35]. For example, *Smart Refresh* implements a mechanism that skips the refresh operations for those rows that have been accessed recently and reduces the DRAM energy by 12.3% [34], while *ESKIMO* uses a semantics-aware DRAM refresh and saves 39% of the DRAM energy on average [35]. *RAIDR* increases $T_{REFP}$ for the rows with a high retention time [1] and thus reduces the DRAM power by 16.1%. However, all these schemes have never been implemented on real hardware, and the reported savings are estimated using simulators but not real servers, as in our study.

Overall, our solution does not require intrusive changes to hardware, and can be easily deployed in state-of-the-art data centers. The proposed scheme can be also used in Edge deployments, where power budgets are stringent. Finally, unlike previous studies [1], [34], [35], [36], [37], we implement a mechanism which automatically detects any abnormal DRAM error behavior and ensures that scaling the DRAM parameters will not lead to a system crash with a probability of $1 - 4 \times 10^{-45}$.

### 7 Related Work

**Statistical prediction and modeling of faults:** Considerable research has been done on statistical prediction of different types of hardware faults, including DRAM errors, in supercomputers [38], [39], [40], [41]. The majority of these studies proposed different techniques, based either on rules or Machine Learning for prediction of failures that may happen in various hardware components using history of errors [40]. Some research studies made attempts to find out what input features should be used for training and if failure prediction models should be time-driven or event-driven [41]. Other works proposed techniques, such as a rule dynamic metalearning [39], to accelerate the model training phase. Recent studies applied statistical analysis and Machine Learning to predict errors at a fine-grained level, e.g., error prediction for a certain DIMM location (rank/row/column) [38]. Even though all these studies provide valuable insights, they ignore workload-dependent error behavior and provide models that are valid only for hardware operating under nominal conditions.

**Scaling $T_{REFP}$ and $V_{DD}$:** Many studies [1], [7], [8], [9], [10], [34], [42], [43], [44] tried to improve DRAM performance and energy efficiency by adopting a low refresh period for weak cells. Chang et al. [3] provided results of a comprehensive study on reduced-voltage operation in DDR3L memory devices. To reduce the DRAM access latency and energy, several studies proposed to relax the DRAM timing parameters or change the DRAM frequency, which we are also going to use in our governor in future research [6], [45].

Other research studies proposed various fine-grained schemes to reduce the number of refresh operations and thus improve DRAM energy efficiency [1], [34], [35], [36], [37]. However, to the best of our knowledge, all these schemes have never been implemented on real servers, and the reported savings are estimated using simulators. Moreover, previous studies have not proposed any mechanism to prevent an abnormal DRAM errors behavior that can be observed when DRAM circuit parameters are relaxed.

### 8 Conclusion

In this paper, we present an automatic technique for scaling DRAM operating parameters, such as $T_{REFP}$ and $V_{DD}$, that minimizes the probability of possible failures. The main idea behind the proposed approach is that DRAM error behavior is workload-dependent and it can be predicted based on specific program inherent features, which we demonstrated in our previous study. We use a Machine Learning method to build a workload-aware DRAM error behavior model based on the extracted program inherent features, DRAM parameters and temperature. In this study, we show that the developed model can be efficiently applied to identify the marginal DRAM refresh period under lowered supply voltage which is likely to not trigger errors when running any possible workload for a specific DRAM device. We implement a governor that automatically sets the marginal
DRAM parameters based on the measured DRAM temperature and thus reduces the DRAM power. Our governor enables us to save 24% of the DRAM power on average, while minimizing the risk of data integrity violations. Finally, unlike previous studies, our scheme, which does not require intrusive changes to hardware, is implemented on a real server and can be easily deployed in state-of-the-art data centers.

ACKNOWLEDGMENTS
This work was funded by the H2020 Framework Program of the European Union through the UniServer Project (Grant Agreement 688540, http://www.uniserver2020.eu) and OpreComp project (Grant Agreement 732631, http://oprecomp.eu). We are grateful to Dr. Philip Hodgers (ECIT) for providing the thermal testbed.

REFERENCES