Fault-Aware Prediction-Guided Page Offlining for Uncorrectable Memory Error Prevention

Xiaoming Du*, Cong Li*, Shen Zhou*, Xian Liu†, Xiaohan Xu†, Tianjiao Wang†, Shijian Ge‡
* Intel Corporation, Shanghai, China, {xiaoming.du, cong.li, shen.zhou}@intel.com
† ByteDance, Beijing, China, {liuxian.1, xuxiaohan, wangtianjiao.joyce}@bytedance.com
‡ ByteDance, Shanghai, China, geshijian@bytedance.com

***Abstract*—Uncorrectable memory errors are the major causes of hardware failures in datacenters leading to server crashes. Page offlining is an error-prevention mechanism implemented in modern operating systems. Traditional offlining policies are based on correctable error (CE) rate of a page in a past period. However, CEs are just the observations while the underlying causes are memory circuit faults. A certain fault such as a row fault can impact quite a few pages. Meanwhile, not all faults are equally prone to uncorrectable errors (UEs). In this paper, we propose a fault-aware prediction-guide policy for page offlining. In the proposed policy, we first identify row faults based on CE observations as the preliminary candidates for offlining. Leveraging the knowledge of the error correction code, we design a predictor based on error-bit patterns to predict whether a row fault is prone to UEs or not. Pages impacted by the UE-prone rows are then offlined. Empirical evaluation using the error log from a modern large-scale cluster in ByteDance demonstrates that the proposed policy avoids several times more UEs than the traditional policy does at a comparable cost of memory capacity loss due to page offlining.**

***Index Terms*—memory reliability, uncorrectable error prevention, page offlining, row fault identification, uncorrectable error prediction**

I. Introduction

Dynamic random access memory (DRAM) is extensively used in modern computing systems as the main memory for fast data storage and retrieval. While we expect the data read from DRAM to be the same as that stored, in reality it may not. Various factors may contribute to the problem, e.g., the faulty circuit, the leak-off of electric charges before getting restored, unstable data transmission over bitlines, etc.

Error correction codes (ECCs) have been developed to detect and correct errors. Errors corrected by ECC are referred to as correctable errors (CEs). When more errors occur simultaneously, the error pattern goes beyond the error correction capability of the ECC, but mostly still stays within the error detection capability of the ECC. An uncorrectable error (UE), or more precisely, a detected uncorrectable error, then happens, typically leading to a system crash. Memory failures have become one of the leading causes of hardware failures in datacenters.

The alternative strategy to mitigate memory errors is to stop using the faulty memory cells before a UE happens. Operating systems (OSs) such as Windows, Linux, and Solaris [1] provide the mechanism of page offlining to avoid memory errors based on predefined policies [1]–[5]. When the condition of page offlining in a policy is satisfied, the OS copies the content of the page to another location and stops using the original page. The cost of page offlining is the loss of the memory capacity of the pages offlined. The traditional policy is to offline a page when it reaches X CEs in a past period of T hours (denoted as X/T). The default configurations are $10/24$ in Linux and $16/24$ in Windows respectively.[1] Such a policy offlines a page according to the CE rate in a page in a certain period.

However, the common page offlining policy based on CE rate has two weaknesses.

- Memory errors are just the observations while the underlying causes are memory circuit faults. Faults such as a row fault, a column fault, or a bank fault can impact a few to a lot of pages. Counting CEs per page does not comprehend the nature of the cross-page faults.
- Not all the faults (or the pages with the CE rate satisfying a certain condition) are equally prone to future UEs. The CE rate in the past period is not a good predictive indicator of future UEs [6].

Suffering from the two weaknesses, only a few UEs can be avoided using the traditional page offlining policy.

In this paper, we propose a new fault-aware prediction-guided page offlining policy to overcome the two weaknesses. The proposed policy takes the row fault type, the most appropriate fault type, as the basis of page offlining. This is because that unlike a column or a bank, a row maps only to a limited number of memory pages, making the cost of memory capacity loss acceptable. In the new policy, we first examine the locations of the past CEs in a row to infer whether the row is faulty or not. We next leverage the knowledge of the ECC to design a predictor based on error-bit patterns observed in the CE history. The predictor is then applied to the inferred faulty rows to predict whether a row fault is prone to UEs or not. Pages impacted by the UE-prone rows are offlined.

We perform empirical evaluation of the new approach on the DRAM error log from a modern large-scale cluster in ByteDance. While in the previous work page offlining policies are evaluated in terms of the number of CEs avoided [2]–[5], we are the first to present the evaluation in terms of the number of UEs avoided. Experimental results indicate that at

[1]See for reference, http://www.mcelog.org/badpageofflining.html and https://docs.microsoft.com/en-us/windows-hardware/drivers/whea/whea-pfa-registry-settings.

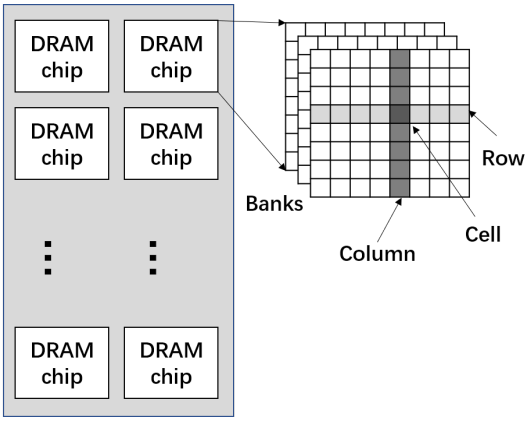

Fig. 1: Simplified logical view of a DIMM.

the similar cost of memory capacity loss due to page offlining, the proposed fault-aware prediction-guided policy is able to avoid several times more UEs then the traditional policy does.

II. Background

A dual in-line memory module (DIMM) contains a series of DRAM integrated circuits. Each chip contains several banks of two-dimensional arrays of cells. Each cell contains a set of data bits and is addressed by the row index and the column index in the array. Figure 1 shows a simplified logical view of a DIMM. In each memory access, a 64-byte cache line is read or written from the cells spanning over multiple chips. Each chip only provides part of the data bits through several beats.

In the multi-channel memory architecture on modern server systems, each memory controller in a processor governs several memory channels. DIMMs are populated over those channels. To accelerate memory access in parallel, a contiguous region of memory addresses is typically interleaved over those DIMMs. On a contemporary Intel SkyLake or Cascade Lake server, on average a row can contain data from 48 4K-byte pages at most (when the channels are fully populated as the worst case). In contrast, a column contains data from a lot more pages, e.g., 131,072 pages in a 32GB DIMM.

We refer to the underlying cause of a memory error as a *fault*. We refer to non-reversible damages in hardware, e.g., cells of stuck-at bits, as *hard faults*. Faults can get developed on the low-level DRAM circuits with a large granularity such as rows, columns, banks, etc. We refer to the observed symptom of a fault as an *error*, e.g., after storing a bit of 0 to a cell of a stuck-at bit of 1, a bit of 1 is read. We refer to errors from hard faults as *hard errors*. *Soft faults* and *soft error* observations are random, e.g., bit flips caused by particle strikes. A fault may not necessarily trigger errors, e.g., storing a bit of 1 in a stuck-at bit of 1, or storing a bit of 0 in it without further accesses.

ECCs have been developed to detect and correct errors. For example, the chipkill ECC [7] is able to correct any number of error bits in a single chip. If the error pattern goes beyond the correcting capability of the ECC, e.g., the coincidence of a hard error in a chip with a soft error in a second chip in chipkill, a UE happens leading to a system crash.

Note that the occurrences of both CEs and UEs depend not only on the faulty status of the hardware but also on the implicit runtime context, e.g., memory access pattern, data content, etc. In practice it is intractable to observe the hardware faulty status (e.g., to which extent the hardware wears out), since an offline comprehensive memory test is required to identify the underlying faults, which is disruptive to the serviceability of the servers. In practice it is also intractable to track the runtime context (e.g., memory access pattern and data content), since the overhead in collecting those details becomes prohibitively high. As a result, memory errors are highly uncertain.

Page offlining stops the use of certain pages in the OS, preventing the access to potential faulty cells in those pages and consequently masking the memory errors caused by the faults [1]–[5]. Algorithm 1 shows the traditional page offlining policy in OSs. We keep CEs in the past T hours tracked in the error history. Whenever a new CE is observed, we identify the corresponding page and count the CEs in the page in the history. If the count reaches X, the page offlining action is triggered. The typical choice of T is 24.[2] The choice of X is used to trade error avoidance with capacity loss in page offlining. The default choice of X is 10 in Linux and 16 in Windows.

Algorithm 1 Traditional page offlining policy in OSs with the predefined parameters of X and $T = 24$

```
procedure OnReceivingCE(errorAddress)
    errorHistory.TrackErrorsInTime(errorAddress, T = 24)
    page ← GetPage(errorAddress)
    errorCount ← errorHistory.CountCEsInPage(page)
    if errorCount ≥ X then
        OfflinePage(page)
```

III. Weaknesses of the Traditional Policy

We re-examine the traditional page offlining policy and demonstrate its two weaknesses through real examples.

Weakness 1: Memory errors are just the observations while the underlying causes are memory circuit faults. Faults such as a row fault, a column fault, or a bank fault can impact a few to a lot of pages. Counting CEs per page does not comprehend the nature of the cross-page faults.

Here we take a real example from a modern large-scale cluster in ByteDance. A DIMM locates in slot 1, channel 0, memory controller 1, CPU 1 on a server. There is a row with ID 0x1A37C coming from bank 2, bank group 1 in the DIMM. Table I shows the error history of 3 pages associated with the row. 2 of the 3 pages have CEs in the history. Specifically, the first page encounters 10 CEs in 6 minutes. In contrast, the third page has no CEs in the history. Actually in this example, all the CEs from the pages locate in the row. Not shown in the table, there are other pages encountering CEs in the same

[2] A 24-hour period is also the common choice on contemporary server platforms for the memory error tracking in firmware for predictive failure analysis.

TABLE I: A real example of 3 pages in a row.

Page ID	0xF5B7912	0xF5b7937	0xF5B7936
Row ID	0x1A37C	0x1A37C	0x1A37C
CE occurrence	10 in 6 minutes	1 in the history	0
UE occurrence	No	No	Yes

TABLE II: A real example of 2 rows from 2 servers.

	Row 1	Row 2
CE number	908	128
Unique CE locations	52	20
Pages with CEs	19	6
Pages with 10 CEs in 24 hours	4	0
UE occurrence	No	Yes

row as well. Based on the CE observations, very likely the row is faulty. Assuming that the fault is prone to future UEs, all the pages associated the row are likely to experience future UEs. However, if we only consider the CEs in the individual pages, only the first page will be offlined with the default Linux policy, but the third page will never be considered as a candidate due to its 0-CE history. Unfortunately, a UE actually occurs in the page with a 0-CE history.

Some other simplified policies are studied in [2]–[4]. While the details are different, those policies suffer from the same weakness of examining the CEs in the individual pages only. Those policies cannot correctly handle this example as well.

Weakness 2: Not all the faults (or the pages with the CE rate satisfying a certain condition) are equally prone to future UEs. The CE rate in the past period is not a good predictive indicator of future UEs [6].

Here we take another real example from the same cluster. In Table II, we show two rows. The first row with ID 0x7F52 comes from bank 1, bank group 1, DIMM slot 1, channel 0, memory controller 0, CPU 0 on a server. The row encounters more than 900 CEs in the past. Those CEs spread over more than 50 locations in the row as well as in many pages. 4 pages reach the rate of no less than 10 CEs in 24 hours. In contrast, the second row with ID 0x13AF1 comes from bank 2, bank group 0, DIMM slot 0, channel 0, memory controller 1, CPU 0 on another server. The row experiences less CEs on less unique locations as well as in less pages. None of those pages reach the rate of no less than 10 CEs in 24 hours. However, no UEs occur in the first row or in pages associated with the first row while a UE occurs in a page associated with the second row.

In [5], a new page offlining policy is proposed to combine the error statistics with a heuristic hint based on the prediction of future CE occurrence in the page. However, the hint is based on the likelihood of experiencing future CEs, but not future UEs.

The weaknesses described above inspire us to develop a new policy that comprehends both the underlying DRAM faults and the likelihood of experiencing future UEs.

IV. Fault-Aware Prediction-Guided Policy

In this paper, we propose a new fault-aware prediction-guide policy for page offlining:

- Instead of counting the historical CEs in an individual page, the new policy examines the historical CEs in a row to infer whether a row is faulty or not. The pages associated with those inferred faulty rows become the preliminary candidates for page offlining.
- We leverage the knowledge of the ECC to design a predictor based on error-bit patterns observed in the CE history. The predictor is able to predict whether future UEs are likely to occur in the faulty rows or not. Pages impacted by the UE-prone faulty rows are offlined.

In the next several subsections, we describe the details of identifying faulty rows and predicting future UEs in the rows. We then summarize the algorithm.

A. Faulty Row Identification

Row faults can be caused by different reasons such as row address decoding failure, circuit damage on a word-line or a sub-word-line driver, etc. In our approach, we identify faulty rows as the preliminary candidates for page offlining. We do not try to identify faulty columns or faulty banks. This is because that mitigating a column fault or bank fault with page offlining results in a significant loss of memory capacity. For example, in a 32GB DIMM, a column contains the content from 131,072 4KB pages implying a capacity loss of 512MB in offlining. In contrast, the number of pages impacted by a row is limited. The actual number depends on memory interleaving. On contemporary Intel SkyLake or Cascade Lake server platforms, the worst case is that all the channels are fully populated. Given a fully populated setting, a row contains the content from 48 pages on average.

Memory errors are the observed symptoms of the underlying faults of the hardware revealed by certain runtime context. One may try to discover the latent faulty status from the historical error observations. A scheme to identifying column faults has been proposed in [6]. In this paper we extend the scheme to identify the row faults as follows. When the circuit of a row is faulty, likely quite a few CEs will occur on different locations in a row. If the CEs observed in a row in the past $T = 24$ hours scatter in a region spanning no less than a length of l_r, and the unique error location number[3] reaches a predefined threshold θ_r, we infer that the row is faulty. Figure 2 shows an example of an inferred row fault.

Pages associated with the inferred faulty rows are regarded as the preliminary candidates for page offlining. We use a reverse address translation module to map the row to the pages. Such a module is available in BIOS on contemporary Intel server platforms.

[3]In [6], it is demonstrated that counting unique error locations is more indicative than counting CEs.

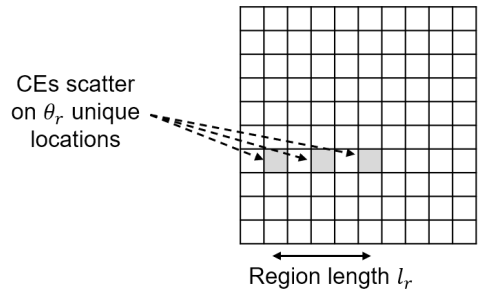

Fig. 2: Example of an inferred row fault.

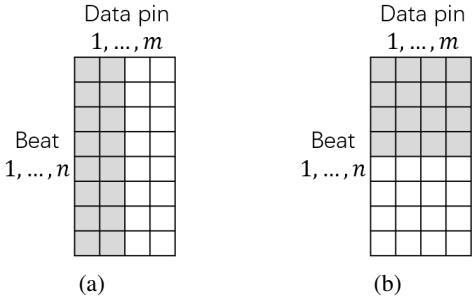

Fig. 3: Examples of (a) a fully correctable error-bit pattern; and (b) a partially correctable error-bit pattern.

B. UE Prediction

Not all the inferred row faults are equally prone to future UEs. The internal architecture of the DIMMs is among the highly confidential information of the DIMM manufacturers and is not shared even with the partners. Inferred row faults with similar observations may actually come from circuit damages on different sub-row components. Whether a certain fault may lead to future UEs depends on how well the ECC can covers the error bits revealed from the fault by the runtime context. Leveraging the ECC knowledge but not the knowledge on the DIMM internal architecture, we now design a predictor to predict how likely a faulty row will experience future UEs.

Unlike the chipkill ECC, ECCs on contemporary Intel server platforms do not guarantee a full coverage of all the possible errors on data bits from an individual chip during a cache line access. While the exact ECC algorithms are never exposed (see., e.g., [8]), coarse-grained error-bit patterns that are fully correctable or only partially correctable can be obtained, e.g., as the confidential information under certain non-disclosure agreements. While here we cannot expose the details of the confidential information, we use a publicly available example to illustrate how we design such a predictor from those patterns.

We assume that a chip in a DIMM provides $n \times m$ bits of data during a cache line access. The data bits are served through n beats over m data pins. We visualize the $n \times m$ bits in a bitmap, as is shown in Figure 3(a) or (b). Note that the visualization is for illustrative purpose only. The actual architecture of the bit storage depends on the design of the DIMM manufacturer and can vary significantly from the visualization, which is unknown to us. Patterns of a certain ECC apply only to the logical view of the bitmap, but not to the actual architecture. In a certain ECC implementation described in [9], if all the bits with errors are bounded within the left half of the bitmap, i.e., the first $\frac{m}{2}$ data pins over the n beats as is shown in Figure 3(a), the error pattern is fully correctable by the ECC. In contrast, if the bits with errors exist in all the m data pins and also in all the the first $\frac{n}{2}$ beats, i.e., the upper half of the bitmap as is shown in Figure 3(b), the error pattern is only partially correctable by the ECC.

We design the predictor of predicting UEs on faulty rows for server platforms with Intel SkyLake and Cascade Lake architecture (the most recent server architecture from Intel) as follows. We obtain the ECC knowledge of the platforms in terms of fully correctable error-bit patterns and partially correctable error-bit patterns. For each CEs in the row, we get the detail error-bit information and match the error bits against the patterns. We count the number of CEs that match *at least one* of the partially correctable patterns but do not match *any* of the fully correctable patterns. If the count reaches a predefined threshold of θ_{ECC}, the faulty row is predicted to experience future UEs. A page offlining action is triggered to offline the pages impacted by the row.

Algorithm 2 Fault-aware prediction-guided page offlining policy with the predefined parameters of l_r, θ_r, and θ_{ECC}

```
procedure OnReceivingCE(errorAddress, errorBits)
    errorHistory.TrackErrorsInTime(errorAddress, T = 24)
    row ← GetRow(errorAddress)
    rowErrorInfo ← errorHistory.GetErrorInfoInRow(row)
    if rowErrorInfo.GetMaxLocIdx() −
            rowErrorInfo.GetMinLocIdx() ≥ l_r and
            rowErrorInfo.CountUniqueLoc() ≥ θ_r then
        row.faulty ← True
    if MatchPartiallyCorrectablePatt(errorBits) and
            not MatchFullyCorrectablePatt(errorBits) then
        row.matched ← row.matched + 1
    if row.faulty and row.matched ≥ θ_ECC then
        pages ← row.GetPages()
        for each page in pages do
            OfflinePage(page)
```

C. Algorithm and Overhead Analysis

Algorithm 2 shows the algorithm of the new fault-aware prediction-guided page offlining policy. Different from the traditional page offlining policy, the new policy requires the additional input of error-bit information. The information is available today on contemporary Intel platforms from the read retry registers in the memory controller.

Both the traditional policy and the new policy are invoked with a new CE observed. While the traditional policy groups the historical CEs by pages, the new policy groups the historical CEs by rows. The overhead from CE grouping is similar. The additional execution overhead brought in by the new policy is to calculate the CE statistics (i.e., unique locations with CEs and the length of the spanning region) in a row and to match the fully correctable and partially correctable patterns. We accelerate the process of calculating the CE statistics in a row

through a simple refinement of the CE history tracking process. For each row with historical CEs in the past T hours, we keep a sorted set of unique locations with CEs along with their last observed CE timestamps. Given a newly incoming CE, we create or update the corresponding element and also remove the earlier elements falling out of the past T-hour window. Note that at the granularity of cache line accesses, the maximum list size is at most 128 per row. Therefore the *worst-case* execution-time overhead is no more than $O(n)$ where $n \leq 128$ is the maximum number of unique locations with CEs in a row in the T-hour Window. While an optimized implementation using a monotonic stack can ensure the *average* complexity of $O(1)$, here we regard that the simple implementation is good enough given the small scale of the problem. Matching patterns with error-bit information is trivial and is invoked once for a newly incoming CE. The execution-time overhead is a small constant. Overall, the execution-time overhead is small.

The new policy stores the CE history in unique CE locations while the traditional policy stores the CE history in CEs. The new policy is at least no worse than the traditional policy in terms of the storage overhead. Note that the error-bit information is consumed at the error time but is not stored.

V. Experiments

We perform experiments to evaluate the performance of the fault-aware prediction-guided policy on the DDR4 memory error log collected from a modern large-scale cluster in ByteDance. The dataset comes from around 10,000 of servers running a mix of various workloads including, e.g., internet services, online data feeding, offline data analytics, etc. All the servers are with the SkyLake or Cascade Lake architecture, i.e., the most recent server architecture from Intel. The DIMMs on the servers come homogeneously from a major DIMM manufacturer which we anonymize here due to the concern of inappropriate information disclosure.

The CE data is collected through the Linux error detection and correction driver. The data collection period spans from Sept. 2020 to Feb. 2021. Neither hardware sparing technologies (e.g., partial cache line sparing, row, bank, or chip sparing) nor software sparing technologies (e.g., software page offlining in OSs) are enabled during the period. Around 0.9 millions CEs from more than 0.2 million unique memory address locations are observed on more than 1,500 DIMMs from around 1,200 servers during the period. There are also 195 UEs recorded from the machine check events with the error address information on 195 servers.

A. UE Avoidance

We replay the error log and run the new fault-aware prediction-guided policy in a trace-driven simulation. A policy under evaluation takes the time series data of the CEs as the input. Pages are offlined accordingly.[4] If an UE arrives, we check whether or not the UE locates in a page which has already been offlined. We are then able to determine whether the UE is avoided.

[4]Note that once a page is offlined, future CEs in the page are ignored during the replay.

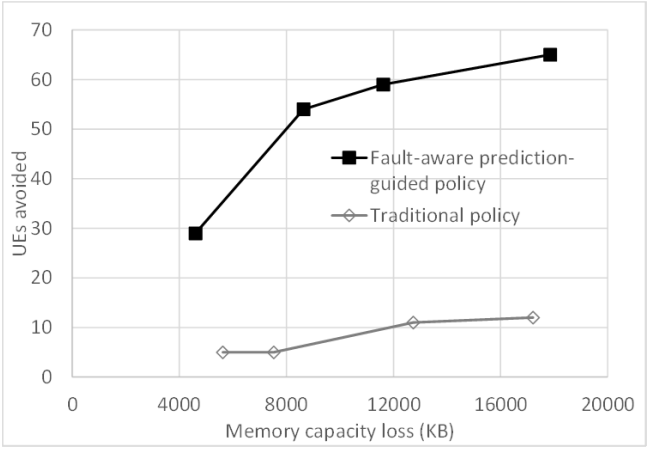

Fig. 4: Number UEs avoided vs. memory capacity loss in KB by the new policy and the traditional policy.

To evaluate how well the policy performs, we examine the number of UEs avoided and the memory capacity loss due to page offlining. While in the empirical evaluation using DRAM error log in the previous work [2]–[5], page offlining policies are evaluated in terms of CEs avoided, we are the first to present the empirical study of page offlining performance in terms of UEs avoided.

Both the number of UEs avoided and the memory capacity loss may vary with respect to the thresholds of l_r, θ_r, and θ_{ECC} in Algorithm 2. With a set of more rigid thresholds, less pages are offlined but less UEs are voided. We vary the thresholds by gradually making them more rigid as $(l_r, \theta_r, \theta_{ECC}) \in \{(\frac{1}{64}L_r, 2, 2), (\frac{1}{32}L_r, 3, 3), (\frac{1}{16}L_r, 5, 5), (\frac{1}{8}L_r, 10, 10)\}$ where L_r denote the length of a row. The performance metrics using those different choices are reported.

We compare the performance of the new policy against that of the traditional $X/24$ policy as the most appropriate baseline. Since the performance of the traditional policy may also vary with respect to different choices of X, here we use different thresholds to see how the metrics vary accordingly.

Note that in both the traditional policy and the new policy, we always configure T to 24, since tracking CEs in the past 24 hours is a universally-agreed choice in Windows, Linux, and BIOS.

Figure 4 plots the curves of the number of UEs avoided versus the memory capacity loss due to page offlining by both policies. As we can see from the figure, only a few UEs can be avoided by the traditional policy with a reasonable cost of memory capacity loss. The new fault-aware prediction-guided policy consistently outperforms the traditional policy, avoiding several times more UEs given the similar memory capacity loss due to page offlining.

Table III shows the numeric results of the default Windows and Linux page offlining policies as well as some other policies studied in [2]–[5]. In this table, we also introduce an auxiliary

TABLE III: Numeric results of the typical page offlining policies.

Policy	Memory capacity loss (KB)	UEs avoided	Cost per UE avoided (KB)
1-error [2], [4]	724184	51	14200
2-error [2]	135244	40	3381
Repeat [2]	138436	28	4944
Proposed $(\frac{1}{64}L_r, 2, 2)$	**17856**	**65**	**275**
10/24 (Linux)	12740	11	1158
12/24+ [5]	12672	10	1267
Costa et al. 2014 [3]	11644	15	776
Proposed $(\frac{1}{32}L_r, 3, 3)$	**11616**	**59**	**197**
16/24 (Windows)	7528	5	1506
18/24+ [5]	7560	5	1512
Proposed $(\frac{1}{16}L_r, 5, 5)$	8640	**54**	**160**

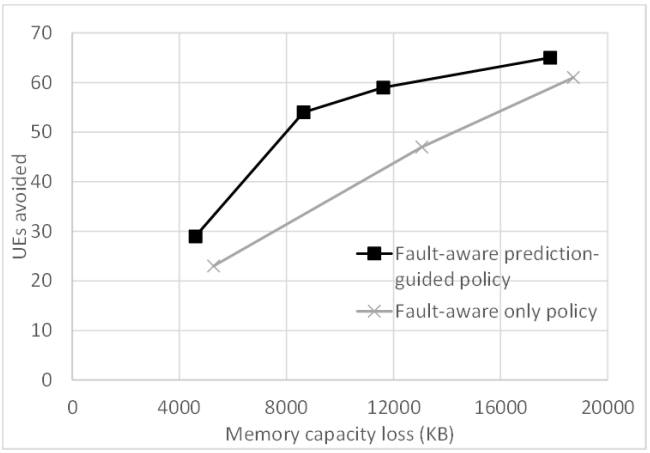

Fig. 5: Number UEs avoided vs. memory capacity loss in KB by the fault-aware only policy.

metric, cost per UE avoided in memory capacity loss, as a measurement of efficiency in page offlining.

'1-error' and '2-error' denote the simple policies of offlining a page if 1 error or 2 errors have been observed in the page [2], [4]. 'Repeat' denotes the simple policy that offlines a page once an address in the page experiences repeat errors [2]. Although those policies are able to avoid more UEs than the default Windows or Linux page offlining policy does, the UE avoidance is at the much higher cost per UE avoided, resulting in a much lower efficiency of page offlining.

Furthermore, as long as a policy only considers the historical CEs in an individual page, even if the pages are offlined in the most aggressive manner using the '1-error' policy at an extremely high cost per UE, it can only avoid 51 UEs at most. The fault-aware prediction-guided policy examines the historical CEs in a row that impacts multiple pages. Given the additional visibility, the proposed policy is able to avoided 65 (27% more) UEs using the least rigid set of thresholds. At the same time, the cost per UE avoided in page offlining is still much lower than that by any of the other baseline policies.

In addition to the default policies used in Linux and Windows, we also examine the performance of the X/24+ policy proposed in [5] which combines the error statistics with the hints from prediction of future CE occurrences [10]. However, the policy still does not address the two weaknesses of the traditional policy: 1) It only considers the CEs in the individual pages; 2) The hint comes from predicting future CEs, but not from predicting future UEs. Table III shows the performance of the X/24+ policy with the appropriate parameters to reach the similar cost of memory capacity loss to that by the traditional X/24 policy. The performance is similar to that of X/24, much worse than that of the new policy.

The policy proposed in [3], denoted as 'Costa et al. 2014', examines both the repeat CEs on the same location and the time interval between the last two CEs. The policy targets at avoiding chipkill CEs with page offlining. While a chipkill CE likely indicates a severe hardware wear-out, avoiding chipkill CEs is far from equivalent to avoiding UEs. The policy is not aware of row faults and is not predictive of UEs. As a result, though with a slightly better performance, it still performs much worse than the proposed policy does.

In summary, comparing with the traditional policies, the proposed fault-aware prediction-guided policy is able to avoid several times more UEs at the comparable cost of memory capacity loss. Comparing with the brute-force baselines (such as the '1-error' policy), the proposed policy is able to avoid considerably more UEs with a much lower (more than one magnitude lower when comparing with the '1-error' policy) cost.

B. How UE Prediction Helps

We now examine the role that the UE prediction plays in the proposed policy.

We start with an ablation study. In the ablation study, we evaluate a fault-aware only policy which is not guided with UE prediction. The policy infers row faults from CE observations and offlines the pages impacted by the inferred faulty rows, but does not check whether the faulty rows are prone to future UEs or not. Figure 5 plots the curve of the number of UEs avoided versus the memory capacity loss due to page offlining by the fault-aware only policy. For a lower number of UEs avoided, a larger memory capacity loss is required by the fault-aware only policy. This indicates that without UE prediction, some unnecessary page offlining actions are performed for faulty rows which are not prone to future UEs.

We next examine how well the UE predictor performs in row level UE prediction. Here, the prediction problem is to predict whether or not a future UE will occur in a specified row based on the past CE history. Note that in the problem setting we disregard whether an UE is actually caused by a row fault or not. We derive a row level UE predictor from the fault-aware prediction-guided policy. We measure its performance in terms of the common metrics of precision and recall. Over the rows predicted to experience future UEs, *precision* measures the percentage of rows with actual UEs in the future. A higher precision implies that more likely the page offlining operation triggered by the policy will avoid an actual UE in the future. Over all the rows with actual UEs, *recall* measures

TABLE IV: Performance of the row level UE predictor derived from the new policy and that of the page level UE predictor derived from the traditional policy.

	Precision	Recall
Row level UE predictor		
$(\frac{1}{64}L_r, 2, 2)$	34.95%	33.33%
$(\frac{1}{32}L_r, 3, 3)$	48.76%	30.26%
$(\frac{1}{16}L_r, 5, 5)$	60.00%	27.69%
$(\frac{1}{8}L_r, 10, 10)$	60.42%	14.87%
Page level UE predictor		
6/24	0.22%	7.18%
10/24	0.35%	5.64%
16/24	0.27%	2.56%
22/24	0.36%	2.56%

TABLE V: Comparison with the previous work on UE prediction.

	Prediction level	Input	Precision	Recall
[11]	Server	CEs, memory events, Performance counters	60%∼96%	7%∼31%
[6]	DIMM	Micro-level CE data	45%∼49%	24%∼35%
[12]	DIMM	Micro-level CE data, boot time, etc.	<2%	63%∼80%
[13]	DIMM	Micro-level CE data	27%∼75%	26%∼64%
Ours	Row	Micro-level data and error bits of CEs	35%∼60%	15%∼33%

the percentage of rows predicted to experience future UEs. A higher recall implies that more likely an UE will get avoided by the page offlining policy.

For reference, we also derive a page level UE predictor from the traditional page offlining policy, $X/24$. The predictor targets the problem at a different level, i.e., predicting whether or not a future UE will occur in a specified page based on the past CE history.

Table IV shows the performance of the row level UE predictor derived from the new policy and that of the page level UE predictor from the traditional policy. The row level predictor derived from the new policy achieves much higher recall scores than the page level predictor derived from the traditional policy does. (Note that the row level UE predictor is preconditioned with an inferred faulty row and therefore cannot cover UEs caused by other fault types.) This explains why the new policy is able to achieve a better coverage in UE avoidance. At the same time, the precision of the row level predictor is two magnitudes higher. Although many pages are offlined after identifying a UE-prone row fault, the new policy still enjoys a much lower cost per UE avoided, as is shown in Table III.

UE prediction has been studied empirically in [6], [11]–[13]. The problem formulations in different studies are different. Table V summaries the prediction levels, the input to the predictors, and the performance reported in those studies for a comparison with ours. Even though the UE prediction in this paper is at the fine-grained row level, the predictor is able to achieve a decent performance comparing with the other work of UE prediction at the coarse-grained DIMM or server level. (Note that UE prediction at the DIMM or server level does not help for page offlining.) We attribute the decent performance in such a challenging setting to the use of ECC knowledge and error-bit information as well as the homogeneous dataset with DIMMs from a same manufacturer.

In summary, UE prediction is an important component in the new policy. The decent performance in UE prediction is critical to the decent performance in avoiding UEs with a small cost in page offlining.

TABLE VI: Discriminating the UE-prone row from the two in Table II.

	Row 1	Row 2
Row fault inferred	Yes	Yes
CEs with the right patterns	0	14
Predicted to experience future UEs	No	Yes
UE occurrence	No	Yes

C. Case Studies

We now re-examine the two real examples demonstrating the weaknesses of the traditional policy in Section III to illustrate how the new policy overcomes the weaknesses.

In the first example in Section III, in row 0x1A37C, there are more than 20 unique locations encountering CEs. The locations span over a region with the length large than $\frac{3}{4}L_r$. Even if we limit our focus on CEs in a time window of 24 hours, there are enough CE observations to infer that the row is faulty. Furthermore, the number of CEs with a certain partially correctable pattern matched but without any fully correctable patterns matched also reaches the threshold θ_{ECC}. As a result, in the new policy, before the occurrence of the UE, all the pages associated with the row are offlined including the third page shown in Table I (i.e., the one with a 0-CE history). Consequently the UE is avoided. This case study shows that the new policy identifies the cross-page fault. It offlines a page with a 0-CE history by properly comprehending the CE information from other related pages to successfully avoid the UE.

In the second example in Section III, row faults are inferred on both the two rows based on their CE history, as is shown in Table VI. In the second row, 14 CEs match certain partially correctable patterns but do not match any fully correctable patterns based on the ECC knowledge. In contrast, in the first row, none of the CEs is with a right pattern. As a result, the second row is both faulty and UE-prone, and is offlined by the new policy, but the first row is preserved without page offlining. By predicting future UE occurrences, the new policy does not waste the memory capacity loss on pages impacted by a row which is not prone to UEs. Consequently it reduces the cost in UE avoidance.

In summary, the two case studies demonstrate how the new policy overcomes the weaknesses of the traditional policy

through inferring the underlying fault and predicting the likelihood of future UE occurrences.

VI. Other Related Work

Empirical studies on memory errors in clusters and datacenters have been performed in, e.g., [2], [4], [14]–[18]. The observations from those studies on the dominance of hard errors and the faults on micro-level DRAM circuits form the basis of our work.

Predicting future memory failures (see, e.g., [6], [11]–[13]) provides the opportunities to mitigate UEs, e.g., through proactive DIMM replacement. Specifically, in [6] and more recently in [13], fault identification mechanisms are proposed as the basis for UE prediction at the DIMM level. However, error-bit information is not used in [6], [13]. We enhance the UE prediction with the error-bit information, extend the UE prediction to the fine-grained level of rows in DIMMs, and use that as the guidance of page offlining, a different mitigation approach not explored in [6], [11]–[13].

In addition to software page offlining, hardware sparing mechanisms can be employed to avoid memory errors such as partial cache line sparing (see, e.g., [19]), sparing at the row or column level (see, e.g., [20]), sparing at the bank or chip level (see, e.g., [21]), etc. We believe that it is straightforward to extend the proposed approach to a policy for hardware row sparing.

VII. Conclusion

In this paper, we have proposed a new fault-aware prediction-guided policy for page offlining. In the new policy, we use the CE observations to infer the faulty status of a row. We also examine the error-bit information of the CEs and leverage the ECC knowledge to predict how likely a faulty row will experience future UEs. The pages impacted by those UE-prone faulty rows are offlined. Experimental results indicate that the new policy outperforms the traditional policy significantly.

References

[1] D. Tang, P. Carruthers, Z. Totari, and M. W. Shapiro, "Assessment of the effect of memory page retirement on system RAS against hardware faults," in *Proceedings of the International Conference on Dependable Systems and Networks*, ser. DSN '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 365–370.

[2] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic rays don't strike twice: Understanding the nature of DRAM errors and the implications for system design," in *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVII. New York, NY, USA: ACM, 2012, pp. 111–122.

[3] C. H. A. Costa, Y. Park, B. S. Rosenburg, C.-Y. Cher, and K. D. Ryu, "A system software approach to proactive memory-error avoidance," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 707–718.

[4] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," in *Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, ser. DSN '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 415–426.

[5] X. Du and C. Li, "Combining error statistics with failure prediction in memory page offlining," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '19. New York, NY, USA: ACM, 2019, pp. 127–132.

[6] X. Du, C. Li, S. Zhou, M. Ye, and J. Li, "Predicting uncorrectable memory errors for proactive replacement: An empirical study on large-scale field data," in *2020 16th European Dependable Computing Conference*, ser. EDCC '20, 2020, pp. 41–46.

[7] T. J. Dell, "A white paper on the benefits of chipkill-correct ECC for PC server main memory," Jan. 1997.

[8] M. Patel, J. S. Kim, T. Shahroodi, H. Hassan, and O. Mutlu, "Bit-exact ECC recovery (BEER): Determining DRAM on-die ECC functions by exploiting DRAM data retention characteristics," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '20. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2020, pp. 282–297.

[9] K. Criss, K. Bains, R. Agarwal, T. Bennett, T. Grunzke, J. K. Kim, H. Chung, and M. Jang, "Improving memory reliability by bounding DRAM faults: DDR5 improved reliability features," in *The International Symposium on Memory Systems*, ser. MEMSYS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 317–322.

[10] X. Du and C. Li, "Memory failure prediction using online learning," in *Proceedings of the 4th International Symposium on Memory Systems*, ser. MEMSYS '18. New York, NY, USA: ACM, 2018, pp. 38–49.

[11] I. Giurgiu, J. Szabo, D. Wiesmann, and J. Bird, "Predicting DRAM reliability in the field with machine learning," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track*, ser. Middleware '17. New York, NY, USA: ACM, 2017, pp. 15–21.

[12] I. Boixaderas, D. Zivanovic, S. Moré, J. Bartolome, D. Vicente, M. Casas, P. M. Carpenter, P. Radojković, and E. Ayguadé, "Cost-aware prediction of uncorrected DRAM errors in the field," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. IEEE Press, 2020.

[13] X. Du and C. Li, "Predicting uncorrectable memory errors from the correctable error history: No free predictors in the field," in *Proceedings of the 7th International Symposium on Memory Systems*, ser. MEMSYS '21. New York, NY, USA: ACM, 2021, to appear.

[14] X. Li, M. C. Huang, K. Shen, and L. Chu, "An empirical study of memory hardware errors in a server farm," in *Proceedings of the 3rd Workshop on on Hot Topics in System Dependability*, ser. HotDep '07. Berkeley, CA, USA: USENIX Association, 2007.

[15] ——, "A realistic evaluation of memory hardware errors and software system susceptibility," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIX ATC '10. Berkeley, CA, USA: USENIX Association, 2010, pp. 6–6.

[16] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: A large-scale field study," in *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '09. New York, NY, USA: ACM, 2009, pp. 193–204.

[17] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory errors in modern systems: The good, the bad, and the ugly," in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: ACM, 2015, pp. 297–310.

[18] V. Sridharan and D. Liberty, "A study of DRAM failures in the field," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 76:1–76:11.

[19] X. Du and C. Li, "DPCLS: Improving partial cache line sparing with dynamics for memory error prevention," in *2020 IEEE 38th International Conference on Computer Design*, ser. ICCD '20, 2020, pp. 197–204.

[20] C.-S. Hou, Y.-X. Chen, J.-F. Li, C.-Y. Lo, D.-M. Kwai, and Y.-F. Chou, "A built-in self-repair scheme for drams with spare rows, columns, and bits," in *Proceedings of the 2016 IEEE International Test Conference*, ser. ITC '16, 2016, pp. 1–7.

[21] X. Jian and R. Kumar, "Adaptive reliability chipkill correct (ARCC)," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture*, ser. HPCA '13, Feb 2013, pp. 270–281.