Predicting Memory Failures with a Two-stage Machine Learning Method

Zhuo Yin[1,2] and Jiacheng Lu[3]

[1] National Engineering Laboratory for Urban Rail Transit Communication and Operation Control, Beijing, China
`zhuoyin94@163.com`
[2] Traffic Control Technology Co., Ltd., Beijing, China
[3] China Merchants New Intelligence Technology Co., Beijing, China
`lujiacheng1@cmhk.com`

Abstract. Memory failures are widely considered as a common source of server breakdown, which significantly affects the reliability and availability. To reduce the effects of memory failures, memory failure prediction has become a research hotspot in recent years. This paper proposes a two-stage machine learning method to tackle the memory failure prediction problem. In the feature engineering process, we extract features to capture the underlying characteristics of error samples by employing a window-based strategy. At the first stage of the proposed method, a retrieval model is used to mitigate the issue of imbalanced data. At the second stage, we formulate the prediction problem as a regression problem instead of a classification problem to enhance the prediction performance. In the PAKDD2021 2^{nd} Alibaba Cloud AIOps Competition, the proposed method is verified using a large-scale dataset, where the method ranks second with the score of 64.52 in the preliminary contests. The rank is 7 with the score of 28.63 in the semi-finals.

Keywords: Memory failure prediction · Retrieval strategy · XGBoost.

1 Introduction

With the tremendous growth of the Internet and mobile communications, the large-scale data center has become an essential infrastructure in recent years. In a data center, maintaining system reliability is considered as a challenging issue [1]. However, the massive throughput of data leads to a high capacity utilization of the devices, which may cause more failures and service disruptions. Among these failures, errors and faults in Dynamic Random Access Memory (DRAM) give rise to most of the server breakdown [2]. Furthermore, the amount of the DRAM continues to increase in data centers, resulting in a far more serious problem in terms of memory failures. To ensure service reliability, how to detect and predict memory failures has become a research hotspot [3–5].

To detect and correct memory errors, error correcting codes (ECC) mechanisms are applied. Single-error correction and double-error detection (SECDED) as well as chipkill are widely used in large-scale systems [6]. The core idea of these mechanisms is to use parity-check code. Though additional storage overheads are required, certain memory errors can be successfully corrected. These kinds of errors can be caused by a lot of reasons, such as alpha particles or cosmic rays [7]. However, once errors cannot be corrected or detected, it may lead to memory failures, resulting in a great impact on system performance. To avoid this situation, various methods of predictive analysis have been proposed [4, 8, 9].

Among these methods, the machine learning approach is treated as a straightforward and effective way for memory failure prediction, as the memory errors are directly recorded as error events by the machine-check architecture(MCA). In [4], an online learning method was proposed to predict memory failure, in which the DRAM cells, ranks, rows, and columns are considered. Authors in [5] used a random forest classifier to predict DRAM uncorrected errors using error logs. Inspired by these studies, we propose a two-stage machine learning method to solve the problem of memory failure prediction, where the large-scale dataset is constructed from servers in Alibaba datacenters. The memory failure prediction task is defined as predicting whether a certain server may fail within the next 7 days. The main challenges in this task include problem modeling, class imbalance, data sparseness, etc.

In this paper, we deliberately explore a detailed feature engineering process to seize the internal relationship of the error data. In the proposed two-stage framework, the prediction problem is first expressed as a retrieval problem. By employing a weighting metric, both the F1 score of the error samples and the recall rate of the faulty servers are taken into account to mitigate the problem of imbalanced data. After the first stage, a regression model is applied to predict probable server crash caused by memory failures. The proposed method is submitted to PAKDD2021 2^{nd} Alibaba Cloud AIOps Competition. Experiment results demonstrate the superior performance of the two-stage scheme.

The rest of the paper is organized as follows. The related work is presented in Section 2. The proposed memory failure prediction scheme is detailed in Section 3. Section 4 verifies the performance of the proposed scheme. Section 5 concludes the paper.

2 Related Work

Schroeder et al. [11] presented an early replacement policy. This paper indicates that a large number of correctable errors lead to uncorrected errors. Costa et al. [9] proposed an OS-based approach to predict memory errors. The memory error patterns are analyzed to identify possible failures. Baseman et al. [10] explored an online machine learning-based approach to classify detected errors. The performance of random forest, logistic regression, and naive Bayes are compared. Du et al. [3] solved the DRAM failure prediction problem using correctable errors. Giurgiu et al. [12] used ensemble machine learning techniques to predict uncorrected DRAM errors, where the daily event logs and sensor measurements are considered. The relationship between the uncorrected errors and the corrected errors is revealed by these literatures. In large-scale datasets, the prediction accuracy suffers from the class imbalance issue heavily [13]. The sampling techniques which aim at adjusting the size of negative and positive classes are widely used to deal with this issue, such as random under-sampling and random oversampling [14]. However, the prediction performance is deteriorated since a great deal of false positives are introduced [15]. Xu et al. [15] presented a ranking model to cope with this problem using disk error datasets. In our work, we propose to employ a recall strategy to tackle this issue.

3 The Proposed Memory Failure Prediction Scheme

3.1 Background

The DRAM architecture in this competition consists of a series of dual-inline memory modules (DIMM). Typically there are 24 DIMM chips deployed in a server, which are indexed from 0 to 23. Each DIMM belongs to a channel which is connected to a certain memory controller. Each DIMM chip is made up of 2 ranks(ID 0 and ID 1). Each rank consists of 16 banks indexed from 0 to 15, and the banks can be accessed in parallel. Only one bank can be accessed during one memory access process. At the micro level, a bank can be treated as a two-dimensional array, in which each element is defined as a cell indexed with rows and columns. One bank stores 2^{17} rows and 2^{10} columns.

In this competition, the organizers provide three error log files. That is:

– Memory-Mce-Log. This log file reports the errors at the bank level using the machine-check architecture (MCA) registers, which includes the server ID, the server manufacturer ID, the memory vendor ID, the error bank ID, the mcelog transaction information, and the collect time.
– Memory-Address-Log. This log file can be regarded as a detailed description of the Memory-Mce-Log, in which the physical addresses of the errors are recorded at the micro level. Besides the information mentioned in the Memory-Mce-Log, the Memory-Address-log describes the memory ID, the rank ID, the row ID, and the column ID of a certain error.
– Memory-Kernel-Log. This log file collects information from the Linux kernel module, which may be associated with the memory errors.

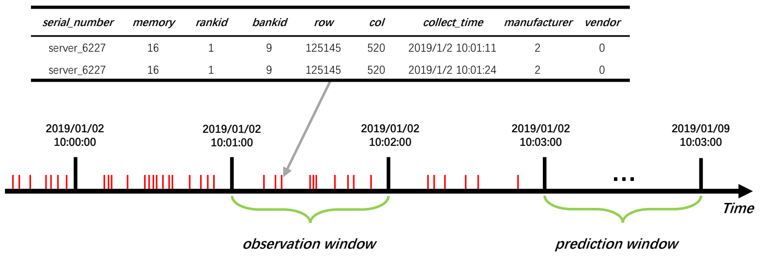
The dataset formed from these error log files contains huge volumes of data. As an example, Memory-Address-Log records 23442751 errors which are found from 17401 servers, 19869 memories and 23168 banks. The detailed information of these log files can be found in [16].

As aforementioned, a wide range of prior work focuses on capturing the relationship between the uncorrected errors and the correct errors. This paper is concerned with the errors collected in the error log files, both correctable and uncorrectable. A certain error is defined as an **error sample (event)**.

The organizers also provide a failure log file termed as Memory-Failure-Tag, which contains actual server crashes caused by memory failures. One record in this file consists of the server ID, the server manufacturer ID, the memory vendor ID, and the failure time. A certain record is defined as a **failure** and used as a label for the memory failure prediction problem. That is to say, this problem can be expressed as using error samples (bank/cell level) to predict whether a failure may occur at the server level.

3.2 Feature Engineering

In this subsection, we explain the feature engineering process. We propose to use a window-based strategy for feature extraction to mitigate the data sparseness issue. As can be seen in Fig. 1, at present time, the potential occurrence of a server failure is to be predicted in the future. The length of the time interval to make failure prediction is defined as the prediction window. For a certain server, the error events constitute a time series. We equally divide the error events into segments by time. A segment is defined as an observation window. Note that the number of error events differs between segments.



serial_number	*memory*	*rankid*	*bankid*	*row*	*col*	*collect_time*	*manufacturer*	*vendor*
server_6227	16	1	9	125145	520	2019/1/2 10:01:11	2	0
server_6227	16	1	9	125145	520	2019/1/2 10:01:24	2	0

Fig. 1. The window-based strategy for feature extraction.

The features can be mainly classified into the following categories:

Frequency features. Frequency features represent statistical characteristics of the frequency of error events. For instance, we count the number of error events in the past period(e.g., 60 minutes) for each error sample of each server at a specific timestamp, which is used as a feature. The average time interval for error events of each server within a certain period(e.g., 60 minutes) is also calculated and defined as a feature. These features reflect the change rate of error events, whereby the surge of error events can be captured.

Principle features: Principle features are designed according to the failure mechanism of the memory. Using the error detection driver, repeated accesses of one faulty location may generate a large number of error events, which implies a memory failure. The probable reason is stuck bits, where a cell is permanently stuck at a particular value [7]. Besides, prior work [4] shows that errors tend to propagate from cells to nearby cells. Based on these observations, for each error event, we count the number of the unique cell error addresses in the past periods. Similar features of other components in terms of memory, rank, bank, row, and column are also calculated.

Furthermore, we aggregate the aforementioned features based on the observation window and obtain the window-based features(e.g., the maximum value of unique cell error address in an observation window). By comparing the window-based features between the current observation window and kth past observation window, the differences of the window-based features can be seized, which can indicate whether a server may fail within the prediction window(e.g., 7 days).

Table 1. Feature extraction.

Feature ID	Feature Description	Log File
feat_0	- For each sample located in the same observation window, the maximum of the number of unique banks in the past k minutes.	Address log
feat_1	- For each sample located in the same observation window, the maximum of the number of unique rows/columns/cells in the past k minutes.	Address log
feat_2	- For each sample located in the same observation window, the maximum of the number of unique memories in the past k minutes.	Address log
feat_3	- The maximum, mean, variance, and the time shift value of samples located in the same observation window. - For each sample located in the same observation window, the maximum of the error event count in the past k minutes.	Address log
feat_4	- For each sample located in the same observation window, the maximum of the number of unique ranks in the past k minutes.	Address log
feat_5	- For each sample located in the same observation window, the maximum of the number of unique MCA IDs in the past k minutes.	MCE log
feat_6	- The one hot vector for vendor ID.	\
feat_7	- The one hot vector for manufacturer ID.	\
feat_8	- For each sample located in the same observation window, the maximum of the number of unique transaction IDs - The number of different types of the transaction ID in the past k minutes.	MCE log
feat_9	- For each sample located in the same observation window, the number of different types of the kernel error ID in the past k minutes.	Kernel log

Due to space restrictions, as can be seen in Table 1, we only list some features with high feature importance scores. Note that some features listed below may contain a group of features. In Table 1, $k \in \{15, 120, 360, 720, 1440, 2880\}$.

3.3 The First Recall Stage

At this stage, we formulate the prediction problem as a retrieval task. That is to say, instead of directly judging whether a certain server will fail, we first recall some meaningful error samples in order to improve the prediction accuracy in the second regression stage. The reason is that the retrieval model has the ability of filtering a large number of less significant error samples to mitigate the problem of imbalanced data.

The XGBoost model is trained with the features of the error samples, and the corresponding server failures are used as labels. The XGBoost model is selected since it is a robust and efficient model. As the labels indicate that most servers are healthy in the prediction window, we need to recall error samples whose corresponding servers are faulty. Meanwhile, the F1 score of the error samples also needs to be considered. Therefore, we propose a novel weighting metric which can be written as:

$$M = \alpha * F_1^{sample} + \beta * Recall^{server}, \alpha + \beta = 1 \tag{1}$$

F_1^{sample} represents the F1 score of the error samples, while $Recall^{server}$ means the proportion of faulty servers correctly judged to the total faulty servers. α and β are defined as the tuning parameters. There exists a trade-off between these two attributes. An excessively high F_1^{sample} implies that only a few samples are retained, resulting in a low value of $Recall^{server}$, which is obviously inappropriate.

We choose a threshold that can maximize the metric M. If the prediction probability of an error sample reaches the threshold, this sample is retrieved. Through this recall stage, one can see that the class imbalance issue is mitigated.

3.4 The Second Regression Stage

Normally, the memory failure prediction problem is transferred into a classification problem. And the label is set to 1 if the server fails within 7 days. Otherwise, the label is set to 0. Obviously, some meaningful information is lost in this rough method. In order to utilize more information, the samples retrieved from the recall stage are used to train a XGBoost regression model. Similar to [17], if a server failure is observed within 7 days, the label is set to the server failure time minus the current time. In addition, if a server failure occurs after 7 days, the label is directly set to 7.

To improve prediction accuracy, a threshold needs to be meticulously selected to make the final decision. By observing the distribution of the labels, we search the threshold within the label range and find that a smaller value of the threshold can achieve better performance. This can be explained that earlier failures which are successfully predicted represent higher confidences [17].

3.5 Training Process

The dataset provided from the organizer contains huge samples over a time period from January 2019 to May 2019. At the recall stage, the error samples from January 2019 are employed to train the XGBoost model, while the error samples from February 2019 are selected to form the validation set. The early stopping mechanism is applied in the process, using the ROC-AUC as the early stopping criterion. Multiple XGBoost classifiers with different hyper-parameters are trained to ensure robustness. After the models are trained, if the prediction probability of a certain sample calculated by these XGBoost models meets the threshold introduced in 3.3, the sample is retrieved. Otherwise, the sample will be ignored. Though only limited samples are used to train the model, the superior performance demonstrates the effectiveness of the retrieval strategy.

Several XGBoost regression models are trained at the second stage. The difference between these models is the training data. As we know, cross-validation is often used in prediction problems. But, it cannot be applied in the memory failure prediction problem. Therefore, similar to time series cross-validation, in the first model at the second stage, we only use the samples from January and February. In the next model, we add the samples from the next month. This process chronologically continues, until no more sample remains. We ensure that there exists no time overlap between the train and the validation data. We average the prediction results to obtain the final prediction result.

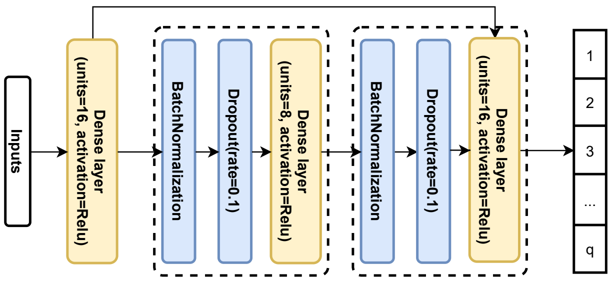

Inputs
Dense layer (units=16, activation=Relu)
BatchNormalization
Dropout(rate=0.1)
Dense layer (units=8, activation=Relu)
BatchNormalization
Dropout(rate=0.1)
Dense layer (units=16, activation=Relu)
1
2
3
...
q

Fig. 2. The architecture of the neural network classifier.

The proposed scheme is applied in the preliminary and semi-finals. Due to the change of the task and the evaluation metric, in the semi-final competition, we add a feedforward neural network classifier with residual connection to accurately predict the time to failure of faulty servers. This design can avoid the overestimate of the failure time compared with the regression methods. The rank is 7 with the score of 28.63 in the semi-finals. The architecture of the neural network classifier can be seen in Fig. 2.

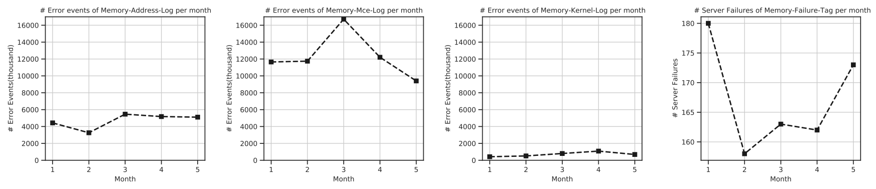


Fig. 3. The error events and the server failures per month.

4 Experiment Study

4.1 Data Analysis

Fig. 3 shows the total number of error events logged in three log files per month and the total number of server failures per month in the Memory-Failure-Tag.

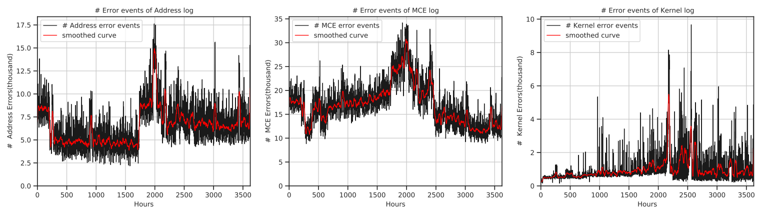


Fig. 4. The error events in three log files per hour.

A large number of error events are recorded in the log files. The peak value can be found in Memory-Mce-Log, where over sixteen million errors are stored in March. However, compared to the error log files, the number of failure records in the Memory-Failure-Tag is significantly reduced. Fig. 4 presents the number of error events per hour in the competition dataset. It is observed that the number of errors obviously varies in the error log files. The surge of error events indicates that the corresponding servers may fail in the future.

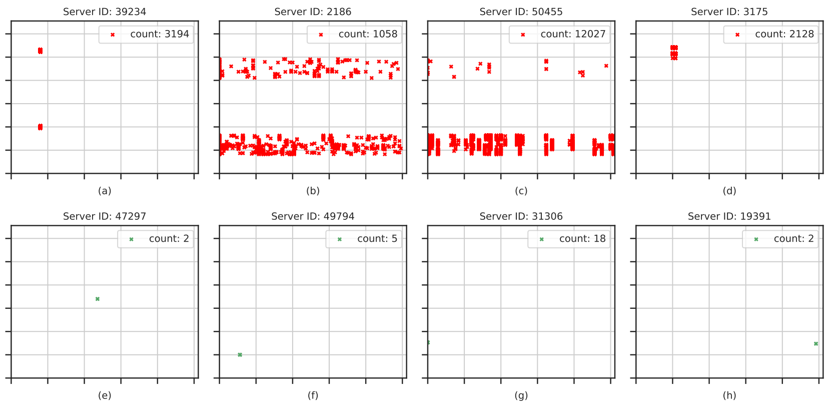


Fig. 5. The visualization of error patterns.

In Fig. 5, we randomly select 8 servers to invest the error patterns. Servers in the first row are reported to be faulty in the Memory-Failure-Tag, while servers in the second row still work properly. In Fig. 5(a) and Fig. 5(d), errors tend to cluster along nearby rows and columns. The error cases in Fig. 5(b) and Fig. 5(c) are comprised of multiple rows and columns. As for the healthy servers, errors are observed to map to a single cell. Clearly, the faulty and healthy servers have marked differences in terms of error patterns. Besides, if we treat the bank as a two-dimensional array with the size of $2^{17} \times 2^{10}$, errors only occur at a small number of cells, resulting in the issue of data sparseness.

4.2 Results and Discussion

In Fig. 6(a), the recall of the faulty servers versus the F1 score of the error samples is plotted. It can be observed that when the F1 score of the error samples increases, the recall of the faulty servers tends to be worse. It is obvious that there is a trade off, which is consistent with the analysis in 3.3. By using Equation (1), the weighting metric M can be calculated. The feature importance of the recall stage is reported in Fig. 6(b), where the definitions of the features can be found in Table 1. Similar to our understanding, the maxima of the unique faulty addresses in terms of banks, rows, and columns are crucial for the memory failure prediction problem. This is reasonable since a high error rate may come from a small number of failures. That is to say, calculating the unique address is better than just counting the error samples.

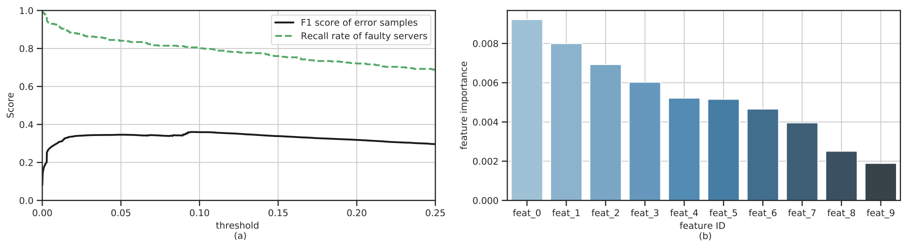


Fig. 6. The weighting metric and the feature importance at the recall stage.

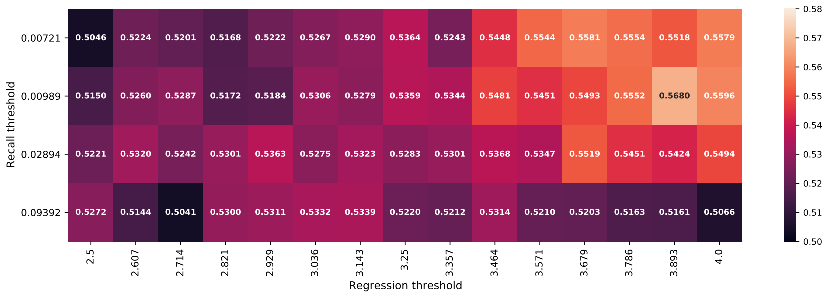


Fig. 7. Performance comparison with different thresholds.

The impacts of the thresholds introduced in the recall stage and the regression stage are presented in Fig. 7. Given a fixed value of the threshold of the recall stage, one can see that the score increases at first and then decreases. This can be understood by the fact that the increase of threshold of the regression stage helps raise the recall, while the precision is decreased. The experimental result shows that we can select the threshold of the recall stage within the range of 3.6-3.8. When we choose a relatively large value of the threshold of the recall stage, the proposed scheme suffers a performance reduction. The reason is that a larger number of error samples are filtered in this situation leading to a lot of information loss. The scheme without the recall stage versus the proposed scheme is provided in Fig. 8. The value of the threshold of the recall stage is set to 0.009893. It is obvious that the prediction accuracy is enhanced compared with the case with no recall stage.

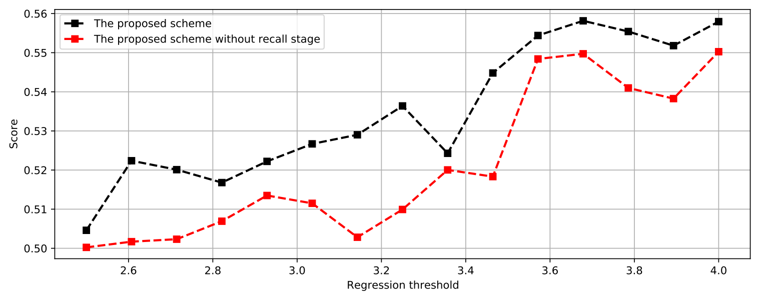


Fig. 8. Performance comparison with/without the recall stage.

5 Conclusions

This paper proposes a two-stage machine learning method to tackle the memory failure prediction problem. At the first stage, a XGBoost retrieval model is employed to mitigate the issue of imbalanced data. A weighting metric is designed to leverage the F1 score of the error samples and the recall rate of the faulty servers. In order to utilize more information, a regression model instead of a classification model is used at the second stage, where a threshold is carefully selected to improve prediction accuracy. Besides, a window-based strategy is introduced in the feature engineering process. In the PAKDD2021 2^{nd} Alibaba Cloud AIOps Competition, the proposed method is verified using a large-scale dataset, where the method ranks second with the score of 64.52 in the preliminary contests. The rank is 7 with the score of 28.63 in the semi-finals.

Acknowledgment

We would like to express our gratitude to Alibaba and PAKDD for hosting, creating and supporting this competition.

References

1. Bautista-Gomez, L., et al.: Unprotected Computing: A Large-Scale Study of DRAM Raw Error Rate on a Supercomputer. In: International Conference for High Performance Computing, Networking, Storage and Analysis.(2016)
2. Sridharan, V., Liberty, D.: A study of DRAM failures in the field. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis.(2013)
3. Du, X., et al.: Predicting Uncorrectable Memory Errors for Proactive Replacement: An Empirical Study on Large-Scale Field Data. In: 16th European Dependable Computing Conference.(2020)
4. Du, X., Li, C.: Memory failure prediction using online learning. In: Proceedings of the 4th International Symposium on Memory Systems.(2018)
5. Boixaderas, I., et al.: Cost-aware prediction of uncorrected DRAM errors in the field. In: The International Conference for High Performance Computing, Networking, Storage and Analysis.(2020)
6. Sridharan, V., et al.: Memory Errors in Modern Systems: The Good, The Bad, and The Ugly. ACM SIGPLAN Notices **50**(4),297–310(2015)
7. Hwang, A.A., Stefanovici, I., Schroeder, B.: Cosmic rays don't strike twice: understanding the nature of dram errors and the implications for system design. ACM SIGPLAN Notices **47**(4)(2012)
8. Baseman, E., et al.: Physics-Informed Machine Learning for DRAM Error Modeling. In: IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems.(2018)
9. Costa, C., et al.: A System Software Approach to Proactive Memory-Error Avoidance. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.(2015)
10. Baseman, E., Debardeleben, N., Ferreira, K., Levy, S., Raasch, S., Sridharan, V., et al.: Improving DRAM Fault Characterization through Machine Learning. In: IEEE/IFIP International Conference on Dependable Systems Networks Workshop.(2016)
11. Schroeder, B., et al.: Dram errors in the wild: a large-scale field study. SIGMETRICS Perform. Eval. Rev. **37**(1), 193–204(2009)
12. Giurgiu, I., Szabo, J., Wiesmann, D., Bird, J.: Predicting DRAM reliability in the field with machine learning. In: Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track.(2017)
13. Liu, Y., et al.: An Introduction to PAKDD CUP 2020 Dataset. In: He C., Feng M., Lee P., Wang P., Han S., Liu Y. (eds) Large-Scale Disk Failure Prediction. AI Ops 2020. Communications in Computer and Information Science, vol 1261. Springer, Singapore.(2020)
14. He, H., Garcia, E.: Learning from Imbalanced Data. IEEE Transactions on Knowledge and Data Engineering **21**(9),1263–1284(2008)
15. Xu, Y., et al.: Improving service availability of cloud systems by predicting disk error. In: Proceedings of USENIX ATC.(2018)
16. https://tianchi.aliyun.com/competition/entrance/531874/introduction
17. Zhang J., Sun Z., Lu J.: First Place Solution of PAKDD Cup 2020. In: He C., Feng M., Lee P., Wang P., Han S., Liu Y. (eds) Large-Scale Disk Failure Prediction. AI Ops 2020. Communications in Computer and Information Science, vol 1261. Springer, Singapore.(2020)