Cost-Aware Prediction of Uncorrected DRAM Errors in the Field

Isaac Boixaderas
Barcelona Supercomputing Center
isaac.boixaderas@bsc.es

Darko Zivanovic
Barcelona Supercomputing Center
darko.zivanovic@bsc.es

Sergi Moré
Barcelona Supercomputing Center
sergi.more@bsc.es

Javier Bartolome
Barcelona Supercomputing Center
javier.bartolome@bsc.es

David Vicente
Barcelona Supercomputing Center
david.vicente@bsc.es

Marc Casas
Barcelona Supercomputing Center
marc.casas@bsc.es

Paul M. Carpenter
Barcelona Supercomputing Center
paul.carpenter@bsc.es

Petar Radojković
Barcelona Supercomputing Center
petar.radojkovic@bsc.es

Eduard Ayguadé
Barcelona Supercomputing Center,
Universitat Politècnica de Catalunya
eduard.ayguade@bsc.es

***Abstract*—This paper presents and evaluates a method to predict DRAM uncorrected errors, a leading cause of hardware failures in large-scale HPC clusters. The method uses a random forest classifier, which was trained and evaluated using error logs from two years of production of the MareNostrum 3 supercomputer. By enabling the system to take measures to mitigate node failures, our method reduces lost compute time by up to 57%, a net saving of 21,000 node–hours per year. We release all source code as open source.**

We also discuss and clarify aspects of methodology that are essential for a DRAM prediction method to be useful in practice. We explain why standard evaluation metrics, such as precision and recall, are insufficient, and base the evaluation on a cost–benefit analysis. This methodology can help ensure that any DRAM error predictor is clear from training bias and has a clear cost–benefit calculation.

***Index Terms*—Memory system, Reliability, Error prediction, Machine learning, Random forest, Cost–benefit analysis.**

I. Introduction

One of the main causes of hardware failure in large-scale clusters is an uncorrected error in main memory [1]–[4]. Node failures are especially problematic in high-performance computing (HPC) systems, where a single tightly-coupled job may execute for days on thousands of nodes. If any of these nodes fails, the whole job is terminated, typically wasting all CPU hours since the last checkpoint. Memory system reliability is therefore an important limit on the ability to scale to larger systems.

Various studies propose use of machine learning methods to predict DRAM errors. These studies are valuable to understand correlated factors and features that can be used for DRAM error prediction. It is not trivial, however, to quantify the impact of the proposed methods on HPC system reliability. There are two main reasons for this. First, most prior studies focus on *corrected* DRAM errors. System reliability, however, is impacted only by *uncorrected* errors [5]–[9], and there is no direct relation between corrected and uncorrected errors [2], [3], [8], [10], [11]. Second, previous studies evaluate the proposed methods using classical prediction metrics, such as precision, recall and F1-score. Although these classical metrics are often suitable, various studies [12]–[15] and our results show that they are insufficient for evaluation of HPC failure predictors because they cannot be used to determine whether prediction is useful in practice.

The main objective of our study is to help the community set a basis for any following work on uncorrected DRAM error prediction that would be practical in the field. This paper makes two main contributions. Firstly, we present and evaluate a method to predict DRAM uncorrected errors (UEs). This method can enable the system to take active measures to mitigate the predicted UE, e.g. perform a checkpoint or live job migration. Secondly, we discuss and clarify several aspects of methodology, relating to cost–benefit analysis and potential sources of bias, that are essential for such a prediction method to be useful in practice. We compare six machine learning classifiers and design an error prediction method based on random forest. The presented method predicts UEs based on preceding warnings and errors (corrected and uncorrected), as well as DIMM characteristics, and node–level events such as reboots and DIMM installations. The method is trained and evaluated using error logs from the MareNostrum 3 supercomputer [16], one of six Tier-0 HPC systems in Europe. At the time of the study, the system comprised 3056 nodes with more than 25,000 memory DIMMs. The error logs cover a production period of more than two years, from October 2014 to November 2016, during which we detected 4.5 million corrected errors and 333 uncorrected errors.

The objective of the HPC failure prediction mechanism is to increase effective use of the HPC system by reducing the compute time lost due to failures. We pay special attention to the evaluation methodology and base the evaluation on a cost–benefit analysis that compares the system resources needed for training, failure prediction and failure mitigation against the

DOI 10.1109/SC41405.2020.00065

saved compute time due to successful failure prediction and mitigation [9]. Our results show that the precision, recall and F1-score are not correlated with saved compute time or mitigation costs, and therefore cannot be used to decide whether and for which model parameters the prediction is useful in practice. This is because precision and F1-score put the same weight on diverse prediction outcomes, whose costs differ by orders of magnitude, and the recall metric ignores false positives, which incur unnecessary and costly mitigation measures. Overall, these standard data prediction metrics are insufficient to evaluate HPC failure predictors, and we suggest to complement them with cost–benefit analysis in future studies.

We are the first to show that only a small fraction of uncorrected errors have an impact on system reliability. Uncorrected DRAM errors, as many failure events, appear in bursts. In a burst of UEs occurring in a short interval of time on the same node, only the first UE has an impact on system reliability. In our study, out of 333 detected UEs, only 67 (one fifth!) have any impact and are therefore used for model training and evaluation. The rest are much easier to predict (at least in our dataset) but have no impact on the cost–benefit calculation.

We also address the bias that may have been introduced by existing resiliency techniques employed in a production supercomputer. State-of-the-art production systems, including MareNostrum, already implement advanced resilience mechanisms in hardware and software. During the monitoring period, MareNostrum incorporated a pre-failure alert mechanism that caused any DIMMs that were considered to be close to failure to be retired from production. These retired DIMMs can cause a bias in the model training. This is the first DRAM error prediction study that discusses this bias and proposes a methodology for its mitigation. All future failure prediction methods trained on production logs should also carefully consider and mitigate this bias.

Finally, we analyze the effect of the model decision threshold, prediction frequency and prediction window on prediction coverage and proficiency. We quantify the impact of these parameters on the cost–benefit calculation, and explain the behavior using the number of errors that the model can predict, the number of correctly predicted errors and model overheads.

In summary, this paper presents and evaluates a method that predicts DRAM uncorrected errors, reducing the compute time lost due to DRAM failures by up to 57%. In our production systems, the savings would be measured as 21,000 node–hours per year. We release the prediction methods' source code as open source [17]. We also aim to help the community to define standard methodology for log pre-processing, model training, parameter exploration and evaluation. This methodology will help ensure that any DRAM error prediction method is clear from bias during training and has a clear cost–benefit calculation.

II. Environment description

A. MareNostrum 3 error logs

Our prediction method is trained and evaluated on memory error logs from the MareNostrum 3 supercomputer [16] over a production period from October 2014 to November 2016. At the time, MareNostrum 3 was one of the six Tier-0 (largest) HPC systems in the Partnership for Advanced Computing in Europe (PRACE) [18]. It comprised 3056 compute nodes, each with two eight-core Intel Sandy Bridge-EP E5-2670 sockets with a 2.6 GHz nominal clock frequency. We use the logs from the compute nodes only, excluding the login and test nodes which are not part of the same monitoring infrastructure and whose failures do not impact large-scale compute jobs. The MareNostrum 3 compute nodes included more than 25,000 DDR3-1600 DIMMs, and during the observation period we collected measurements on more than 2,000 billion MB-hours. The main workloads executed on MareNostrum 3 were large-scale scientific HPC applications and the system utilization typically exceeded 95%. We analyze DIMMs from all three major memory manufacturers. These manufacturers have been anonymized to protect the interested parties, and are referred to as *Manufacturer A*, *B* and *C*. There are 6694, 5207 and 13,419 DIMMs from *Manufacturer A*, *B* and *C*, respectively.

MareNostrum 3 employed a Single Device Data Correction (SDDC) ECC scheme, which could correct all errors coming from a single x4 device, a level of resiliency commonly referred to as Chipkill. For x8 devices, SDDC ECC can correct up to 4-bit errors coming from the same DRAM chip. The ECC check is performed on each application memory read and by a patrol scrubber which periodically traverses the whole physical memory and performs an ECC check on each location.

During production, any DIMM that showed early signs of failure was flagged by a pre-failure alert and retired by the system administrators. This action was recorded in the system log together with the date and time. Over the two-year period analysed by this paper, 51 DIMMs were retired for this reason.

B. Data collection

Corrected errors (CEs) were logged by a daemon, based on the mcelog Linux kernel module [19], that periodically extracts information about corrected errors from the CPU machine-check architecture (MCA) registers [19]. Each corrected error was recorded in a log file, which specifies the error time stamp, node and DIMM id, and the physical location of the error in the DIMM including rank, bank, row and column.[1] The log entry also indicates whether correction was done during an application memory read or by patrol scrubbing.

The daemon accessed the MCA registers with a period of 100 ms, which we selected because it was the shortest time interval that caused negligible overhead to the production applications. If more than one error occurred in a 100 ms time interval, the MCA registers records the number of errors, but only provide detailed information for one error in the interval. Our logs therefore specify the exact total number of corrected errors and provide detailed error information for a subset of the errors. Increasing the sampling frequency would

[1]The address mapping to exact physical location is non-diclosed manufacturer information, and was done using help from a memory manufacturer.

increase the number of errors with detailed information, but also increase the performance overhead of the error logging daemon. Previous studies perform similar readings of the memory error registers with a period of a few seconds [10], [20], [21] or once per hour [22].

Uncorrected errors (UEs) were logged by IBM firmware [23], which is part of the MareNostrum 3 monitoring software. For each uncorrected error, the log specifies the DIMM that failed and the cause of the error, i.e. whether it occurred during an application memory read or patrol scrubbing. The log also contains over-temperature conditions, which are considered to be uncorrected errors. Finally, the log records **UE warnings** generated when memory modules are throttled to prevent an over-temperature condition and when the correctable ECC logging limit has been reached.

Our prediction method is trained and evaluated using the production MareNostrum 3 error logs. These production error logs are considered sensitive, and as is the case for other field studies, the production error logs cannot be released. However, in order to enable future studies to quantify the real-world impact of DRAM uncorrected errors and evaluate proposed resiliency techniques, we have generated and released a synthetic UE log that resembles the spatial and temporal distributions of the MareNostrum 3 production errors [17].

C. UE reduction

Various studies have observed burstiness in HPC node failure events [5], [7], including uncorrected DRAM errors [8]. This phenomenon affects both training and evaluation of UE prediction methods. In a burst of UEs that occurred in a short time period on the same node, only the first UE has an impact on system reliability. The first UE leads to a node reboot and the killing of all affected jobs. Subsequent UEs have no real impact, but they are still reported by the system software. In MareNostrum, after detecting one UE, the node is removed from production for testing for one week. This means that all UEs on the same node within one week after the first UE occurrence have no impact on a production workload. Performing data reduction, i.e. focusing only on UEs that impact production workloads, significantly reduces the number of UEs under study, from 333 UEs to 67 UEs. This makes a major impact on the model design and evaluation.

D. MareNostrum 4 job logs

The cost–benefit evaluation is done in two ways: firstly, based on assumptions on the average UE cost, which allows an estimation of the behaviour on any system, and, secondly, based on a real distribution of HPC job sizes. We unfortunately do not have a log of the jobs running on MareNostrum 3, so instead we employ a log of the jobs executed on the general-purpose block of the successor system, MareNostrum 4 [24]. This log covers the period from March 2018 to March 2019. MareNostrum 4 has 3456 nodes, each with two 24-core Intel Xeon Platinum sockets with a 2.1 GHz clock frequency.

The MareNostrum 4 job size distribution has been taken from a different and higher performance machine and a different production time period than was used for the MareNostrum 3 error logs. Nevertheless, we believe that this does not significantly change any of the conclusions.

III. Error prediction

A. Prediction methods

We perform uncorrected DRAM error prediction with six classification methods: random forest [25], gradient boosting decision tree [26], Gaussian naïve Bayes [27], logistic regression [28], support vector machine [29] and deep neural network [30]. All the classifiers are implemented based on the corresponding *Python* libraries. For random forest, Gaussian naïve Bayes, logistic regression and support vector machine we use the *sklearn* library [31]. Gradient boosting decision tree is implemented based on *LightGBM* [32], and the deep neural network is deployed using TensorFlow [33].

The classifiers make separate predictions for each DIMM as to whether or not it will experience an upcoming uncorrected error. As described in the following section, prediction is done periodically, e.g. every minute, and each prediction covers a fixed length of time, e.g. the next day. Each classifier is trained for all nodes. We use offline learning, where the model is trained once and used to make predictions. It is common practice to schedule a time interval for retraining, after which new data is added to the existing data and the model is retrained. We tried two training intervals, every day and every week. Both provided similar results. We trained and tested each classifier applying the methodology described in Section IV-A, with hyperparameter tuning and cross-validation.

We present detailed results for the random forest because it provides the best results over a large range of uncorrected DRAM error costs (see Section VI). Also, random forests are relatively easy to tune and less likely to overfit than other methods, they can quantify the importance of different features, and training can be easily parallelized. Our results confirm the findings of previous studies that have shown that random forest works well for problems that involve predicting failures over time, such as predicting corrected and uncorrected DRAM errors [2], [34] and disk failures [35].

B. Time windows

Figure 1 illustrates how prediction is performed periodically. In this figure, the **prediction window** refers to the length of the (future) time interval for which a prediction is made. The **observation window** refers to the length of the (past) period that is observed in order to make the prediction. In our study the observation window extends from the beginning of error logging until the prediction moment, which is at the beginning of the prediction window. The observation window therefore increases in length over time, leading to better predictions. As in most prior work, the **prediction frequency** is a constant, with the time between successive predictions known as the **prediction interval** and illustrated using Δt in Figure 1. The prediction interval must be no greater than the prediction window, and is a trade-off between overhead, precision in time and coverage of UEs. A full discussion is given in Section V-B.

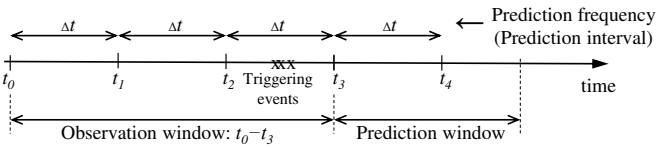


Fig. 1: Periodic prediction process over time, showing prediction frequency, observation window and prediction window.

We propose a triggered prediction that provides high responsiveness with low prediction overhead. In Figure 1, predictions are scheduled at regular time intervals, t_1, t_2, t_3 and t_4, but a prediction will only be made if the preceding prediction interval contains one or more trigger events: *Corrected Errors, Uncorrected Errors, Uncorrected Error Warnings, Node boots* or *DIMM installs*. In the figure, a prediction will only be made at t_3, triggered by the events detected in the t_2–t_3 interval. At t_1, t_2 and t_4 the system performs only a low-overhead check to determine whether any trigger events have been detected.

C. DIMM retirement bias

MareNostrum incorporates a pre-failure alert that retires from production any DIMMs that are considered to be close to failure. Preventive DIMM retirement such as this is known to cause bias when analyzing the dependency between corrected and uncorrected errors [8]. Whether or not it also introduces bias in the training or evaluation of an UE prediction method depends on the characteristics of the method. In our case, DIMM replacement after the end of the prediction window has no impact on the model training and evaluation. For most DIMMs, i.e. those that were not replaced, prediction can therefore be evaluated until the end of the log files.

If the DIMM is replaced inside the prediction window, we could not determine whether it would have experienced an UE had it not been retired, because we could not obtain monitoring data after DIMMs were replaced. We therefore remove these samples from training and evaluation, which introduces a bias. In order to remove this bias and to enable comparison between the existing pre-failure mechanism and the new predictor, we recommend that device health is monitored for some time, on back-up or service nodes, after retirement from production.

D. Feature engineering

The **features** (input data structures) used by our prediction method are listed in Table I. Most features are obtained directly from the logs (see Section II-B). In addition, we include features that describe the DIMM characteristics, such as manufacturer, chip and DIMM capacity. Categorical attributes are transformed into features using one-hot encoding, which is known to improve the performance of machine learning classification and regression methods. We compute the feature values at the moment of the prediction, and also account for the **feature variation over time**. For example, feature variation in the last minute before the prediction moment is computed as:

$$\textit{Feat. variation (1min)} = \frac{\textit{Feat. value (Prediction moment)}}{\textit{Feat. value (Prediction moment – 1 min)}}$$

TABLE I: Observation features used for the prediction

Feature for prediction method
Per DIMM:
Number of corrected errors (CEs)
Number of ranks, banks, rows and columns with CEs
Average and standard deviation of errors per rank, bank, row and col.[a]
Number of uncorrected errors (UEs)
Number of warnings
Time since the DIMM was installed in its current position
Number of times the DIMM has changed its position in production
Manufacturer[b]
DIMM Capacity: 4, 8 or 16 GB[b]
Chip Capacity: 2 Gbit or 4 Gbit[b]
Data Width: x4 or x8[b]
Per socket:
Number of DIMMs with corrected errors
Sum of corrected errors in all the DIMMs
Number of DIMMs with uncorrected errors
Sum of uncorrected errors in all the DIMMs
Number of DIMMs with warnings
Sum of warnings in all the DIMMs
Per node:
Sum of each per-socket feature across sockets in the node
Number of node boots (starts) in the last minute, hour and day

[a] Considering only the ranks, banks, rows and columns with errors.
[b] Transformed into features using one-hot encoding.

It is set to zero if the denominator is zero. We consider lagged time intervals of 1 minute, 1 hour and 1 day. The feature variation over time is computed for the features listed in Table I, except the DIMM characteristics, which do not change in time, and the numbers of node boots in the last minute, hour and day, which already include the notion of time.

E. Prediction classes: Positive and negative

The outcome of our prediction model is either:

- **Positive**: The model predicts that the DIMM will experience at least one UE in the prediction window, or
- **Negative**: The model predicts that the DIMM will experience no UE in the prediction window.

The classifiers estimate the probability that the sample belongs to each class and classifies the sample as positive if the probability of belonging to the positive class is greater than a threshold known as the **decision threshold**, otherwise it is classified as negative. The lower the decision threshold, the more often the outcome is a positive prediction.

F. Class imbalance

For the vast majority of prediction windows, the correct prediction is that there will be no UE. This case is about 300 times more frequent than the alternative, that of an UE in the prediction window. We are therefore dealing with a problem of highly imbalanced data. The classifiers aim to minimize the **overall** error rate so they may not perform well for the minority class of highly imbalanced data. Two common approaches to address this problem are: data sampling during training and the use of class weights.

Sampling techniques adjust the ratio of the sizes of the majority and minority classes, in our case the negative and positive classes respectively. We tried multiple sampling methods: random under-sampling [36], random over-sampling [36], SMOTE [37] and balanced random forest [38], all of them using the implementation in the *Python imblearn* library [39]. Sampling techniques take a parameter that specifies the desired ratio of the sizes of the minority and majority classes. We set this parameter to 1 so that training is done with a random subset of the majority class of size chosen to obtain equal numbers of samples with and without UEs.

Class weights address class imbalance by weighting the misprediction penalty for different classes during training. In our case the misprediction penalty is increased for samples in the minority (positive) class and decreased for samples in the majority (negative) class. We implement class weights with the *balanced* option available in the *sklearn* library [31].

We tested all classifiers with class weights and different training data sampling methods, and for each classifier we selected the method that provided the best results. Random under-sampling gave the best performance for random forest, Gaussian naïve Bayes, support vector machine and neural networks. Class weights was best for logistic regression and gradient boosting decision tree.

IV. Evaluation methodology

A. Method evaluation and cross-validation

1) Time Series Cross-validation: Our evaluation methodology is based on time series cross-validation, which is a well-known technique to determine how well a predictive model of time series data can predict new data that was not available while training. The aim is to identify possible problems (such as overfitting) and to quantify how well the model will generalize to an independent dataset.

We perform a sequence of training and testing steps. For each week of production time in the log, we train the model, which includes hyperparameter tuning, on all data in the dataset that precedes that week (training set). We then evaluate the tuned model by using it to predict the upcoming week (testing set). This process continues, week by week, until no more data remains. The final results given in Section V cover all iterations (weeks).

2) Evaluation on different HPC systems: Ideally, we would demonstrate the general applicability of our method using error logs from multiple HPC systems. We have carefully reviewed previous DRAM errors studies (see Section VII) and public repositories [40] and found no DRAM error logs that could be used for additional verification of our model. This is not a surprise because HPC system failure data is considered to be sensitive. To increase the confidence in the generality of our method, we have partitioned the MareNostrum 3 error logs by DRAM manufacturer. MareNostrum 3 comprises 6694, 5207 and 13,419 DIMMs from anonymized *Manufacturer A*, *B* and *C* respectively. Apart from a small number of exceptions, all DIMMs in a given node are from the same DRAM manufacturer.

The evaluation was performed with two sets of experiments. Firstly, we trained and evaluated the method on the whole system (*MN3/All*). Secondly, we performed separate training and testing for each subsystem comprising a single DIMM manufacturer: *MN3/A*, *MN3/B* and *MN3/C*. All experiments correspond to the case where the users download the open source model [17] and train it on their systems. We believe that this is the only reasonable approach given the variation in system logs (i.e. model features), UE rates and UE cost among production HPC systems.

B. Precision, recall and F1-score

Each prediction outcome can be classified as one of:

- **True Positive (TP)**: UE predicted and UE occurs.
- **True Negative (TN)**: UE not predicted and none occurs.
- **False Positive (FP)**: UE predicted, but none occurs.
- **False Negative (FN)**: UE not predicted, but UE occurs.

Previous studies that proposed error prediction methods [2], [41]–[43] are evaluated using standard data prediction metrics: precision, recall and F1-score.

Precision refers to the percentage of observations classified as positives that are true positives. In our case, the precision refers to the ratio between correctly predicted UEs (TPs) and the total number of predicted UEs (TPs and FPs):

$$Precision = \frac{Correctly\ predicted\ UEs}{Total\ predicted\ UEs} = \frac{TPs}{TPs\ +\ FPs}$$

Recall is the proportion of actual positives that are correctly identified as such. In our case, this metric refers to the fraction of UEs that are correctly predicted (TPs):

$$Recall = \frac{Correctly\ predicted\ UEs}{Total\ UEs\ occurred} = \frac{TPs}{TPs\ +\ FNs}$$

False Negative Rate (FNR) is the proportion of actual positives that are not identified as such. It is complementary to the recall: $FNR = 1 - Recall$.

F1-score is the harmonic mean of precision and recall:

$$F_1 = 2 \times (Precision \times Recall)\ /\ (Precision + Recall)$$

Although these classical metrics are often suitable, various studies [12]–[15] show that they are insufficient for evaluation of HPC failure predictors. This is because, as shown in Section V-D, they are not correlated with a cost–benefit analysis, and therefore cannot be used to decide whether and for which model parameters the prediction is useful in practice.

C. Cost–benefit calculation

The objective of the HPC failure prediction mechanism is to increase the effective use of the HPC system by reducing the compute time lost due to failures [12], [13]. We therefore perform a cost–benefit analysis that compares the system resources needed for training, failure prediction and failure mitigation against the saved compute time due to successful failure prediction and mitigation.

1) Lost compute time without UE prediction and mitigation: If an UE is detected, the job must typically be terminated and all job node–hours since the last checkpoint (if any) are lost:[2]

$$\textit{Total UE cost} = \textit{Number of UEs} \times \textit{Average UE cost}$$

where *Average UE cost* is the average cost of a single UE, measured in node–hours. To cover HPC systems of different scales we perform a sensitivity analysis and consider average UE costs of 5, 50 and 500 node–hours. We refer to these UE costs as *small*, *medium* and *large*, respectively. We also show results for the real distribution of HPC jobs executed on the general purpose block of MareNostrum 4 (see Section II-D). The average UE cost was derived using a Monte Carlo simulation of a system executing jobs according to the real distribution of MareNostrum 4 production, in which the UE cost is the elapsed compute time, in node–hours, since the beginning of the currently executing job.

After an uncorrected error is detected, the affected node is removed from production for further testing (see Section II-C). We do not include the testing node–hours as part of the overall UE cost because we assume that the system has sufficient spare nodes that could easily replace the one removed from the production. In HPC systems for which this is not the case, this node testing time should be added to the UE cost.

2) Net lost compute time with UE prediction and mitigation: The total cost is the cost of the UEs that were not predicted plus the total overheads of prediction model training, UE prediction and mitigation measures.[3]

The **total cost of the UEs that were not predicted** is given by *Non-predicted UEs* $\times$ *Average UE cost*. As discussed in Section II-C, only the first UE in a burst has an impact on system reliability. For this reason, we only consider the cost of the first non-predicted UE in a burst of UEs that occur on the same node within a week. It is also important to consider that it is only useful to predict an error if this is done sufficiently ahead of time, so that the system has time to complete the error mitigation measure. This time interval is typically referred to as the **lead time**. The lead time depends on the system, application and mitigation strategy.

A recent study of Das et al. [44] analyzes various actions that can mitigate the impact of node failures, such as live job migration, node cloning and checkpointing. The authors conclude that 2 min suffice for most of these actions. We therefore select a lead time of 2 min and treat an UE that is predicted with less than 2 min notice as a non-predicted UE.

The **cost of model training** depends on the amount of data used for the training, which increases with the prediction frequency. The model training is executed on a single CPU core running a single process. With a prediction interval of 10 seconds, the model training requires 6.8 seconds and 3.1 GB; with the interval of 1 day, the training requires 0.3 seconds and 2.5 GB of main memory. We also measure the cumulative cost corresponding to model training while processing the whole data log, i.e. corresponding to more than two years of the system production. The cumulative cost for training is in the order of node–minutes, which is a negligible factor in the overall cost–benefit analysis. This paper reports the results when the model training is performed once per week. We repeated the experiments with a model training frequency of 1 day and we detected an insignificant difference in the model predictions and the cost–benefit analysis.

The **cost of UE prediction** is incurred each time a new event is detected in the log, as the algorithm processes the logs and performs UE prediction. Each prediction is performed on a single CPU core (running a single process) and requires 6 ms and 4 MB. We measure the cumulative cost for UE prediction to be on the order of node–minutes, which is also negligible.

The **cost of mitigating UEs** is incurred when an UE is predicted, as the system performs an UE impact mitigation. The UE impact mitigation overhead should be counted each time the model predicts an UE (each time the model predicts a positive), independently of whether the UE indeed occurred or not (whether the prediction is a true or false positive). Following the suggestions of Das et al. [44], we consider that the overhead of a single UE impact mitigation is 2 minutes.

The net lost compute time is the sum of the above four terms: total cost of the non-predicted UEs plus the total cost of model training, UE prediction and UE mitigation. For the HPC jobs that execute on healthy nodes the model introduces only a negligible overhead due to the training and UE prediction.

V. Results

A. Cost–benefit analysis

Figure 2 summarizes the results of the cost–benefit analysis for the prediction model and mitigation. The assumptions used in this figure, and throughout Sections V and VI are given in Table II.

The x-axis of Figure 2 is the UE cost, covering generic UE costs of 5, 50 and 500 node–hours and the average UE cost calculated using the job size distribution from production MareNostrum 4 HPC job logs (see Section IV-C1). The y-axis in Figure 2a is the saved node–hours, which is the reduction in lost compute time due to UE prediction and mitigation (see Section IV-C2, model overheads are counted as additional lost compute time), compared with the baseline system without UE prediction and mitigation (Section IV-C1). The y-axis in Figure 2b is the saved node–hours normalized to the number of node–hours lost in the baseline system. Different bars show the results for MareNostrum 3 as a whole (MN3/All) and its different subsystems corresponding to the DRAM manufacturer: MN3/A, MN3/B and MN3/C (see Section IV-A2). Bars MN3/ABC show the overall results in the case of the system partitioning: the sum of the node–hours saved for MN3/A, MN3/B and MN3/C in Figure 2a and their weighted average percentage savings in Figure 2b.

These results show that the effectiveness of the method is similar across all scenarios considered: whether applied

[2]In HPC, nodes are usually not shared among multiple jobs. It is safe to assume that reboot of an HPC node usually kills a single HPC job.

[3]In addition to this, on each prediction interval we have to check whether there is a new event in the log (in the preceding observation window). In this study, however, we estimate that this check causes negligible overhead.

TABLE II: Parameters used in Sections V and VI

Item	Value
Cost of UE impact mitigation	2 minutes
Prediction frequency	1 minute
Prediction window	1 day
Decision threshold	Optimal for given UE cost

and evaluated to MareNostrum 3 as a whole or separately to MN3/A, MN3/B and MN3/C. The MN/ABC results also closely match those for MN3/All. In terms of the percentage savings in node–hours, in Figure 2b, all results are similar to those for the whole system, MN3/All.

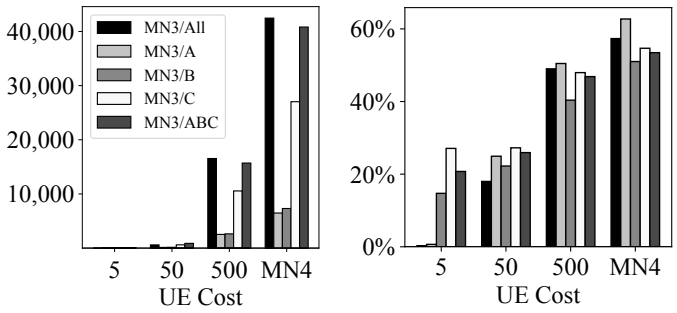
The results also clearly show that the cost–benefit calculation is strongly influenced by the average UE cost. For a small UE cost of 5 node–hours, UE prediction has zero effect on the saved compute time. We explore the reasons in the following sections and see that in this case, the optimal decision threshold is 1, so proactive UE mitigation is never performed. For medium and large UE costs, however, savings are seen in Figure 2a, of 586 node–hours (medium) and 16,541 node–hours (large). These are reductions of 18% and 49% respectively (Figure 2b).

The node–hours savings computed based on the production job logs reach 57% which is equivalent to 42,000 node–hours or 21,000 node–hours per year. These savings are constrained by the low number of uncorrected DRAM errors of MareNostrum 3, and may appear low relative to the overall system size. Other studies, however, of field errors of DDR1, DDR2, FBDIMM [3], DDR3 and GDDR5 [6] memory systems report UE rates over a large range that in some cases [3] exceed ours by up to three orders of magnitude. In these systems, the UE costs and saved node–hours would increase in proportion, reaching tens of percents of the total production time.

Another important finding of our study is that the saved node–hours increase superlinearly with the average UE cost. This is for two reasons. Firstly, the larger the UE cost, the larger the savings from correct UE prediction, while the overheads of the model and mitigation actions remain the same. Secondly, larger UE costs allow the model to be more aggressive, leading to a greater number of predicted errors that more than compensates for the increased number of mitigation actions (as discussed in Section V-C). We reach this conclusion for all six classifiers explored in our study (Section VI). This is because the finding is based on the cost–benefit analysis in terms of the costs of the true and false positives. It is not specific to any method and can therefore be applied to error prediction methods in general.

B. Prediction window and frequency

The prediction window and frequency (Section III-B) provide a trade-off among the prediction model overhead, its precision in time and the maximum coverage of UEs. The overhead is mainly influenced by the prediction frequency. The higher the prediction frequency, the higher the frequency



(a) Number of saved node–hours (b) Percentage saved node–hours

Fig. 2: The model cost-efficiency depends on the UE cost. For large UE cost, the savings are significant, measured in thousands of node–hours over the two-year production period.

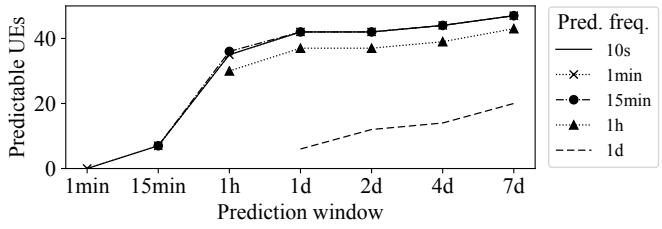
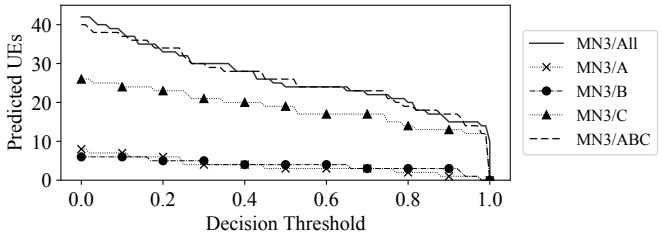


Fig. 3: Maximum number of UE that the model could potentially predict depends on the prediction window and frequency.

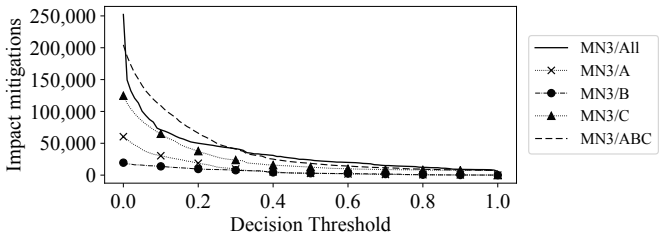
of running the model and the higher the overhead of false positives. The precision in time also depends on the length of the prediction window, since a shorter prediction window isolates any predicted UE to a shorter time window. The result is that, on a true positive, i.e. if an UE indeed happens, less time is wasted, on average, between the mitigation measure and the UE error.

The relationship between the prediction window and frequency and the maximum potential coverage of UEs is shown in Figure 3. Of the 67 UEs that impact system reliability (Section II-C), 13 UEs have no preceding event at all. These UEs cannot be predicted by our method since there is no information available to the model that could indicate that an UE is likely. Assuming that there are preceding events, however, if the prediction window is too short, then the preceding events may be too far in the past. This happens when the time since the preceding events is larger than the prediction window, and it is a consequence of event triggered prediction (Section III-B). On the other hand, if the prediction interval is too long, then the preceding events may be followed too quickly by an UE, without time to invoke prediction and perform successful mitigation.

In general, as can be seen in Figure 3, the longer the prediction window and the higher the prediction frequency, the higher the number of UEs that are visible to the prediction model. There is, however, a non-linear increase in the number of predictable UEs, and a point of diminishing returns. Increasing the prediction window from 1 min to 1 hour and



(a) The lower the threshold, the more correctly predicted UEs.



(b) The lower the threshold, the more UE impact mitigations.

Fig. 4: Optimal threshold selection is a trade-off between the number of predicted UEs and UE impact mitigations.

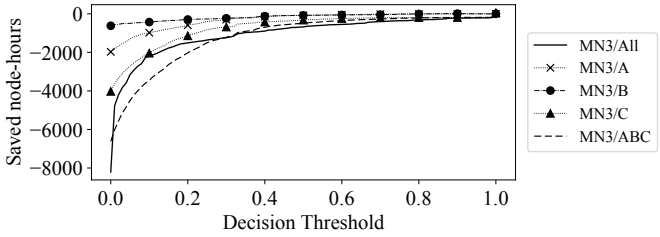
1 day significantly increases the number of visible UEs, while further increasing it to 2, 4 or 7 days leads to a minor increase.

Taking into account all three factors (overhead, precision and coverage), we select a prediction window of 1 day and a prediction interval of 1 min. Due to the event triggered prediction (Section III-B) and low-overhead prediction method (Section IV-C), even at this high prediction frequency the overhead of our method predictions is in the order of node–minutes.

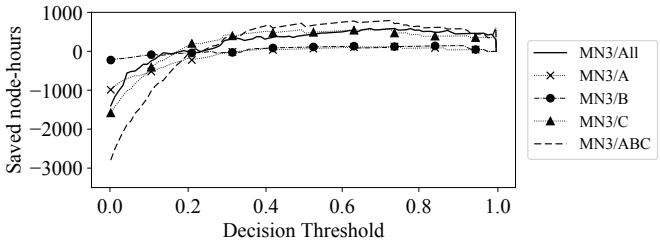
C. Decision threshold

As explained in Section III-E, an UE is predicted if the probability of an UE assigned by the random forest is greater than the decision threshold. Varying the decision threshold, which can be done at runtime or when the model is deployed, provides a trade-off between true positives (which benefit in lost node–hours) and the total number of positives (which require mitigation measures).

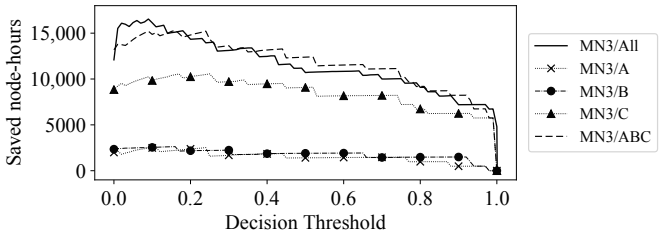
This trade-off is shown in Figure 4. Again we show the results for MareNostrum 3 as a whole (MN3/All), its different partitions (MN3/A, MN3/B and MN3/C) and the cumulative results in the case of the system partitioning (MN3/ABC). In both subplots, the x-axis is the decision threshold. In Figure 4a, the y-axis is the number of correctly predicted UEs, which are the true positives. The lower the decision threshold, the more positive outcomes and the more correctly predicted UEs. In Figure 4b, the y-axis is the number of UE impact mitigations, which are all the positive outcomes. For small values of the decision threshold, there are up to 250,000 UE impact mitigations, which is almost 4000 UE impact mitigations per UE error. The MN3/ABC curves show similar values to MN3/All. The individual partitions MN3/A, MN3/B and MN3/C show similar trends with values approximately proportional to the size of the partition.



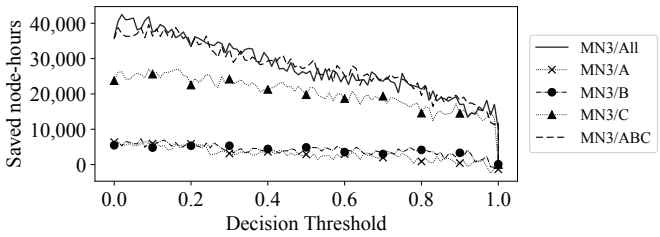
(a) UE cost = 5 node–hours (small)



(b) UE cost = 50 node–hours (medium)



(c) UE cost = 500 node–hours (large)



(d) MN4 Monte Carlo simulation

Fig. 5: Number of saved node–hours for different threshold values. The threshold has a significant impact on the model efficiency and the optimal value depends on the UE cost.

Figure 5 shows how this trade-off is reflected in the cost–benefit analysis. In all plots, the y-axis is the saved node–hours and the x-axis is the decision threshold. We focus our explanation on the MN3/All results. As in previous figures, the MN3/ABC curves are similar to MN3/All, while MN3/A, MN3/B and MN3/C show the same trend with values approximately proportional to the partition size. Figure 5a shows the results for an UE cost of 5 node–hours. Although the overhead of a single UE impact mitigation (2 minutes per node) is much lower than the cost of an uncorrected error (5 node–hours), the relatively large number of false positives mean that the optimal decision threshold is 1. With a decision threshold of 1, the model never suggests an error mitigation and the only loss is the model overhead of a

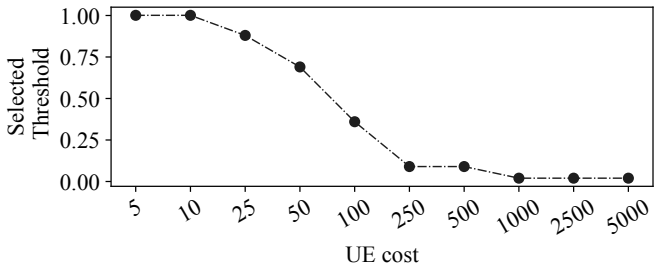


Fig. 6: Optimal threshold depending on UE cost: equal to 1 for small UE costs and converges to zero for large UE costs.

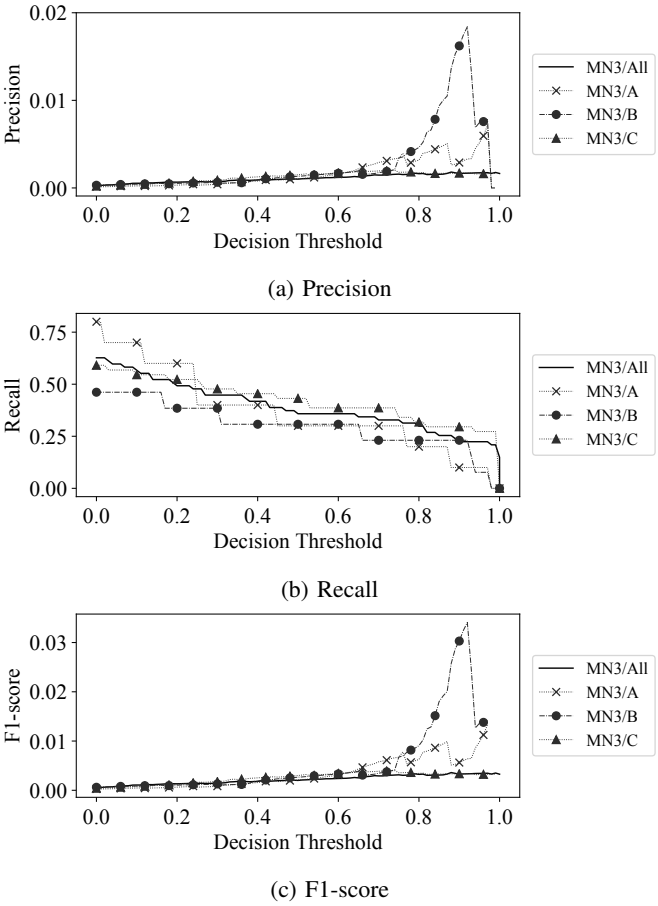
few node–minutes over the whole study. As the threshold is reduced, the model losses increase, reaching 8200 node–hours for a threshold of 0. For an UE cost of 50 node–hours, the optimal threshold reaches a plateau around the value of 0.7. Outside this range, the model efficiency drops especially as the threshold decreases. For thresholds below 0.2, the model generates additional losses that reach 6400 node–hours for a threshold of 0 (not shown in the chart). For an UE cost of 500 node–hours, the model reaches the highest savings for a decision threshold of 0.09, with a sharp decline on both sides, as shown in Figure 5c. In this case, a good threshold selection saves thousands of node–hours. Finally, for the job size distribution from MareNostrum 4 production, the optimal decision threshold is 0.02.

To further explore the dependency between the UE cost and the decision threshold, Figure 6 shows the optimal threshold as the UE cost varies between 5 and 5000 node–hours. The chart has three areas. For small UE costs (5 and 10 node–hours), the optimal threshold equals 1, so the system should not perform error mitigation actions because their overhead would exceed the benefits. For large UE costs (over 1000 node–hours) the optimal threshold converges to 0. In this case, the system should perform error mitigations whenever triggering events are detected in the preceding prediction interval. In these two UE cost areas, the error prediction is trivial—*predict no UEs* for small UE cost and *always predict an UE* for large costs. In the area of moderate UE costs, between 10 and 1000 node-hours in our case, the prediction is more complex, requiring proper prediction methods and careful threshold selection.

Production HPC systems execute jobs of different sizes and with different failure mitigation strategies. On a given system, the UE cost varies among jobs, and even within a single job depending on the time since it started or last performed a checkpoint. Our study shows that the UE prediction decision threshold should be determined at runtime based on the UE cost, i.e. based on the characteristics of the running HPC job. We hope to motivate further discussion and development of error prediction and mitigation methods that could be adapted to the diversity of workloads executed on production HPC systems.

D. Precision, Recall and F1-score

Next we evaluate our prediction model using standard data prediction metrics: precision, recall and F1-score (see Section IV-B). The results are presented in Figure 7. The **precision** is the ratio between correctly predicted UEs (true positives) and the total number of predicted UEs (true and false positives).



(a) Precision

(b) Recall

(c) F1-score

Fig. 7: Precision, recall and F1-score for different threshold values. These standard data prediction metrics are not correlated with a cost–benefit analysis (Fig. 5), and therefore cannot be used to decide whether prediction is useful in practice.

In our case, the dominant factor for the calculation of the precision is the large number of false positives, which leads to small values of the precision, typically below 1%. Increasing the threshold reduces the number of false positives and improves the precision, but its value remains small.

For MN3/A and MN3/B, and threshold values between 0.8 and 1.0, we detect higher variability in the precision. For these threshold values, the number of predicted UEs in the MN3/A and MN3/B subsystems is small (see Figure 4a), and each predicted UE therefore makes a significant relative difference to the value of the precision. The **recall** is the ratio between the numbers of correctly predicted UEs (true positives) and the total detected UEs (true positives and false negatives). The recall chart has the same form as Figure 4a (number of correctly predicted UEs), relative to the number of detected errors in each system partition: MN3/All, MN3/A, MN3/B or MN3/C. For *Threshold*=0 the recall reaches 0.63 for MN3/All

and 0.8 for MN3/A. The **F1-score**, as the harmonic mean of precision and recall, is dominated by the low precision numbers. It practically matches the precision curve, with slightly higher values.

The results show that the precision, recall and F1-score are not correlated with a cost–benefit analysis. There is nothing in Figure 7 to indicate that the optimal decision threshold is 1.0, 0.69 and 0.09 for small, medium and large UE cost, respectively, or 0.02 for the MareNostrum 4 job distribution, or that the net benefit for large UE cost is high. This is because precision puts the same weight on different prediction outcomes, true positives and false positives, whose costs differ by orders of magnitude. Also, it does not consider false negatives, which have a very high cost. Recall does not consider false positives, which incur costly mitigation measures when they are not needed. Also, in our case, the F1-score is dominated by the low precision values. Overall, these standard data prediction metrics are insufficient to decide whether and for which parameters the prediction is useful in practice, and we would suggest to complement them with cost–benefit analysis in future studies.

E. Feature importance

We quantify the importance of the features used for prediction (introduced in Section III-D) using the Gini importance [25], which is a common metric in the context of random forest classifiers. Figure 8 shows the Gini importance, grouped by category and excluding the first year of the study. We see large error bars, indicating large changes in importance over the course of the last 14 months of the study, which is due to the small number of UEs and large changes in importance as a consequence of individual UEs. The error bars would be larger if the first year were included or if individual features rather than categories were plotted.

We detect a high Gini importance (relevance to prediction) of the features relating to the exact error location: rank, bank, row and column. The column features have the highest prediction relevance, with a Gini importance of 0.22, while the cumulative rank, bank, row and column features have a Gini importance reaching 0.38. This is important because corrected DRAM error logs do not often include this level of information. The machine-check architecture registers record only the error address, and the mapping to the exact physical location is not trivial and requires non-disclosed manufacturers information. In our case, the mapping is done by a custom daemon that is designed with help from a memory manufacturer. We also see the importance of socket– and node–level events, such as CEs, UEs, UE warnings (Section II-B) and boots. DIMM characteristics, UEs and UE warnings, however, show a small importance. After detailed analysis of the logs, for instance, we detected that only eight DIMMs experienced an UE warning within 24 hours before the uncorrected DRAM error, and only five DIMMs had an UE warning within an hour before the error. This is interesting because it contradicts the intuition that DIMM UE warnings would be expected to precede an uncorrected error.

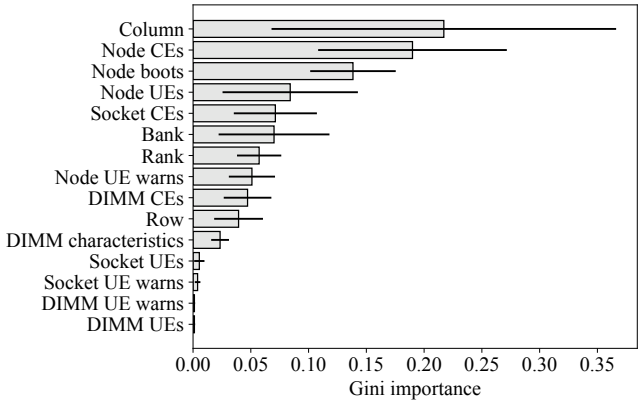


Fig. 8: Gini importance for different feature categories.

F. Importance of UE reduction

In Section II-C we explained that UEs often arrive in bursts but only the first UE in a burst impacts the system's reliability. In this section we illustrate why UE reduction (focussing only on the UEs that impact production workloads) is important for correct prediction model design and evaluation.

Table III shows the impact on the numbers and percentages of UEs, with and without node–level UE reduction. The table shows the results for a prediction frequency of 1 minute, a prediction window of 1 day and a decision threshold of 0.5. We obtained similar results and the same conclusions for different values of these parameters.

We first consider the classification of UEs, which is clearly affected by UE reduction. Without reduction, the total number of UEs increases five-fold, from 67 to 333 UEs. The number of critical over-temperature UEs increases by a factor of 22, from 12 to 262 errors, which is an increase from 18% of all UEs to 79% of UEs without reduction. This implies that over-temperature UEs are especially likely to appear in bursts. We also performed a DIMM–level reduction (not in the table), which removed bursts of UEs in the same DIMM rather than the whole node. A 1 week DIMM reduction reduces the number of UEs to 105, indicating that most of the UE bursts occur in the same DIMM. Nevertheless, as explained in Section II-C, since the whole node is rebooted and removed for testing following the first UE on that node, we continue the analysis with node–level UE reduction.

The predictability of UEs is also affected by UE reduction. Without UE reduction, a much higher fraction of errors have a preceding event (95% rather than 81%) and could potentially be predicted (90% rather than 63%). Also, since UEs in bursts are easier to predict, the number of correctly predicted UEs increases from 40% to 62%. Nevertheless, since bursts of errors occur very fast, most of the successfully predicted UEs are predicted with less than two minutes' lead time, leaving insufficient time to complete a mitigation measure before the UE halts the node. Without UE reduction, out of 62% of the correctly predicted UEs, only 19% are predicted with sufficient lead time. With UE reduction, this problem is negligible.

TABLE III: Quantitative comparison of UE classification and predictability with and without UE reduction.

	With UE reduction: 1 week–node		No UE reduction	
Detected UEs	67	100%	333	100%
UE classification:				
Over-temperature	12	18%	262	79%
Application read	38	57%	43	13%
Patrol scrub	17	25%	28	8%
UE predictability:				
UEs with preceding event	54	81%	318	95%
Predictable UEs[(a)]	42	63%	301	90%
Predicted UEs[(a)]	27	40%	205	62%
Predicted UEs $\geq$ 2min[(a)]	25	37%	63	19%

[(a)] Pred. freq = 1 min, Pred. window = 1 day, Decision threshold = 0.5

In summary, the numbers of UEs, their classification and predictability are quantitatively different depending on whether or not reduction is performed. Omitting UE reduction would change not only the model evaluation, but also its design. Keeping the UE bursts in the training datasets would encourage the model to predict such events. For example, when UE reduction is correctly applied, the importance of features that capture preceding node UEs is moderate, with an Gini importance of 0.08, as in Figure 8. If UE reduction is omitted, these features become the most relevant ones, with an importance of 0.28. These results would lead to different evaluation outcomes, independent of the metrics used.

VI. Other prediction methods

This section compares the results of six classifiers explored in the study: random forest (RF), logistic regression (LR), gradient boosting decision tree (GBDT), Gaussian naïve Bayes (GNB), support vector machine (SVM) and deep neural network (NN). The assumptions used in this section are the same as in Section V: cost of the UE impact mitigation is 2 minutes, prediction frequency and window are 1 minute and 1 day, respectively, and decision threshold is selected to be optimal for the given classifier and UE cost.

Figure 9 shows the results of the **cost–benefit calculation**. The figure plots the saved node–hours (y-axis) for various UE costs (x-axis). Different bars show results for different classifiers. Random forest shows the best overall results. It is closely followed by logistic regression and then gradient boosting and Gaussian naïve Bayes methods. Support vector machine shows the lowest savings, up to 20% below random forest for a UE cost of 500 node–hours. Figure 9 shows that for all the classifiers the UE cost strongly influences the cost–benefit calculation. Also the saved node–hours increase superlinearly with the average UE cost, which confirms the analysis and findings presented in Section V-A.

Next, we compare the different classifiers using standard data prediction metrics: **precision, recall and F1-score**. The trends and conclusions closely match those for the random forest (Section V-D). A large number of false positives leads to small values of the **precision**, below 1% for all the methods.

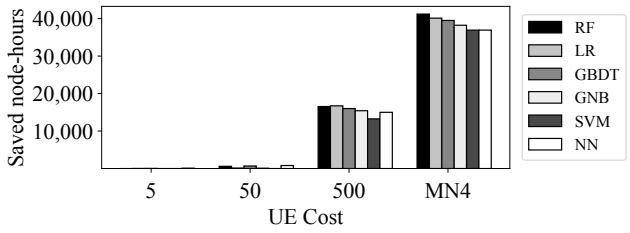


Fig. 9: For all the classifiers the UE cost strongly influences the cost–benefit calculation. Random forest shows the best results.

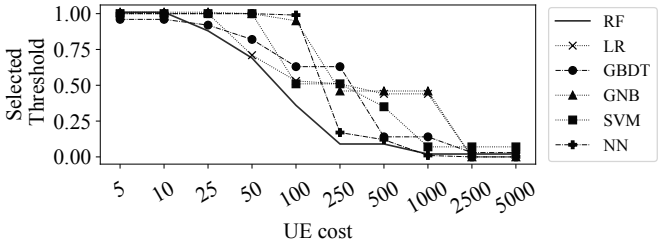


Fig. 10: For small UE costs, the optimal threshold equals 1; for large UE costs it converges to 0. For moderate UE costs, each prediction method requires a careful threshold selection.

For all classifiers the highest value of **recall** is 0.63 (for *Threshold*=0), for which all classifiers always predict a UE if there is a preceding event. For all classifiers, the range of recall is zero (for *Threshold*=1) to 0.63, but there are slight differences in the recall–threshold curves (outside the scope of this paper). Finally, the **F1-score**, as the harmonic mean of precision and recall, is dominated by the low precision. Recall is 1% or below for all classifiers.

To further explore the dependency between UE cost and the decision threshold for different classifiers, Figure 10 shows the optimal threshold as the UE cost varies between 5 and 5000 node–hours. Similar to Figure 6, the chart has three areas. For small UE costs, the optimal threshold equals 1 while for large UE costs it converges to 0. In the area of moderate UE costs, the prediction is more complex, and it requires a careful threshold selection for each prediction method. Figure 10 confirms the need for runtime-adaptive error prediction and mitigation methods that would adjust to the characteristics of the HPC jobs and associated UE cost.

VII. Related work

A. Uncorrected DRAM errors

A study of Giurgiu et al. [2] uses random forest to predict uncorrected DRAM errors. The proposed model is based on preceding corrected errors and measurements from over 100 sensors that monitor system functioning. The goal of Giurgiu et al. is to predict UEs while minimizing false positives. For this reason, the method targets high precision and tolerates low recall. Our goal is to maximize the number of saved node–hours. We therefore target mid-high recall and tolerate a higher number of false positives and low precision.

Our study extends the work of Giurgiu et al. in various aspects. We perform detailed corrected error logging with the exact error location (rank, bank, row, column) and show the importance of observation features based on this data. In the model evaluation we consider the lead time analysis and perform a sensitivity analysis on the prediction frequency and window. We also analyze burstiness of UEs and show that it has an important impact on the design and evaluation of the prediction model. Our study also considers a potential bias of the DIMMs that are replaced due to the pre-failure alerts already implemented in the system under study. While Giurgiu et al. evaluate their model with the precision, recall and balanced accuracy, we also perform a detailed cost–benefit analysis.

B. Corrected DRAM errors

A few recent studies present machine learning methods to classify or predict future corrected DRAM errors.

Costa et al. [42] propose an OS mechanism that monitors memory health and predicts memory error repetition. Based on the number of spatial repetitions and the error rates, the proposed mechanism classifies memory pages as healthy, unhealthy or fatal. The pages classified as fatal are offlined to prevent future DRAM errors. The proposed OS mechanism is implemented and evaluated on a Blue Gene/Q system.

Baseman et al. [34] use naïve Bayes, logistic regression, random forest and gradient boosted random forest classifiers to categorize memory fault modes as Single bit, Single word, Single row, Single column, Single bank, Multi bank or Multi rank. Their objective is to classify the faults based on prior detected errors, rather than predicting future errors. A follow-up work [45] extends the proposed prediction methods to predict which memory pages would experience future faults, similar to the study of Costa et al. [42]. The proposed methods are trained and tested on two large HPC systems, Hopper and Cielo, and they show predictive performance improvement compared with deterministic rule-based systems.

Sun et al. [41] use neural networks to predict disk and DRAM errors. DRAM error prediction uses the features memory usage, memory corrected errors, memory speed, memory power, other MCE-related errors, memory age and node load. The DRAM error predictions are evaluated using logs that contain hundreds of DRAM failures from an in-house system. The proposed scheme outperforms the baseline long short-term memory and random forest in precision, recall and F1-score.

Du and Li [43] propose a method to predict DRAM errors in micro-level components, such as cell rows and columns. The method is based on a kernel function that measures the similarity between the current observation and a certain previous observation in history. The advanced method also accounts for memory failure propagation. The method is evaluated on DRAM error logs from the IBM Blue Gene/P HPC cluster (DDR2 memory) containing 140 thousand corrected errors. The proposed error prediction method shows better precision, recall and F1-score than various baseline approaches that use the aggregated error data in history.

These studies are valuable to understand corrected DRAM error rates, distributions, correlated factors and features that can be used for their prediction. An important direction of future work would be to explore how these findings could lead to measurable improvements in system reliability. An important issue is that reliability is only impacted by *uncorrected errors* [5]–[8], and there is no direct relation between corrected and uncorrected errors [2], [3], [8]–[11]. Modifying the proposed corrected error predictors to instead predict uncorrectable errors may be challenging [43].

C. GPU memory errors

A few recent studies have analysed GPU errors in the field [46], [47]. Nie et al. [47] analyze the GPU errors on the Titan supercomputer, which comprises 18,688 K20X GPUs. The authors study the system conditions and workload characteristics that trigger GPU errors, and they propose and evaluate several machine learning-based models: logistic regression, gradient boosting decision tree, support vector machine, and neural network. The study analyzes all corrected single-bit errors together and does not distinguish between errors in different memory structures: register files, caches and device memory. A previous study that analyzes GPU errors on the same system [48] reports that 98% of the detected errors come from the L2 cache. The findings of Nie et al. [47], therefore, could not be directly applied to device DRAM error prediction.

VIII. Conclusions

This paper presented and evaluated a method to predict DRAM uncorrected errors that lead to node failure. Our cost–benefit analysis shows that the prediction method reduces the lost compute time by up to 57% for a real HPC production workload mix, which is a net savings of 21,000 node–hours per year. After comparison of six machine learning approaches, we use a random forest classifier, which is trained and evaluated using error logs from more than two years of production of MareNostrum 3. All prediction methods' source code is released as open source. We also discuss and clarify several aspects of methodology that are essential for any prediction method to be useful in practice. Our cost–benefit analysis shows that the effectiveness of our prediction scheme is highly dependent on system and workload characteristics, pointing the way to future work on adaptive resiliency techniques. Overall, we hope that future researchers will build on our work to improve the throughput of production HPC systems as demonstrated by a clear cost–benefit calculation.

Acknowledgment

This work was supported by the Spanish Ministry of Science and Technology (project PID2019-107255GB), Generalitat de Catalunya (contracts 2014-SGR-1051 and 2014-SGR-1272) and the European Union's Horizon 2020 research and innovation programme and EuroEXA project (grant agreement No 754337). Paul Carpenter and Marc Casas hold the Ramon y Cajal fellowship under contracts RYC2018-025628-I and RYC2017-23269, respectively, of the Ministry of Economy and Competitiveness of Spain.

REFERENCES

[1] HP, "How memory RAS technologies can enhance the uptime of HPE ProLiant servers," Hewlett Packard Enterprise, Technical white paper 4AA4-3490ENW, Feb 2016.

[2] 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*, 2017, pp. 15–21.

[3] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM Errors in the Wild: A Large-scale Field Study," in *Proceedings of the International Joint Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2009, pp. 193–204.

[4] 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," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 111–122, 2012.

[5] B. Schroeder and G. A. Gibson, "A Large-Scale Study of Failures in High-Performance Computing Systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 337–350, Oct 2010.

[6] C. D. Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer, "Lessons Learned from the Analysis of System Failures at Petascale: The Case of Blue Waters," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2014, pp. 610–621.

[7] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari, "Failures in Large Scale Systems: Long-term Measurement, Analysis, and Implications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2017, pp. 44:1–44:12.

[8] D. Zivanovic, P. E. Dokht, S. Moré, J. Bartolome, P. M. Carpenter, P. Radojković, and E. Ayguadé, "DRAM Errors in the Field: A Statistical Approach," in *Proceedings of the International Symposium on Memory Systems (MEMSYS)*, 2019, pp. 69–84.

[9] P. Radojkovic, M. Marazakis, P. Carpenter, R. Jeyapaul, D. Gizopoulos, M. Schulz, A. Armejach, E. Ayguade, F. Bodin, R. Canal, F. Cappello, F. Chaix, G. Colin de Verdiere, S. Derradji, S. Di Carlo, C. Engelmann, I. Laguna, M. Moreto, O. Mutlu, L. Papadopoulos, O. Perks, M. Ploumidis, B. Salami, Y. Sazeides, D. Soudris, Y. Sourdis, P. Stenstrom, S. Thibault, W. Toms, and O. Unsal, "Towards Resilient EU HPC Systems: A Blueprint." European HPC resilience initiative. White paper, April 2020. [Online]. Available: https://resilienthpc.eu/blueprint2020

[10] 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 (SC)*, 2012, pp. 76:1–76:11.

[11] S. Levy, K. B. Ferreira, N. DeBardeleben, T. Siddiqua, V. Sridharan, and E. Baseman, "Lessons Learned from Memory Errors Observed over the Lifetime of Cielo," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2018, pp. 554–565.

[12] N. Taerat, C. Leangsuksun, C. Chandler, and N. Naksinehaboon, "Proficiency Metrics for Failure Prediction in High Performance Computing," in *International Symposium on Parallel and Distributed Processing with Applications*, 2010, pp. 491 – 498.

[13] Z. Zheng, Z. Lan, R. Gupta, S. Coghlan, and P. Beckman, "A practical failure prediction with location and lead time for Blue Gene/P," in *International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2010, pp. 15–22.

[14] D. Jauk, D. Yang, and M. Schulz, "Predicting Faults in High Performance Computing Systems: An in-Depth Survey of the State-of-the-Practice," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2019, pp. 1–13.

[15] A. Frank, D. Yang, A. Brinkmann, M. Schulz, and T. Süss, "Reducing False Node Failure Predictions in HPC," in *IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2019, pp. 323–332.

[16] Barcelona Supercomputing Center, *MareNostrum 3 User's Guide*, Apr. 2016.

[17] I. Boixaderas, D. Zivanovic, S. Moré, J. Bartolome, D. Vicente, M. Casas, P. M. Carpenter, P. Radojković, and E. Ayguadé, "UEPREDICT: A method for predicting DRAM Uncorrected Errors and evaluating its model's performance," 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3872777

[18] "PRACE Research Infrastructure," http://www.prace-ri.eu.

[19] A. Kleen, "MCELOG: Memory Error Handling in User Space," in *International Linux System Technology Conference (Linux Kongress)*, 2010.

[20] V. Sridharan, J. Stearley, N. DeBardeleben, S. Blanchard, and S. Gurumurthi, "Feng Shui of Supercomputer Memory: Positional Effects in DRAM and SRAM Faults," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2013, pp. 22:1–22:11.

[21] 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 International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015, pp. 297–310.

[22] X. Li, M. C. Huang, K. Shen, and L. Chu, "A Realistic Evaluation of Memory Hardware Errors and Software System Susceptibility," in *Proceedings of the USENIX Conference on USENIX Annual Technical Conference (USENIXATC)*, 2010, pp. 6–6.

[23] *System x iDataPlex dx360 M4 Types 7912 and 7913: Problem Determination and Service Guide*, IBM, Apr 2014.

[24] Barcelona Supercomputing Center, "MareNostrum 4 (2017) System Architecture," https://www.bsc.es/marenostrum/marenostrum/technical-information, 2017.

[25] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[26] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[27] G. Bonaccorso, *Machine learning algorithms*. Packt Publishing Ltd, 2017.

[28] J. A. Nelder and R. W. Wedderburn, "Generalized linear models," *Journal of the Royal Statistical Society: Series A (General)*, vol. 135, no. 3, pp. 370–384, 1972.

[29] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[30] K. L. Priddy and P. E. Keller, *Artificial neural networks: an introduction*. SPIE press, 2005, vol. 68.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[32] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in neural information processing systems*, 2017, pp. 3146–3154.

[33] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, 2016, pp. 265–283.

[34] E. Baseman, N. DeBardeleben, K. Ferreira, S. Levy, S. Raasch, V. Sridharan, T. Siddiqua, and Q. Guan, "Improving dram fault characterization through machine learning," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. IEEE, 2016, pp. 250–253.

[35] M. M. Botezatu, I. Giurgiu, J. Bogojeska, and D. Wiesmann, "Predicting Disk Replacement towards Reliable Data Centers," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016, p. 39–48.

[36] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[37] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[38] C. Chen, A. Liaw, and L. Breiman, "Using Random Forest to Learn Imbalanced Data," *University of California, Berkeley*, vol. 110, no. 1-12, p. 24, 2004.

[39] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017.

[40] USENIX: The Advanced Computing Systems Association, "The Computer Failure Data Repository (CFDR)," https://www.usenix.org/cfdr, Jun 2020.
[41] X. Sun, K. Chakrabarty, R. Huang, Y. Chen, B. Zhao, H. Cao, Y. Han, X. Liang, and L. Jiang, "System-Level Hardware Failure Prediction Using Deep Learning," in *Proceedings of the 56th Annual Design Automation Conference (DAC)*, 2019, pp. 1–6.
[42] 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 (SC)*, 2014, p. 707–718.
[43] X. Du and C. Li, "Memory Failure Prediction Using Online Learning," in *Proceedings of the International Symposium on Memory Systems (MEMSYS)*, 2018, p. 38–49.
[44] A. Das, F. Mueller, P. Hargrove, E. Roman, and S. Baden, "Doomsday: Predicting Which Node Will Fail When on Supercomputers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*, 2018, pp. 108–121.
[45] E. Baseman, N. DeBardeleben, K. Ferreira, V. Sridharan, T. Siddiqua, and O. Tkachenko, "Automating dram fault mitigation by learning from experience," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2017, pp. 137–140.
[46] B. Nie, D. Tiwari, S. Gupta, E. Smirni, and J. H. Rogers, "A large-scale study of soft-errors on gpus in the field," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016.
[47] B. Nie, J. Xue, S. Gupta, T. Patel, C. Engelmann, E. Smirni, and D. Tiwari, "Machine Learning Models for GPU Error Prediction in a Large Scale HPC System," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018, pp. 95–106.
[48] D. Tiwari, S. Gupta, J. Rogers, D. Maxwell, P. Rech, S. Vazhkudai, D. Oliveira, D. Londo, N. DeBardeleben, P. Navaux, L. Carro, and A. Bland, "Understanding GPU errors on large-scale HPC systems and the implications for system design and operation," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 331–342.