Predicting Remediations for Hardware Failures in Large-Scale Datacenters

Fred Lin, Antonio Davoli, Imran Akbar, Sukumar Kalmanje, Leandro Silva,
John Stamford, Yanai Golany, Jim Piazza, Sriram Sankar
Facebook Inc.

***Abstract*—Large-scale service environments rely on autonomous systems for remediating hardware failures efficiently. In production, the autonomous system diagnoses hardware failures based on the rules that the subject matter experts put in the system. This process is increasingly complex given new types of failures and the increasing complexity in the hardware and software configurations.**

In this paper, we present a machine learning framework that predicts the required remediations for undiagnosed failures, based on the similar repair tickets closed in the past. We explain the methodology in detail for setting up a machine learning model, deploying it in a production environment, and monitoring its performance with the necessary metrics. We also demonstrate the prediction performance on some of the repair actions.

I. Introduction

Efficient hardware failure remediation is the foundation for sustaining a fleet of hardware at high availability for a large-scale service environment. Autonomous systems such as Borg from Google [1], Autopilot from Microsoft [2], and FBAR from Facebook [3], have been deployed in datacenters to diagnose and remediate the hardware failures. The goal of these autonomous systems is to triage the hardware failures quickly and accurately, so the servers can be released back to production in a healthy state without the same failure reoccurring in the near future.

However, due to the high complexity in the hardware failures introduced from new hardware, firmware, and software, there are sometimes new failure modes for which we do not have remediation rules in the autonomous remediation system. These new failure modes therefore become undiagnosed failures, which require human investigation for the required remediation. The human investigation could be time-consuming, leading to lower server availability and lower server-to-administrator ratio at the datacenters [4]. Given the geographically distributed nature of the datacenters, it is also challenging for the engineers to be fully synchronized on the most effective remediations for each undiagnosed issue.

In [3], a machine learning framework that predicts the resolutions for undiagnosed and misdiagnosed failures was briefly discussed. In this paper, we extend the discussion by presenting the details about model development, prediction deployment criteria, and the continuous operational support required for the framework in production. We also demonstrate the prediction performance for some of the repair actions. Using natural language processing (NLP) techniques, the machine learning model learns from how past hardware failures have been remediated given the failure signals from the server and tooling logs, and predicts the required remediation for a new undiagnosed failure. With the predicted diagnoses, we can shorten the server downtime from manual investigation, and analyze the patterns the model finds to learn about how failures are commonly resolved across all datacenters.

Given the dynamic and complex nature of a large-scale service environment, the machine learning model needs to be retrained periodically and its performance needs to be monitored in production to detect regressions. In addition to the prediction accuracy, *i.e.* whether the predicted repair could bring the server back to a healthy state, we monitor the *repeat offender rate* to make sure the remediation actually fixes the underlying failures, instead of temporarily masking them by actions such as server reboots.

The rest of the paper is organized as the following: We introduce the hardware failure detection and remediation flow in a production environment in Section II. The machine learning model, its input data, and an accuracy upper bound are discussed in Section III, while a practical flow for deploying the machine learning framework in production is illustrated in Section IV. The prediction performance is presented in Section V, and the conclusion is presented in Section VI.

II. Hardware Failure Remediation in a Large-Scale Service Environment

Hardware fails in datacenters for various reasons, such as material degradation from use (*e.g.* the mechanical parts of a spinning hard drive or the flash cells of a solid state drive), environmental impacts (*e.g.* corrosion due to humidity or damage from electrostatic discharge), and manufacturing defects [5]. In this section we introduce an autonomous flow for remediating hardware failures across distributed datacenters.

A. Failure Detection

The first step for fixing a broken hardware is to detect the failure. As illustrated in [3], a tool named MachineChecker [6] runs a collection of checks on each server periodically to check for server failures. Examples of the checks are host and out-of-band (OOB) ping, power supply check, sensor check, network interface card (NIC) speed, memory error check, and hard disk drive (HDD) or solid state drive (SSD) S.M.A.R.T. check.

The frequency of MachineChecker runs is decided based on the trade-off between the time-to-detect of the failures and the overhead on the servers when running the checks. At a

lower level, we can also adjust the hardware interrupt handling mechanism and polling frequency for detecting machine errors while minimizing the CPU stalls [5].

B. Failure Remediation

Once a hardware failure is detected by MachineChecker, Facebook Auto-Remediation (FBAR) [7] and Cyborg [6] would try to auto-remediate the failure [3]. If neither FBAR or Cyborg could auto-remediate the hardware failure, a repair ticket will be created.

Repair tickets are created for two main reasons: 1) a physical repair is required or 2) the remediation system does not have a rule for the failure signals. In the second case, an *undiagnosed* repair ticket is created. The repair actions can be categorized as *component* or *non-component* repairs. A component repair requires actions on a hardware component, *e.g.* swapping or reseating a motherboard. A non-component repair is done without interacting with a hardware component, *e.g.* reimage, firmware upgrade, or a reboot. To sustain the hardware fleet at high availability, our goal is to create as few undiagnosed tickets as possible, and execute as many required non-component repairs in auto-remediation as possible.

Note that although non-component repairs do not incur component cost, they are not free. The associated overhead from non-component repairs include additional server downtime and the potential to lose the failure signals. We will illustrate this with an example in Section IV-C.

III. The Machine Learning Framework

A. Input Data

For the machine learning model to learn about the similarities between repair tickets, each ticket is characterized with a set of *features*. The features include the failure signals collected from the MachineChecker checks and server attributes such as the model of the server. Other potentially useful server information, including the hardware and software configurations, environmental variables, resource utilizations, and the repair history of the server, are potentially useful and are being incorporated to the dataset.

Another aspect of the input data is the time window of the repair tickets that we want the model to learn from. Including tickets from a longer time window as the *training set* provides more samples per repair action for the machine learning model to learn from. However, the production environment is very dynamic with new failures modes and evolving repair behaviors, therefore it is also important to have the model focus more on the recent samples. We have found a training window of 6 months to be practical, and we implemented a supplementary mechanism to detect new failure modes and blacklist them when the prediction accuracy is low.

Since the server failure signals collected from the server logs are largely free-form texts, we need to first convert the text to a vector representation before the machine learning model could consume them. Common techniques for the conversion include word frequency based approaches such as Term Frequency-Inverse Document Frequency (TF-IDF) [8] and word embedding such as fastText [9]. While TF-IDF represents the text by the relative frequency of the exact words seen in the training data, text embedding could map related words to vectors that are close in the embedding feature space. In our setting, since the vocabulary found in the server failure logs is relatively small and it is rare to see new or ambiguous words, we have found that TF-IDF and fastText word embedding would perform similarly on our dataset. For simplicity, we productionized the model using TF-IDF vectorization.

B. The Machine Learning Model

After the free-form texts are vectorized, the features can be concatenated with the vectors representing the server attributes and repair history. The categorical features need to be presented as binary vectors through one-hot encoding [10]. We chose to use Gradient Boosted Decision Tree (GBDT) [11] as the classifier for its simplicity and comparable performance with the other state-of-the-art methods. Fig. 1 shows the data flow of the model. With the compatibility for text, categorical, and parametric features, the framework can be easily extended to incorporate more features in the future.

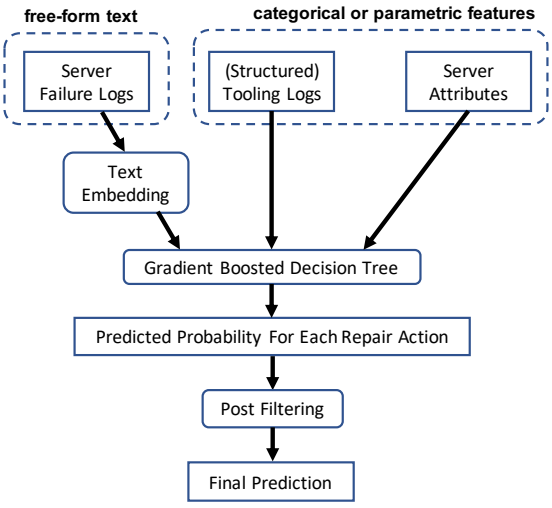

Fig. 1. Data flow for the machine learning framework.

C. The Upper Bound on Prediction Accuracy

As the machine learning model learns from how similar repair tickets have been closed for new undiagnosed tickets, many similar tickets in the training dataset would be undiagnosed tickets closed with human investigation and remediation. There is typically a higher variance in the repair actions from human remediation, because the engineers can be unfamiliar with the failures, and the diagnosis practice could slightly vary across different datacenters. As a result, given exactly the same feature values, repair tickets could have been closed with different remediations. When calculating the prediction accuracy against the actual ticket resolutions, this variation in ticket resolutions, which we measure with a metric *repair consistency*, imposes an upper bound to the prediction accuracy,

because the machine learning model would always make the same prediction given the same signals. Without additional information to model the variation, the predictions cannot match all the actual ticket resolutions. The repair consistency can be weighted based on the quality of the historical repairs on groups of tickets and engineers.

IV. Deploying the Predictions in Production

A. Confidence Thresholds

The machine learning model makes predictions with a *score*, a value between 0 and 1. The prediction score does not directly indicate the probability of the prediction being correct, but shows the relative confidence the model has about the prediction. In practice, we can set a threshold on this confidence score to act only on the predictions with higher confidence. Different confidence thresholds would result in different values of *precision* and *recall*, or similarly, *true positive rate (TPR)* and *false positive rate (FPR)* [12]. Precision measures the correctness of the positive predictions, and recall measures the coverage of positive samples by the correct predictions. In other words, precision is a constraint, and recall is the gain given the constraint. Fig. 2 is an example of the *precision-recall curve* for motherboard swap predictions.

For hardware remediation, the cost of an incorrect repair action includes wasted parts, unnecessary human labor, shipping and engineering cost for testing the returned device, and additional server downtime. Once the saved and incurred resources from correct and incorrect predictions are derived based on these factors, we can normalize the values and convert the precision-recall curve into a cost-saving curve, and pick an operation point with a reasonable pair of cost and saving values. While it is intuitive to find the operation point with maximized total saving, sometimes there can be additional constraints on the incurred cost, *e.g.* a limit on the number of parts we are willing to throw away due to incorrect predictions.

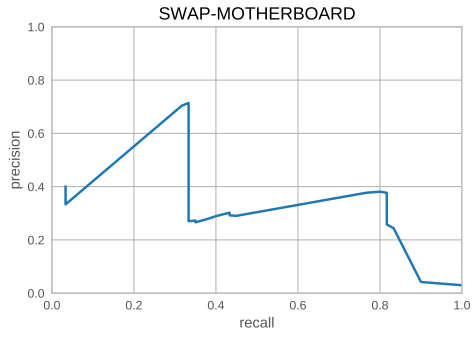

Fig. 2. The precision-recall curve for swap motherboard predictions.

B. Executing the Predictions

The repair flow incorporating the predicted repairs is illustrated in Fig. 3. The repair handler is a new step in the remediation flow, right before when repair tickets are created. The repair handler sends the collected features about the failure to the machine learning model, and retrieves the most confident predictions along with their confidence scores. The repair handler then tries the high-confidence non-component predicted repairs by descending order of their confidence scores. If the hardware issue remains after the list of non-component repairs is exhausted, depending on whether there is a high-confidence predicted component repair, the repair handler would create either a diagnosed or undiagnosed repair ticket for human remediation.

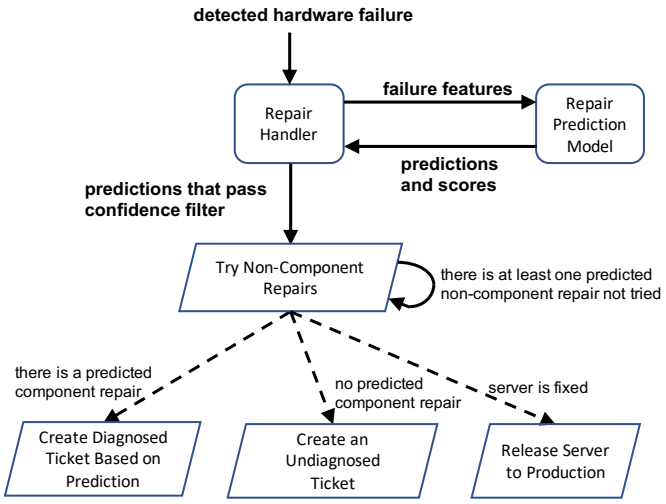

Fig. 3. Repair prediction execution flow.

For each repair type, we set a confidence threshold and a switch to turn the prediction off in case an emerging issue in production needs to be left untouched for investigation. The configuration can be specified at finer granularity to include other constraints such as server model and datacenters.

C. Performance Monitoring

The precision and recall of the model are continuously monitored and evaluated for replacing the prodiction model with a retrained model candidate based on more recent data. For the framework built with TF-IDF and GBDT, each retraining takes less than an hour to finish.

In addition to the precision and recall per repair type, we monitor the *repeat offender rate* of the hardware failures for evaluating the quality of the repairs that close the repair tickets. In some cases, a hardware failure is transient and can be temporarily alleviated by resetting program states or rebooting the server. For example, PROCHOT is a signal that could indicate CPU overheating and would throttle CPU frequency [3], [13]. One common reason for PROCHOT is that the thermal compound has worn out over time, and it is more likely to be triggered when the server is running heavy production load. Rebooting the server can naturally clear the PROCHOT condition because the CPU temperature drops when the server stops serving production traffic during the reboot. However, a reboot does not fix the underlying issue

about the thermal compound, and PROCHOT is likely to be triggered again in production. Therefore, the repeat offender rate of the failures is an important metric for capturing how the failures are properly fixed, instead of being temporarily remediated.

V. Prediction Performance

Table I presents the precision and recall of some of the most accurately predicted repair actions for undiagnosed repair tickets. The result is based on a high volume of representative repair tickets that were sampled from the production repair tickets in 2019. In this setting, we take the prediction with the highest confidence score per failure without a confidence threshold, or equivalently, a confidence threshold at 0.

Without a confidence threshold, the predictions for some repair types such as SWAP FLASH show lower precision but very high recall. If we execute all the predicted SWAP FLASH repairs without checking the prediction confidence, we will cover more actual SWAP FLASH repairs at the cost of additional unnecessary SWAP FLASH repairs. For this type of repairs, we need to raise the confidence threshold to move the operation point towards higher precision and lower recall to arrive at a practical operation point, based on the discussion in Section IV-A. SWAP CABLE, on the other hand, has perfect precision at a relatively lower recall. This means that the model predicts SWAP CABLE less frequently, but when it does the predictions are all correct. If we simply follow all of these predictions, we can correctly diagnose 33% of the undiagnosed repair tickets that require SWAP CABLE repairs.

Similar to Table I, we can also analyze the precision and recall by different failure signals. This analysis would give us hints about which failure signals are of less quality, *i.e.* less specific for pointing to an effective repair, and we can improve the failure signal logging accordingly.

TABLE I
Precision and recall of the most accurately predicted repair types.

Repair Action	Precision	Recall
SWAP CABLE	1.00	0.33
SWAP CPU	0.35	0.39
SWAP HDD	0.53	0.95
SWAP FLASH	0.44	1.00
SWAP MEMORY	0.28	0.69
SWAP MOTHERBOARD	0.57	0.33
RAID CONFIG	1.00	0.50
RESEAT RAID CARD	0.33	0.50

VI. Conclusion

As the hardware fleet scales and the complexity in the hardware and software configuration increases, rule-based diagnosis can be lagging for the new failure modes. In this paper, we present a machine learning framework that bridges the gap by predicting the proper repair actions based on the human repair behaviors observed at the datacenters.

In addition to providing diagnoses to undiagnosed repair tickets, this framework can also correct misdiagnosed repair tickets when the predicted repair is different from the rule-based diagnosis but has very high confidence. For improving the prediction performance, we are exploring other state-of-the-art text embedding techniques as well as incorporating more features including the hardware and software configurations, environmental variables, relative rack positions, and resource utilities.

While broader issues such as software configurations would be captured and escalated as repeat offenders, we prefer to capture these issues as early as possible. We have deployed the dimensional analysis presented in [14] on the undiagnosed and misdiagnosed tickets, when they are created and closed respectively, for the first time instead of waiting for them to become repeat offenders. Out of hundreds of server configuration variables, the dimensional analysis finds combinations of variable values associated with the target tickets automatically.

Acknowledgement

The authors would like to thank Matt Beadon, Stuart Cannon, George Chewning, Preston Connor, Pavan Daggubati, Ricardo Delfin, Harish Dattatraya Dixit, Brian Fredericks, Xiang Gao, Anna Jacobi, Ranjith Meka, Michał Naruniec, Varun Nirantar, Eoin O'Kane, Sathia Pitchai, Olivier Raginel, Joao Rechena, Lakhwinder Singh, Amandeep Singla, Gianni Vialetto, Seth Weidman, and Chi Zhou for their contribution to this work.

References

[1] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *European Conference on Computer Systems (EuroSys)*, Apr. 2015.

[2] M. Isard, "Autopilot: Automatic data center management," in *ACM SIGOPS Operating System Review*, Apr. 2007.

[3] F. F. Lin, M. Beadon, H. D. Dixit, G. Vunnam, A. Desai, and S. Sankar, "Hardware remediation at scale," in *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops*, Jun. 2018.

[4] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," in *ACM SIGCOMM Computer Communication Review*, Jan. 2009.

[5] H. D. Dixit, F. Lin, B. Holland, M. Beadon, Z. Yang, and S. Sankar, "Optimizing interrupt handling performance for memory failures in large scale data centers," in *ACM/SPEC International Conference on Performance Engineering*, 2020.

[6] R. Komorn. (2016) Python in production engineering. [Online]. Available: https://engineering.fb.com/production-engineering/python-in-production-engineering/

[7] (2011) Making facebook self-healing. [Online]. Available: https://www.facebook.com/notes/facebook-engineering/making-facebook-self-healing/10150275248698920/

[8] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," in *Journal of Documentation*, vol. 28, no. 1, 1972.

[9] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *arXiv:1607.01759*, 2016.

[10] D. Harris and S. Harris, *Digital design and computer architecture*, 2nd ed. Morgan Kaufmann, 2012.

[11] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," in *The Annals of Statistics*, vol. 29, no. 5, 2001.

[12] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *Internation Conference on Machine Learning*, 2006.

[13] Intel, "Intel 64 and ia-32 architectures software developer manuals," 2016.

[14] F. Lin, K. Muzumdar, N. P. Laptev, M.-V. Curelea, S. Lee, and S. Sankar, "Fast dimensional analysis for root cause investigation in a large-scale service environment," in *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, 2020.