Hardware Remediation At Scale

Fan (Fred) Lin, Matt Beadon, Harish Dattatraya Dixit, Gautham Vunnam, Amol Desai, Sriram Sankar
Facebook Inc.

***Abstract*—Large scale services have automated hardware remediation to maintain the infrastructure availability at a healthy level. In this paper, we share the current remediation flow at Facebook, and how it is being monitored. We discuss a class of hardware issues that are transient and typically have higher rates during heavy load. We describe how our remediation system was enhanced to be efficient in detecting this class of issues. As hardware and systems change in response to the advancement in technology and scale, we have also utilized machine learning frameworks for hardware remediation to handle the introduction of new hardware failure modes. We present an ML methodology that uses a set of predictive thresholds to monitor remediation efficiency over time. We also deploy a recommendation system based on natural language processing, which is used to recommend repair actions for efficient diagnosis and repair. We also describe current areas of research that will enable us to improve hardware availability further.**

I. Introduction

Large scale services are hosted on servers that are distributed across multiple datacenters in different regions. There are several thousands of servers in each datacenter, and to manage them effectively there are distributed management systems [1]–[4]. An important piece of the management system is to detect when there are hardware issues and react to the failure. Remediation actions for typical failures include actions such as migrating services, rebooting, reimaging or off-lining the machine for manual repair. To make sure our servers are supporting the services properly, we have been continuously working on improving our hardware remediation efficiency and extracting insight for hardware health at scale. In 2011, we introduced an auto-remediation tool named Facebook Auto-Remediation (FBAR) [3], [4] to automate remediation routines for fixing servers in the datacenters. FBAR is now performing automatic remediations on our servers every day, and additional tools have been developed to improve the effectiveness of our remediation flow.

The effectiveness of any system designed to handle failures at scale starts with its ability to diagnose issues when they happen. In order to detect hardware failures at scale, we run a set of checks periodically on each server. Whenever a check fails, FBAR picks up the failure and tries to remediate the issue based on the configuration specified by the owner of the service that this particular server supports. If no automatic remediation is available for FBAR to fix the issue, FBAR passes the failure to a repair engine for trying lower-level system fixes such as firmware update or reimaging. If none of the attempts could fix the issue, a repair ticket is created for datacenter technicians to manually repair the server.

To ensure that we remediate the server failures correctly and efficiently, we track all the failures throughout the remediation process. One challenge we have observed in the remediation flow is that *transient failures* such as processor and memory overheating are not persistent, i.e. when the server is drained to stop receiving production workload, these transient failures would often disappear. Later when FBAR tries to remediate these failures, it may not see all of the failure signals and therefore treat the original failure detection as false alarms. To capture and remediate the transient failures effectively, we focused on *hardware error reproducibility*. We added a set of benchmarks for processor, memory, and network to act as a proxy for the production workload and traffic behavior, and this helps reproduce the transient failures during remediation.

Another challenge in system health management is to keep up with the changes in technology, such as the introduction of new components, which changes production behavior. As distributed system changes occur, we see issues that our detection system has not been designed to diagnose. It might detect a symptom of an issue happening, but would need additional improvements to diagnose accurately. For this aspect, we use machine learning approaches to automatically identify areas of opportunities. Facebook has a well-developed infrastructure for developing and deploying machine learning applications in production quickly [5]. From a pool of developed algorithms, teams can easily reuse the machine learning code developed by other teams in different disciplines. For monitoring the efficiency of the remediation flow, we monitor the results of each tool in the flow (e.g. percentage of failures closed as false alarms by FBAR), with anomaly detection on time series signals. Deploying the automatic anomaly detection helps us scale the monitoring to a large number of metrics and notify stakeholders immediately upon anomaly detection.

We also focus on improving the repair diagnostics for issues introduced by new hardware and software. We apply machine learning techniques to learn from past repair data, and then recommend actions for undiagnosed repair tickets. Undiagnosed tickets require datacenter technicians to manually debug the issue and would sometimes need to be escalated for further investigation. The application of machine learning results in much easier hardware debugging, and reduces the time the server is unavailable to serve production load. We apply a natural language processing machine learning model fastText [6], developed by Facebook AI Research (FAIR), to learn from the text logs populated for each repair ticket and recommend repair actions.

The rest of the paper is structured as follows: Section II covers the hardware remediation architecture, Section III describes the challenges in handling transient errors, the error reproducibility enhancements for detecting transient errors,

and the delay induced by logging events to System Event Log (SEL). We discuss our machine learning applications for anomaly detection in the remediation flow in Section IV, and for improving repair diagnosis in Section V. Section VI concludes with current and future areas of research for applying machine learning for hardware remediation at scale.

II. Hardware Remediation at Facebook

In this section, we explain the hardware remediation flow and the tools involved, and present how the results are being monitored.

A. Remediation Flow and Tooling

- ***MachineChecker***: Hardware remediation begins with failure detection. On each server, a tool called MachineChecker [7] runs a set of checks periodically to check for server failures. A non-exhaustive list of the server checks includes checks for host ping, out-of-band (OOB) ping, power status check, memory error check, sensor check, secure shell (SSH) access, network interface card (NIC) speed, dmesg check, S.M.A.R.T. check, and power supply check.
- ***FBAR***: Once a check fails, i.e. a potential hardware failure is detected, an *alert* is created with detailed failure information. Facebook Auto-Remediation (FBAR) [3], [4], a tool that automates remediation routines, then picks up the alert, processes the information logged in the alert, and executes the necessary remediation, e.g. reboot or custom-made scripts, if possible. Service owners can write customized remediation for different syndromes based on the specific needs of the service, and set proper rate limits for remediation to make sure there are always sufficient servers running to keep the service up.
- ***Cyborg***: When none of its remediations could fix the issue, FBAR passes the case to a repair engine named Cyborg [7]. In addition to FBAR's remediations, Cyborg is designed to execute low-level software fixes such as firmware updates and reimaging. If Cyborg's attempted fix also fails, it will create a repair ticket for a datacenter technician to manually debug the server and potentially carry out a physical hardware repair.

During the remediation flow, FBAR and Cyborg both run MachineChecker again to verify the failure. The entire detection and remediation flow for server failure is demonstrated in Fig. 1.

B. Monitoring the Auto-Remediation

We track all the failures through the remediation flow for evaluating the effectiveness of our remediations. Fig. 2 shows the remediation results of motherboard failures within two weeks in 2017. (Note: we have abstracted the count of failures to make the discussion agnostic of actual failure numbers). The percentages marked on each branch indicate the proportion of failures coming from the previous remediation stage. For all of the motherboard alerts, which were created once server check

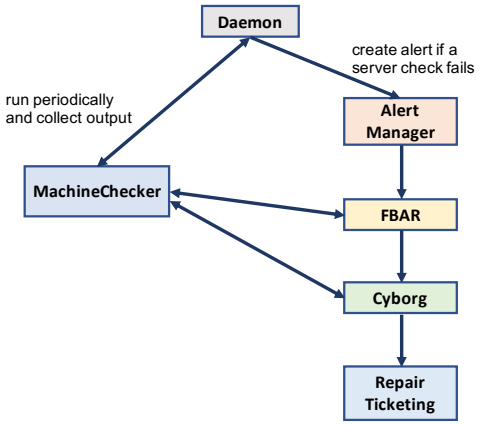

Fig. 1. The hardware failure detection and remediation flow.

failures were detected, 38.6% were cleared by FBAR's auto-remediation, 32.0% were closed as false alarms by FBAR, 10.9% fell into the other categories (e.g. remediation rate-limited by the service owner to maintain sufficient capacity for running the service, remediation blocked by an existing repair ticket created against the server, or exceptions caused by improper manual interactions during the remediation flow). Only 18.5% of the failures were sent to repair by FBAR. For the failures that FBAR decided to send to repair, Cyborg also attempted to remediate the issue with low-level system fixes and ran MachineChecker to validate if the issue had been resolved. Out of all the failures that were sent to Cyborg, 45.6% were successfully resolved by the low-level software fix attempt, 28.2% could not be remediated and ended up as repair tickets, and 26.2% fell into the other categories. Overall, only 5% of the motherboard alerts ended up as repair tickets.

III. Challenges for Hardware Remediations

While the above-mentioned remediation flow is designed for each hardware failure to be fixed either by a hardware repair (e.g. component swap/reseat) or a software repair (e.g. custom-made script, firmware update, reimage), we have observed that there are failures that fall out of the expected remediation flow. These affected servers were not remediated properly before they were released back to production, and some of them have a high likelihood of having the same failures in the near future. In the example of motherboard alerts shown in Fig. 2, 32% of the failures were closed as false alarms by FBAR because FBAR did not see the failures when running MachineChecker again. As we investigated further, we found that many of these false alarms were failures that only occurred when the servers were under heavy load, and they were cleared when the servers were drained to stop receiving production load. After the failures were closed as false alarms and the servers were released back to the production fleet, these failures were likely to be triggered again soon.

One example of these *transient failures* is PROCHOT [8], [9], a signal originally designed to indicate processor or memory overheating, and as a result throttles processor frequency.

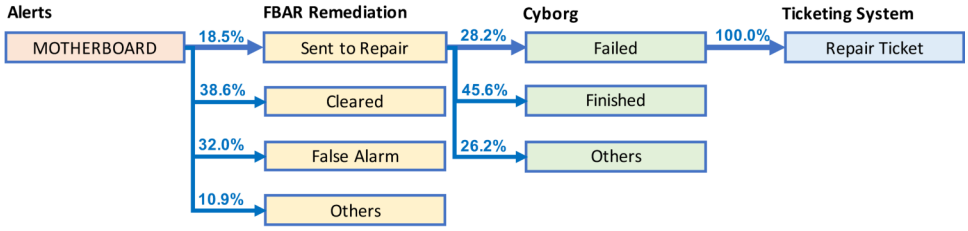

Fig. 2. Percentage of failures ending in each branch from the previous stage.

In practice, there are several other conditions that could trigger PROCHOT as well, but we will not discuss those situations due to the limited length of this paper. We present one scenario as a typical example for such issues at scale: In the processor/memory overheating scenario, PROCHOT is usually detected on a server running heavy workload, potentially because the thermal compound has worn out over time. After the server is drained before the remediation starts, the processor/memory is no longer generating excessive heat. When FBAR runs MachineChecker it does not detect PROCHOT anymore, and closes the failure as a false alarm. Out of all the servers that had PROCHOT failures that were closed as false alarms, 30% of them had detected PROCHOT again in less than 30 days. As PROCHOT failures are usually cleared during remediation, the servers would be stuck in this loop of processor/memory overheating, drained for remediation, released to production by FBAR as false alarms, and processor overheating again. This loop impacts the server availability. Other examples of transient failures include memory correctable/uncorrectable errors and network Cyclic Redundancy Check (CRC) errors. Similar to PROCHOT, these memory and network errors mostly occur only when the server is running heavy load, and therefore may not be detected when FBAR runs MachineChecker again to verify the failure.

To handle these transient failures effectively, we run a set of processor, memory, and network benchmarks to create synthetic loads for reproducing the transient failures during remediation. The benchmarks we included are CoreMark [10], stream-scaling [11], MPrime [12], stressapptest [13], SPEC CPU2006 [14], and iperf3 [15]. For memory correctable/uncorrectable errors, we found that stressapptest could effectively reproduce all detected errors within 3 hours for 256GB RAM. The required runtime increases for larger memory capacity. iperf3 has been very effective in reproducing all CRC errors detected by MachineChecker. For PROCHOT failures, we could only reproduce approximately 65% of the failures because not all PROCHOT failures are actually caused by the heavy load on the servers, and using MPrime to drive processor utilization to 100% was found to be the most effective approach for reproducing PROCHOT. We continuously optimize the collection and ordering of the benchmarks, in order to evolve our base benchmarks for the production scenarios of different failure modes.

In addition to transient errors, we have also found that some error reporting mechanisms could impact the server performance. One example is that when events are being logged to the System Event Log (SEL), System Management Interrupts (SMIs) are created for the system to enter System Management Mode (SMM), where all but one processor cores are temporarily disabled until the interrupt is cleared [16], [17]. Since memory correctable errors are more frequent and not as critical as the other events logged in SEL, we are switching to use Error Detection And Correction (EDAC) [18] for monitoring memory correctable errors.

IV. Machine Learning Enhanced Anomaly Detection

To monitor for anomalous behaviors in the remediation flow, we set up anomaly detection on the time series based on the percentages marked in Fig. 2 using common methodologies including Holt-Winters, exponential, normal distribution, and Gaussian mixture models [19]. An example of such anomaly would be a spike in the proportion of failures closed by FBAR as false alarms when we introduce a new type of server check in MachineChecker. In addition to pre-defined static thresholds, our anomaly detection system also checks if the data points are outside of the *predictive thresholds*, which are learned from the time series using machine learning. This anomaly detection is able to filter out the seasonality component of the data when detecting anomalies. Fig. 3 shows an example of the time series (blue line), static thresholds (horizontal red lines) and predictive thresholds (the band that adapts to the time series), and the violations (the vertical bands) that would trigger anomaly alerts, based on the server reboot rate for a specific service.

Deploying the automatic alerting based on the anomaly detections helps us scale the monitoring to hundreds of hardware health metrics, at a granular level (e.g. per service, server model, or datacenter), with immediate notifications. In addition to hardware remediation results, the automatic anomaly detection has been helping us detect early syndromes in hardware health metrics such as unexpected reboots, System Event Log (SEL) events, and hardware failure alerts, to triage the issues as soon as possible.

V. Machine Learning for Undiagnosed Repair Tickets

When FBAR and Cyborg both agree that a repair ticket is required for fixing the server, a repair ticket is created in the ticketing system. Most of the time, a repair ticket is created with an attached repair action for datacenter technicians to

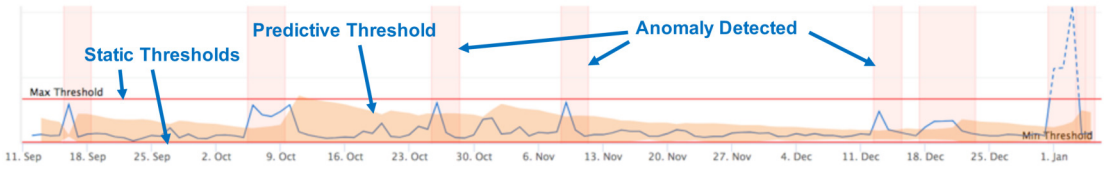

Fig. 3. Server reboot rate for a specific service.

follow and fix the issue. However, not all repair tickets are created with repair actions. Given the various failure modes and the continuous introduction of new hardware and software in our fleet, there are failures for which we do not have a corresponding repair action in the system, and would rely on manual debug of the issue. The repair tickets for these failures would be created without repair actions, and are therefore *undiagnosed tickets*.

Since undiagnosed tickets are created without repair actions, we have to start debugging the issue based on the logs collected in the remediation flow, i.e. logs from MachineChecker, FBAR, and Cyborg. This manual debugging process usually takes much more time than the repair for diagnosed tickets and often requires escalations for further investigation. To reduce the manual debugging time for the undiagnosed tickets, we deploy machine learning to learn from closed repair tickets, and recommend repair actions for new undiagnosed tickets based on how similar tickets have been closed in the past.

We use the fastText [6] library developed by Facebook AI Research (FAIR) for building the machine learning model. fastText trains and yields very competitive results compared to traditional text classification models in far less time, and its natural language processing ability fits our problem perfectly since the features that describe the tickets, i.e. the logs in MachineChecker, FBAR, and Cyborg, are largely unstructured raw text. Formulated in machine learning language, our problem becomes: *Using the raw text logs as features, learn from the closed tickets in the past and predict the repair actions for new undiagnosed tickets.* The well-supported machine learning framework [5] inside Facebook helped us quickly build and deploy the fastText model.

Overall, when we recommend up to 5 repair actions for each undiagnosed ticket, the recommendations could cover the correct repairs for between 50% to 80% of the undiagnosed tickets, which significantly reduces the manual debugging time required for these tickets. The prediction accuracy varies over time as we introduce new server checks and failure modes, so the prediction results are being continuously monitored and the model is retrained based on a new collection of closed repair tickets when the prediction accuracy drops. We also investigate the prediction accuracy for each failure mode to provide feedback to upstream tooling for improving the server checks and logging, e.g. emphasizing the keywords and relevant failure messages in the log for a specific failure mode.

VI. Conclusion

In this paper we present the current remediation flow and the difficulties in capturing transient failures such as PROCHOT and memory/network failures. New hardware and software introduce new failure types, which lead to new challenges in hardware remediation. While we are actively working on improving our remediation flow, we would also like to share the experience we had in hardware remediation at scale with researchers for new opportunities in this field.

Our machine learning deployment in hardware remediation also demonstrated promising results in providing insightful recommendations for debugging server issues. The natural language processing ability of fastText allows us to use unstructured raw text log as the input feature. The prediction results help us analyze the diagnostics and logging for different failure modes, and provide actionable feedback for improvements in upstream tooling. We focus on continuously optimizing our hardware remediation flow and using machine learning has improved our ability to operate at scale.

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] A. Power, "Making facebook self-healing," *https://www.facebook.com/notes/facebook-engineering/making-facebook-self-healing/10150275248698920/*, 2011.

[4] R. Komorn, "Making facebook self-healing: Automating proactive rack maintenance," *https://code.facebook.com/posts/629906427171799/making-facebook-self-healing-automating-proactive-rack-maintenance/*, 2016.

[5] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied machine learning at facebook: A datacenter infrastructure perspective," in *International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2018.

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

[7] R. Komorn, "Python in production engineering," *https://code.facebook.com/posts/1040181199381023/python-in-production-engineering/*, 2016.

[8] Intel, "Cpu monitoring with dts/peci," 2010.

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

[10] *https://www.eembc.org/coremark*.

[11] *https://github.com/gregs1104/stream-scaling*.

[12] *https://www.mersenne.org/download*.

[13] *https://github.com/stressapptest/stressapptest*.

[14] *https://www.spec.org/*.

[15] *https://iperf.fr*.

[16] Intel, "System event log (sel) troubleshooting guide," *Rev 3.2*, 2017.

[17] B. Delgado and K. L. Karavanic, "Performance implications of system management mode," in *IEEE International Symposium on Workload Characterization (IISWC)*, Sep. 2013.

[18] *http://bluesmoke.sourceforge.net/*.

[19] D. C. Montgomery, C. L. Jennings, and M. Kulahci, *Forecasting and Time Series Analysis*. McGraw-Hill, 1990.