Detecting silent data corruptions in the wild

Harish Dattatraya Dixit
Meta Platforms, Inc.
hdd@fb.com

Laura Boyle
Meta Platforms, Inc.
lauraboyle@fb.com

Gautham Vunnam
Meta Platforms, Inc.
gauthamvunnam@fb.com

Sneha Pendharkar
Meta Platforms, Inc.
spendharkar@fb.com

Matt Beadon
Meta Platforms, Inc.
mbeadon@fb.com

Sriram Sankar
Meta Platforms, Inc.
sriramsankar@fb.com

ABSTRACT

Silent Errors within hardware devices occur when an internal defect manifests in a part of the circuit which does not have check logic to detect the incorrect circuit operation. The results of such a defect can range from flipping a single bit in a single data value, up to causing the software to execute the wrong instructions. Silent data corruptions (SDC) in hardware impact computational integrity for large-scale applications. Manifestations of silent errors are accelerated by datapath variations, temperature variance, and age, among other silicon factors. These errors do not leave any record or trace in system logs. As a result, silent errors stay undetected within workloads, and their effects can propagate across several services, causing problems to appear in systems far removed from the original defect. In this paper, we describe testing strategies to detect silent data corruptions within a large scale infrastructure. Given the challenging nature of the problem, we experimented with different methods for detection and mitigation. We compare and contrast two such approaches - 1. Fleetscanner (out-of-production testing) and 2. Ripple (in-production testing). We evaluate the infrastructure tradeoffs associated with the silicon testing funnel across 3+ years of production experience.

KEYWORDS

silent data errors; data corruption; system reliability; hardware reliability; bitflips; large scale infrastructure

1 INTRODUCTION

Meta Infrastructure serves numerous applications like Facebook, Whatsapp, Instagram, Messenger and Oculus workloads. All these applications expect computational integrity and reliability from the underlying infrastructure. Silent data corruptions challenge these fundamental assumptions and impact applications at scale. In our previous paper [10], we shared insights on the impact of silent data corruptions with a case study within the Spark workloads at Meta. In the shared example, a simple computation like $(1.1)^{53}$ resulted in the wrong answer (0 instead of 156.24), resulting in missing rows within the database, which subsequently led to data loss for the application. Within Meta infrastructure, we have observed hundreds of instances of unique silent data corruptions. Meta runs several detection and testing frameworks, and we prevent the impact to our applications before the corruption can propagate. We have employed these detection strategies since 2019 within our fleet. Within this paper, we provide insights into the different strategies which majorly fall into 2 buckets: 1. Out-of-production testing and 2. In-production testing. We summarize the tradeoffs and test metrics associated with different stages within the silicon lifecycle.

The paper is structured in the following way - Section 2 provides insights on related work within this domain. Section 3 dives deep into the testing philosophies at different stages within the silicon lifecycle. Section 4 provides the motivation for fleetwide testing. Section 5 elaborates on the infrastructure testing strategies employed at Meta by exploring the in-production and out-of-production testing mechanisms. Section 6 provides insights into the results associated with the different strategies and evaluates tradeoffs associated with them. Section 7 concludes the paper.

2 RELATED WORK

There has been recent interest in the area of silent data corruptions [10], [14], [28]. Prior studies [6], [22], [7] within this domain focused on the soft errors induced due to cosmic rays. Fault injection studies [17], [8], [11] focused on fault modeling using soft error occurrence rates which were modeled at one fault in a million silicon devices. Meta published one of the first studies on large scale impact of silent errors [10], and showed that the SDC occurrence rate of one in thousand silicon devices is reflective of fundamental silicon challenges, and not limited to particle effects or cosmic rays. Google also published their observations [14], where mercurial cores were identified to disobey the fundamental rules of computation and produce erroneous results. This is an industry wide problem. At OCP 2021 [3], a panel of experts [4] from both industry and academia within the silent error domain gathered to discuss the strategies for the domain moving forward. More research and articles [2], [1], [13], [21], [29] establish the importance of this domain. The research focus for industry and academia has strongly been on identifying strategies and mitigating silent data corruptions not only in CPUs but also in all silicon devices.

3 SILICON TESTING FUNNEL

Before a silicon device reaches the Meta infrastructure fleet, the silicon device goes through different stages as part of the silicon development process. In this section, we will not go into elaborate details regarding the silicon development process. We are comparing the testing strategies employed at different stages to understand the cost associated with testing at fleetwide scale, and why that can be challenging. The **silicon testing funnel** in figure 1 provides a high level comparison for different stages within this section. Following subsections provide primitive descriptions of the different stages. It is to be noted that each stage is a dedicated research topic on its own, and the testing model varies for hyperscalars versus enterprise scale companies. In this paper, we focus on three important parameters: testing volume, test time, and the impact of a fault at that stage.

3.1 Design and verification

For any silicon device, once the architectural requirements are finalized, the silicon design and development process is initiated. Testing is usually limited to a few design models of the device, and simulations and emulations are used to test different features. The device is tested regularly with implementation of novel features. Test iterations are implemented on a daily basis. The cost of testing is low relative to the other stages, and the testing is repeated using different silicon variation models. Design iteration at this stage is faster than any other stage in the process. Faults can be identified based on internal states that are not visible in later stages of the development cycle. The test cost increases slowly with placement of standard cells for ensuring that the device meets the frequency and clock requirements, and also with the addition of different physical characteristics associated with the materials as part of the physical design of the device. There is plenty of available industry research on testing optimizations within these stages [19], [12], [20], [5], [16]. The testing process here lasts usually for many months to a couple of years depending on the chip and the development stages employed.

3.2 Post silicon validation

At this stage, numerous device samples are available for validation. Using the test modes available within the design of the device, the design is validated for different features. The number of device variations has grown from models in the previous stage to actual physical devices exhibiting manufacturing variance. Significant fabrication costs have been incurred before obtaining the samples, and a device fault at this stage has a higher impact since it typically results in a re-spin for the device. Additionally, there is a larger test cost associated with precise and expensive instrumentation for multiple devices under test. At the end of this validation phase, the silicon device can be considered as approved for mass production. The testing process here typically lasts for a few weeks to a few months.

3.3 Manufacturer testing

At mass production, every device is subjected to automated test patterns using advanced fixtures. Based on the results of the testing patterns, the devices are binned into different performance groups to account for manufacturing variations. As millions of devices are tested and binned, time allocated for testing has a direct impact on manufacturing throughput. The testing volume has increased from a few devices in the previous step to millions of devices, and test cost scales per device. Faults are expensive at this stage, as they typically result in respin or remanufacturing of the device.

3.4 Integrator testing

After the manufacturing and testing steps, the devices are shipped to an end customer. A large scale infrastructure operator typically utilizes an integrator to coordinate the process of rack design, rack integration and server installation. The integrator facility typically conducts testing for multiple sets of racks at once. The complexity of testing has now increased from one device type to multiple types of devices working together in cohesion. The test cost increases from a single device to testing for multiple configurations and combinations of multiple devices. An integrator typically tests the racks for a few days to a week. Any faults require reassembly of racks and reintegration.

3.5 Infrastructure intake testing

As part of the rack intake process, infrastructure teams typically conduct an intake test where the entire rack received from the integrator is wired together with datacenter networks within the designated locations. Subsequently, test applications are executed on the device before executing actual production workloads. In testing terms, this is referred to as infrastructure burn-in testing. Tests are executed for a few hours to a couple of days. There are hundreds of racks containing a large number of complex devices that are now paired with complex software application tools and operating systems. The testing complexity has increased significantly relative to previous test iterations. A fault is challenging to diagnose due to the larger source of fault domain.

3.6 Infrastructure fleet testing

Historically, the testing practices concluded at infrastructure burn-in testing. The device is expected to work for the rest of its lifecycle, and any faults if observed would be captured using system health metrics and reliability-availability-serviceability features built into devices, which allow for collecting system health signals.

However, with silent data corruptions, there is no symptom or signal that indicates there is a fault with a device. Hence without running dedicated test patterns to detect and triage silent data corruptions, it is almost impossible to protect an infrastructure application from corruption. As a result, it has become imperative to test periodically within the fleet using different strategies. At this point within the lifecycle, the device is already part of a rack and serving production workloads. The testing cost is high relative to other stages, as it requires complex orchestration and scheduling while ensuring that the workloads are drained and undrained effectively. Tests are designed to run in complex multi-configuration, multi-workload environments. Any time spent in creating test environments and running the tests is time taken away from server running production workloads.

A fault within a production fleet is expensive to triage and root-cause as the fault domains have evolved to be more complex with ever changing software and hardware configurations. As a result, advanced strategies are required to detect silent data corruptions with expensive infrastructure tradeoffs.

4 WHY IS THIS A HARD PROBLEM ?

With millions of devices, within a large scale infrastructure, there is a probability of error propagation to the applications. With an occurrence rate of one fault within a thousand devices, silent data corruptions have the ability to impact numerous applications. Until the application exhibits noticeable difference at higher level metrics, the corruption continues to propagate and produce erroneous

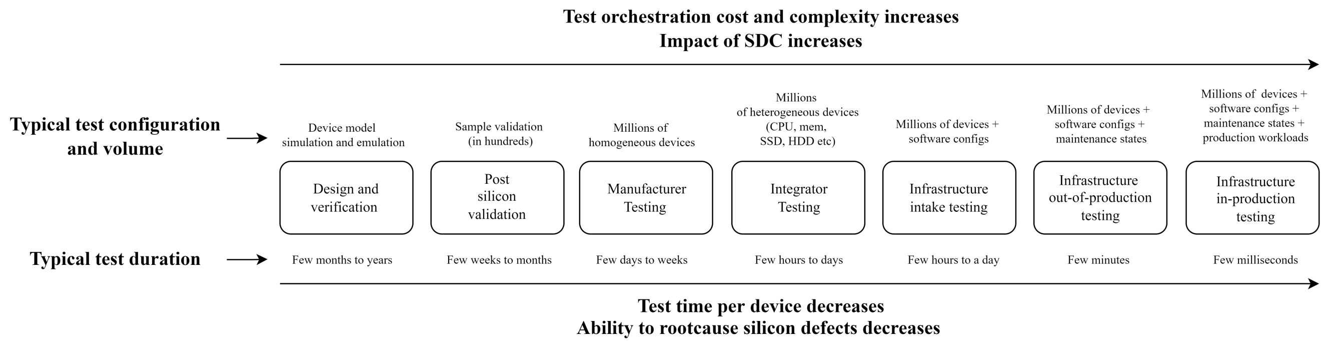
Test orchestration cost and complexity increases
Impact of SDC increases
Typical test configuration and volume
Device model simulation and emulation
Sample validation (in hundreds)
Millions of homogeneous devices
Millions of heterogeneous devices (CPU, mem, SSD, HDD etc)
Millions of devices + software configs
Millions of devices + software configs + maintenance states
Millions of devices + software configs + maintenance states + production workloads
Design and verification
Post silicon validation
Manufacturer Testing
Integrator Testing
Infrastructure intake testing
Infrastructure out-of-production testing
Infrastructure in-production testing
Typical test duration
Few months to years
Few weeks to months
Few days to weeks
Few hours to days
Few hours to a day
Few minutes
Few milliseconds
Test time per device decreases
Ability to rootcause silicon defects decreases

Figure 1: Silicon testing funnel

computations. This scale of fault propagation presents a significant challenge to a reliable infrastructure. We have observed that faults can be due to a variety of sources or accelerants. We categorize these into four major sections. We turn to periodic testing with dynamic control of tests to triage corruptions and protect applications. These observations are based on testing and aggregating samples for ≈3 years within Meta infrastructure. We are using an example product computation of *3 times 5* to demonstrate our observations:

- **Data randomization:** We observe that the corruptions are data dependent by nature. For example, we observe numerous instances where the majority of the computations would be fine within a corrupt CPU but a smaller subset would always produce faulty computations due to certain bit pattern representation. For example, we may observe that 3 times 5 is 15, but 3 times 4 is evaluated to 10, and thus until and unless 3 times 4 is verified specifically, we cannot confirm computation accuracy within the device for that specific computation. This leads to a fairly large state space for testing.
- **Electrical variations:** In a large scale infrastructure, with varying nature of workloads and scheduling algorithms, the devices undergo a variety of operating frequency (f), voltage (V) and current (I) fluctuations. We observe that changing operating voltages, frequency and current associated with the device can lead to acceleration of occurrence of erroneous results on faulty devices. While the result would be accurate with one particular set of f, V and I, the result may not hold true for all the possible operating points. This leads to a multi-variate state space. For example, we may observe that 3 times 5 is 15 in some operating conditions, but repeating the same calculation may not always result in 15 under all operating conditions.
- **Environmental variations:** We observe that variations in location dependent parameters also accelerate occurrence of silent data corruptions. It is well documented that temperature [15], [30], [31], [27], and humidity [26], [9], [25] have a direct impact on the voltage and frequency parameters associated with the device due to device physics. In a large-scale datacenter, while the temperature and humidity variations are controlled to be minimal, there can be occurrences of hot-spots within specific server locations due to the nature of repeated workloads on that server and neighboring servers. Also the seasonal trends associated with a datacenter location can create hotspots across data halls within a datacenter. For example, we may observe 3 times 5 is 15 in datacenter A, but repeated computations can result in 3 times 5 computing to 12 in datacenter B.
- **Lifecycle variations:** We observe that silicon continually changes in performance and reliability with time. This has been well documented in bathtub curve failure modeling across the literature [18], [24], [23]. However, with silent data corruptions we observe that certain failures can manifest earlier than the traditional bathtub curve predictions based on device usage. As a result, a computation producing a correct result today provides no guarantee that the computation will produce a correct result tomorrow. In one specific experiment, we repeated the exact same computation sequence on the device once every day for a period of 6 months and the device failed after 6 months indicating degradation with time for that computation. In essence, a computation like 3 times 5 equals 15 can provide a correct result today but tomorrow may result in 3 times 5 being evaluated to an incorrect value.

As a result of all four observations, we conclude that the only way to measurably protect the fleet against silent data corruptions is to repeatedly test the infrastructure with ever improving test routines and advanced test pattern generation. By building engineering capability in finding hidden patterns across hundreds of failures, and feeding the insights into optimizations for test runtimes, testing policies and architectures, the fleet resiliency can be improved. Sharing these insights with vendors, industry and academia on a periodic basis also enables the collective research growth within this domain.

5 INFRASTRUCTURE TESTING

As part of Meta infrastructure, we have implemented 2 broad categories of testing at fleet scale. When a fleet is made up of millions of machines spread across multiple regions and fault domains, it is important that testing is efficient and tactical. The 2 broad categories of testing are:

- Out-of-production testing.
- In-production testing.

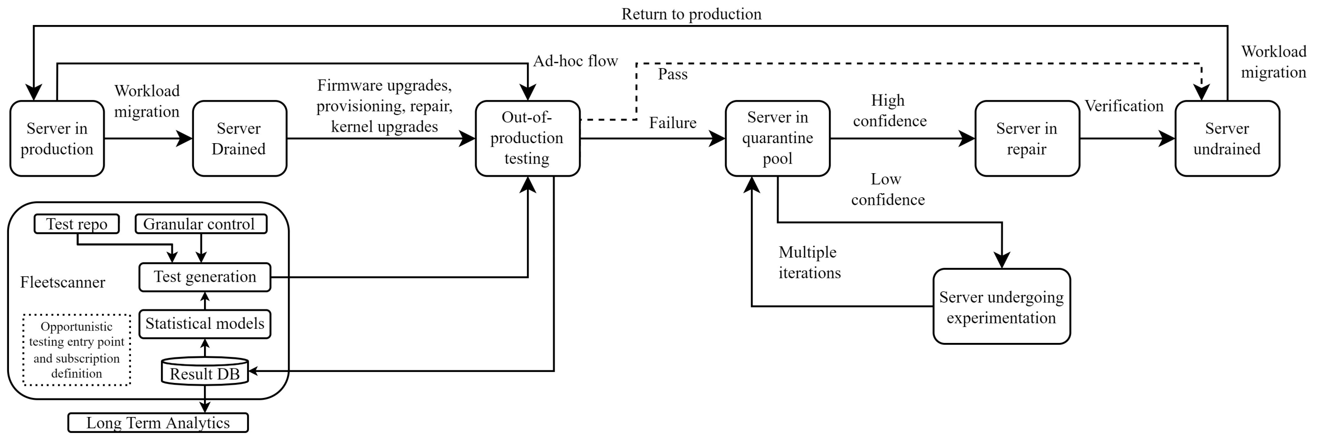

Figure 2: Out-of-production testing

5.1 Out-of-production testing

Out-of-production testing refers to the ability to subject machines to known patterns of inputs, and comparison of its expected outputs with known reference values across millions of different execution paths. Tests are executed across different temperatures, voltages, machine types, regions etc. while the machine is idle and not executing production workloads.

The test patterns are generated based on our production experience and understanding of silicon architectures as well as obtained from silicon vendors. The instructions are carefully crafted in sequences to match known defects or target a variety of defect families using numerous state search policies within the testing state space.

Typically in a large scale infrastructure, there are always sets of machines going through maintenance. Before any of these maintenance are started, the workload is safely migrated off the machine, typically referred to as a draining phase. Post a successful drain phase, we observe one or many of the following maintenance:

- **Firmware upgrades:** There are numerous devices within a given server and there may be new firmware available on at least one component. These component firmware upgrades are required to keep the fleet up to date for fixing firmware bugs as well as security vulnerabilities.
- **Kernel upgrades:** Similar to component level upgrades, the kernel on a particular server is upgraded at a regular cadence, and these provide numerous application and security updates for the entire fleet.
- **Provisioning:** While the above two mechanisms refer to the process of upgrading a server. Provisioning refers to the process of preparing the server for workloads with installation of operating systems, drivers and application-specific recipes. There could also be instances of reprovisioning where-in within a dynamic fleet a server is moved from one type of workload to another.
- **Repair:** Each server that encounters a known fault or triggers a match to a failing signature ends up in a repair queue. Within the repair queue, based on the diagnoses associated with the device, a soft repair (without replacing hardware components) is conducted or a component swap is executed. This enables faulty servers to return back to production.

Any machine exiting the maintenance phase is then undrained to make the machine available to production workloads. With these maintenances already available within the fleet, we at Meta developed and integrated a tool called *Fleetscanner*. Fleetscanner opportunistically identifies machines entering and exiting maintenance states and schedules the machines to undergo silent data corruption testing. The architecture for fleetscanner and its integration at a very high level is represented in Figure 2. In all the cases, based on the time available and the type of machine identified, fleetscanner runs optimized versions of tests and provides a snapshot for the device's response to sensitive architectural codepaths, and verifies the computations to be accurate. A number of machine specific parameters are captured at this instant to enable understanding the conditions that result in device failures. Any machine identified to fail for silent data corruption routines are routed to the quarantine pool for further investigation and test refinements.

The four out-of-production workflows are independent complex systems with orchestration across millions of machines, and fleetscanner enables a seamless methodology to orchestrate silent data corruption tests within a large fleet by integrating with all the workflows. It is extremely important to minimize the time spent in drain and undrain phases and piggyback on existing maintenance. It is also important to minimize disruption to existing workflows with significant time overheads and orchestration complexities. This allows the testing cost to be noticeable yet minimal per machine while providing reasonable protection against application corruptions.

5.2 In-production testing

While out-of-production testing allows for testing opportunistically when machines transition across states, there are many instances within our fleet where a novel signature identified must be immediately scaled to the entire fleet. Waiting for out-of-production scanning opportunities and subsequently ramping up fleetwide

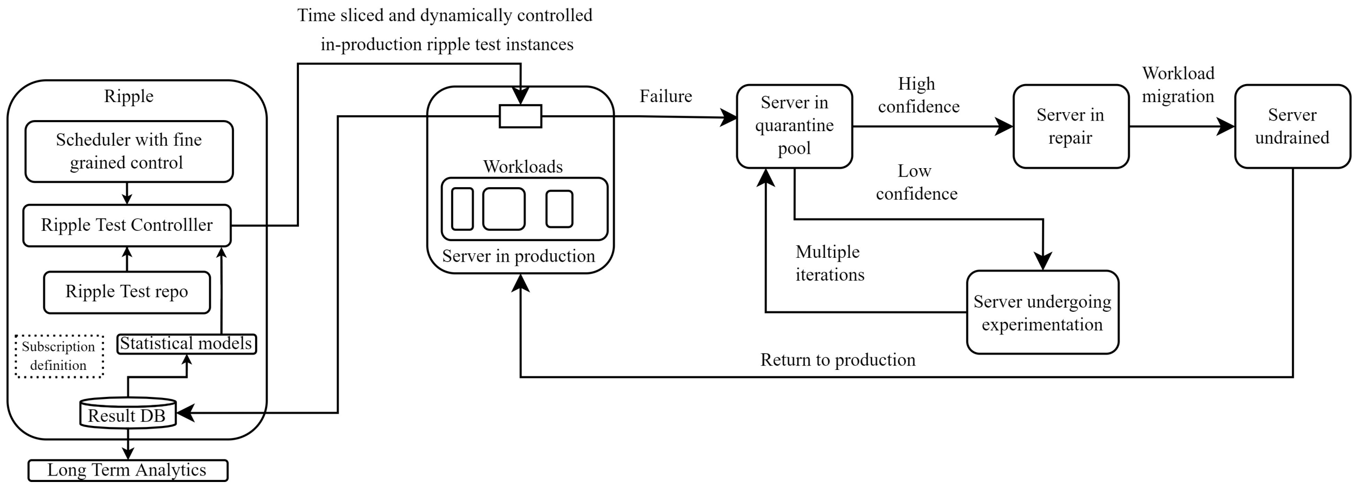

Figure 3: In-production testing

coverage is slow. While fleetscanner has its own benefits in implementing longer running tests, with test runtime in minutes, we observe a requirement for an alternate light-weight method to test within the fleet while the machines are running production workloads. This is tricky to achieve without a granular understanding of the workload and modulation of testing routines with the workloads.

At Meta, we have implemented a testing methodology called *Ripple* which co-locates with the workload, and executes test instructions for millisecond level intervals. The architecture for ripple testing is described in figure 3. Test sequences used in out-of-production testing are modified specifically to be conducive to run through Ripple. Typically intrusive tests are used as part of infrastructure burn-in testing; changing them to run in ripple mode requires fine-tuning of tests along with test coverage tradeoff decisions. The test orchestration is implemented with extreme care as any variation within the test could immediately affect the application. This test is *live* within the entire fleet and provides granular control on test subsets, cores to test, type of workloads to co-locate with as well as in scaling the test up and down to multiple sets of cores based on the workload.

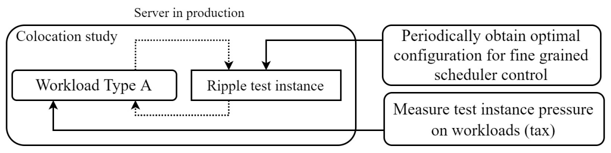
Server in production
Colocation study
Workload Type A
Ripple test instance
Periodically obtain optimal configuration for fine grained scheduler control
Measure test instance pressure on workloads (tax)

Figure 4: Shadow testing

5.2.1 Shadow testing: We implemented and fully rolled out ripple across multiple sets of workloads. We have also crafted the *ripple* test architecture to be able to have safeguards to prevent fleetwide fallout in case of a test defect. We implemented shadow testing by running a wide variety of workloads with A/B testing for different instruction sequences with different seasonality and across different workloads. A major challenge in shadow testing is enabling colocation. Based on the scaling of the workload, the testing mechanism had to descale. For each type of workload at Meta, we identified an evaluation process for the scaling factor. Based on instrumentation, we established the footprint tax associated with the test. Each workload type's colocation study provides the tax, and the goal for the tool is to minimize its tax below a certain threshold. A simplified version of the architecture is referenced in Figure 4. With repeated sets of experimentation, we established control structures and safeguards for enabling different options for different workloads, and then scaled the solution to the entire fleet.

5.2.2 Always-on in production: This mechanism is *always on*. Only the scale at which it is operating is dynamically controlled through configurations within the fleet. This methodology is powerful in finding defects which require thousands of iterations of the same data inputs, as well as in identifying devices undergoing degradation. A novel signature identified within the fleet for a device could be scaled to the entire fleet with satisfiable test randomization and defect pattern matching within a couple of weeks. This methodology is also uniquely effective in identifying silicon transition defectsl. In the comparison of results below, we share the statistical value of this method in identifying silent data corruptions for a large subset of defects for one CPU defect family.

5.2.3 Recommendations to the industry: We have recommended this mechanism of testing as an important evolution for silent data corruption tools from large scale internal studies. Based on our findings and data-sharing practices, our vendors have enabled modes within their tests that make them suitable for ripple test. Vendors have published white-papers [29] around a testing methodology called *trickle testing* which derives from the in-production testing flow and its fleetwide success within Meta. We would like to thank our industry partners in taking lessons from our fleetwide studies and making them available to industry and academia.

Metric	Fleetscanner	Ripple
Total tests executed	≈68 million (lifetime)	≈2.5 billion (per month)
Testing time	≈4 billion fleet seconds (lifetime)	≈100 million fleet seconds (per month)
Performance aware	No	Yes
Unique SDC coverage	23 percent	7 percent
Time to equivalent SDC coverage	≈6 months (70 percent)	≈15 days (70 percent)

Table 1: Comparison of Fleetscanner and Ripple

6 RESULTS

We are sharing results below from around 3 years of data aggregation regarding the effectiveness of these two different testing strategies. The result dataset here in comparison is for a large subset of defects for one CPU defect family, with tests being executed on a significantly large percentage of the fleet. Equivalent coverage within this section refers to the ability to detect the same set of failures through different methods of testing.

6.1 Out-of-production testing (Fleetscanner)

Using fleetscanner as part of the fleetwide out-of-production detection across millions of machines, we have obtained a total of around 68 million unique test iterations within the lifetime of the tool. Test runtimes vary based on type of maintenance available and the type of integration sequence in place. In total, we have tested for around 4 billion test seconds. The tests are inherently intrusive in nature and hence conducted out-of-production. We observe that fleetscanner provides 93 percent coverage among all detected silent data corruptions for defect family under study. Fleetscanner also achieves 23 percent unique coverage which is not reachable by ripple. Based on different cadences of maintenance, we observe that some machines may undergo significantly more testing than others. However, fleetscanner achieves approximately full fleetwide coverage within five months to six months based on average deployments and maintenance.

From a test cost perspective, this is expensive. The fleet spends a significant compute time executing tests. However, it is our observation that this is an important cost with increasing sightings of silent data corruptions.

6.2 In-production testing (Ripple)

In comparison to the out-of-production testing, the ripple test framework provides its own set of unique coverage metrics. Since ripple is always-on, we are able to achieve around 2.5 billion unique test instances any given month because of the non-intrusive nature of the tests and granular control and co-location. Test runtimes vary based on workload intensity and the subscription configurations. However, given that each test is limited to hundreds of milliseconds at best, we obtain a total test runtime of around 100 million fleet seconds every month.

Ripple testing offers a unique coverage of 7 percent among the set of all detectable machines. We observe that this coverage is impossible to achieve with fleetscanner due to the inherent nature of testing and the underlying silicon defects. To elaborate, certain failures are detected via ripple due to frequent transitions of test instructions along with workloads, and are not detected with continuous long running tests. While 7 percent coverage is unique to ripple, it can detect 70 percent within the 93 percent coverage that fleetscanner provides within 15 days. While ripple can achieve this coverage within 15 days, fleetscanner requires around 5 to 6 months. This scaling effect makes ripple a powerful framework within a large fleet.

6.3 Comparison

A comparison of the numbers presented in the above 2 sections is provided in table 1. From the table, we observe that for defect family under study within this paper, 70 percent of the common coverage detection could be completed within 15 days using ripple. Fleetscanner ramps up to the remaining 23 percent of the coverage over 6 months. A unique 7 percent coverage is through repeated ripple instances within the fleet. Ripple provides a total coverage of 77 percent with significantly lower total test runtimes than fleetscanner. There are benefits to both models of testing. We also consistently revisit and evaluate these coverage metrics to inform and update our fleetwide testing strategies around test vectors, test cadences and test runtimes. We observe that with different types of defects, the coverage split varies.

Historically, each CPU only went through a few hours of testing as part of infrastructure burn-in tests. Further testing was typically conducted via sampling. We observe that novel detection approaches are required for application health and fleet resiliency. In this paper, we demonstrate the ability to test at scale and get through billions of fleet seconds of testing every month across a large fleet consistently. These novel techniques enable us to detect silent data corruptions and mitigate them at scale.

7 CONCLUSIONS

Detecting silent data corruption is a challenging problem for large-scale infrastructures. Applications show significant sensitivity to these problems and can be exposed to such corruptions for months without accelerated detection mechanisms. It can also result in data loss and require months to debug and resolve software level residue of silent corruptions. This research shows novel techniques resulting from years of experience observing silent corruptions and in categorizing their occurrence patterns and faster time to detection. Impact of silent data corruption can have a cascading effect on applications and we have to address this as a critical problem. As a result, detecting these at scale as quickly as possible is important towards enabling a safer and reliable fleet.

Acknowledgement The authors would like to thank Aslan Bakirov, Melita Mihaljevic, Thiara Ortiz, Tejasvi Chakravarthy, Manish Modi, Vijay Rao, T.S. Khurana, Nishant Yadav, Aravind Anbudurai and other infrastructure engineers for their inputs in the implementation of solutions and valuable technical suggestions. The authors would also like to thank Bill Holland, Chris BeSerra, Marty Humphrey, Fred Lin and Daniel Moore for their thoughtful comments, feedback and efforts towards improving this paper.

REFERENCES

[1] 2021. *2021 Program – International Test Conference.* http://www.itctestweek.org/wp-content/uploads/2021/10/2021-Final-Program.pdf

[2] 2021. *2021 Program – Silicon Errors in Logic – System Effects.* https://selse.org/previous-workshops/2021-archive/2021-program/

[3] 2021. *Open Compute Project. 2021.* https://www.opencompute.org/summit/global-summit

[4] 2021. *PANEL OCP HW Operation at Scale Reliability to Address Silent Data Corruptions.* https://www.youtube.com/watch?v=3yhg4Gt8M_E

[5] M.S. Abadir, J. Ferguson, and T.E. Kirkland. 1988. Logic design verification via test generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 7, 1 (1988), 138–148. https://doi.org/10.1109/43.3141

[6] R. C. Baumann. 2005. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability* 5, 3 (2005), 305–316. https://doi.org/10.1109/TDMR.2005.853449

[7] D. Bossen. 2002. CMOS Soft Errors and Server Design - IRPS. Tutorial Notes - Reliability Fundamentals. (2002).

[8] G. C. Cardarilli, F. Kaddour, A. Leandri, M. Ottavi, S. Pontarelli, and R. Velazco. 2002. Bit flip injection in processor-based architectures: a case study. In *Proceedings of the Eighth IEEE International On-Line Testing Workshop (IOLTW 2002).* 117–127. https://doi.org/10.1109/OLT.2002.1030194

[9] Davide Cimmino and Sergio Ferrero. 2020. High-voltage temperature humidity bias test (HV-THB): Overview of current test methodologies and reliability performances. *Electronics* 9, 11 (2020), 1884.

[10] Harish Dattatraya Dixit, Sneha Pendharkar, Matt Beadon, Chris Mason, Tejasvi Chakravarthy, Bharath Muthiah, and Sriram Sankar. 2021. Silent Data Corruptions at Scale. *arXiv preprint arXiv:2102.11245* (2021).

[11] James Elliott, Frank Mueller, Frank Stoyanov, and Clayton Webster. 2013. *Quantifying the impact of single bit flips on floating point arithmetic.* Technical Report. North Carolina State University. Dept. of Computer Science.

[12] P.E. Gronowski, W.J. Bowhill, R.P. Preston, M.K. Gowan, and R.L. Allmon. 1998. High-performance microprocessor design. *IEEE Journal of Solid-State Circuits* 33, 5 (1998), 676–686. https://doi.org/10.1109/4.668981

[13] Nicole. Hemsoth. 2021. *How Facebook Architects Around Silent Data Corruption.* https://www.nextplatform.com/2021/03/01/facebook-architects-around-silent-data-corruption/

[14] Peter H Hochschild, Paul Turner, Jeffrey C Mogul, Rama Govindaraju, Parthasarathy Ranganathan, David E Culler, and Amin Vahdat. 2021. Cores that don't count. In *Proceedings of the Workshop on Hot Topics in Operating Systems.* 9–16.

[15] Eric Humenay, David Tarjan, and Kevin Skadron. 2006. *Impact of parameter variations on multi-core chips.* Technical Report. VIRGINIA UNIV CHARLOTTESVILLE DEPT OF COMPUTER SCIENCE.

[16] Warren A Hunt. 1989. Microprocessor design verification. *Journal of automated reasoning* 5, 4 (1989), 429–460.

[17] X. Iturbe, B. Venu, and E. Ozer. 2016. Soft error vulnerability assessment of the real-time safety-related ARM Cortex-R5 CPU. In *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT).* 91–96. https://doi.org/10.1109/DFT.2016.7684076

[18] Georgia-Ann Klutke, Peter C Kiessler, and Martin A Wortman. 2003. A critical look at the bathtub curve. *IEEE Transactions on reliability* 52, 1 (2003), 125–129.

[19] William KC Lam. 2005. *Hardware design verification: simulation and formal method-based approaches.* Prentice Hall Professional Technical Reference.

[20] Erik Larsson, Zebo Peng, and Krishnendu Chakrabarty. 2002. An integrated framework for the design and optimization of SOC test solutions. In *SOC (System-on-a-Chip) Testing for Plug and Play Test Automation.* Springer, 21–36.

[21] John Markoff. 2022. *Tiny Chips, Big Headaches.* https://www.nytimes.com/2022/02/07/technology/computer-chips-errors.html

[22] S. S. Mukherjee, J. Emer, and S. K. Reinhardt. 2005. The soft error problem: an architectural perspective. In *11th International Symposium on High-Performance Computer Architecture.* 243–247. https://doi.org/10.1109/HPCA.2005.37

[23] Michael G Pecht, Riko Radojcic, and Gopal Rao. 2017. *Guidebook for managing silicon chip reliability.* CRC press.

[24] William J Roesch. 2012. Using a new bathtub curve to correlate quality and reliability. *Microelectronics Reliability* 52, 12 (2012), 2864–2869.

[25] N.L. Sbar and R.P. Kozakiewicz. 1979. New acceleration factors for temperature, humidity, bias testing. *IEEE Transactions on Electron Devices* 26, 1 (1979), 56–71. https://doi.org/10.1109/T-ED.1979.19380

[26] Dieter K Schroder. 2015. *Semiconductor material and device characterization.* John Wiley & Sons.

[27] Dieter K Schroder and Jeff A Babcock. 2003. Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing. *Journal of applied Physics* 94, 1 (2003), 1–18.

[28] Kostya Serebryany, Maxim Lifantsev, Konstantin Shtoyk, Doug Kwan, and Peter Hochschild. 2021. SiliFuzz: Fuzzing CPUs by proxy. *CoRR* abs/2110.11519 (2021). arXiv:2110.11519 https://arxiv.org/abs/2110.11519

[29] Arjan. Van De Ven. 2021. *Intel Data Center Diagnostic Tool Now Available for Deployment.* https://www.intel.com/content/www/us/en/developer/articles/tool/intel-data-center-diagnostic-tool-now-available.html

[30] David Wolpert and Paul Ampadu. 2012. Temperature effects in semiconductors. In *Managing temperature effects in nanoscale adaptive systems.* Springer, 15–33.

[31] Christian Zorn and Nando Kaminski. 2014. Temperature humidity bias (THB) testing on IGBT modules at high bias levels. In *CIPS 2014; 8th International Conference on Integrated Power Electronics Systems.* VDE, 1–7.