Multi-stage Approach for Identifying Defective Hardware in Frontier

Nick Hagerty
National Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, United States of America
hagertynl@ornl.gov

Andy Warner
Hewlett Packard Enterprise
Bloomington, MN, United States of America
andy.warner@hpe.com

Jordan Webb
National Center for Computational Sciences
Oak Ridge National Laboratory
Oak Ridge, United States of America
webbjw@ornl.gov

***Abstract*—In June 2022, the long-awaited exaflop compute barrier (1 quintillion floating-point operations per second) was surpassed on the TOP500 list by Frontier, an HPE Cray EX supercomputer at Oak Ridge National Laboratory (ORNL). Drawing peak power of 21.1 MW, Frontier demonstrated 1.1 exaflops/second of computational capability, much of which is supplied by more than 37,000 AMD Instinct MI250X graphics processing units (GPUs). With a single GPU drawing up to 560W thermal design power (TDP), each AMD MI250X draws 2x more power under load than the NVIDIA V100 GPUs used in Frontier's predecessor at ORNL, the 200 petaflop IBM POWER9 supercomputer, Summit. There are many other major technological advances in the memory, compute, power, and infrastructure of Frontier that are new to production environments. Frontier's mission to enable ground-breaking research in U.S. energy, economic and national security sectors is fulfilled through leadership-class workloads, which are workloads that demand greater than 20% of the supercomputer. These large workloads are vulnerable to defective and failing computing hardware. The rate of failing hardware is quantified through the mean time between failures (MTBF), which is the length of time between a hardware-level failure anywhere in the system. In this work, we describe the multi-staged approaches to stabilizing and maintaining the functionality of the hardware on Frontier. There are three strategies discussed; the first two utilize leadership-class tests to target improving the MTBF of Frontier, the third utilizes single-node validation to efficiently identify individual instances of defective hardware in Frontier. We provide summarized data from each of the three strategies, then classify the diverse set of failures and discuss trends in defective hardware, before discussing several key challenges to identifying defective hardware and improving the MTBF of Frontier.**

***Index Terms*—hardware, reliability, leadership-class**

I. Introduction

The Oak Ridge Leadership Computing Facility (OLCF) is a user facility within Oak Ridge National Laboratory's National Center for Computational Sciences tasked with providing users with the computational resources needed to tackle the most challenging scientific problems in the world. Frontier is the latest deployment under OLCF, and is the first supercomputer in the world to break the exascale barrier (1 quintillion floating-point operations per second) on the High-Performance Linpack benchmark [2], showcasing the achievement in the June 2022 edition of the TOP500 list[1] [1], [5]. Frontier was deployed to enable leadership-class workloads; that is, workloads that consume more than 20% of the supercomputer. Large leadership-class computation is not possible if the more than 37,000 AMD Instinct GPUs and 9,000 AMD EPYC CPUs in Frontier are not working together. Defective hardware residing within Frontier jeopardizes the fulfillment of leadership-class computation by synthetically lowering the mean time between failures (MTBF). The MTBF quantifies the length of time between hardware-level failures in the system, which shares a direct relationship with the success of leadership-class workloads; each hardware failure may cause workloads currently computing to fail.

Frontier's AMD Instinct MI250X GPUs draw a thermal design power (TDP) of 560W, which is double the TDP of the NVIDIA V100 GPUs utilized by Frontier's predecessor at ORNL, the 200 petaflop IBM POWER9 supercomputer, Summit. In addition to the large difference in power, the MI250X also packs 64 GB of high-bandwidth memory (HBM), 4 times more than the V100. The MI250X also decreases the physical size of the process nodes by 75% with 6 nanometer (nm) FinFET process technology, while the V100 utilizes 12 nm FFN technology. These factors contribute to a supercomputer that is more complex than previous generations, and as such,

[1]TOP500 June 2022: https://www.top500.org/lists/top500/2022/06

demands unique approaches for deploying and maintaining the system.

In this work, we discuss a multi-staged approach for identifying defective hardware in Frontier, building on previous work which discussed developing an effective periodic node screen for Frontier [10]. This multi-staged approach not only identifies defective hardware, but begins to uncover key trends throughout the lifetime of Frontier. Summarized data from each stage is presented, then two key observations from this data are presented; (1) there are discernible seasons of classifiable failures and (2) there is a constant population of "bad actor" nodes. Following analysis, key challenges are discussed. There are several key challenges to multi-staged efforts for identifying defective hardware discussed in this work; (1) discovering trends in failures, (2) balancing resources allocated to testing with resources allocated to production, (3) failure modes may be triggered by bad hardware OR buggy code, and (4) maintaining a relevant set of tests.

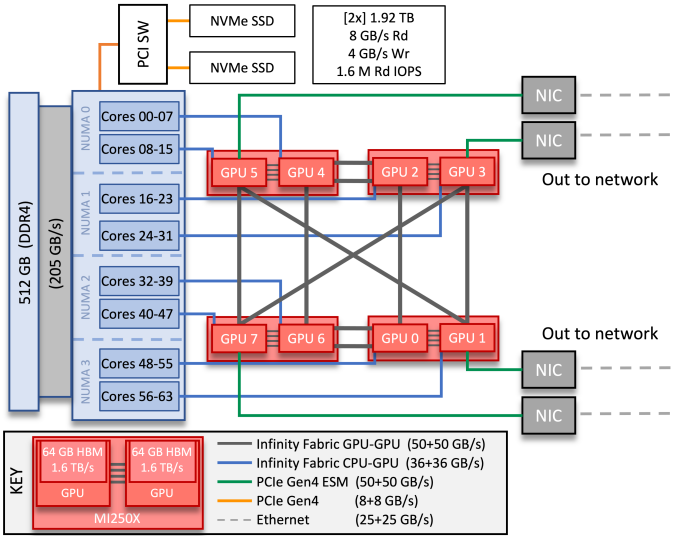

Fig. 1. A visual depiction of the architecture of a Frontier node.

II. Background

A. Frontier node architecture

Frontier is a 9,408-node HPE Cray EX system based on the Shasta architecture and Slingshot interconnect. Each node of Frontier is powered by one AMD optimized 3rd-gen EPYC 64-core processor and four AMD Instinct MI250X GPUs. Each MI250X GPU contains two graphics compute dies (GCDs) which are seen as two distinct GPUs by applications and resource managers. Each GCD contains 64 GB high-bandwidth memory (HBM) accessed at 1.6 TB/s. GCDs within a MI250X GPUs are connected via Infinity Fabric, with a peak bi-directional bandwidth of 200+200 GB/s (for a total of 400 GB/s). GCDs on different MI250X GPUs are connected via Infinity Fabric GPU-GPU, as shown in Figure 1, with peak bi-directional bandwidth of up to 50+50 GB/s (for a total of 100 GB/s), based on the number of Infinity Fabric links between GCDs. In total, Frontier contains 37,632 GPUs, 9,408 CPUs, and 9.2 PB of total memory (CPU + GPU).

The single AMD EPYC processor on each node is partitioned into 4 NUMA domains. Each NUMA domain contains 2 L3 caches, with each L3 cache associated with a single GCD within a MI250X. Frontier utilizes a feature called Low-Noise Mode (LNM), which binds all system processes to core 0. Frontier reserves the first core in each L3 cache by default, leaving 56 cores available. System process reside on core 0 and the other 7 cores are used by Lustre to improve I/O stability. The `sbatch`/`salloc` flag `-S` can be used to override this default by setting `-S 0`, though all system process will still be bound to core 0. Each physical core on the AMD EPYC CPU contains two hardware threads, though only one thread is active by default. The second core can be activated using the `sbatch`/`salloc` flag `--threads-per-core=2`. Each node has 512 GB of DDR4 dynamic random-access memory (DRAM) accessible at speeds of 205 GB/s.

Each compute node on Frontier also contains 2 non-volatile memory (NVME) storage devices supplying 3.8 TB of local storage that can be accessed at a peak sequential performance of 5,500 MB/s (read) and 2,000 MB/s (write).

B. Previous Work

This paper directly builds on work presented at the SC'23 HPC System Testing Workshop, entitled "Experiences Detecting Defective Hardware in Exascale Supercomputers" [10], which presented two generalized methods for detecting defective hardware in supercomputers, demonstrating the utility of such methods on Frontier. These methods are briefly summarized in the following sections. This paper extends the methods discussed in the previous work to include two more strategies targeting improving the MTBF of Frontier, and data from each approach is used to build a cohesive picture of the evolution of Frontier, with analysis of results and discussion of trends and key challenges.

1) Slurm backfill: The first method utilizes short jobs with low node counts designed to be scheduled by Slurm's `backfill` scheduler, to avoid impacting production workloads. On Frontier, the periodic node screen is typically run on 1000 nodes with a time limit under one hour, with most tests are under 30 minutes. This allows node screen jobs to backfill around production jobs; for example, while a full-machine job is waiting for the last few required nodes to complete their current allocation, node screen jobs can run on the idle nodes that the production job is holding without delaying the start of the production job. This method effectively screens nodes without interfering with production workloads, but lacks the ability to enforce an maximum time allowed between tests.

2) Slurm epilog: The second method was developed using lessons learned from deploying the Slurm `backfill`-based method. Slurm `epilog` runs at the end of each job to clean up temporary file systems and kill any remaining processes. Within `epilog`, a device health check script called `checknode` is run. This script examines system logs, file system mounts, and free memory, among other metrics, to

identify any defects in a node. The regularity with which `checknode` runs was identified as a convenient mechanism for enforcing strict time intervals for periodically screening nodes. `checknode` was extended to check the last date and result of a node screen test on a node, and if the last test was greater than 1 week prior or failed, `checknode` would remove the node from the queue and run node screen tests on it. These tests run and report results independent from the Slurm `backfill`-based tests. Tests run by this method must not use MPI and must be very short to minimize the time a node is forcibly taken out of the queue.

C. Related Work

There are many existing generalized approaches to identify data corruption or transient hardware faults. One such framework attempts to detect silent data corruption (SDC) of real-world HPC applications based on certain data properties such as smoothness, demonstrating a success rate of over 90% [6]. This work is extended further to also consider the impact of a SDC on the application execution, which reduces the overhead of such detection techniques [8]. These works have many parallels to the defective hardware identification in this work. First, both efforts emphasize the importance of data corruption and the challenges of detecting such events at extreme HPC scales. We approach the same problems from two sides; this SDC detection method is intended to be implemented within a production job, while our node screen is designed to identify defective hardware contributing to SDC's using small tests with well-known and reproducible behavior.

A second common approach utilizes machine learning (ML) techniques. One example of this approach utilized ML to build a model for how "soft" hardware faults propagate through an application [4]. Broadly, these approaches seek to identify the signatures of transient hardware faults or SDC's in the behavior of applications.

Engineers at Facebook recently published a paper on ArXiv detailing the investigation of silent data corruptions on internal production machines [9]. The findings and best practices from their paper are in good agreement with our findings as well, though unique approaches are taken. Distinctly, Facebook focused on SDC's, while we focus on any result of defective hardware (including SDC's).

In contrast with the first two of the discussed related topic areas, we seek to proactively identify and repair the hardware components responsible for SDC's or hardware-level faults, rather than detect the presence of fault in a production job. In some cases, there may be transient faults that cannot be pinned to a specific component, in which case, efforts like those described above certainly become critical. At this new frontier of computing scales, multi-pronged approaches to solving problems become increasingly effective for enabling ground-breaking science.

III. Approaches

Since Frontier's installation and acceptance, several efforts have been undertaken to purge defective hardware and improve the MTBF of Frontier. These efforts are integral to the continued success of Frontier's ground-breaking computations. The following sections describe the three strategies utilized to improve the MTBF and remove defective hardware on Frontier.

A. Targeted Application Failure Investigation

The first targeted strategy to improve the MTBF of Frontier was to run a single application with well-known behavior repeatedly on Frontier and investigate all failures to identify any defective hardware or software bugs. The application chosen for this task was LAMMPS, specifically the ReaxFF potential within LAMMPS [3], which had exhibited an above-average propensity to discover or cause a hardware failure in previous testing. LAMMPS was already highly-optimized on Frontier and easily customized from the command line to fit any node count and duration, making it a suitable choice for this effort. The ReaxFF potential of LAMMPS performs memory access patterns that stress the memory management hardware on the AMD MI250X GPU, while utilizing the majority of available memory on each GPU, making this code especially effective for identifying GPUs with under-performing or defective memory characteristics while maintaining a significant computational load.

In this strategy, leadership-class LAMMPS jobs were launched on Frontier, and when a failure was encountered, increasingly smaller jobs were launched on the targeted nodes as needed until a single failing part could be identified. This study also investigated the effects of varying the GPU's TDP, voltage of the GPU HBM controller, and HBM frequency on the rate of hardware-level errors, while also identifying and removing defective hardware. These parameters were varied as part of targeted investigation into power and memory-related failures. There were four experiments with these parameters; the first two performed a sweep over job sizes between 1728 and 9261 nodes with the following parameters:

- GPU TDP set to 560W
- GPU TDP set to 500W

The third and fourth experiments focused on 4096 nodes with the following parameters:

- GPU TDP set to 500W, HBM voltage increased +100mV
- GPU TDP set to 500W, HBM voltage increased +100mV, memory frequency reduced to 1200 MHz

Each experiment was constructed based on data from the current experiment; for example, the second experiment, setting the GPU TDP to 500W, was designed because power failures were observed at the default 560W TDP. This effort began in September 2022 and concluded in January 2023.

B. Formal Job Completion Study

The second strategy targeting the improvement of the MTBF of Frontier utilized an optimized version of the HACC[2] cosmological simulation code, which was chosen instead of LAMMPS to diversify the tests used to study and improve

[2]HACC: https://cpac.hep.anl.gov/projects/hacc/

Frontier. Similar to the first strategy, all failures were triaged to identify a root cause. Each phase examined the behavior of Frontier at a set of specific leadership-class job sizes and time limits; for example, phase 4 targeted jobs of 2500 nodes with a time limit of 75 minutes. This configuration was selected for this phase because of the trade off between job size and scheduling. This strategy is considered an extension of the LAMMPS investigation, so the five phases of this strategy were numbered 2 through 6, building upon phase 1 from LAMMPS. This effort began in June 2023 and concluded in January 2024.

C. Periodic Node Screen

The final strategy to improve the MTBF of Frontier is to periodically run a set of single-node or single-rank workloads on each compute node in Frontier independently. This strategy was first deployed in November 2022 and is described in previous work, summarized in Section II-B. Two methods for periodically screening nodes were developed; the first was deployed November 2022 using Slurm's `backfill` scheduler to schedule jobs on idle nodes, the second was deployed April 2023 using a script called `checknode` called by Slurm's job `epilog` to enforce a weekly testing frequency. The Slurm `backfill`-based method continues to run behind production workloads, but the Slurm `epilog`-based method was removed from production in December 2023 after it was found to be too disruptive to user workloads on the system. This method continually took large groups of nodes out of the queue to run screens, instead of scavenging cycles while nodes are idle.

Similar to the other strategies, this employs well-known applications that have previously failed on Frontier due to defective hardware or software. Table I lists the base applications currently supplying tests for Frontier. The set of available tests has grown over the lifetime of the effort and continues to grow to adapt to emergent behavior, though not all available tests are consistently run.

TABLE I
CURRENT PERIODIC NODE SCREEN APPLICATIONS ON FRONTIER.

Base code	Description
AMG[a] [3]	An algebraic multi-grid solver, CORAL-2 source
HACC [4]	An extreme-scale cosmological simulation code, CORAL-2 source
rocHPL[a] [5]	High-Performance Linpack benchmark [2], AMD source
LAMMPS[a,b] [6]	Several potentials from the LAMMPS Molecular Dynamics code [11]
oblex	A closed-source application provided by AMD, designed to test the GPU's high-bandwidth memory
rocPRIM [7]	A set of unit tests provided by the `rocprim` library.
BabelStream[c] [8]	A memory bandwidth microbenchmark for GPUs. [7]

[a]Both single-node and single-rank configurations used.
[b]Multiple configurations available for both single-node and single-rank.
[c]Newly developed since SC'23, November 2023.

IV. RESULTS

For each of the strategies used to identify defective hardware, we provide high-level results below. These results quantify the approximate cost in node-hours that each strategy consumes and data relevant to each strategy. The focus of the LAMMPS and HACC investigations was primarily on Frontier's MTBF, so the results for those strategies are centered on MTBF.

A. LAMMPS Application Failure Investigation

The LAMMPS application failure investigation was comprised of four experiments varying parameters such as GPU TDP, HBM controller voltage, and HBM frequency, as discussed in Section V. Figure 2 shows the failure rate as a function of node count for TDP's of 500W and 560W. There were 14 jobs between 40 and 240 minutes launched at 4096 nodes and below, and 11 jobs between 40 and 120 minutes launched from 6859 to 8000 nodes, and 8 jobs launched at 9261 nodes between 40 and 90 minutes. In total, 144 jobs were launched in this TDP experiment.

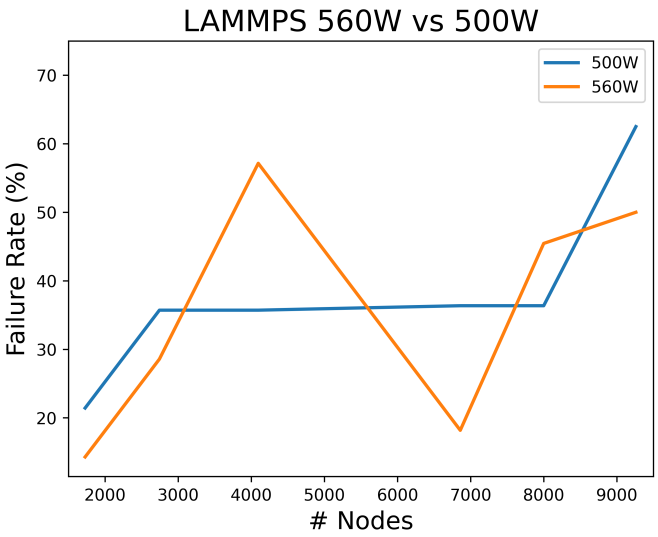

Fig. 2. The failure rate of LAMMPS jobs by node count for each examined TDP.

After observing that changing the TDP does not substantially improve the failure rate of large leadership-class LAMMPS jobs, the voltage to the HBM controller was increased by 100 mV and the memory frequency was varied while leaving the TDP at 500W. 4096 nodes was selected as the specific node count of interest for this experiment.

We ran 50 1-hour jobs for each configuration at 4096 nodes, with the following results:

[3]AMG: https://github.com/LLNL/AMG
[4]HACC: https://cpac.hep.anl.gov/projects/hacc/
[5]rocHPL: https://github.com/ROCmSoftwarePlatform/rocHPL
[6]LAMMPS: https://github.com/lammps/lammps
[7]rocPRIM: https://github.com/ROCmSoftwarePlatform/rocPRIM
[8]BabelStream: https://github.com/UoB-HPC/BabelStream

- 17 of 50 jobs failed with TDP of 500W, +100mV to HBM controller, and the memory frequency left as the default 1800 MHz
- 11 of 50 jobs failed with TDP of 500W, +100mV to HBM controller, and the memory frequency reduced to 1200 MHz

A total of 244 jobs were run with this strategy. During this study, 19 cases of defective hardware were successfully identified and repaired.

B. HACC Formal Job Completion Study

The HACC job completion study was comprised of five phases, targeting specific job node counts and time limits. Phase 5 was truncated for reasons outside of the study. Table II shows the number of jobs in each phase, as well as the minimum and maximum node counts in that phase. All jobs performed 200 steps in HACC, which takes between 75 and 90 minutes for all job sizes, on average.

TABLE II
NUMBER OF JOBS AND NODE COUNTS TARGETED FOR EACH HACC PHASE

Phase #	# jobs	Node counts
2	57	4000-9072
3	66	2500-7500
4	78	2500
5	19	7500
6	70	2500

Table III shows the count of each failure mode encountered during each phase of the HACC job completion study. The `Other` category is primarily composed of file system errors, unknown causes, and uncommon failure modes. This data is analyzed further in Section V.

TABLE III
FAILURE MODES IN THE HACC AND LAMMPS STUDIES

Phase	Power	GPU HBM	Network	Bad Actor	Other
2	20	2	3	1	10
3	10	3	0	0	8
4	5	0	0	1	13
5	2	0	0	0	5
6	5	0	8	2	1

C. Periodic Node Screen

There are two methods of running the periodic node screen: (1) utilizing the Slurm backfill scheduler to schedule short/small jobs on idle nodes, and (2) utilizing Slurm epilog to enforce a weekly screen on each node. In previous work, we provided detailed analysis of tests performed in the month of June 2023 [10]. Expanding upon this, we provide results for a recent 6-month span, from October 1, 2023 to April 1, 2024.

1) Slurm backfill scheduling: The first method of periodic node screening was deployed in November 2022 and has administered nearly 6 million single-node tests since deployment as of April 2024. From October 1, 2023 to April 1, 2024, this method of screening completed 1.9 million node tests, with summary statistics aggregated by test name shown in Table IV. 5 jobs impacted by system-wide issues have been excluded from these results. Note that when a test fails, the node is typically removed from the queue and the test is repeatedly re-run on that node to reproduce the failure and work with the triage team to identify the failure mechanism as needed. Failures that were fatal to a node (ie, power faults) are not currently tracked in this screening method.

TABLE IV
NUMBER OF RUNS AND FAILURES FOR EACH TEST RUN BETWEEN OCTOBER 1, 2023 AND APRIL 1, 2024 IN THE PERIODIC NODE SCREEN.

Test name	MPI?	Count Total	Count Failed
amg-nompi	n	337684	129
amg	y	192	0
coral2-hacc	y	3517	0
hpl-nompi	n	328576	162
hpl	y	184	0
lammps-nompi	n	175418	12
lammps-cpu	y	4225	0
lammps	y	2083	0
oblex	n	184424	124
rocprim	n	304361	24
stream-gpu	n	562671	2796

All 2796 `stream-gpu` failures are reproducible performance issues on a small subset of nodes caused by a suspected software bug that is under further investigation. Among the other 451 test failures, we identified 99 unique failing nodes. The failure modes from these nodes are summarized in Table V. This table includes all unique failures, so the sum of the right column is greater than the 99 unique failing nodes, since some nodes exhibited multiple failures throughout the 6 month window. The `Unknown` classifier is used for failures where there is not sufficient information to fully classify the failure.

TABLE V
PERIODIC NODE SCREEN FAILURES ITEMIZED BY FAILURE MODE.

Failure mode	Count
Hang, not reproducible	3
Crash, not reproducible	1
Transient performance failure	27
GPU HBM uncorrectable error	12
Numerical instability	11
Software bug	54
Unknown	13

There were three cases of a node screen test hanging and one case of a test crashing where no root cause could be identified. The hang or crash could not be reproduced upon further testing. Transient performance failures are tests that failed the performance threshold set in the node screen once and passed the performance threshold on subsequent screens. The 12 GPU HBM uncorrectable memory errors were identified by automated device health checks and the nodes were designated for hardware repair.

The 11 cases of numerical instability were promptly removed from the system for further testing and repair, and are under further investigation.

The 54 software bugs are primarily associated with software responses to edge cases. An edge case includes but is not limited to cases where an application leaves a node in a bad state or a hardware component fails but software is not able to identify that failure correctly. These software bugs are discussed in greater depth in Section V.

2) Checknode weekly screen: The second method of periodic node screening performs all tests consecutively and performed 39,437 screens as shown in Table VI, where each screen consists of the 3 tests listed in the table executed independently on all 8 GPUs on a node. During this time, no failures were observed from these tests. This may be due in part to this method utilizing shortened versions of each test. For example, the `lammps` tests used in the first method consume 24 minutes, while the `lammps` test in this method consumes less than one minute. Analyzing an additional 6 months prior to October 1, 2023 (beginning April 1, 2023), there were 3 failures identified by this testing method, all of which were found to be GPU HBM UE's triggered by the `rocprim` application. The ineffectiveness of this screening method led to removing this method from Frontier in December 2023.

TABLE VI
NUMBER OF RUNS AND FAILURES FOR EACH TEST RUN BETWEEN OCTOBER 1, 2023 AND APRIL 1, 2024 BY THE CHECKNODE PERIODIC NODE SCREEN

Test name	Count Total	Count Failed
lammps	39437	0
rocprim	39437	0
mini-hacc-fpe	39437	0

V. ANALYSIS

Given the complexity of an exascale supercomputer, it is often challenging to identify common themes among failures. In Frontier, most failures can be classified into one of several failure modes, including but not limited to: timeout/hang, application abort, power fault, network error, and numerical instability. These failures often are not random, but can be observed in "seasons" of each failure mode; for example, a month-long period in time where power faults are the most prevalent failure mode. In addition to these "seasons" of classifiable failure modes, there is a constant small population of nodes that are "bad actors"; that is, nodes that fail with an uncommon signature that is often very difficult to identify. These assertions are discussed further in the following sections.

A. Seasons of classifiable failures

The majority of hardware failures in Frontier fall within several failure modes. These failure modes often fluctuate in "seasons", where there is one prevalent failure mode and many other ancillary failure modes. In Table V, periodic node screen failures from October 1, 2023 to April 1, 2024 are presented. There are 54 software bugs identified in this 6-month span. 51 of these software bugs were triggered in the latter 4 months of the 6-month span. Software bugs, especially, are an excellent example of the "seasons" of prevalent failure modes, due to the lag in identifying and patching the software bug. It must be acknowledged that "seasons" of failure modes are influenced by the resources dedicated to identifying or repairing this failure mode. In large systems, there may be dozens of defective components that share the same failure mode. These components are removed in one of two ways: (1) the failing part is found during normal operations and replaced or (2) a proactive targeted investigation seeks to identify more parts that might be suffering from the same failure mode. When the latter approach is taken, this "seasons" of failure modes observation may be influenced by the targeted investigation, since there is greater effort to find a specific failure mode.

An example of this behavior in physical hardware is a season of power faults observed during the HACC study. After observing a number of power faults, a targeted investigation into this failure mode was performed during the HACC job completion study. A specific part within the system was successfully identified as the cause of these power faults, and a campaign was undertaken to replace the affected parts and mitigate this failure mode. This campaign overlaps the HACC job completion study, so we can see that from Phase 2 to Phase 4 and beyond of the HACC study, the number of power faults was reduced by 75%.

B. Constant population of bad actors

During each of the targeted defective hardware investigations, there was a constant very small population of so-called "bad actors", quantified in Table III as typically 0-2 nodes. These are most often nodes that produce an obscure or difficult-to-identify failure. Investigating and repairing these cases is often very challenging, and error messages may also be misleading; for example, if a component such as a CPU or GPU periodically produces a failure that seems clearly to be a defect within the compute device, but is later found to be a defect with the socket on the motherboard or some other component.

One example is a node that was identified in January 2024. This node intermittently failed only the MPI-enabled variant of one test in about 10% of test runs. Single-rank variants do not identify a failing part and the system log does not contain any incriminating messages identifying a specific part. In cases like this, a brute-force approach must be used to shuffle parts one-by-one to a known healthy node until the failure follows a specific part. Each "bad actor" case is thoroughly analyzed to harden software such as drivers and hardware repair processes against any future similar failures.

VI. KEY CHALLENGES

Since breaking the exascale barrier, Frontier has demonstrated the need for a new set of challenges to be addressed; the challenges to *maintain* exascale. In the following sections, we discuss several key challenges to maintaining Frontier,

specifically regarding maintaining a high MTBF and identifying defective hardware.

A. Discovering trends in failures

Discovering trends in failures is one of the most important yet most difficult tasks when maintaining a supercomputer. Frontier has many factors that make discovering trends in failures very difficult. Fist, with over 37,000 GPUs, 37,000 DRAM DIMMs, and 9,000 CPUs, along with a multitude of other parts, there are many components that need to be tracked. In addition to the staggering parts count, hardware is repaired by a team of technicians working in shifts, not a single person, which means that 10 faults of the same failure mode in the same day might be repaired by 10 different technicians and never get noticed. Finally, exascale demanded more powerful and more efficient, but more complex hardware than previous generations of supercomputers, and it is difficult to determine if the rate of failing parts exceeds the expected rate of failure, since many parts are new to production environments.

The first two factors discussed are logistical; efficient monitoring and tracking practices address both of these factors. While these factors are challenging, they are by no means novel. The third factor is perhaps the most challenging, and aligns with the motivation for this study. The new generation of hardware that enabled exascale is more complex than past supercomputers, and the behavior of such hardware in production environments is to be examined carefully. Computational studies such as this aid in forecasting the future health of the system, and lays the ground work to identify further trends in hardware failures.

B. Balancing testing with production

Several testing strategies were discussed in this work, and balancing the resources to dedicate to these testing strategies (and as a result, not to production jobs) is another key challenge to identifying defective hardware in Frontier. Minimizing the resources consumed is viewed as the "most efficient" method of testing, but it may not be the most effective. For example, the `checknode`-based periodic node screen prioritized minimizing the resources dedicated to the node screen. There are many differences between that method and the Slurm `backfill`-based method, but one consistency is that both methods use the exact same non-MPI LAMMPS test, except `checknode`'s LAMMPS test is more than 20 times shorter than the Slurm `backfill` LAMMPS test. Using the data presented above, we find the `checknode`-based LAMMPS test did not identify any failures through nearly 40,000 tests, whereas the `backfill`-based LAMMPS test averaged identifying one failure per 14,618 tests. Effectiveness and efficiency must be balanced, this looks different in every deployment. Given Frontier's typical workload (leadership-class jobs consuming 20% or more of the system), using 1000-node jobs less than 1 hour long to backfill in the queue has demonstrated effectiveness and has not shown significant disruption to production workloads. This remains a challenge as we continue to develop policies to inform this balance.

C. Bad hardware vs bad code

One of the key challenges in maintaining an exascale system is the ability to distinguish between cases of bad hardware and software bugs. Some error messages can occur for either defective hardware or an application code bug. One example is an error message that is occasionally associated with a GPU HBM UE that begins with "Memory access fault by GPU". This same error message can also be triggered from compiler bugs and application segmentation faults, among other errors. GPU HBM UE's are innocuous, they leave behind a well-known trace in system logs, but identifying a bug in the compiler from an application code bug can be very difficult.

D. Maintaining relevant tests

As failure modes and software stacks evolve, tests also need to evolve. At a minimum, tests need to keep up to the default software stack, but tests should also be representative of current and recent prevalent failure modes. This can be difficult, depending on how fast defaults move, how many tests there are, or if a test source version is updated which could require change to other parts of node screen infrastructure.

The number of tests also cannot be allowed to grow too large. Math dictates that running more tests either requires the resources dedicated to testing to increase, other tests to be decommissioned, or each test to run less. In the year and a half since initial deployment, the Slurm `backfill`-based method of periodically screening nodes has increased in size from 7 initial tests to 11 tests. In Table IV, there are 6 tests run more than 100,000 times, and the next most-run test had fewer than 5,000 runs. So while the number of tests has increased, the number of actively-used tests has not changed significantly, as some tests were replaced with newer or more-efficient versions and less effective tests were decommissioned.

VII. Conclusion

As the first exascale supercomputer in the U.S., Frontier enables leadership-class problem-solving in a new and groundbreaking way. The majority of Frontier's computational capability is provided by more than 37,000 AMD Instinct MI250X GPUs, but is supplemented by thousands of other parts including more than 9,000 CPUs, 37,000 DRAM DIMMs, and numerous components unseen. Frontier's GPUs are much more complex than the GPUs utilized by the previous generation supercomputer at OLCF, Summit, due to an increase in the power consumption and device memory, and reduction in the size of the compute chip. This new generation of hardware exhibits a more frequent failure rate than the previous generation components of Summit. Frontier's productivity would grind to a halt if defective hardware and software bugs were not repaired swiftly. In this work, we presented three strategies that were employed during and after the installation and acceptance of Frontier that have successfully removed dozens of defective components and identified several prevalent hardware and software issues that would have otherwise greatly hindered Frontier. This multi-staged approach informed the formulation of two key observations: (1) there are discernible seasons

of classifiable failures and (2) there is a constant population of "bad actor" nodes. Four key challenges of this work are outlined: (1) discovering trends in failures, (2) balancing resources allocated to testing with resources allocated to production, (3) failure modes may be triggered by bad hardware OR buggy code, and (4) maintaining a relevant set of tests.

We find that swiftly identifying seasons of failure modes and mitigating "bad actor" nodes is key to maintaining exascale performance in Frontier. Further, there must be a constant vigilance to maintaining all infrastructure designed to serve this purpose.

ACKNOWLEDGEMENT

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] https://www.ornl.gov/news/frontier-supercomputer-debuts-worlds-fastest-breaking-exascale-barrier, August 2023.

[2] "Hpl: High-performance linpack benchmark," http://icl.cs.utk.edu/hpl/index.html, 2023.

[3] H. M. Aktulga, J. C. Fogarty, S. A. Pandit, and A. Y. Grama, "Parallel reactive molecular dynamics: Numerical methods and algorithmic techniques," *Parallel Computing*, vol. 38, pp. 245–259, 2012.

[4] R. A. Ashraf, R. Gioiosa, G. Kestor, R. F. DeMara, C.-Y. Cher, and P. Bose, "Understanding the propagation of transient errors in hpc applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2807591.2807670

[5] S. Atchley, C. Zimmer, J. Lange, D. Bernholdt, V. Melesse Vergara, T. Beck, M. Brim, R. Budiardja, S. Chandrasekaran, M. Eisenbach, T. Evans, M. Ezell, N. Frontiere, A. Georgiadou, J. Glenski, P. Grete, S. Hamilton, J. Holmen, A. Huebl, D. Jacobson, W. Joubert, K. Mcmahon, E. Merzari, S. Moore, A. Myers, S. Nichols, S. Oral, T. Papatheodore, D. Perez, D. M. Rogers, E. Schneider, J.-L. Vay, and P. K. Yeung, "Frontier: Exploring exascale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3581784.3607089

[6] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, and F. Cappello, "Lightweight silent data corruption detection based on runtime data analysis for hpc applications," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 275–278. [Online]. Available: https://doi.org/10.1145/2749246.2749253

[7] T. Deakin, J. Price, M. Martineau, and S. McIntosh-Smith, "Evaluating attainable memory bandwidth of parallel programming models via babelstream," *International Journal of Computational Science and Engineering*, vol. 17, no. 3, pp. 247–262, 2018. [Online]. Available: https://www.inderscienceonline.com/doi/abs/10.1504/IJCSE.2018.095847

[8] S. Di and F. Cappello, "Adaptive impact-driven detection of silent data corruption for hpc applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 10, pp. 2809–2823, 2016.

[9] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, "Silent data corruptions at scale," 2021. [Online]. Available: https://arxiv.org/pdf/2102.11245.pdf

[10] N. Hagerty, J. Webb, V. Melesse Vergara, and M. Ezell, "Experiences detecting defective hardware in exascale supercomputers," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 619–626. [Online]. Available: https://doi.org/10.1145/3624062.3624134

[11] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Comp. Phys. Comm.*, vol. 271, p. 108171, 2022.