# Systemic Assessment of Node Failures in HPC Production Platforms

Anwesha Das[1], Frank Mueller[1], Barry Rountree[2]
[1]*North Carolina State University*, adas4@ncsu.edu, mueller@cs.ncsu.edu
[2]*Lawrence Livermore National Laboratory*, rountree4@llnl.gov

***Abstract*—Production HPC clusters endure failures reducing computational capability and resource availability. Despite the presence of various failure prediction schemes for large-scale computing systems, a comprehensive understanding of how nodes fail considering various components and layers of the system is required for sustained resilience. This work performs a holistic diagnosis of node failures using a measurement-driven approach on contemporary system logs that can help vendors and system administrators support exascale resilience.**

**Our work shows that external environmental influence is not strongly correlated with node failures in terms of the root cause. Though hardware and software faults trigger failures, the underlying root cause often lies in the application malfunctioning causing the system to fail. Furthermore, lead time enhancements are feasible for nodes showing fail slow characteristics. This study excavates such helpful empirical observations, which could facilitate better failure handling in production systems.**

***Index Terms*—Root Cause, Node Failures, Holistic Analysis**

## I. Introduction

Powerful supercomputers require high availability to run scientific applications, enabling researchers from diverse technical domains to address grand challenges in computational simulations. As engineers are designing efficient exascale nodes, current computing platforms require robust failure handlers to keep up with system scale, density, software complexity and computational speed. This necessitates the need for proactive solutions that can flag ahead of time an impending component failure. Before failure mitigation, having a better understanding of how failures materialize is essential. Recent research on log mining-based failure characterization [17], [34], prediction [9], [24] and recovery [6] have revealed helpful insights to address failures in HPC. The time elapsed between failure manifestation and the timestamp of the precursor log message when an impending failure is flagged is defined as the lead time. Proactive fault tolerant solutions [9], [24] when supported by root cause diagnosis can improve lead times and help in responding to both manifested or imminent failures effectively. Prior studies on failure evaluation [11], [28], [39] either provide a high-level categorization of faults with limited systemic correlations or lack effective analysis for proactive resilience [20] that can ameliorate the failure impact with an effective fix. Root cause analysis of any complex system requires awareness towards the events encountered by its components. While researchers have focused their attention on specific components [3], [25] and interfaces depending on their target problem, answering how compute nodes fail needs a more integrated approach towards correlation-based log mining. Our methodology adopts a system-wide view to track causes of node failures prevalent in computing systems. Our work is novel in that it considers system environment conditions along with inter-component dependencies to increase lead times to failures boosting failure prediction schemes.

**Background**: The current state-of-the-art lacks in the following aspects for better fault tolerance in HPC:
1. While few studies consider the full software stack to design resilient systems conforming to the vision of cross layer resilience [7], various system layers (software [17], hardware [41], application [29]) are often studied separately without exploiting their correlations during causal analysis [13].
2. Diverse system components affect each other (e.g., GPU [31], SSD [3], DRAM [5]). Focusing on a specific component in isolation provides a local view, yet lacks a global perspective. As an example, analyzing SSD failures [3] w.r.t. workloads separately may miss indicators in event logs generated elsewhere. In the context of root cause, having an understanding of how much these components correlate in failure manifestation can prevent recurring faults and re-investigations, thereby reducing unnecessary overhead.
3. Once a failure manifests, corrective actions need to be enhanced (e.g., failed interconnect failovers [22] exacerbate failure recovery). A deeper understanding of how failures happen (beyond spatio-temporal traits [11]) can aid in choosing the appropriate curative action for long-term system health.

This work investigates causes of node failures considering software, hardware, and application malfunctioning across diverse components, with recommendations for effective system health checkers for applicability in practice.

**Challenges**: The following impediments make root cause analysis challenging but important for computing systems:
1. Events occurring transiently may not get manifested as logs. Moreover, production logs occasionally contain missing (specific time duration) or partial information (absence of certain environmental logs) due to logging discrepancies. Deciphering transient faults that lead to cascading failures is non-trivial.
2. Components can exhibit fail-slow characteristics, different from fail-stop behavior [16]. The analysis of the former is important to assess predictable lead times.
3. Additional inputs may be required from operators to understand the implications of low-level system logs in order to analyze the root causes of failures accurately.

After the detailed failure analysis studies on Blue Wa-

1530-2075/21/$31.00 ©2021 IEEE
DOI 10.1109/IPDPS49936.2021.00035

TABLE I. HPC System Details

| System | Duration | Log Size | #Nodes | Type | Interconnect | Job Scheduler | FileSystem/OS | Processors | GPUs/ Burst Buffer |
|---|---|---|---|---|---|---|---|---|---|
| S1 | 10 mons | 373GB | 5600 | Cray XC30 | Aries Dragonfly | Slurm | Lustre/SuSE | IvyBridge | × |
| S2 | 12 mons | 150GB | 6400 | Cray XE6 | Gemini Torus | Torque | Lustre | Haswell | × |
| S3 | 8 mons | 39GB | 2100 | Cray XC40 | Aries Dragonfly | Slurm | Lustre/SuSE | Haswell | Burst Buffer |
| S4 | 10 mons | 22GB | 1872 | Cray XC40/XC30 | Aries Dragonfly | Torque | Lustre/CLE | Haswell/IvyBridge | Burst Buffer |
| S5 | 1 mon | 3.1GB | 520 | Institutional | Infiniband | Slurm | Lustre/RedHat | Haswell | GPUs |

ters [28] and Titan [34] for contemporary HPC systems, this paper investigates how node failures happen with beneficial insights to their reasons. Unlike prior works [27], [34] on a single Cray system considering manual failure reports [11], [28], our diagnosis is purely log-based from five different production sites with no human written reports. Similar to several past statistical analyses [11], [20], our study did find cases that cannot be analyzed further because of insufficient data in logs to determine a root cause decisively.

**Contributions**: This paper answers the following questions to enhance resilience in HPC systems:
1. Are there spatial or temporal correlations amongst failed nodes w.r.t. similar root causes?
2. How much do the environmental factors (e.g., heartbeat faults, temperature violations) directly influence node failures? If there exist early external indicators, can the lead times be enhanced considering the same?
3. What faults do not lead to failures? Under what conditions?
4. How do applications executing on nodes contribute to failures? While jobs fail because of node failures, jobs can trigger nodes to fail as well. Is there any presence of temporal locality for the failed nodes running the same specific application?

Past works [11], [28], [34] have performed high-level characterization of potential root causes without analyzing the external influence over correlated failures. These investigations viewed failures in isolation with limited causal diagnosis. In contrast, we adopt a global view. To summarize, our work makes the following contributions:

- We diagnose compute node failures across five HPC systems, four of which are well used production clusters, incorporating environmental correlations. We provide estimates of commonly occurring faults not leading to anomalous shutdowns.
- We quantify lead time enhancements if feasible using the external indicators for the failed nodes.
- We analyze job attributes on failed nodes to drill down to the root cause, apart from spatio-temporal correlations. Based on the insights we obtain from such system-wide measurements we discuss their implications for enhanced system health.

## II. Preliminaries

Table I entails the characteristics of peta-scale HPC clusters whose logs have been analyzed. As evident, four out of five are Cray systems of substantial scale, all used heavily for various scientific applications. We have briefly mentioned the system configuration details to provide a clear description about the log sources. The time line of the logs spanned across 3 years (2014 to 2016). S5 is a small scale institutional cluster with a local file system using an Infiniband interconnect, unlike the rest. We did not have external environmental logs for S5, we discuss our findings on 4 weeks data, only to quantify application-based failures compared to the other machines. Apart from S2, which has Gemini interconnect, all the Cray systems use Aries, with a Lustre file system. While S1, S3 and S5 use Slurm as their job scheduler, the rest use Torque.

TABLE II. Log Data Details

| Node Logs | Content Description |
|---|---|
| Internal | console/messages/consumer (p0-directories) |
| External | controller/event (ERD)/SEDC/Slurm/Torque |

Table II depicts the major portions of the logs consulted for this study. The compute node internal logs (console, consumer, and messages) in the p0-directories are used to obtain the node-specific events on Cray systems. Log messages pertaining to blade, cabinet, and environmental data are analyzed using controller and event router daemon (ERD) logs. These contain SEDC (System Environmental Data Collections) warnings and additional hardware fault alerts to aid failure analysis. The job scheduler logs from Torque or Slurm are analyzed to investigate job-based failures. *We have published sample logs used in this study* at *https://doi.org/10.5281/zenodo.4114171 to facilitate further research by the community.*

### A. Methodology

Figure 1 highlights the integrated components of a Cray machine from a finer to coarser granularity at a high-level. We consider system-wide environmental logs and blade/cabinet characteristics along with the node-specific internal events during the *unhealthy* time frame. We move from node to blade to cabinet to understand fault conditions and derive early indicators of impending failures. The controller logs coupled with event router messages provide deviations (higher/lower than the normal range) in sensor measurements (e.g., fan speed, temperature) to warn about health problems. Incorporating such features with job-based events aid in failure diagnosis.

We have consulted the Cray documentation [1] and relevant findings published in the CUG (Cray User Group) [2] for our work. Figure 2 highlights our investigation procedure. We perform failure analysis in the following manner:
1. We track confirmed failure indications in the node-specific logs. These encompass the *console*, *messages* and *consumer* logs of the compute node internals. This initial step involves cluster administrator's knowledge about anomalous failure symptoms validating the ground truth.
2. Considering the time-frame of failed nodes derived in (1), we investigate the nodes' health residing in the same blade as that of the failed nodes, mining the *controller* logs containing blade- and cabinet-specific information. This helps us to

understand the presence of spatial correlation. We correlate the SEDC warnings of the *event* logs to elicit any external influence evident over the blades with unhealthy nodes.
3. Additionally, we analyze the jobs allocated on the failed nodes from the *scheduler* logs to understand their effect on the compute nodes. This helps us to narrow down the root cause in terms of deciphering whether application-triggered node failures appear in conjunction with early external indicators such as hardware errors and sensor reading variations or not.

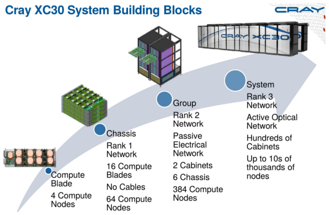

Fig. 1: Cray System Source URL

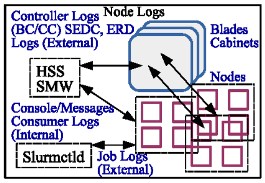
Node Logs
Controller Logs (BC/CC) SEDC, ERD Logs (External)
Blades Cabinets
HSS SMW
Nodes
Console/Messages Consumer Logs (Internal)
Slurmctld
Job Logs (External)

Fig. 2: Methodology

Figure 2 shows the SMW (System Management Workstation) with its HSS (Hardware Supervisory System) that manages the states of blades and cabinets. The Slurm workload manager, with ALPS (Application Level Placement Scheduler), coordinates resource allocation and job scheduling.

Additional consideration of user workload details [27], performance logs (e.g., LDMS [19]), or resource usage based diagnosis [8] is beyond the scope of this work. While some production sites neither maintained manual reports nor collected detailed system metrics [20], others may not release them because of restricted distribution policies, making our statistical inference difficult to confirm. However, with failure reports [26] and performance monitoring tools [30] currently being integrated into HPC systems, validating such analyses has become practically attainable.

## III. Evaluation Results

System-wide outages (SWOs) making the entire system unavailable are present in our logs and tend to be mostly either service related, intended node shutdowns, or file system caused failures. They contribute to less than 3% of the overall anomalous failures. We recognize and exclude intended shutdowns. Our study addresses single and multiple node failures[1], unlike SWOs, caused by any system malfunctioning. We evaluated over 1200 node failures for our analysis. In most Cray systems, 4 nodes reside in a single blade (slot). We provide representative samples carefully chosen over time-intervals to make observations, i.e., changing the duration of time or time-frame does not alter the overall inference.

### A. Inter-Node Failure Times

Before digging into the root cause, we checked how far apart failures are per day and how many nodes share the dominant failure reason. We consider failures over 7 weeks and calculate the cumulative node failures over different inter-node failure times. Figure 3 shows that 92.3% and 76.2% of the node failures happen within 1 to 16 minutes of each other in S1, over W1 and W7. The mean time between successive failures (MTBF) for those weeks are 1.5 (±0.56), and 12.1 (±4.2) minutes, respectively. Similar observations on different days on all systems indicate that time between adjacent node failures ranges from a few seconds to more than 2 hours. Though there are days without failures, on other days nodes fail just minutes apart. Unlike SWOs appearing more than 6 hours apart in Blue Waters [28], and server failures between 12 to 13 hours in Google [15], we observe shorter MTBFs.

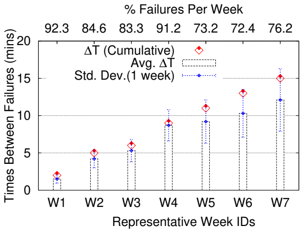

Fig. 3: Time b/w Failures

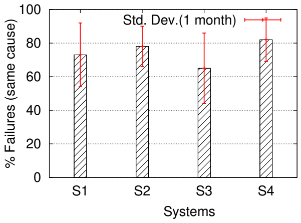

Fig. 4: Same Failure Cause

Next, we identify single or multiple *dominant* failure reasons per day and compute the fraction of failed nodes on the same day corresponding to the dominant failure reason (e.g., H/W MCEs, kernel oops, Lustre Bugs). Figure 4 indicates that in 30 days, 65% to 82% of the nodes share the same failure cause with variation between 12 and 21 across all four systems. Certain days have high job-triggered failures when most nodes run the same application. On other days, failures relate to diverse causes over short temporal distances. It is interesting to note that, if the dominant fault gets fixed, over 50% of the node failures can be recovered per day.

**Observation 1**: *Time between failures has reduced (hours to minutes) in recent years compared to prior work [15], [28]. On average, more than 65% of the failures per day are caused by the same malfunctioning, indicating the importance of analyzing short lead times and root causes of node failures.*

### B. External Influence on Node Failures

In order to understand the environmental influence on nodes we consider the blade and cabinet-specific health faults logged during the unhealthy time frames of the corresponding node failures. Blades encounter health faults and SEDC warnings (e.g., Electronic Circuit Breaker (ECB) faults) triggered by the blade controller (BC) software related to power monitoring. The cabinet controller (CC) software logs similar cabinet health status and sensor reading deviations such as temperature, voltage, and air velocity. A breakdown of some observed controller faults and warnings is shown in columns 1 and 2 in Table III. We correlate the blade identifier (ID) and cabinet ID of the corresponding failed node ID and inspect the logs around the failure time for systems S1 to S4.

Figure 5 highlights that 67% to 97% of the observed node voltage faults (NVF) correspond to failed nodes over 5 different months. These occur rarely, but when they do, they often relate to failures. On the contrary, 21% to 64% node heartbeat faults (NHF) actually manifest as failed nodes. This is because if a node fails a health test or skips a heartbeat, it is suspected to be dead, but empirically we observe that

[1] Here, *failures* typically refer to *node* failures, unless otherwise mentioned.

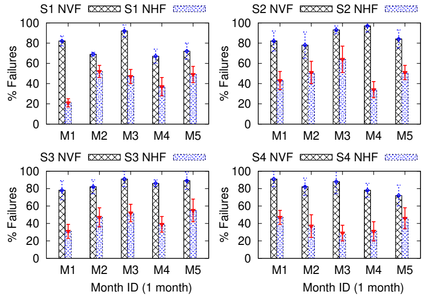

Fig. 5: Node Faults

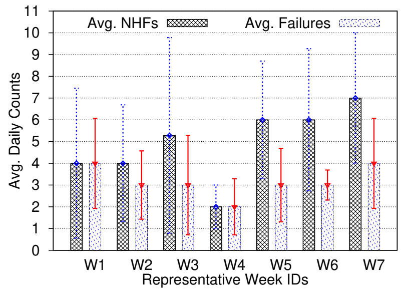

Fig. 6: Node Heartbeat Faults

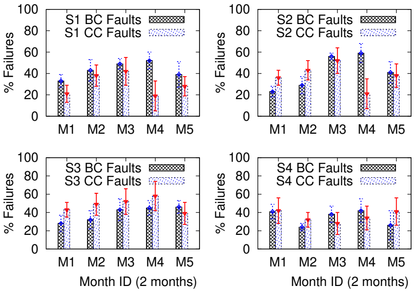

Fig. 7: Blade and Cabinet Faults

only about ≈43% NHFs actually fail. Figure 6 shows a finer breakdown of the NHFs, over 7 weeks. Most NHFs were failures in W1 and W4, while in the others more than 50% NHFs eventually caused a node to fail. Many NHFs turn out to be failures caused by hardware machine check exceptions (MCE), while NHFs that do not fail are nodes with their power turned off or those with skipped heartbeats. Unlike prior work [35] (2% of the NHFs fail), we observe higher correspondence of NHFs with failures. For job-caused failures, NHFs may not be present in the logs, because the node passed the health checks at the communication level, but later job-caused malfunctioning launches the node health checker (NHC), which, when in *suspect mode*, may turn the node to *admindown* [35] based on failed tests. Early NHF indicators for blades can warn about likely defects affecting some nodes.

TABLE III. Fault Breakdown

| Health Faults | SEDC Warnings |
|---|---|
| NHF, NVF, BC Heartbeat (BCHF) | Temperature, Voltage |
| ec_heartbeat_stop, ec_l1/l0_failed | ECB Faults |
| get sensor reading failed | Air Velocity |
| Cabinet Power & Micro Controller Communication Faults | ec_environment (e.g., Fan Speed, Air Flow) |
| Module Health & RPM Faults | Cabinet Sensor Check |

Figure 7 indicates, over periods of 2 months, 23% to 59% of the failures belong to faulty blades, and 19% to 58% to cabinets that elicited some warnings or faults. While this alone does not give any clear indication about correlations between blade or cabinet health and specific failures, hardware bugs and file system errors are observed in the node internal logs during these times. For days with no failures, such health faults are seldom found. For specific BC heartbeat faults, we observed only a fraction of the nodes in that blade to fail, but not all.

**Observation 2**: *A small fraction of failed nodes correspond to blade-cabinet-related faults, implying weak correlation. NVFs and NHFs can be utilized as early indicators of malfunctioning in failure prediction schemes to improve lead times.*

### C. What faults do not cause failures?

We have identified that certain recurring environmental warnings and errors do not cause failures and are mostly benign. A small fraction of the blades and cabinets in the event logs (ERD) encounter *ec_sedc_warnings* with BC and CC sensor reading deviations. These predominantly contain warnings for temperature, voltage or velocity falling below the minimum allowed system threshold. Figure 8 shows that the unique blade count with various SEDC *warnings* varies between 5 and 226, while the cumulative count of blades and cabinets experiencing faults ranges from 24 to 240 (±21) over a week for S1. The blade count for encountered health *faults* is mostly higher than the *warnings*. Figure 9 depicts the frequency of multiple BC-CC warning types occurring throughout the day for S2. While blades 1, 5 and 8 experienced more than 1400 mean recurring warnings, 7 stopped seeing them after a certain hour of the day. Over 3 weeks, 8 blades underwent voltage and temperature violations, but the specific failed nodes did not belong to any of these blades.

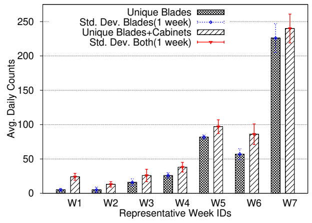

Fig. 8: Unique Blade Counts

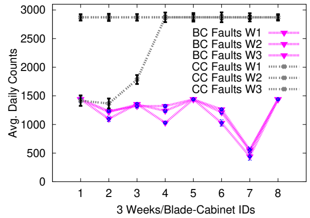

Fig. 9: BC-CC Warnings

Cabinet-level faults are logged more frequently than those of blades, with more than 1400 mean daily counts. Only 32.14% (e.g., 9 out of 28) failures belonged to these faulty cabinets over a week. Cabinet faults do not have correlations with a certain failure type since the corresponding blade of the faulty node on several occasions does not encounter such signals. Furthermore, many healthy blades with no failures experience similar temperature or voltage violations. Temperature variations can trigger hardware errors affecting sensor readings of an entire blade causing the air velocity to be automatically reduced by the firmware in the cabinet. However, presence of similar incessant faults during healthy time frames as well do not help in pinpointing the potential root cause. Hence, we did not investigate further. Our observation conforms to prior work on temperature variability [12], i.e., there is no clear evidence that hotter nodes contribute to higher node outages based on studies with LANL HPC clusters. This contradicts the presumption [34] that hot air triggers failures for Titan nodes. Titan nodes have GPUs known to be temperature sensitive [31], unlike ours. Moreover, there is no diagnosis made that heat is the root cause, apart from a high-level spatial analysis. Our studies did not find notable features in threshold violations to help quantify node reliability.

**Observation 3**: *Blade and cabinet-level indications are not primary causes of failures. Though health faults persist on*

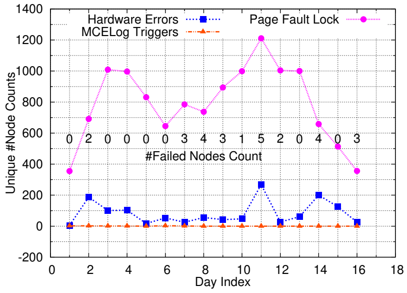

Fig. 10: Node Errors in S3

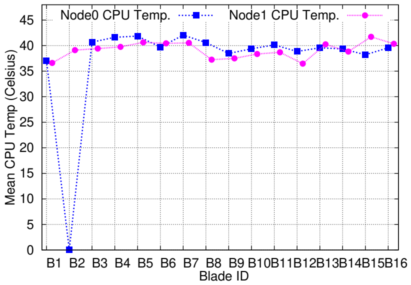

Fig. 11: SEDC Analysis in S1

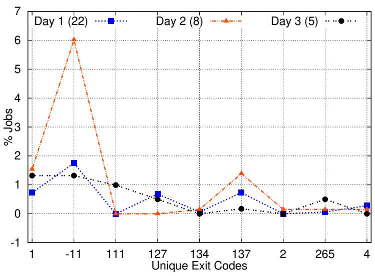

Fig. 12: Job Errors in S4

*days with observed failures, there is no clear evidence of sensor reading violations causing nodes to fail.*

Figure 10 indicates that over 16 consecutive days the total nodes experiencing hardware errors (e.g., correctable/non-correctable memory errors, buffer overflow), MCE log triggers (page/cache/DIMM; caused when the error count exceeds a predefined threshold), and Lustre I/O errors (caused by deadlocks and page fault locks), are much higher than the failed nodes (<6). While recent work [27] has similar findings in terms of error concentration on a few nodes w.r.t. the system scale, we further notice that most of these erroneous nodes do not fail in the due course. More nodes experience page fault locks signaling I/O problems (job-triggered) than hardware errors. Figure 11 shows the mean CPU temperature of 2 nodes per blade across 16 blades located in the same chassis and cabinet for a specific day with 1 failure in B2. Except 1 node turned off in B2 (Node0 of B2 has 0°C in Fig. 11), the remaining blades exhibit a steady temperature (≈40°C). As hinted earlier, these temperature values do not aid in root cause analysis. Figure 12 shows that over 3 specific days with 22, 8, and 5 failures, respectively, only 0.06% to 6.02% of the jobs finish with non-zero exit codes, while 90.43% to 95.71% of the jobs complete successfully. Of these erroneous jobs, some are caused by configuration errors (e.g., exceeding wall time/memory limit, user killed) leaving a few errors caused by node problems or application bugs. ≈10% of the failed nodes correlated with application malfunctioning, but not all.

**Observation 4**: *Increase in error counts need not necessarily degrade system reliability since only a small number of nodes with errors eventually fail. However, more errors are tangible with application configuration problems, which indirectly spawn filesystem and hardware errors. Such errors may relate to unpredictable user behavior impacting the system.*

### D. Lead Time Enhancements

Most studies [24] have referred to lead time considering internal console logs containing a sequence of fault indicative messages (e.g., fatal) leading to failure. We inspect if additional environmental messages appearing before or along with the indicative internal logs aid in improving the lead time. While many external faults and warnings are not the primary root causes, certain failures possess early indicators, e.g., *ec_hw_errors* in the event logs indicating hardware malfunctioning. Typically, such errors appear in conjunction with multiple indicative root causes such as processor corruptions, node heartbeat faults, firmware bugs, kernel panics etc. in the internal logs. Hardware errors do appear during healthy times as well. However, additional internal failure patterns affirm their correlations with node failures. Such cases of blade-level faults and SEDC warnings along with the evidence of failed nodes with buggy console messages enable lead time increments. These environmental alerts imply fail-slow characteristics unlike fail-stop, similar to fail-slow hardware evident in production data centers [16]. For certain failures, hardware errors sustain for a long time while they have very low frequencies during other times. Indicators in the external logs timestamped earlier are correlated to node, blade, and cabinet IDs over the failure times in the internal logs. The time difference between the relevant external and internal timestamps are computed to obtain viable lead time increments. Figure 13 shows that considering the external faults, the mean lead times can be increased by about 5 times, compared to just the internal node logs by themselves, in systems S1 to S4. For these failures, the early indicators were absent during normal operation. To confirm that lead times cannot be enhanced further if failure is solely caused by the application, several external faults are analyzed. Figure 13 illustrates that for 10% to 28% of node failures lead times could be enhanced considering early indicators over 4 different weeks. For 72% to 90% of the failures, absence of external warnings prevented lead time enhancements. This fraction depends on the contribution of application-caused failures.

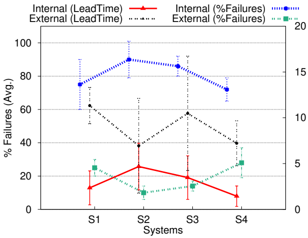

Fig. 13: External Influence

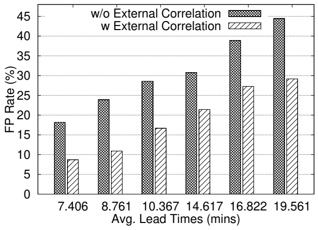

Fig. 14: Lead Times with FP

To assess the effect of additional external correlations on the false positive rate, the existence of similar correlations in the healthy node logs around similar failure times are analyzed. Figure 14 highlights that the false positive rate is lower (e.g., 30.77% down to 21.43%) with external correlations considered than otherwise. This is because healthy node logs that appear *similar* to multiple correlations (for an unhealthy node) occur less frequently, reducing the number of false positives.

**Observation 5**: *Certain failures caused by hardware errors or file system bugs possess early indicators in the external logs.*

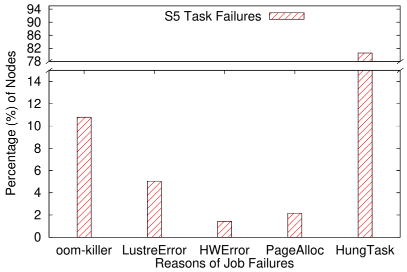

Fig. 15: Job Failures in S5

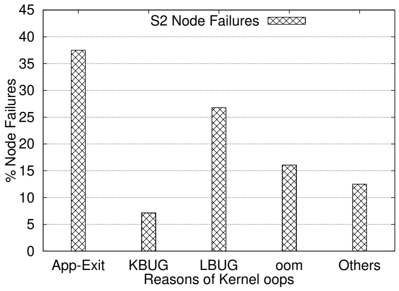

Fig. 16: Job Failures in S2

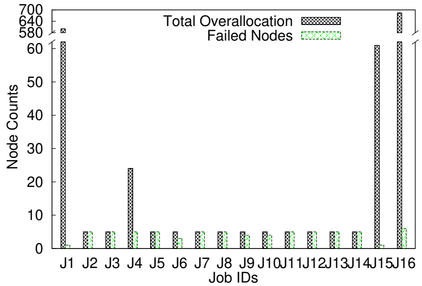

Fig. 17: Resource Overallocation

*Lead times can be enhanced by about a factor of 5 times in such cases with a lower false positive rate w.r.t. the failures without external correlations. However, such enhancements are not possible for application-triggered node failures, since early failure indicators are absent in this case.*

### *E. Application Triggered Failures*

The following patterns are observed primarily for failures triggered (directly or indirectly) by applications:

- The internal node logs often contain major hardware or software errors apart from file system or interconnect errors and node shutdown messages. These occasionally appear in conjunction with job-specific errors due to NHC tests.
- Typically external environmental indicators do not exist for these failures, making lead time enhancements infeasible.
- Multiple nodes or blades failing at similar times of the day often share the same job ID. For such cases, failed nodes need not be quarantined as these nodes recover once new jobs run on them since the problem is with the application.
- Once *oom-killer* (out of memory) is invoked and processes are killed, associated modules in the stack traces (e.g., xpmem, dvsipc, lustre) often indicate file system inconsistencies. Processes also get killed by the epilogue of the job scheduler that removes any user job from a node before it is reallocated.

The key root causes of kernel oops observed across the studied systems are *Lustre* and *kernel* bugs, *hung-task timeouts* and *oom-killers*. Hardware, software and application faults tend to trigger callback traces. Across all 5 systems, out-of-memory faults, job-triggered kernel bugs and Lustre bugs are prevalent reasons for node failures. These events are often accompanied by NHC test failures (e.g., abnormal application exits) and stack traces. Even without application malfunctioning, MCEs, CPU corruptions and file system bugs result in kernel panics. Jobs do get aborted because of failed nodes. But nodes also fail because of the running application's computational requirements. Our measurements show that spatially distant nodes have temporal locality of failures because of the common jobs running on them. For example, in Figure 3 week W1 experienced 92.31% of the failures within 2 minutes. More than 42% of those failures belonged to different blades distant from each other on a specific day; however, all of them executed under the same job ID during the time of failure.

We analyzed the call traces of S5's logs. Figure 15 shows that 10.59% of the nodes were running low on memory, which triggered the *oom-killer* that killed processes and added *kernel oops* messages to the log. 5.04% of the nodes had Lustre errors without any call trace causing jobs to fail. While 1.43% of the nodes encounter hardware errors, such as GPU or disk errors, 2.16% had software errors, such as page allocation faults and segmentation faults. Several jobs got canceled in the interactive session, around 11% of the jobs failed to complete because they got affected by the state of the allocated nodes. 80.57% of the nodes encountered *hung task timeout* errors followed by a call trace indicating slow system I/O, unable to flush the data to free memory within the stipulated time (2 mins). Hung task-based kernel oops are only seen in S5 (institutional cluster with a local file system), but they do not cause nodes to fail and are not observed on the Cray systems.

For S2, Figure 16 shows that 37.5% of the failures occur due to anomalous app-exits (application), failing NHC tests turning the node down. 7.14% of the failures were caused due to critical kernel bugs (e.g., invalid opcode) and 26.78% because of file system bugs (e.g., race in threads spawned in the code), all prompted by the compute jobs. 16.07% of the failures happened due to memory resource exhaustion without additional software bugs. The other 12.5% of kernel oops were due to CPU stalls and other driver and firmware bugs. On the surface, these bugs (`KBUG` and `Others` in Fig. 16) seem to emanate from the OS, but careful analysis reveals the potential of application-triggered file system bugs, which indirectly propagate as kernel bugs or CPU stalls. Finer inspection included examining the beginning of the stack traces. E.g., presence of `dvsipc` related modules indicate an affected file system triggered by the application when running out of resources. Runtime errors triggering kernel oops cannot be handled ahead of time since tangible software or hardware errors appear only after the affected kernel module is invoked. This indicates the need for fine-grained application performance diagnosis studies [37], which consider such runtime anomalies to reduce application-triggered errors.

Figure 17 illustrates how memory overallocation can cause failures in production HPC systems. On a specific day, 53 failures occur over just 16 jobs. Of the allocated nodes, a subset of them suffer resource *overallocation* errors. As evident for jobs J5 and J8, all overallocated nodes fail, while only a few of them fail for jobs J4 and J15. J1 and J16 had 1 and 6 failures for 600 and 683 overallocated nodes, respectively. When a small fraction of the overallocated nodes fail, the jobs fail to complete. Consequently, job re-allocations are performed for recomputations. In this case, Slurm allocated more memory than was available for the node. Such job triggered node failures suggest the need for carefully choosing the job submission parameters (e.g., number of cores, tasks per core etc.) for a specific application (e.g., MPI, Matlab) that the user intends to execute.

**Observation 6**: *Unlike institutional clusters, file system bugs are more frequent in Cray systems indicating the application's influence on Lustre contention and resiliency. ≈37% of the app-exits occur because of incapable nodes implying the need for better resource-aware scheduling. When job requirements exceed a node's resource capacity, quarantining [18] may not be effective. Instead, those applications can be monitored and their corresponding users can be informed.*

### *F. Node Internal Failure Analysis*

Failure causes evident from internal node logs have been studied in the past [17], [28]. We observed similar failure causes such as MCEs, processor corruptions, and file system bugs. We performed additional correlations with external heath faults and application impacts to scrutinize external influences. We summarize our findings here:

1. Many driver or firmware bugs appear after *application exit* messages from NHC and result in kernel oops. These may or may not relate to hardware faults, (e.g., *ec_hw_errors*) that are observed in the external logs. Such hardware bugs may get incited only when specific modules of the code are executed (e.g. row hammer [23] attacks on DRAM).
2. Many segmentation faults originate from applications. Programs invoking page allocation requests often cause nodes to fail due to memory limits.
3. Software traps (e.g., invalid opcode) generally do not fail nodes, unless exception handling disturbs the file system, which may eventually fail the node.
4. Disk and job induced inode errors can render a file system inaccessible, causing nodes to be slow or non-responsive. Finer analysis of such bugs indicates the root cause to be at the application, though the failures manifest inside the OS kernel.

Even though job-specific malfunctioning is not indicated in the node failure logs upfront, the root cause often lies in the application. For Lustre or kernel bugs, several kernel oops, firmware bugs, and fork or memory allocation errors, the original fault propagates from the job running on the compute nodes. The high-level breakdown of the failure causes has been discussed in the literature [17], [28] and is consistent with Table IV column 1. Additionally, our work affirms most of such root causes to be application-triggered by analyzing the specific system modules present in the stack traces. While some of the root causes can help create fault-aware solutions (e.g., CPU corruptions, MCEs), several high impact tangible indications happen only during application runtime (e.g., invalid opcode, segmentation fault, resource exhaustion). This calls for better performance-aware scheduling in HPC systems.

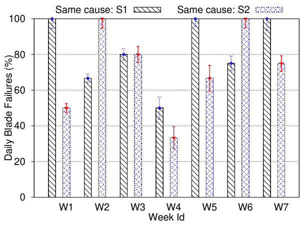

Fig. 18: Blade Failures

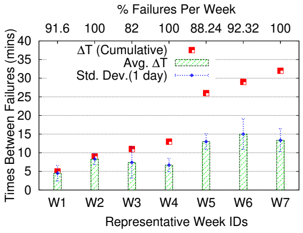

Fig. 19: Temporal Locality

Table IV indicates the failure causes w.r.t. the predominant kernel modules reported by stack backtraces. Since kernel oops are frequently observed, we examined the preliminary calltraces indicating the modules linked to the trace such as *dvs_ipc_mesg*, *mce_log* etc. While few modules clearly refer to file system problems, *ldml_bl* and *sleep_on_page* are job-triggered. Hang or concurrency bugs due to application code cause kernel panics. We did not investigate the entire trace, but there are indications of application-caused (which in turn may affect the file system) versus file system-caused failures.

TABLE IV. Failure Causes and Stack Modules

| Reasons | Stack Trace Modules |
|---|---|
| SegFault, Page Fault, MCEs, Application, Kernel, FileSystem Bugs, Corrupt CPUs | sleep_on_page, ldml_bl, dvs_ipc_mesg/lnet_mapuvm, mce_log, rwsem_down_failed |

For S3 over 4 months, hardware faults (BIOS/disk errors, MCEs) contribute to 37% of the failures, software faults (Kernel/Lustre bugs) contribute to 32%, and applications account for 31%. 27% of the failures were caused by memory exhaustion. In terms of failure locality, we observed two primary trends in the context of root cause:

1. Days with multiple node failures, spatially diverse in physical location, usually fail with different root cause, unless they are correlated in terms of sharing the same job executable.
2. Multiple blade failures are often caused by the same application executing on those nodes around the failure time. Blade failures display temporal locality in systems with similar failure reasons with short (≈5 mins) inter-node failure times.

Figure 18 depicts the fraction of blade failures with the same failure reasons for S1 and S2 over 7 different weeks. Most days experience single or multiple node failures on diverse blades. Considering blades with all failed nodes, the manifested symptoms often appear to be similar. On certain days, all blade failures were due to the same hardware faults and application-triggered software faults. For both S1 and S2, errors are less than ±7.2 hinting at the consistent temporal locality of nodes in terms of the root cause. When blades fail at the granularity of $\mu$secs, they share the same system malfunction as the root cause. Figure 19 indicates that the MTBFs over 7 specific weeks in S3 for job-triggered failures does not exceed 32 minutes. As evident, W1 encounters on average 91.6% of the failures within 5 mins. W6 and W7 experienced more than 90% of the failures within 29 to 32 mins. Temporal locality (same job) caused failures are well evident. These are much shorter than the MTBFs observed in LANL systems in prior work [36] (> 5 hours).

**Observation 7**: *Finer analysis implies that the origin of many bugs lies in the application. Performance-aware job scheduling, and stack trace analysis in conjunction with failure prediction schemes has the potential to improve system health.*

**Observation 8**: *Nodes sharing an application often fail during similar times. These nodes may belong to different blades*

TABLE V. Sample Failure Cases

| # | #Failures | Internal Indicators | External Indicators | Root Cause Inference |
|---|---|---|---|---|
| 1 | 1 failure | L0_sysd_MCE followed by NHC warnings, other nodes of the same blade encountered correctable H/W errors and SSID errors | No environmental indications or job malfunctioning reported around the failure time | Potential root cause could not be deduced |
| 2 | 3 failures | Neither spatially nor temporally close (4 am, 12.38 pm and 3.21 pm), however, similar patterns (H/W error → MCEs → kernel oops) | Aries link error and temperature threshold violations distant from the failure time, no job malfunctioning indications | CPU corruptions and MCEs affecting the file system causing failure |
| 3 | 6 failures | oom-killer invoked → kernel oops (app-based call trace) with similar times and patterns on all nodes | No external indications around the failure time, same application running on all the nodes | Application-caused memory exhaustion, nodes fail NHC tests leading to failure |
| 4 | 1 failure | LustreErrors → Unable to handle kernel paging request, other nodes of the blade did not fail | Link errors, temp. threshold violations distant from the failure time, scheduled job aborted | Application-triggered file system bug causing failure |
| 5 | 1 failure | H/W MCEs → critical errors, other nodes of the blade encountered benign events | Early indicators of ec_hw_errors & link errors prior to the failure time, no job errors evident | Fail-slow symptoms of memory failing the node (degraded h/w triggered by s/w) |

*(spatially distant), but exhibit temporal locality w.r.t. the root cause. Tracking buggy application IDs (APIDs) and job abortions can prevent multiple node failures in such cases.*

**Unknown Causes**: During our analysis, we could not infer potential root causes for three failure patterns. First,
*Type:2; Severity:80; Class:3; Subclass:D; Operation: 2*
may indicate BIOS problems. However, these are commonly seen in the systems for benign healthy cases as well. Several nodes encountered anomalous shutdowns with these errors without any other helpful patterns. Application or hardware do not show faults that influence these failures. It is not clear what conditions could trigger failures with this error.

Second, *L0_sysd_mce* related to memory errors appear before nodes fail. However, insufficient information prohibited us to understand what causes these to appear. During failure times, no other nodes in the blade fail, and there is no correlation over time. The event name implies hardware problems related to the blade controller (L0→BC MCEs), but the observed symptoms did not unveil anything helpful as we aim at more decisive fine-grained causal inference.

Finally, there are failures with simply shutdown messages or no prior anomaly symptoms. The affect of cosmic radiations on hardware and silent data corruptions have been shown in the research literature [5], [11], [16]. We suspect such solar flares could cause nodes to fail, which are undetectable in the logs. We could not examine the weather reports for such uncertain failures, hence this remains speculative. What is more probable is operator error, i.e., manual shutdowns of good nodes by accident (e.g., wrong button, IPMI command).

**Observation 9**: *Several failures have insufficient information, where we cannot deduce any potential root cause. Such cases may or may not relate to known operator errors or rare cosmic radiations. These cases require more investigation for root cause inference subject to operator-level or vendor support.*

### G. Case Studies

Let us briefly analyze 5 cases of root cause inference giving inklings of problems arising in production HPC. The abridged analysis of these failures is summarized in Table V.

**Case 1**: A single node failure occurs with *L0_sysd_mce* errors in the console logs without additional hardware, software, or application problems. The other nodes in the same blade do not fail and experience correctable hardware MCEs (which are mostly benign) and hardware SSID errors. No additional external influence or job errors are reported around the failure time. These indications did not lead to potential root cause and do not suffice to understand why the node was shutdown.

**Case 2**: 3 nodes from different blades fail hours apart with similar internal failure patterns and no temporal correlations. All 3 nodes encounter hardware errors followed by MCEs and kernel oops. Interconnect errors and temperature threshold violations are present in the SEDC logs, but not around the failure time. No application problems are reported. Processor corruptions and critical MCEs turn out to be the root cause.

**Case 3**: 6 nodes fail around similar times (seconds to minutes apart) with similar node internal failure patterns. We confirmed that the nodes were running the same job around the failure time. No environmental faults were reported. *oom-killer* was invoked because of memory exhaustion. Several processes were killed followed by kernel oops. The modules linked to the call trace were indicative of the running application and NHC warnings were observed. This is a resource exhaustion caused failure with no additional hardware problems.

**Case 4**: A failure occurs with a kernel bug unable to fulfill a paging request preceded by Lustre errors. External indicators include interconnect errors and temperature threshold violations, distant from the failure time. The job running on this node did not terminate gracefully. In this case, the application triggered a file system bug eventually failing the node.

**Case 5**: A node fails with critical MCEs with no kernel oops or major software bugs. In the external logs, *ec_hw_errors* and link errors were reported several minutes before the failure time. For the other blades such sustained environmental errors around that time were absent. No application misbehavior was evident. This case is a hardware caused failure with fail-slow symptoms and feasible lead time enhancements.

**Discussion**: From these results we infer that a generic approach with a formal algorithm is unsuitable as a means to identify failure causes. Our statistical inference over 4 production clusters helps to become cognizant of the extent to which the external factors combined with node-specific logs affect the systems. The fact that major external health faults are not the primary culprits of failures is not obvious (e.g., temperature variability influence node reliability in the data centers [12], [38]). Besides, the stack trace logs indicating the associated modules (hints to root cause) and job-triggered failures (that can be *indirectly* caused) imply the importance of application diagnosis [40]. Further effective action items are

TABLE VI. Findings and Recommendations

| # | **Major Findings** | **Suggested Recommendations** |
|---|---|---|
| 1 | While higher error count need not always fail nodes, certain faults (e.g., NVF) and short-term multiple blade failures often indicate unhealthy system state. Several daily failures relate to similar root causes | Non-critical health faults (e.g., NHF) and temporal locality of failures can be considered before launching checkpoint/restarts making reactive approaches more aware of the potential root cause |
| 2 | Major Blade and Cabinet level health indicators are not strongly correlated with the primary root cause | Frequent appearance of SEDC warning and threshold violations can be ignored unless major indicators are observed in the node internal logs |
| 3 | Fail-slow hardware symptoms exist for certain software triggered hardware failures aiding in lead time improvements | Node failure prediction schemes can incorporate external correlations for possible lead time enhancements for proactive fault tolerance |
| 4 | Node quarantining can be ineffective when the root cause is triggered by application misbehavior | Instead of sequestering nodes, users can be intimated about their malfunctioning job (by cluster operators) or buggy jobs can be blocked (by NHC) |
| 5 | Across all the systems, a considerable number of node failures involve Kernel oops with long stack traces. These can be triggered by the hardware, software, and application based on the fault propagation chain | Conducting a machine learning guided study of call traces from large-scale systems to narrow down the buggy code or function emanating from the application or file system can segregate job-triggered (which in turn affect the file system or produce driver bugs) versus job-caused failures |
| 6 | Spatial-temporal correlations of node failures exist w.r.t. the application-caused failures; Jobs can trigger filesystem/interconnect errors without failing nodes | System administrators can incorporate additional health tests in NHCs to account for the nodes failing incessantly due to abnormal application exit to track the buggy APID besides *rebooting* or turning the node to *admindown* |
| 7 | A significant number of failures are primarily triggered by the applications, which in turn may affect the file system or hardware | Application resilience schemes (performance diagnosis) can be used in conjunction with system failure prediction tools to infer future system health |

TABLE VII. Large-scale System Evaluation

| **Studies** | **Focus** | **Some Findings** |
|---|---|---|
| [6], [17], [21], [34] | Overall System | Regime detection, Wasted time analysis, Integrated monitoring infrastructure design, Reliability consistency over time, Failure categorization |
| [16], [38] | Server Failures, Hardware Faults | Spatio-Temporal distribution, Root causes: Hard disk, Raid controller and Memory faults, Fail-slow symptoms, Environmental causes |
| [3], [5], [12] | Disk, DRAM, SSDs | Multi-bit error corruption, , Temp. and Energy influences, Volatility of DRAM error rates, Infant drives with easier predictive features fail more |
| [14], [22], [25], [31], [40] | Interconnect, Soft Errors, GPUs | Load imbalance effect on lane degrades, SWOs, Failover, Lane recovery impact, Radiation expts. and GPU temp. sensitivity, scheduling |
| [10], [26], [27], [29], [32] | Application, Jobs | Resource contention exists; Increase job wait times; Jobs with higher CPU usage are more fallible; Time-out jobs with high run times consume higher core-hours; H/W errors concentrated on few jobs/nodes/users |
| [11], [28] | Node Failures | Power, Temperature effects on LANL HPC Nodes, Root cause breakdown of SWOs and node failures for Blue Waters |
| Our work | Node Failures | Fine-grained external influence analysis, Faults not leading to failures, Feasible lead time enhancements for 4 production systems |

TABLE VIII. Comparison

| # | **Features** | **Our Work** | **[28]** | **[16]** | **[11]** |
|---|---|---|---|---|---|
| 1 | Root Cause | ✓ | ✓ | ✓ | ✓ |
| 2 | Node Failures | ✓ | ✓ | × | ✓ |
| 3 | Stack Trace | ✓ | × | × | × |
| 4 | External/Job Correlations | ✓ | ✓ | × | ✓ |
| 5 | Fail-slow Symptoms | ✓ | × | ✓ | × |
| 6 | Lead Time | ✓ | × | × | × |
| 7 | Cloud | × | × | ✓ | × |
| 8 | HPC | ✓ | ✓ | ✓ | ✓ |

subject to developmental efforts with the production clusters. Table VI summarizes our findings with recommendations.

We specifically observe that environmental indications are weakly correlated to failures and the existence of fail-slow symptoms. While operators can be less concerned about external health warnings in the absence of internal faults, it is better to equip the health checkers, failure prediction and checkpoint/restart (C/R) schemes to be aware of early indicators of health faults and malfunctioning jobs to reduce recomputation cost. Instead of quarantining a node, buggy jobs can be monitored with intimation to the user. A full stack trace can provide finer details about the file system and application errors. Automated diagnosis tools for anomaly inference from call traces [4] can further help in pinpointing the root cause. These recommendations are suitable for HPC systems with extreme heterogeneity as well, since augmented hardware (e.g., accelerators) or complex workload characteristics (e.g., deep learning on multi-core processors) can also benefit from performance-aware scheduling or stack trace analysis.

**Comparative Analysis**: Table VII encapsulates major failure analysis studies performed for large-scale systems. Table VIII compares our work with three related studies. [16] studied detailed hardware faults for 12 production clusters with anecdotal evidences without any empirical analysis. [28] performed statistical analysis without external correlations on the Blue Waters. [11] studied non-Cray systems focusing on power and temperature unlike ours. None of these address lead time enhancements. Our study is based on 5 contemporary systems providing lessons learned through assiduous environmental correlations and stack trace diagnosis.

## IV. Related Work

[14], [22], [25], [31], [33], [40] efficiently schedule jobs and characterize power consumption, soft errors, GPU, and interconnect faults. [17], [27], [34] conduct general system fault measurements and study the relationship between hardware errors and job logs. [10], [26], [32] characterize job failures on non-Cray systems, and [41] point to the limitations of DRAM error characterization. These focus on specific components or layers and affirm the need for a holistic study to understand how node failures happen from a system-wide perspective [20]. [20], [21], [30] perform holistic application monitoring and survey of failure analysis in petascale systems identifying existing gaps. A number of diagnosis techniques [13], [28], [39] either point out high-level failure layer or perform causal analysis without holistic considerations or are application-centric. In contrast, we include controller and event logs to understand external influences and quantify lead times increments. Recently, machine learning (ML)-guided failure prediction schemes [9], [24] have been developed to proactively respond to failures. Our work complements such efforts in providing tenable resilience schemes.

## V. Conclusion

We present node failure diagnosis observed in production HPC systems based on correlations of internal and external logs. We recognize job characteristics that cause nodes to fail and provide an estimate of feasible lead time increments. Our work identifies that environmental influences are not strongly correlated to node failures. Choosing a mitigation action with an understanding of the root cause when a failure is imminent can have long-term benefits in progressive computation instead of restarting from checkpoints, which requires recomputation.

## Acknowledgment

The authors are grateful to the reviewers for their helpful feedback. This work was performed in part under the auspices of the U.S. DOE by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, Lawrence Berkeley National Lab, and NSF grants 1525609 and 0958311.

## References

[1] *Cray Documentation*. [Online]. Available: https://pubs.cray.com/

[2] *Cray User Group*. [Online]. Available: https://cug.org/

[3] J. Alter, J. Xue, A. Dimnaku, and E. Smirni, "SSD failures in the field: symptoms, causes, and prediction models," in *SC*, 2019, pp. 75:1–75:14.

[4] D. C. Arnold, D. H. Ahn, B. R. de Supinski, G. L. Lee, B. P. Miller, and M. Schulz, "Stack trace analysis for large scale debugging," in *IEEE IPDPS*, 2007, pp. 1–10.

[5] L. Bautista-Gomez, F. Zyulkyarov, O. Unsal, and S. McIntosh-Smith, "Unprotected computing: A large-scale study of dram raw error rate on a supercomputer," in *SC*. IEEE Press, 2016, p. 55.

[6] L. A. Bautista-Gomez, A. Gainaru, S. Perarnau, D. Tiwari, S. Gupta, C. Engelmann, F. Cappello, and M. Snir, "Reducing waste in extreme scale systems through introspective analysis," in *IEEE IPDPS*, 2016.

[7] E. Cheng, J. Abraham, P. Bose, A. Buyuktosunoglu, D. Chen, H. Cho, Y. Li, U. Sharif, K. Skadron, M. Stan *et al.*, "Cross-layer resilience: Challenges, insights, and the road ahead," in *IEEE/ACM DAC*, 2019.

[8] E. Chuah, A. Jhumka, S. Alt, J. J. Villalobos, J. Fryman, W. Barth, and M. Parashar, "Using resource use data and system logs for HPC system error propagation and recovery diagnosis," in *IEEE ISPA/BDCloud/SocialCom/SustainCom*, 2019, pp. 458–467.

[9] A. Das, F. Mueller, P. Hargrove, E. Roman, and S. B. Baden, "Doomsday: predicting which node will fail when on supercomputers," in *SC*, 2018, pp. 9:1–9:14.

[10] S. Di, H. Guo, E. Pershey, M. Snir, and F. Cappello, "Characterizing and understanding HPC job failures over the 2k-day life of IBM bluegene/q system," in *IEEE/IFIP DSN*, 2019, pp. 473–484.

[11] N. El-Sayed and B. Schroeder, "Reading between the lines of failure logs: Understanding how HPC systems fail," in *IEEE DSN*, 2013.

[12] N. El-Sayed, I. A. Stefanovici, G. Amvrosiadis, A. A. Hwang, and B. Schroeder, "Temperature management in data centers: why some (might) like it hot," in *ACM SIGMETRICS*, 2012, pp. 163–174.

[13] X. Fu, R. Ren, S. A. McKee, J. Zhan, and N. Sun, "Digging deeper into cluster system logs for failure prediction and root cause diagnosis," in *IEEE CLUSTER*, 2014, pp. 103–112.

[14] R. Garg, T. Patel, G. Cooperman, and D. Tiwari, "Shiraz: Exploiting system reliability and application resilience characteristics to improve large scale system throughput," in *IEEE/IFIP DSN*, 2018, pp. 83–94.

[15] P. Garraghan, I. S. Moreno, P. Townend, and J. Xu, "An analysis of failure-related energy waste in a large-scale cloud environment," *IEEE Trans. Emerging Topics Comput.*, vol. 2, no. 2, pp. 166–180, 2014.

[16] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, G. Grider, P. M. Fields, K. Harms, R. B. Ross, A. Jacobson, R. Ricci, K. Webb, P. Alvaro, H. B. Runesha, M. Hao, and H. Li, "Fail-slow at scale: Evidence of hardware performance faults in large production systems," in *USENIX FAST*, 2018, pp. 1–14.

[17] S. Gupta, T. Patel, C. Engelmann, and D. Tiwari, "Failures in large scale systems: long-term measurement, analysis, and implications," in *SC*, 2017, pp. 44:1–44:12.

[18] S. Gupta, D. Tiwari, C. Jantzi, J. H. Rogers, and D. Maxwell, "Understanding and exploiting spatial properties of system failures on extreme-scale HPC systems," in *IEEE/IFIP DSN*, 2015, pp. 37–44.

[19] R. Izadpanah, N. Naksinehaboon, J. M. Brandt, A. C. Gentile, and D. Dechev, "Integrating low-latency analysis into HPC system monitoring," in *ICPP*, 2018, pp. 5:1–5:10.

[20] D. Jauk, D. Yang, and M. Schulz, "Predicting faults in high performance computing systems: an in-depth survey of the state-of-the-practice," in *SC*, 2019, pp. 30:1–30:13.

[21] S. Jha, J. M. Brandt, A. C. Gentile, Z. Kalbarczyk, G. H. Bauer, J. Enos, M. T. Showerman, L. Kaplan, B. Bode, A. Greiner, A. Bonnie, M. Mason, R. K. Iyer, and W. Kramer, "Holistic measurement-driven system assessment," in *IEEE CLUSTER*, 2017, pp. 797–800.

[22] S. Jha, V. Formicola, C. D. Martino, M. Dalton, W. T. Kramer, Z. Kalbarczyk, and R. K. Iyer, "Resiliency of HPC interconnects: A case study of interconnect failures and recovery in blue waters," *IEEE Trans. Dependable Sec. Comput.*, vol. 15, no. 6, pp. 915–930, 2018.

[23] Y. Kim, R. Daly, J. Kim, C. Fallin, J. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *ACM/IEEE ISCA*, 2014, pp. 361–372.

[24] J. Klinkenberg, C. Terboven, S. Lankes, and M. S. Müller, "Data mining-based analysis of HPC center operations," in *IEEE CLUSTER*, 2017.

[25] M. Kumar, S. Gupta, T. Patel, M. Wilder, W. Shi, S. Fu, C. Engelmann, and D. Tiwari, "Understanding and analyzing interconnect errors and network congestion on a large scale HPC system," in *IEEE/IFIP DSN*, 2018, pp. 107–114.

[26] R. Kumar, S. Jha, A. Mahgoub, R. Kalyanam, S. L. Harrell, and X. Carol, "The mystery of the failing jobs: Insights from operational data from two university-wide computing systems," in *IEEE DSN*, 2020.

[27] S.-H. Lim, R. G. Miller, and S. S. Vazhkudai, "Understanding the interplay between hardware errors and user job characteristics on the Titan supercomputer," in *IEEE IPDPS*, 2020, pp. 180–190.

[28] C. D. Martino, Z. T. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer, "Lessons learned from the analysis of system failures at petascale: The case of blue waters," in *IEEE/IFIP DSN*, 2014.

[29] C. D. Martino, W. Kramer, Z. Kalbarczyk, and R. K. Iyer, "Measuring and understanding extreme-scale application resilience: A field study of 5, 000, 000 HPC application runs," in *IEEE/IFIP DSN*, 2015, pp. 25–36.

[30] A. Netti, M. Mueller, C. Guillen, M. Ott, D. Tafani, G. Ozer, and M. Schulz, "DCDB wintermute: Enabling online and holistic operational data analytics on HPC systems," in *ACM HPDC*, 2020.

[31] B. Nie, D. Tiwari, S. Gupta, E. Smirni, and J. H. Rogers, "A large-scale study of soft-errors on GPUs in the field," in *IEEE HPCA*, 2016.

[32] T. Patel, Z. Liu, R. Kettimuthu, P. Rich, W. Allcock, and D. Tiwari, "Job characteristics on large-scale systems: long-term analysis, quantification, and implications," in *ACM/IEEE SC*, 2020, pp. 1–17.

[33] T. Patel, A. Wagenhäuser, C. Eibel, T. Hönig, T. Zeiser, and D. Tiwari, "What does power consumption behavior of HPC jobs reveal?: Demystifying, quantifying, and predicting power consumption characteristics," in *IEEE IPDPS*, 2020, pp. 799–809.

[34] E. Rojas, E. Meneses, T. Jones, and D. Maxwell, "Analyzing a five-year failure record of a leadership-class supercomputer," in *SBAC-PAD*. IEEE, 2019, pp. 196–203.

[35] J. Stearley, R. Ballance, and L. Bauman, "A state-machine approach to disambiguating supercomputer event logs," in *Workshop on Managing Systems Automatically and Dynamically, MAD*, 2012.

[36] D. Tiwari, S. Gupta, and S. S. Vazhkudai, "Lazy checkpointing: Exploiting temporal locality in failures to mitigate checkpointing overheads on extreme-scale systems," in *IEEE/IFIP DSN*, 2014, pp. 25–36.

[37] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. M. Brandt, V. J. Leung, M. Egele, and A. K. Coskun, "Online diagnosis of performance variation in HPC systems using machine learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 4, pp. 883–896, 2019.

[38] G. Wang, L. Zhang, and W. Xu, "What can we learn from four years of data center hardware failures?" in *IEEE/IFIP DSN*, 2017, pp. 25–36.

[39] Z. Zheng, L. Yu, Z. Lan, and T. Jones, "3-dimensional root cause diagnosis via co-analysis," in *ICAC*, 2012, pp. 181–190.

[40] C. Zimmer, D. Maxwell, S. McNally, S. Atchley, and S. S. Vazhkudai, "GPU age-aware scheduling to improve the reliability of leadership jobs on Titan," in *SC*, 2018, pp. 7:1–7:11.

[41] D. Zivanovic, P. E. Dokht, S. Moré, J. Bartolome, P. M. Carpenter, P. Radojković, and E. Ayguadé, "DRAM errors in the field: a statistical approach," in *MEMSYS*. ACM, 2019, pp. 69–84.