Understanding the Interplay between Hardware Errors and User Job Characteristics on the Titan Supercomputer

Seung-Hwan Lim
Oak Ridge National Laboratory
Oak Ridge, TN 37831
lims1@ornl.gov

Ross G. Miller
Oak Ridge National Laboratory
Oak Ridge, TN 37831
rgmiller@ornl.gov

Sudharshan S. Vazhkudai
Oak Ridge National Laboratory
Oak Ridge, TN 37831
vazhkudaiss@ornl.gov

***Abstract*—Designing dependable supercomputers begins with an understanding of errors in real-world, large-scale systems. The Titan supercomputer at Oak Ridge National Laboratory provides a unique opportunity to investigate errors when an actual system is actively used by multiple concurrent users and workloads from diverse domains at varying scales. This study presents a thorough analysis of $6,908,497$ hardware errors from $18,688$ compute nodes of Titan for $312,215$ user jobs over a 3-year time period. Through careful joining of two system logs - the Machine Check Architecture (MCA) log and the job scheduler log - we show the correlated pattern of hardware errors for each job and user, in addition to individual descriptive statistics of errors, jobs, and users. Since the majority of hardware errors are memory errors, this study also shows the importance of error correcting in memory systems.**

I. INTRODUCTION

The thorough understanding of errors in real-world, large-scale systems is critical to designing dependable supercomputers. The general consensus is that error patterns are related to multiple factors such as system components, applications, system operation (e.g. job scheduling), and user behaviors [7], [13], [21]. Thus, given hardware and software system components, the understanding of error patterns under various applications and users is of historical importance [11]. In this context, high performance computing (HPC) systems like the 17.59 petaflops Titan supercomputer [1] (No. 9 in the Top500 list) hosted at Oak Ridge National Laboratory are sitting in a unique position.

Titan provides computing resources to thousands of diverse users from government, business, and academia. They run diverse workloads ranging from traditional scientific simulation workloads to modern machine learning applications from more than 30 science domains [9]. Additionally, hardware accelerators such as GPUs are becoming more common in the HPC world and Titan is one of the largest systems to use GPUs. Thus, understanding errors from Titan will be a useful guide for building dependable future computing systems in CPU-GPU heterogeneous architecture for a wide range of purposes.

Analyzing system errors has a rich history in the literature [7], [11], [19]–[21], including studies on large scale distributed systems [4]. System error analysis often relies on collected system logs such as CPU hardware error logs [7] and GPU error logs [20]. Additionally, merging logs from multiple log collection systems is also attempted [11], [16], since the analysis on the merged logs uncovers the interaction among system errors, workloads, the user behavior, and system operations. This study is similar to other studies that present an analysis based upon merging hardware errors from CPUs and user job submission logs. However, this study reveals the error pattern of Titan supercomputer after a series of innovations presented in previous studies [12], [21], [22] to improve system performance and system utilization, along with a large scale replacement of failing hardware components (e.g., GPUs).

This study collected hardware errors using the machine check architecture (MCA) provided by AMD, the manufacturer of Titan's CPUs. MCA provides detailed information about hardware errors at various levels, including errors from in-processor caches, DRAM memory, and the inter-core bus. In order to understand the interplay between the hardware errors and jobs and users, we merged this MCA log with the job submission log, collected from the system job scheduler. The job submission log provides information about the user id (UID), the starting time, the end time, and the list of nodes assigned to the job. Since the available job log does not provide information on the exit code or associated hardware errors, we cannot definitively conclude that any particular hardware errors led to job failure. However, merging the job submission logs and MCA logs does allow us to understand the interaction among errors, nodes, users, and jobs.

This study discovers the following notable findings:

- Daily errors are mostly limited to a small set of nodes - less than 400 out of $18,688$ compute nodes. However,

This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

this set changes over the time. Thus, we cannot attribute the overall error trends to specific faulty nodes.

- The distribution of error intervals from all the errors form multivariate distribution, which are associated with the source of errors in the error status code.
- 99.97% of errors in Titan supercomputer were correctable ECC errors, which means that ECC-enabled DRAM can significantly raise the system reliability. On the other hand, an algorithm that can tolerate single bit ECC errors such as low-precision algorithms will be able to significantly raise the reliability without hardware support [3].
- We have found weak correlation between critical uncorrectable errors and other kinds of hardware errors.
- Errors in a node are more associated with user characteristics than the characteristics of nodes and jobs under the current operational practices in production high performance computing systems.

We describe the target system that we consider, along with relevant background and related work, in Section II. After reviewing the background and literature, we present a brief overview of the analysis in Section III, followed by detailed analysis in Section IV.

II. Related work

This section describes a few important studies on the reliable operation of high performance systems, though analysis results in this study can be relevant to other types of computing systems. Meneses et al., [11] correlate failures from the system error log to the job scheduler log, providing a user's perspective on system errors. Their analysis is based upon logs collected in 2014. The authors profiled the system failures in two categories: software and hardware, with a further investigation on the correlation between job submission frequency and the failure rate and its impact on the workloads due to system failures.

Zimmer et al., [21] studied the job scheduling log to understand the relationship between job placement and application execution time variability. They augmented the existing job placement system with an enhanced node-layout strategy. Their job placement system reduced the average hop count between nodes by up to 50%, resulting in an average runtime improvement of 10% for the measured applications.

Gupta et al., [4] studied the reliability characteristics of multiple large-scale HPC production systems hosted at Oak Ridge National Lab, covering more than one billion compute node hours across five different systems over a period of 8 years. This study analyzed the syslog data, which includes machine check exception. Although their study discussed a potential implication of job failures from system failures, they did not provide an in-depth analysis on the relationship between failures and job execution. One of the notable findings in [4] is that hardware errors like machine check exception errors are the major class of errors across systems over the whole observed period. This fact motivated our study on the understanding of the relationship between machine check error (e.g., uncorrectable memory errors) and job execution.

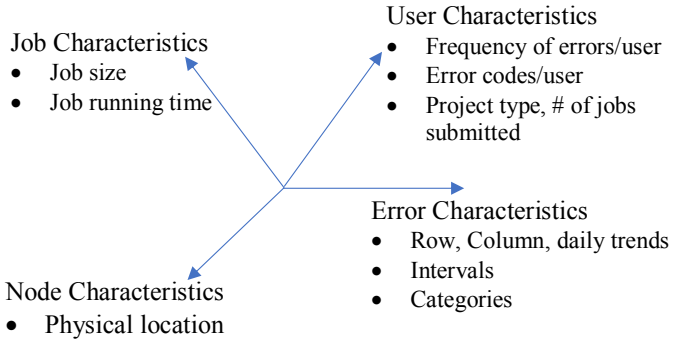

Fig. 1: The dimensions across which the logs are analyzed.

Levy et. al., [7] provide detailed analysis of correctable and uncorrectable memory errors over the entire lifetime of a large-scale system - Cielo - at Los Alamos National Lab. For memory failure logs, their study used logs recorded by the same machine check architecture (MCA) that is used in this study. In addition to the MCA log, they used the resource management log from the application level placement scheduler (ALPS), the kernel panic log, and the hardware inventory log. Key findings in this study are: 1) memory errors do not increase over the time and; 2) correctable errors are not precursors of uncorrectable errors in production systems. However, their study still did not look into the relationship between memory errors and jobs.

Tiwari et al., [20] present an in-depth analysis of GPU errors in a large-scale system with GPUs, using console logs for errors with timestamps and system utility (e.g. nvidia-smi). In their study, single bit errors were predominant, similar to our study. Simarly, Nie et al., [14] studied GPU errors, using machine learning models with a focus on predicting single bit errors. However, both of those studies only analyzed error logs for a single type of hardware components - the GPU - without joining the error logs with job logs. Thus, neither study clearly addressed the impact of errors on actual user applications and workloads.

Park et. al., [16] presented case studies from multiple joined logs such as machine check exception errors; job scheduler logs; reliability, availability and serviceability (RAS) logs from various sensors; and parallel file system logs. Due to the challenges in joining multiple large logs, they implemented a Spark-based big-data analytic infrastructure called LogSCAN. Their analysis usecases include the correlation of job executions and errors, which is similar to the analysis presented in this study. However, their study focused on establishing the data analytics infrastructure, rather than resulting analysis.

III. Methodology

This section provides a brief overview of analysis and data, along with relevant information about the system and error collection methods.

A. Analysis overview

This study is organized into three dimensions based on the idea that errors may stem from the characteristics of nodes, jobs and users. (Figure 1) First, errors may exhibit their own unique spatio-temporal patterns such as frequency of errors and intervals between errors. These patterns may vary according to the characteristics of nodes. A node is a combination of multiple hardware and software components, and the reliability of those components may vary from node to node due to environmental factors such as thermal differences and differing usage caused by, for example, network routing and scheduling preference. We present a detailed analysis of error characteristics and study if it varies based on compute node differences.

In addition, error characteristics may vary based on the jobs being run. Each job may represent a specific application program with a set of input parameters and output data. In distributed environments, each application may show different communication patterns that can lead to a certain distribution of errors among nodes allocated to a job. For example, if a particular process is coordinating communications for all processes in a job, the node running that process might be more vulnerable to errors. We study if error patterns are related to major job features.

Finally, each user can have different ways of executing applications, though they run the same program with other users. In addition, during the lifetime of a project, each user may have different patterns of resource allocation and running time for different sets of applications. We study the relationship between user behavior and errors. The following analysis is focused on understanding if the above hypotheses can be confirmed.

B. Titan supercomputer

The Titan supercomputer is an 18,688 node 17.59 petaflop Cray XK7 system. It was the primary compute platform for the Oak Ridge Leadership Computing Facility (OLCF) up through the end of 2018 and is expected to continue in service for most of 2019.Each compute node consists of a 16-core AMD Opteron CPU with 32 GB of memory and an NVIDIA K20X GPU (Kepler G110 processor) with 6 GB of memory. Titan's compute nodes are connected via a high-performance Gemini interconnection network, which is a 3D torus with XYZ dimensions of $25 \times 16 \times 24$. Since 3D networks are composed of many small routing devices, multiple routing choices are possible between a pair of compute nodes. Physically, compute nodes are organized by 8×25 cabinets, where each cabinet consist of 3 vertical cages. Each cage has 8 circuit boards, or slots. Each slot has 4 compute nodes on it. The OLCF runs around 500 jobs every day on Titan ($\sim$182,000 jobs per year). Jobs are scheduled using the Application Level Placement Scheduler (ALPS) and the MOAB scheduler. ALPS is responsible for enumerating all of the nodes within the system and creating a network-aware ordered list. MOAB handles both job scheduling and resource allocation. Scheduled jobs are provided resources in order from the list created by ALPS.

Attributes	Values
job monitoring period	2018-02-09 to 2018-08-06
error monitoring period	2015-01-27 to 2018-08-06
job entries	312,215
error entries	6,908,297
jobs with errors	24,337
errors during the job monitoring period	1,096,666
errors when jobs run	851,611
# of users	342

TABLE I: Facts of data sets

C. Log collection method and collected errors

For this study, two types of data were gathered: Machine Check Exception errors and job start & end records. Both types of data are logged via standard syslog processes. On Titan, all syslog processes forward messages to a central server which then forwards the messages to a Splunk [2] server. Due to disk space constraints, data in the main Splunk index is only kept for a few months. Thus, every two months, we would extract the latest data from Splunk and save it offline for later analysis.

The Machine Check Architecture (MCA) plays a vital role in the reliability, availability, and serviceability of AMD processors, as well as computer systems based on them. It logs *hardware* and *processor* errors and reports them to system software. It logs the error in specific registers (banks) and then triggers a machine-check exception to notify the system software.

When the MCA detects an error, hardware autonomously acts to either correct the error or contain the propagation of the corrupting effects of an uncorrected error. In other words, there are three categories of hardware errors: *corrected*, *uncorrected*, and *deferred*. If an error can be *corrected* by hardware, no immediate action by software is required. In this case, information is logged, if enabled, to aid in later diagnosis and possible repair. If correction by hardware is not possible, the error is classified as *uncorrected*. As hardware cannot correct uncorrected errors, system software attempts to correct the error and resume the user-level application. If the error cannot be corrected in software, then the system software determines the impact of the error. If the impact of an uncorrected error can be contained, it is classified by hardware as a *deferred* error. Deferred errors are logged, but not immediately reported via a machine check exception.

A common example of deferred error processing and localization is the conversion of globally uncorrected DRAM errors to process-specific consumed memory errors. In this example, uncorrected ECC-protected data that has not yet been consumed by any processor core is tagged as "poison." The hardware reports the uncorrected data as a localized error via a machine check exception only when it is about to be used by an instruction execution stream. In contrast, an error that cannot be contained and is of such severity that it has compromised the continued operation of a processor core, requires immediate action to terminate system processing and may result in a hardware-enforced *shutdown.*

Register Bank	Error type	Error subtype	Description	Counts
Bank0	Data cache error	Data array	load-store unit (LS), including data cache	84
	Tag error	Tag array		8
Bank1	IC read error	IC data load parity	instruction fetch unit (IF), including instruction cache	370
	IC read error	Main tag		39
	Tag probe	Probe tag error		28
	Tag probe	Probe tag valid bit		1
Bank2	L2 Cache	Fill parity error	combined unit (CU), including L2 cache	951
	L2 Cache	FillECC/L1		130,084
	L2 Cache	FillECC/L2		21,130
	L2 Cache	PrqData parity error		1
	L2 Cache	VbData ECC error		24,545
	L2 Cache	VbData Parity error		63
	L2 Cache	WCCTag ECC error		10,165
	Tag	L2Tag		1379
Bank4	Compute unit data error	-	northbridge (NB)	2,171,095
	Link data error	-		4,548,314
	Sync error	-		11
Bank5	Internal error	AG payload array parity	execution unit (EX)	1
	Internal Error	AG1PRF parity		25
	Internal Error	IDRF array parity		1

TABLE II: Error types and subtypes, with associated register banks.

D. Data Overview

Table I describes the summary statistics of the analyzed logs. Since the different logs cover different time frames, we must acknowledge the differences between the job monitoring period (∼6 months) and error monitoring period (∼43 months). The job monitoring period is included in the error monitoring period. We have a total of 7 million errors, while 1 million errors during the job monitoring period. A notable statistic is that 92.3% of jobs do not have a single error, while 88% of errors occurred when jobs are running on the node. We also find that errors occurred on 17,656 nodes out of 18,688 nodes, which is about 94.5 % of the nodes in the system. This is a bit different from the previous reports, where errors were mostly contained in a small number of nodes. We have more detailed findings about the relationship between nodes and error patterns in Section 4-D.

The errors are stored in different register banks, each of which maps to different units in the processor, as shown in Table II. We identify 29 different error signatures from the entire logs, which consists of bank ID, error code, and error code extension. However, we find the same error with multiple error signatures and errors with multiple subcases. After grouping errors by more semantically meaningful error type and error sub-type, we find 20 types of errors as reported in Table II. We find that a majority of errors, $6,719,420$, occur in Bank4, NorthBridge. Errors related to the Combined Unit (CU) follows next, with a total of $88,318$ errors. Critical errors from load-store units and instruction fetch units are relatively rare, yet they happen. In Section 4-B we discuss uncorrected errors by hardware, where a half of uncorrected errors are related to load-store units.

Figure 2 shows the daily trend of the number of errors from all the nodes. We observe that the average number of errors during all the monitored period is $5,376$ errors per day, and max number of errors per day is $26,550$ errors. During the $1,287$ days of the monitoring period, 447 days had a larger than average number of errors, which is about 34% of monitoring period. Two questions arise after observing such a phenomenon: (1)*which factors contribute such a wide fluctuation of the number of errors*; and (2) *which errors cause serious system crashes?* Identifying dominant factors may lead to more reliable system operations and lead to the designing of more reliable hardware components in actual systems. In the following sections, this study examines if this fluctuation stems from the faulty hardware, the fluctuation of workloads, or the specific applications (or users).

CECC	UECC	PCC	UC	counts
0	0	0	0	1,464
			1	7
		1	1	19
	1	1	1	2
1	0	0	0	6,906,805

TABLE III: Basic statistics for each category of errors. Most errors are correcctable ECC errors (CECC=1), while we do have other types of errors. CECC: correctable ECC error. UECC: uncorrectable ECC error, a deferred error if converted to poison data. UC indicates whether the error was actually corrected by the processor (UC=1), poison data consumed or no way to avoid passing it to process. PCC: processor context corrupt.

IV. DATA ANALYSIS

In this section, we analyze four major potential contributing factors about errors. First, we look into the characteristics of errors in order to answer questions around the effectiveness of ECC and any need for additional protection for the memory and processors. In addition, we look into error intervals in order to identify the contributing factors of daily error trends.

A. Error characteristics

In this section, we describe the category of errors that we find in the collected logs and their basic characteristics such as error intervals. This section intends to show if we have serious errors that can cause system or node crashes and if the daily fluctuation of errors stem from specific errors with extremely short intervals.

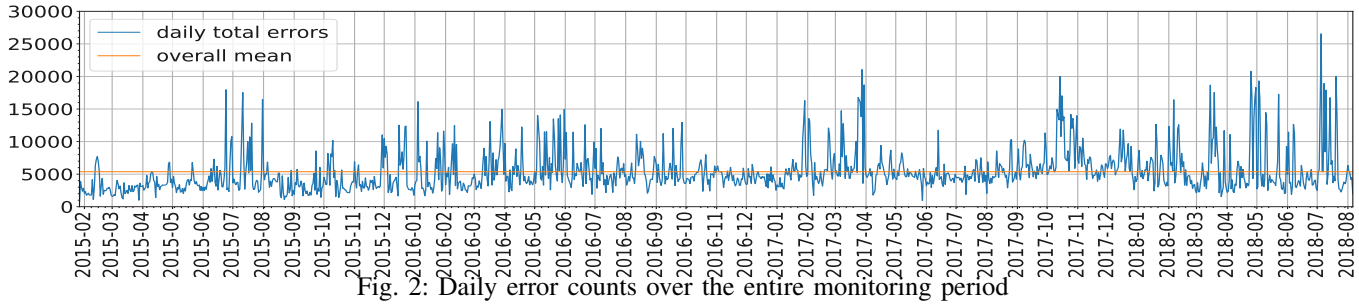

Fig. 2: Daily error counts over the entire monitoring period

1) Error categories: Errors are categorized by whether they are correctable ECC errors (CECC=1) or not (CECC=0). However, CECC=0 does not always mean the error is an uncorrectable ECC error (UECC=1), or multi-bit errors, since some errors can be related to instruction fetches or program context corruption (PCC=1). In addition, the hardware often does not correct errors (UC=0) when the error should be handled by software or when the error does not need to be immediately corrected (deferred errors). Overall statistics on each category of errors are shown in Table III.

The foremost finding from the three years of the error monitoring period is that 99.97% of errors are correctable ECC errors. We find that uncorrectable ECC errors (UECC=1) due to the restrictions of the ECC algorithm are very rare. Only two errors are recorded from the log of 18,688 nodes for 30,888 hours, a total of 577,234,944 node-hours. Thus, DRAM modules with ECC is very critical and effective to ensure the system reliability. In addition, this might be a result of constant attention on DRAM failure in the operation of Titan supercomputer similar to other HPC centers [7].

Next, we look into errors that are not correctable ECC errors (CECC = 0). Although they account for only 0.03% of the total number of errors, they are serious errors such as double bit errors or errors in hardware components other than memory. These errors can cause immediate program exit and, potentially, shutdown of the specific compute node. Among errors that are not correctable ECC errors, we have a total of 28 errors corrected by hardware (UC=1) in Table III. Those 28 errors are mainly from two sources: 16 errors from load-store units (8 data cache errors and 10 tag array errors) and 12 errors from the northbridge (10 northbridge array errors and 2 compute unit data errors).

For further detailed analysis of errors that are not correctable ECC errors, Figure 3 shows the number of errors grouped by error type and error sub-type and whether or not the errors are correctable. In this analysis, bank ID, error type and error subtype are as summarized in Table II. We find that errors that are not correctable ECC errors happen across all the components in the processor. However, 63.5% of non-correctable ECC errors have the error type of L2 Cache with the subtype of fill parity error in Bank 2 (Combined unit). Fill parity error means fill parity error on *instruction fills*. All errors in this category are 100% non-correctable ECC errors in our log. Including fill parity error in L2 cache, we find that

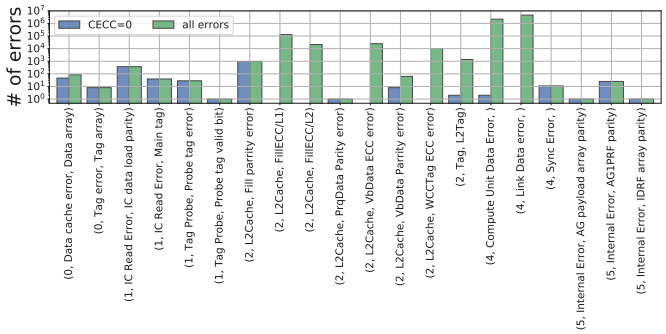

Fig. 3: The number of errors grouped by error codes, bank, error code type, and error code subtype. Errors from north-bridge (bank 4) are the most common, but more critical CECC=0 errors are common from load-store unit (bank 0) and instruction fetch unit (bank 1).

88% of non-correctable ECC errors are related to instruction cache in the load-store unit (Bank 0) and instruction fetch unit (Bank 1). We observe errors in Bank 0 (Load store unit), Bank 1 (Instruction fetch unit), and Bank 5 (Execution unit) are not correctable ECC errors in certain cases. Therefore, we can expect to significantly improve the reliability of large scale systems by raising the reliability related to instruction cache within a processor [5].

2) Error Intervals: Table IV shows basic statistics of error intervals per node. Note that the max wall clock limit of jobs on Titan is 24 hours, which aligns with the max error interval (24 hours and 27 minutes). We find extreme situations such as the situation where two consecutive errors are 134 days apart. In addition, depending upon the number of requested nodes, Titan supports different max wall time. For example, when a user requests less than 125 nodes, the max wall time is 2 hours. Thus, the error intervals when jobs are running on the node tend to be smaller than the overall cases. Notably, we observe a highly temporal locality since 50% of errors have an interval less than 6 seconds when jobs are running on the node, and 0.05 seconds for all the cases.

Next, we analyze the distribution of error intervals for all cases since the overall pattern of error interval is similar in Table IV. Figure 5 shows the log of error intervals, where we can find three peaks: 1) at tens of microseconds scale, 2) tens of milliseconds scale, and 3) tens of minutes scales. The

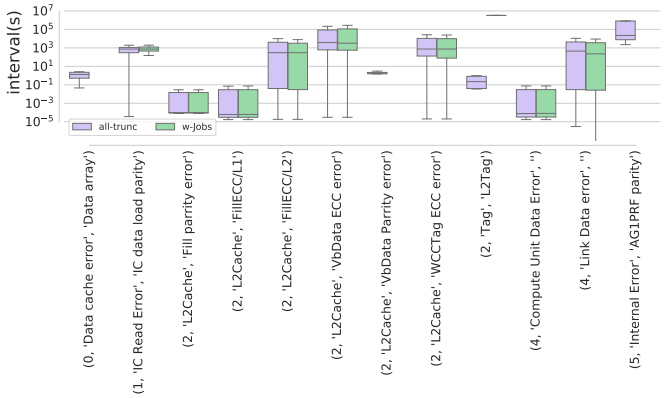

Fig. 4: The boxplot(min,25%,median,75%, max) of error intervals per error code on a node when jobs are running on the node (w-jobs) and all cases during the job monitoring period (all-trunc).

largest interval is more than tens of days. Since we identified three distinct groups of error intervals, we further investigate if each error interval group can be connected to different error types or error subtypes.

Figure 4 shows the distribution of error intervals for each error type and error subtype. Here, the error interval represents the interval between two errors on the same node. Thus, some error types and error subtypes in Figure 3 do not appear in Figure 4 Firstly, the intervals of errors do not change significantly whether a job is running on a node or not. However, we confirm that a few errors occurred when jobs are not running on the node as we have observed in Figure 3 and Figure 4.

In this analysis, we find that error intervals have different distributions depending on error type and error subtype. For errors matching to the first peak centered around 10^{-4} seconds in Figure 5, we find errors with the error subtype of Fill parity error, errors with the error subtype of FillECC/L1, and errors with the error type of Compute Unit Data Error. Errors with the error subtype of Fill parity error happens when we find parity error on instruction fills in instruction cache. Errors with the error subtype of FillECC/L1 happens when we find ECC error on data fills and the data source is L1 cache. Compute unit data error means Northbridge received a data error from a core. This error may occur for the data writes to DRAM, APIC, and IO. Short error intervals of these errors imply that these errors exhibit high temporal locality. The relatively narrow ranges of error intervals of these errors also support the possibility of

statistics	w/ Job	all cases
mean	00:29:20.982055	13:36:52.522612
std	01:35:11.146578	89:48:42.928203
min	00:00:00.000016	00:00:00.000005
25%	00:00:00.000058	00:00:00.000056
50%	00:00:06.040840	00:00:00.055997
75%	00:08:43.046752	00:17:27.961082
max	24:27:30.421187	134 days 14:54:38.632010

TABLE IV: Statistics of error intervals per node.

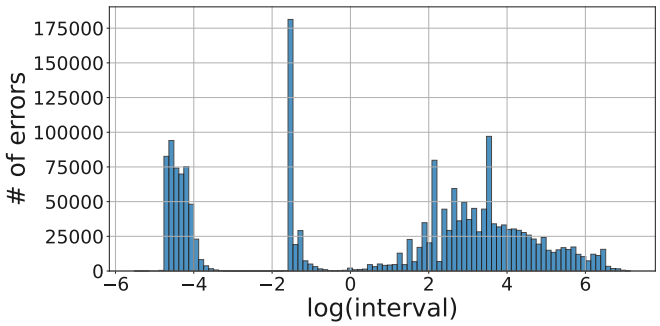

Fig. 5: The histogram of log(error intervals) for all errors, showing three distinctive ranges of error intervals.

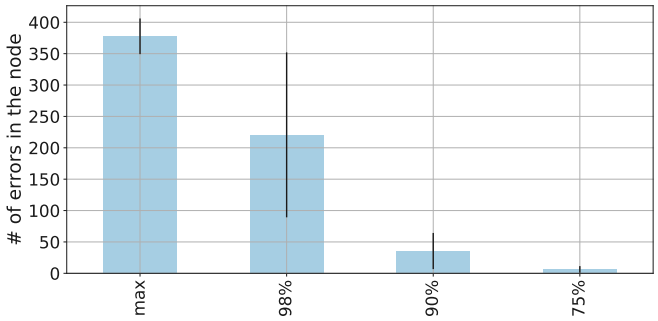

Fig. 6: The daily number of errors per node for each percentile. Error bar represents the standard deviation of each percentile to show daily fluctuation.

high temporal locality of these errors.

For the second peak that is centered on 10^{-1} seconds in Figure 5, we find that errors with the error subtype of L2Tag is the closest. However, the high frequency of errors in this range may stem from the fact that many errors happen around this range. The third category of errors that is centered on 10^3 seconds in Figure 5 can be attributed to errors with fill ECC errors corrupted on L2 cache (FillECC/L2), write coalescing cache tag ECC errors (WCCTag ECC error), victim buffer data ECC errors (VbData ECC error), and link data errors. These errors have a median around 10^3 seconds, or about a half hour between two errors. It implies that temporal locality in these errors is not clear. With respect to the data access pattern, data in L2 cache will show high temporal locality, compared with link data. Thus, this result may indicate that ECC error pattern may not align with the data access pattern since errors in the combined unit that includes L2 cache in the processor (FillECC/L2, WCCTag ECC error, VbData ECC error) show relatively low temporal locality compared to the errors in the northbridge (link data errors). Identifying such a relationship between ECC error pattern and data access pattern should deserve a in-depth study with the actual program.

B. Node Characteristics

In this section, we analyze the relationship between errors and node characteristics. First, we counted the daily number

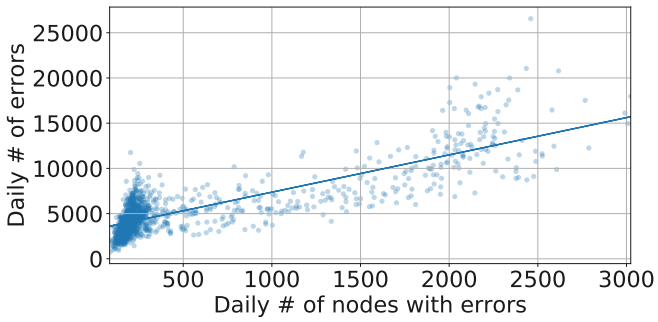

Fig. 7: The number of errors and the number of nodes with errors for each day. The linear regression result shows $R^2 = 0.68$, a medium positive linear relation.

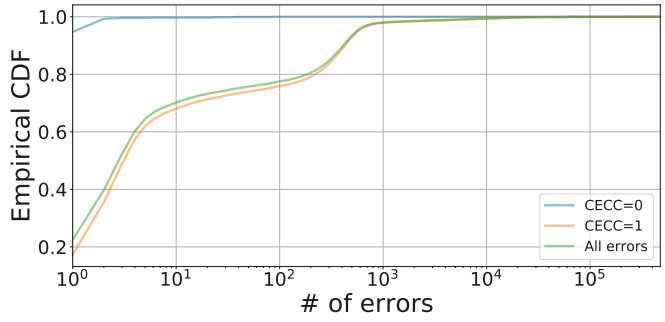

Fig. 8: The empirical cumulative distribution function of the number of nodes according to the total number of errors in a node for all errors, correctable ECC errors (CECC=1), non-correctable ECC errors (CECC=0). We found a total of 1,492 non-CECC errors (CECC=0), including 2 errors that UECC set as 1, as reported in Table III. Note that non correctable ECC errors include UECC errors (e.g., multi-bit ECC errors), processor context corrupt errors and deferred errors to be handled by software.

of errors per node. Figure 6 shows the max number of errors in a node, the 98-th percentile of number of errors in a node, the 90-th percentile of number of errors in a node, and the 75-th percentile of number of errors in a node, where error bars represent standard deviation of their daily fluctuation. We observe that, for each day, the worst node records around 377 errors with relatively small standard deviation, while 98-th percentile records around 220 errors with very wide fluctuation according to day. Meanwhile, 75-th percentile of nodes only generates 6 errors with small fluctuation per day. Thus, we can say that the number of daily errors fluctuates mainly due to a small set of nodes less than top 25-th percentile of nodes with respect to the number of errors. This analysis implies that errors do not occur in a uniform distribution and it might be possible to significantly raise the reliability of large scale systems by focusing on a small set of nodes.

Followed by this finding, the question is if this small set of nodes can explain the daily fluctuation of errors in the system. We analyzed the association between the number of errors and the number of nodes with errors for each day, where we obtained 0.730 of Spearman coefficient that implies positive monotonic correlation between them. Spearman correlation measures the strength and direction of monotonic association between two variables which may or may not be linear. A perfect Spearman correlation of $+1$ occurs when one variable is a monotonically increasing function of the other, or -1 when it's a monotonically decreasing function. In order to identify if a linear relationship exists between the number of errors and the number of nodes with errors for each day, we attempted a linear regression and obtained $R^2 = 0.68$, which means a mild positive linear relationship exists, as shown in Figure 7. That is, when we have a larger number of errors in the system, it is very likely that a larger number of nodes generate errors in the system. In this analysis, we also find that up to $3,024$ nodes (16% of compute nodes) generate errors a day, and the median number of nodes with the error per day is only 221 nodes (1% of compute nodes). Thus, we can say that errors are generated from a small set of nodes for each day and the daily fluctuation would be contributed by more number of nodes with errors, instead of a single dominant node.

Now, we show the overall errors from the empirical cumulative distribution of the number of nodes according to the total number of errors from the node as shown in Figure 8. We discover a bi-modal, log-normal distribution among the nodes according to the total number of errors. In this analysis, we confirm that 76.1% of the nodes have less than 100 errors during the observed time period. The remaining 23.9% of nodes contain more than 100 errors, which forms the second slope beginning with 100 errors in Figure 8. This set of nodes account for 98% of the entire errors. Narrowing down, we find that 373 nodes (2% of compute nodes) with more than 1,000 errors generate 77.5% of the entire errors. This analysis discovers the existence of a set of error-prone nodes that significantly influence the overall number of errors in the system. Thus, if we pay close attention to these nodes, we have a good chance to significantly lower the number of errors in the system.

The majority of errors found in the hardware error log are correctable ECC errors ($CECC = 1$), which can be easily corrected through ECC modules. However, we have found a total of 1,492 non-correctable ECC errors ($CECC = 0$) from 1,253 unique nodes, throughout the entire error monitoring period, as reported in Table III. On average, we have 1.16 of these non-correctable errors per each day. These non-correctable ECC errors are tend to be more critical errors, including UECC errors (multi-bit ECC errors, $UECC = 1$), processor context corrupt errors ($PCC = 1$) and deferred errors to be handled by software. Thus, it means we need to investigate at least one machine in the Titan supercomputer for any given day, which highlights the importance of managing these errors.

We check if these critical non-correctable errors happen more often in the more error-prone nodes, but only 6% of nodes had non-correctable errors among nodes with more than

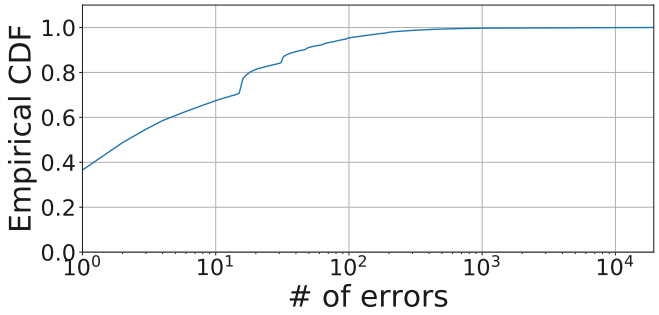

Fig. 9: The empirical cumulative distribution function of the number of jobs according to the number of errors in a job.

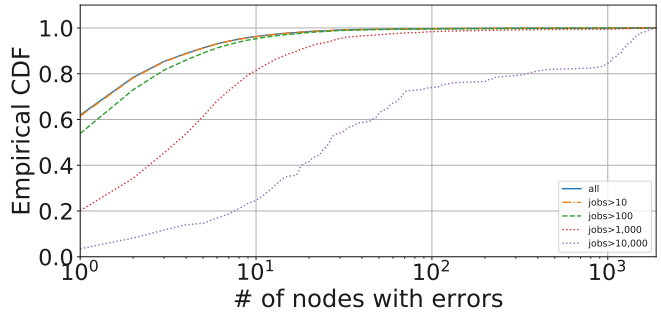

Fig. 10: The empirical cumulative distribution function of the number of jobs according to the number of nodes with errors, for the jobs with different groups of allocated nodes. All is for all jobs. $jobs < n$ represents jobs with more than n allocated nodes.

100 errors. Thus, we claim that the occurrence of correctable ECC errors ($CECC = 1$) does not indicate higher possibility of other serious non-correctable errors ($CECC = 0$). It is because other critical errors are related to instructions rather than data accesses. Thus, although ECC modules dramatically improve system reliability, in order to prevent systems from critical errors, we need to raise fault tolerance on hardware components related to instructions.

In summary, we discover the existence of error-prone nodes whether they are ECC errors or other types of errors. A possible hypothesis of concentrated errors on a small number of nodes is that the jobs scheduled on the node may have impacted the error pattern of the node. Thus, in the following section, we will analyze if the error pattern varies according to the jobs on the node.

C. Job characteristics

Each job log entry records the starting time, the ending time, the list of allocated nodes, and UID associated with the user who submitted the job. Thus, major features of a job are 1) the running time and 2) the number of allocated nodes. This section analyzes if error patterns are related to those major job features.

Figure 9 shows the empirical cumulative distribution function of the number of jobs according to the number of errors in a job. Overall, the empirical CDF shows that the number of jobs according to the number of errors follow zipfian distribution. Note that 24,337 jobs out of 312,215 recorded jobs contain errors (refer to Table I.) The analysis in this section focuses on the jobs with errors, which comprises of only 7.8% of jobs. Among 24,337 jobs with errors, 37% of jobs contain a single error and 80% of jobs contain less than 20 errors. However, there exist jobs with more than $10,000$ errors.

We calculated Spearman correlation coefficients in order to find correlation between the number of errors and features of a job. We computed Spearman coefficients 1) between the number of errors in a job and the running time of the job (0.289); 2) between the number of errors in a job and the allocated number of nodes for the job (0.339); and 3) between the number of errors in a job and the number of nodes with errors iamong the allocated nodes for the job (0.630.) From this calculation, we find no strong association between the number of errors and both job size and running time, while a good association between the number of errors and the number of nodes with errors among the nodes allocated to the job.

Let us investigate into the weak correlation between the number of errors and the number of allocated nodes. If there exist specific less reliable nodes, jobs with large allocation will have greater chance to include them. Thus, the number of errors will have correlation with the number of allocated nodes for a job. However, if errors are related to the computational pattern of a job, errors only may happen in nodes depending on the computational pattern of the job. For instance, there can be a small set of nodes with heavy communication, I/O accesses, and processing.

To address this question, we calculated the cumulative distribution function (CDF) of the number of jobs according to the number of nodes with errors in the job as shown in Figure 10. This analysis shows that 61.9% of all jobs have errors only in a single node, 61.4% of jobs with more than 10 allocated noes, 53.9% of jobs with more than 100 allocated noes, and 20% of jobs with more than $1,000$ allocated nodes. For jobs with $1,000$ allocated noes, 80% of jobs have errors in less than 1% of nodes (10 nodes). Jobs with $10,000$ allocated nodes show similar pattern. Although the chance of having errors in a single is lower than other cases, the chance of having errors in less than 1% of nodes (less than 100) is still high, 71.9%. Thus, this analysis discovers that errors occur in a small set of allocated nodes for a job. This result suggests that focusing on those 1% of nodes will be sufficient in order to understand the reliability of most of user jobs.

However, we also note outlier cases where jobs experienced errors more than 10% of allocated nodes. For instance, at the worst case, a job had errors in $1,884$ nodes out of $11,274$ allocated nodes, or 16.7% of allocated nodes had errors. Interestingly, including this job, a few other outlier jobs are submitted by the same user. The next section will be discussed

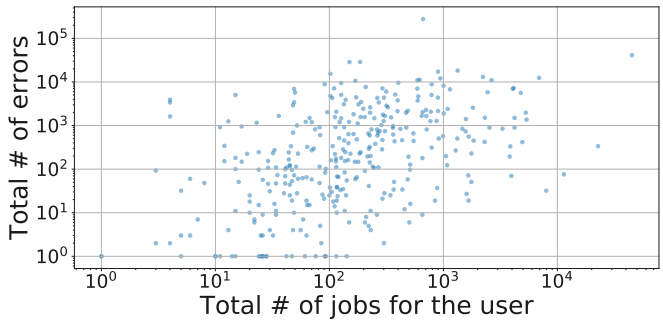

Fig. 11: The total number of errors with respect to the total number of jobs for each user. Note that both x-axis and y-axis are in log-scale.

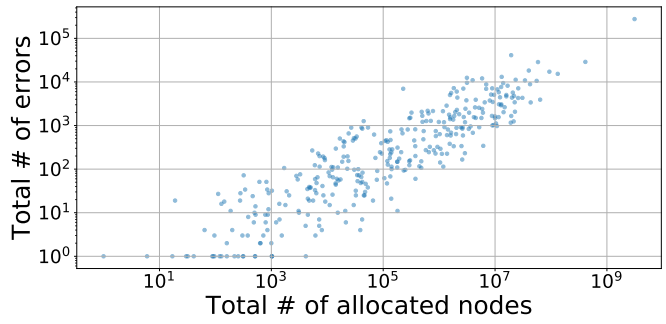

Fig. 12: The total number of errors with respect to the total number of allocated nodes for each user shows a strong correlation.

such correlation between user and the number of errors in detail.

D. User Characteristics

Users may run specific workloads (e.g., simulation and data analysis) within the scope of each project. This section considers the hypothesis that there is a relationship between users' characteristics and errors. For instance, a user may have more errors than others or a user may have a strong association with specific errors. Firstly, we begin with analyzing the frequency of errors per user.

Figure 11 shows the total number of errors versus the number of all jobs per user. The figure shows that moderately positive relationship exhist between the total number of jobs and the total number of errors, since the Spearman correlation between two variables is 0.476 We further investigate if the important feature of jobs per user is the size of allocation or the running time.

Figure 12 shows the relationship between the total number of allocated nodes and the total number of errors for each user. In this analysis, we find a strong monotonic association between the total number of allocated nodes and the total number of errors, according to 0.898 of Spearman correlation coefficient. This indicates that users who have often submitted large jobs will be likely to experience a large number of errors throughout associated projects. It does not necessarily mean all the large jobs will have large number of errors. However, considering the same analysis for each user, we discover the number of allocated nodes is correlated to the number of errors. Therefore, focusing on the users with large jobs can significantly reduce the total number of errors in the system.

On top of this analysis, we can claim that user characteristics are clearly a factor in generating errors in a job, as we have previously discussed in the analysis related to Figure 10. Such a correlation may happen because a user will employ specific applications with specific input parameters, software libraries, and data analysis methods, throughout the project. Indeed, fusion simulation is well-known for the challenging I/O workloads [10] and large-scale collaboration between national labs [18], all of which may contribute to the distinct

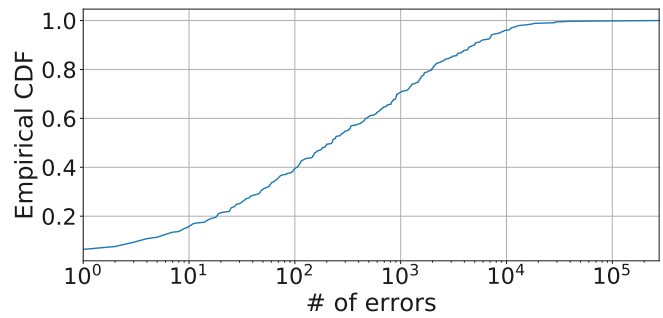

Fig. 13: The empirical cumulative distribution function of the number of users according to the total number of errors.

error pattern of this user. We will again confirm this trend is not only restricted to this top user in the following analysis.

Figure 13 shows empirical cumulative distribution function (CDF) of the number of users according to the total number of errors. This analysis shows that the empirical CDF of the number of users according to the total number of errors follows power-law distribution since the empirical CDF follows normal distribution when x-axis is transformed with logarithm as shown in Figure 10. The highest number of errors for a user is $276,973$, but 288 users out of 342 total users has less than 1% of errors of the top error user.

This trend leads us to detailed analysis of users with high number of errors. The total number of errors from top 10 users is $463,937$, which is 54.4% of total errors. Interestingly, the top 2 users are associated with the same project, fusion energy, and the number of errors from both users is $318,094$, 37.3% of the total errors. Among top 10 users, we have three users in fusion energy area and their total number of errors is $330,152$. Since we are not able to access their codes or simulation parameters, we cannot decisively clarify the root cause of this phenomenon. Those users mostly ran fusion simulation, `xgc` [6], which is one of the important simulation code in high performance computing centers [15]. The I/O scalability of this simulation code has been an important issue [8], [17]. Therefore, it might be worth of studying the

relationship between memory errors and I/O issues in order to identify the root cause of seriously high number of memory errors for this simulation, compared with all other simulations.

V. Conclusions

This paper rigorously studied the hardware error patterns combined with job scheduler logs from one of the fastest supercomputers in the world. We used error logs that were collected over 3 years and job scheduler logs that were collected over 6 months. We discovered three distinctive temporal locality patterns of errors according to the error types. We found a nested power-law distribution on the number of errors according to the nodes, jobs, and users, which implies that errors are highly concentrated in a small number of nodes, jobs, and users. We revealed that such a power-law distribution stems from user behavior, rather than the impacts from user-independent job characteristics (e.g., allocated node size, and the execution time) or specific hardware component defects. The low impact from hardware component defects may be credited to agile system operation practices such as a rapid identification and replacement of defective hardware components. Since CPU-GPU heterogeneous systems are gaining popularity for both the newer generation of applications (e.g., machine learning) and traditional scientific applications in HPC environments (e.g., scientific simulations), this study provides valuable statistical information to help design dependable large scale systems

References

[1] Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x — TOP500 Supercomputer Sites. http://www.top500.org/system/177975.

[2] Splunk, 2019 (accessed April 1, 2019). https://www.splunk.com/.

[3] Hui Guan, Lin Ning, Zhen Lin, Xipeng Shen, Huiyang Zhou, and Seung-Hwan Lim. In-place near zero-cost memory protection for dnn. In *Thirty-third Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[4] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. Failures in large scale systems: long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 44. ACM, 2017.

[5] Imran S. Haque and Vijay S. Pande. Hard data on soft errors: A large-scale assessment of real-world error rates in gpgpu. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID'10)*. IEEE Computer Society, 2010.

[6] S Ku, Choong-Seock Chang, M Adams, J Cummings, F Hinton, D Keyes, S Klasky, W Lee, Z Lin, Scott Parker, et al. Gyrokinetic particle simulation of neoclassical transport in the pedestal/scrape-off region of a tokamak plasma. In *Journal of Physics: Conference Series*, volume 46, page 87. IOP Publishing, 2006.

[7] Scott Levy, Kurt B Ferreira, Nathan DeBardeleben, Taniya Siddiqua, Vilas Sridharan, and Elisabeth Baseman. Lessons learned from memory errors observed over the lifetime of cielo. In *Lessons Learned from Memory Errors Observed Over the Lifetime of Cielo*, page 0. IEEE, 2018.

[8] Jianwei Liao, Balazs Gerofi, Guo-Yuan Lien, Takemasa Miyoshi, Seiya Nishizawa, Hirofumi Tomita, Wei-Keng Liao, Alok Choudhary, and Yutaka Ishikawa. A flexible i/o arbitration framework for netcdf-based big data processing workflows on high-end supercomputers. *Concurrency and Computation: Practice and Experience*, 29(15):e4161, 2017.

[9] Seung-Hwan Lim, Hyogi Sim, Raghul Gunasekaran, and Sudharshan S Vazhkudai. Scientific user behavior and data-sharing trends in a petascale file system. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 46. ACM, 2017.

[10] Qing Liu, Jeremy Logan, Yuan Tian, Hasan Abbasi, Norbert Podhorszki, Jong Youl Choi, Scott Klasky, Roselyne Tchoua, Jay Lofstead, Ron Oldfield, et al. Hello adios: the challenges and lessons of developing leadership class i/o frameworks. *Concurrency and Computation: Practice and Experience*, 26(7):1453–1473, 2014.

[11] Esteban Meneses, Xiang Ni, Terry Jones, and Don Maxwell. Analyzing the interplay of failures and workload on a leadership-class supercomputer. In *Cray User Group Conference*, 2015.

[12] S. Neuwirth, F. Wang, S. Oral, S. Vazhkudai, J. Rogers, and U. Bruening. Using balanced data placement to address i/o contention in production environments. In *2016 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2016.

[13] Bin Nie, Devesh Tiwari, Saurabh Gupta, Evgenia Smirni, and James H Rogers. A large-scale study of soft-errors on gpus in the field. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 519–530. IEEE, 2016.

[14] Bin Nie, Ji Xue, Saurabh Gupta, Tirthak Patel, Christian Engelmann, Evgenia Smirni, and Devesh Tiwari. Machine learning models for gpu error prediction in a large scale hpc system. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018.

[15] Department of Energy. Tapping the next generation of supercomputers to bring fusion energy to the world. https://www.energy.gov/articles/tapping-next-generation-supercomputers-bring-fusion-energy-world.

[16] Byung H Park, Yawei Hui, Swen Boehm, Rizwan A Ashraf, Christopher Layton, and Christian Engelmann. A big data analytics framework for hpc log data: Three case studies using the titan supercomputer log. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 571–579. IEEE, 2018.

[17] Norbert Podhorszki, Scott Klasky, Qing Liu, Ciprian Docan, Manish Parashar, Hasan Abbasi, Jay Lofstead, Karsten Schwan, Matthew Wolf, Fang Zheng, et al. Plasma fusion code coupling using scalable i/o services and scientific workflows. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, page 8. ACM, 2009.

[18] Norbert Podhorszki, Bertram Ludaescher, and Scott A Klasky. Workflow automation for processing plasma fusion simulation data. In *Proceedings of the 2nd workshop on Workflows in support of large-scale science*, pages 35–44. ACM, 2007.

[19] Bianca Schroeder and Garth Gibson. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing*, 7(4):337–350, 2010.

[20] Devesh Tiwari, Saurabh Gupta, James Rogers, Don Maxwell, Paolo Rech, Sudharshan Vazhkudai, Daniel Oliveira, Dave Londo, Nathan DeBardeleben, Philippe Navaux, et al. Understanding gpu errors on large-scale hpc systems and the implications for system design and operation. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 331–342. IEEE, 2015.

[21] Christopher Zimmer, Saurabh Gupta, Scott Atchley, Sudharshan S Vazhkudai, and Carl Albing. A multi-faceted approach to job placement for improved performance on extreme-scale systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 87. IEEE Press, 2016.

[22] Christopher Zimmer, Don Maxwell, Stephen McNally, Scott Atchley, and Sudharshan S. Vazhkudai. Gpu age-aware scheduling to improve the reliability of leadership jobs on titan. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'18)*. IEEE Press, 2018.