Analyzing the Impact of System Reliability Events on Applications in the Titan Supercomputer

Rizwan A. Ashraf and Christian Engelmann
Computer Science and Mathematics Division,
Oak Ridge National Laboratory,
Oak Ridge, Tennessee 37831, USA.
Email: {ashrafra, engelmannc}@ornl.gov

***Abstract*—Extreme-scale computing systems employ Reliability, Availability and Serviceability (RAS) mechanisms and infrastructure to log events from multiple system components. In this paper, we analyze RAS logs in conjunction with the application placement and scheduling database, in order to understand the impact of common RAS events on application performance. This study conducted on the records of about 2 million applications executed on Titan supercomputer provides important insights for system users, operators and computer science researchers. Specifically, we investigate the impact of RAS events on application performance and its variability by comparing cases where events are recorded with corresponding cases where no events are recorded. Such a statistical investigation is possible since we observed that system users tend to execute their applications multiple times. Our analysis reveals that most RAS events do impact application performance, although not always. We also find that different system components affect application performance differently. In particular, our investigation includes the following components: parallel file system, processor, memory, graphics processing units, system and user software issues. Our work establishes the importance of providing feedback to system users for increasing operational efficiency of extreme-scale systems.**

I. Introduction

Reliability, availability, and serviceability (RAS) events play a major role in the maintenance and operation of extreme-scale computing systems. These events are recorded from almost all components in a high-performance computing (HPC) system and therefore provide useful insights into the reliability of the system. The infrastructure to log RAS events is an integral part of the HPC ecosystem, and is used by system operators to assess operational efficiency of the system and to keep up with its maintenance. At the same time, the logged RAS events also provide important insights into the different kinds of faults, errors and failures which can occur in an extreme-scale computing system.

Previous studies on various large-scale machines with diverse architectures have helped to understand multiple failure mechanisms and quantified system time-to-failure [1]–[5]. However, these studies do not correlate failure mechanisms with application executions. Our work is distinct in this regard, since RAS events are analyzed in conjunction with application runs. Such an investigation is critical to understand the impact of faults, errors and failures on user applications executed in extreme-scale systems. Previously, it has been shown that memory errors even when correctable through error-correcting codes (ECC) cause severe performance degradation due to overheads of the error-reporting stack [6]. Their experiments with SPEC CPU2006 benchmarks and a web-search workload on a single machine using a proprietary fault injection tool reveal variation in performance degradation dependent on the workload. The analysis herein does similar analysis in terms of assessing performance impact, however, on a much larger scale and for a complex combination of workloads in a deployed HPC system. Another related work [7] characterizes resiliency of applications executed on Blue Waters supercomputer, however, fails to assess the impact of RAS events on application performance. Based on our knowledge, the study herein is first of a kind, which concurrently analyzes the impact of RAS events in parallel file system, processor, memory, graphics processing units (GPUs), interconnect, and system software on extreme-scale application performance.

In this work, we analyze logs from the Titan supercomputer at Oak Ridge Leadership Computing Facility (OLCF) at Oak Ridge National Laboratory (ORNL). Titan is a Cray system with a hybrid architecture composed of 18,688 nodes. Each node is composed of 16-core AMD Opteron CPUs and NVIDIA Kepler GPUs. Titan supercomputer provides an open resource for scientific computing. A complex mix of workloads from a variety of scientific domains including biology, chemistry, computer science, earth science, engineering, fusion, materials science, etc., are executed on Titan. The resources are allocated to users using three different competitive allocation programs, for up to one year. Our analysis is done on RAS event and application logs of about 13 months, i.e., from Dec. 24, 2015 to Feb. 2, 2017. Based on the allocation cycle, study over one year is sufficient to capture the different kind of workloads. The interested reader is referred to: [1], for an in-depth and multi-year analysis on the RAS logs from Titan supercomputer.

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy 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).

Titan has a hybrid architecture with computing accelerated using GPUs. Use of diverse components from different hardware vendors give rise to a complex ecosystem where RAS logs need to be aggregated from multiple components in order to operate the system efficiently. System administrators use the logs to observe and maintain system health, and to efficiently allocate resources to user applications. For example, memory modules reporting frequent errors are replaced as part of routine maintenance [5]. We leverage this infrastructure and analyze the impact of RAS events on application performance. We investigate whether RAS events slowdown application execution. RAS events may or may not be caused by the users, and we investigate impact in each case. For instance, segmentation faults are caused mostly by user applications. Similarly, congestion in Lustre network can cause RAS events to be logged, which may be caused due to multiple users in the system writing to the parallel file system at the same time. This paper looks into the impact of each component separately and also co-analyzes events from different types of components.

In our analysis, we find that RAS events impact application performance in most cases irrespective of whether a failure of an application occurs or not. This is an important finding since it is normally assumed that non-fatal events do not impact application performance given extreme-scale systems are configured to handle such cases efficiently and have minimal overhead due to the logging infrastructure. Our analysis is facilitated by the presence of multiple executions of applications with same workloads. However, it is not clear why such multiple executions are done by the users, and one of the reasons may be to remove outliers when conducting performance and/or debug runs. Whatever the reason may be, we are able to conduct a useful study using these logs. In particular, our work is important for the following reasons: 1) users of large-scale machines can understand how system RAS events impact the execution of their applications, 2) system operators can mitigate the overheads associated with the logging of RAS events which are observed to impact application performance, and 3) computer science researchers can develop mechanisms in programming models, runtime systems and system software which incorporate information from system RAS events and subsequently lower impact on application performance, e.g., doing another task if the parallel file system is not responding.

II. The Data and its Limitations

In this paper, the following logs are used for our analysis:

- *Event log:* this contains information about RAS events such as the type of event, time of the event, and identity of the node where the event took place.
- *Application Level Placement Scheduler (ALPS):* this contains allocation information for all applications executed on compute nodes such as name and identity of the executable (a single job submission may contain multiple application executions, each is counted separately), identity of the user (running the application), start time, end time, identities of allocated nodes.

TABLE I: Distribution of RAS events occurring in the Titan supercomputer independent of application executions.

Event Class	Category	Percentage
Parallel File System	Hardware/Software	73.7%
Processor Events	Hardware/Software	15.7%
Machine Check Exceptions	Hardware	6.5%
Graphics Processing Unit	Hardware/Software	1.5%
Seg-Faults & Out-of-Mem.	Software	1.9%
Interconnect	Hardware	0.8%

In our study, we only consider RAS events taking place on compute nodes where bulk of applications are executed after their production phases. We remove from our analysis all events taking place on other non-compute type of nodes in the system such as, service nodes (used for production-type work), login nodes, data transfer nodes (user for data transfer to Lustre storage), etc. We also exclude scheduled maintenance periods from our analysis, subsequently removing all recorded events and test applications executed during these periods. Furthermore, given these assumptions and available data, our study has the following limitations:

- We are unable to distinguish between fatal and non-fatal application runs since exit codes are not available. The difficulty of assessing fatal events or severity of events also prevents us from making this classification. Furthermore, the fatal event may or may not be fatal for an application depending on whether some resiliency mechanism is implemented or not. Nonetheless, this limitation does not prevent us from analyzing the performance impact of RAS events on applications. At the same time, we are unable to identify runs which might have terminated early due to a fatal event resulting in a speedup as compared to corresponding cases with no events.
- Our study is oblivious to any resilience mechanisms implemented in the applications, such as checkpoint restart, since we do not have this information. We therefore can not assess the performance benefit of resilience mechanisms.
- Resource utilization information for different system components such as memory utilization, I/O bandwidth, GPU utilization, etc., is not available. This prevents us from determining the role of resource utilization or contention in application slowdown. For example, we can not assess the impact of Lustre I/O utilization on application performance when different Lustre events are recorded during application runs. Similarly, we can not accurately assess the proportion of applications using GPUs which prevents us from normalizing recorded GPU events across only the applications which use them.

III. Categorization of RAS Events and Characteristics of Applications with RAS Events

The type of RAS events analyzed in this work are categorized in Table I. We distinguish our analysis based on these classifications to assess the impact of each system component separately. During our categorization, we do not isolate events triggered by user applications because of the complexity associated with determining the root-cause. For instance, an out of memory error in an application run could be triggered by a system software issue and/or prior application runs on the node. We do however distinguish between hardware- and software-related events. Next, we briefly discuss each RAS event category and analyze the occurrence of events independent of application executions, i.e., the events may or may not take place during the application executions.

A. Categorization of RAS events independent of application executions

During our study period, RAS events from Lustre file system (parallel file system) are the most dominant event type when considered independently of application executions (see Table I). *Lustre file system events* are composed of both the file system software events and Lustre network events. Example of some common RAS events reported by Lustre include: communication errors between nodes, out of memory errors on Lustre nodes, failure of request sent due to slow reply or network error, etc. It is to be noted that this breakdown is not normalized and reporting mechanisms differ for different components. For instance, Lustre tends to generate a large number of RAS events in a short period as compared to other system components.

In terms of number of events, Lustre events are followed by '*Processor Events*.' These include HWERR (Hardware Error), kernel panic, and graphics engine error. HWERR is the most dominant event type within this category. HWERR includes access violation errors, dynamic page retirement, double bit uncorrectable memory errors, etc. Operating system kernel panic included in this category causes a reboot of the node due to fatal exception generated by a hardware or a software error in the processor. For instance, double bit memory errors detected using ECC and uncorrectable in system software (most scenarios) cause kernel panic, which is fatal for the application using the node.

Related to the processor events are *machine check exceptions (MCEs)*, which are part of the machine check architecture of the processor. This architecture facilitates reporting of processor and system hardware errors to system software. It is configurable and includes logging of both correctable and uncorrectable errors. Errors which are uncorrectable are reported to system software via a MCE and if they are not correctable require a system shutdown, for example, through a kernel panic. Notice, the uncorrectable error is also recorded redundantly as a HWERR above, and is not removed from our analysis since we are interested in the overall overhead associated with the chain of events triggered from a single root-cause. Mostly memory errors are reported in DRAM and can often be corrected in hardware using ECC [3]. Although, such reporting does not impact the correctness of the application and does not cause a failure, it is important to quantify the overhead of such a correction event on application performance. It should be pointed out that system operators

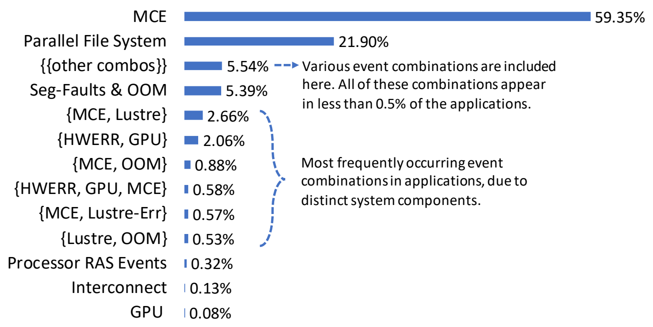


Fig. 1: Distribution of applications based on type of RAS events observed during their executions. All possible combination of events (within a single category and across categories) are considered to classify each application. Only major combinations of events across different categories are included.

replace memory modules reporting frequent ECC corrections over time in order to minimize these errors, and such information is not available nor required as part of this study.

GPU errors are composed of GPU memory hardware errors which are uncorrectable, as well as errors due to user applications, GPU driver, and thermal issues. An example of an error due to user application is when an array is accessed illegally in a kernel. *Segmentation faults and out of memory (OOM) errors* in the GPU and/or CPU are categorized separately in our analysis, and may or may not be caused due to user applications. Note, correctable errors in the GPU memory are not included in our analysis. Related work [8] has devised machine learning models for predicting the occurrence of GPU memory correctable errors in Titan supercomputer.

Interconnect errors are logged whenever there is an issue in any of the network links in the system. In Titan, there are three lanes in a link, and any combination of lanes can go down. The severity of this event can be high if all lanes go down in a link at the same time. In this work, we do not distinguish between the severity of different interconnect failure cases. These events are recorded at gemini routers and mapped to compute nodes. Interconnect errors are the least occurring event type during our study period. Interested reader is referred to following references for a detailed account of interconnect errors in Cray machines: [9], [10].

B. Categorization of applications based on observed RAS events

Next, each application execution is classified based on the types of RAS events (as described above) recorded during their execution (see Fig. 1). Node sharing is not allowed on Titan supercomputer, therefore, there is a one-to-one mapping between event occurrences and scheduled applications on the nodes. We find that about 68k applications in our study period out of a total of about 2 million applications have events recorded during their execution. This number should not be taken literally since all events do not cause an application failure and due to the limitation of not recording all the event

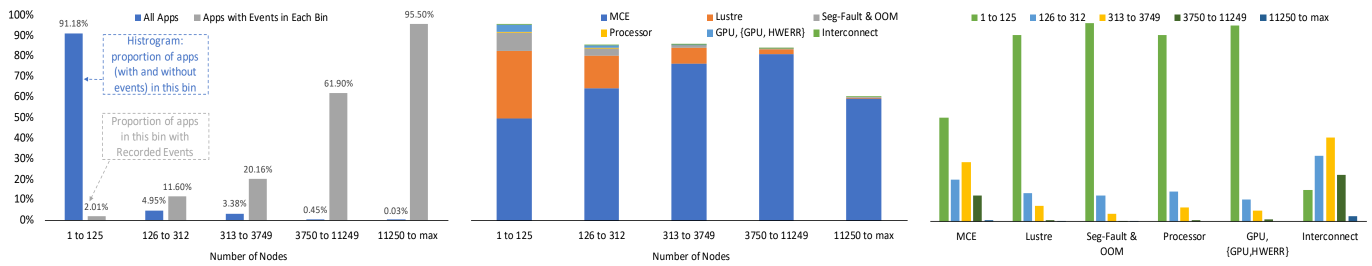


Fig. 2: Left figure: one set of bars show the histogram of all application executions based on number of nodes utilized irrespective of whether there is an event or not, and the second set of bars show the proportion of applications in each bin with events. This provides an estimate of the likelihood of having events based on application size. The bins for application sizes are defined according to prioritization given to applications during scheduling; Center figure: shows the breakdown of applications with different event categories normalized w.r.t. applications of different sizes. Note, all possible event combinations are not included, resulting in incomplete accounting of applications with events in each bin; Right figure: shows the distribution of applications with different event categories based on multiple size bins.

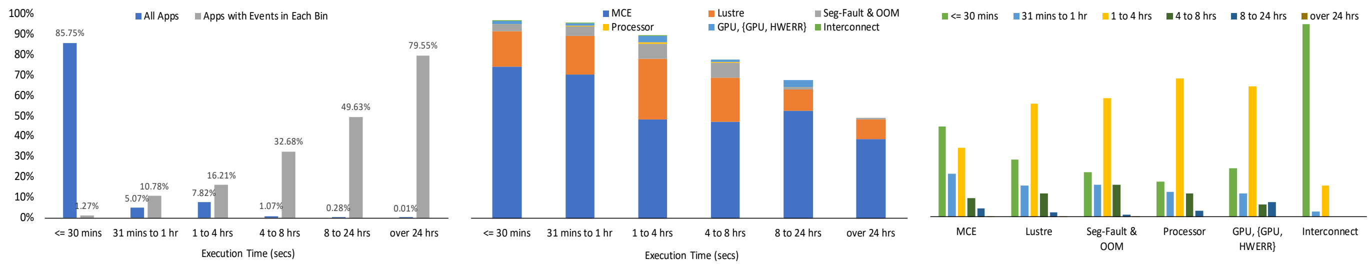


Fig. 3: Left figure: one set of bars show the histogram of all application executions based on execution times irrespective of whether there is an event or not, and the second set of bars shows the proportion of applications in each bin with events; Center figure: shows the breakdown of applications with different event categories normalized w.r.t. applications of different execution times. Right figure: shows the distribution of applications with different categories based on multiple time bins.

types in the system at all times (to keep overheads low). Yet, it gives us a lower bound on the proportion of applications whose performance might be impacted in extreme-scale systems.

The observed RAS events in application executions are used to classify each run as shown in Fig. 1. All combinations of different event types within the same event class are considered (where applicable) to classify each application execution. Majority of the applications have MCEs recorded during their execution, i.e., 59% of applications. This is in contrast to distribution of RAS events considered independent of application executions. Without a joint analysis, one could wrongfully ascertain that Lustre events impact majority of application runs (see Table I). In contrast to previous observation, Lustre events are only recorded in about 22% of the applications. Also, some events from distinct event categories frequently occur together. For example, Lustre and MCE events occur together within the same application execution in about 2.7% of cases, which is plausible. Another common and interesting occurrence is the presence of HWERR with GPU errors in about 2.1% of the cases. As mentioned earlier, the number of applications with GPU events in our analysis are not normalized based on the number of applications which actually use the GPUs.

We also analyze the characteristics of applications with events as shown in Figs. 2 and 3 based on number of nodes utilized and execution times, respectively. We later find out that it is important to identify these characteristics since occurrence of RAS events cause slowdown in most cases. Two important observations can be made from the histograms (figures on the left) in Figs. 2 and 3: majority of applications executed on Titan supercomputer during our study period used only a fraction of the total nodes and had short execution times (less than 30 mins); the likelihood of having events increases as the size of applications increase and/or execution times increase. For instance, only 3.9% of applications in our analysis used more than 313 nodes. And about 96% of applications with size of 11250 or more, had events recorded during their executions.

Results in Fig. 2 also show the distribution of applications with different event types based on the number of nodes utilized. The following observations can be made from these results: most of the applications with Lustre, segmentation faults and OOM errors, processor errors, and GPU errors (most GPU errors occur in combination with HWERR) use less than 126 nodes in the system; MCEs seem to occur uniformly across applications of all sizes; interconnect events mostly appear in medium- to large-scale applications, for instance, they are the second most dominant event type appearing in combination with MCEs within applications of size greater than 11249 (note, this result is not included in Fig. 2); events from distinct event categories appear more often together in large-scale applications (this can be observed by the high proportion of incomplete categorization of applications to individual event classes in the last bin as seen in Fig. 2).

One probable explanation for such a high proportion of segmentation faults and OOM errors at low-scale may be because users tend to test their codes before running their applications at large-scales, even though service nodes are reserved for this purpose. Part of this explanation can also be used to explain the dominance of processor errors and GPU errors at low-scale. Especially, common co-occurrence of HWERR and GPU errors might be due to testing of GPU codes resulting in kernel exceptions in the GPUs. GPU errors are also not so common at large-scale since we suspect that not many applications use GPUs at such scale, although, we do not have the GPU utilization data to back this claim. Finally, a low proportion of applications with Lustre events at large-scale (roughly 80% occur for applications using less than 126 nodes), can be partly explained by small number of jobs competing for Lustre resources at the same time. This also shows that Lustre events are mostly caused due to user applications.

Fig. 3 shows the distribution of applications with recorded events based on execution times. The following observations can be made from these results: majority (more than 50%) of applications with Lustre, segmentation faults and OOM errors, processor errors, and GPU errors have execution times between 1 hour and 4 hours; MCEs once again seem to occur uniformly across applications with different execution times; interconnect events are mostly recorded in short lived executions, which also holds true when they appear in combination with MCEs; events from distinct event categories appear more often together in applications with execution times over 8 hours. In contrast to results in Fig. 2, Lustre, segmentation faults and OOM errors, processor errors and GPU errors seem to be distributed across applications with various execution times. However, presence of their majority in the 1 to 4 hours time window is interesting and needs to be investigated further.

Most of these findings are intuitive, for example, the probability of having an event is higher for large-scale and lengthier executions. Other observations include: MCEs are the most common event type occurring in applications and do not relate to size or runtime, i.e., they mostly occur randomly; Lustre, segmentation faults and OOM errors, and GPU errors occur mostly in small-scale applications, although part of this might be due to users testing their codes or due to contention of resources; interconnect errors mostly occur in medium to large-scale and short-lived applications. These findings can be used by extreme-scale system users as guidelines to deploy resiliency mechanisms in their applications.

IV. Statistical Analysis for Comparing Applications With and Without Events

In this section, we perform statistical analysis between execution times of applications with no events and execution times of applications with events. The goal is to establish whether occurrence of events cause a significant and noticeable difference in performance as compared to the error-free case. We compare the runtimes of applications with same binary names and using same number of nodes. This analysis is possible since we observe that users tend to run multiple instances of the same application. The assumption is that the application runs being compared utilize the same workload. Unfortunately, we do not have resource utilization information, such as memory utilization, available as part of this study, which we could use to verify this assumption. However, it is plausible to make this assumption since users on extreme-scale systems often run well-established workloads.

We perform two sample t-tests on multiple applications, for whom we have executions with events and corresponding executions with no event occurrences. These tests consider both the sample size and variability in the data. We are able to perform this analysis on a wide variety of applications since over 70% users in the system are impacted by events in the system during our analysis period. The two samples represent two different groups for each distinct binary name using distinct number of nodes. Specifically, we test the *null hypothesis* that "*execution times of applications with and without events are same*," i.e., there is no significant statistical difference between the execution times of applications irrespective of whether events take place or not. This hypothesis testing is done using Spark. We use, LogSCAN [11], for this analysis.

The results of two sample t-tests reveal that the null hypothesis can be rejected with a p-value of less than 0.1 in 48.1% cases. Alternatively, the two groups of execution times are from two different distributions. In these cases, the t-scores are significantly high with a low p-value. Note, a high t-score represents that the two samples are significantly different and a corresponding low p-value indicates confidence in the results. A low p-value indicates that the sample is inconsistent with the null hypothesis. As an example, Fig. 4 highlights the distinction in execution times of a select few applications whereby runtimes are grouped and plotted separately to distinguish runs with events from runs without events. The disparity in execution times is obvious by comparing the difference in median values between the two sets. The execution times also show performance variation, common in extreme scale systems. Although not investigated thoroughly in this work, one of the reasons of performance variation is the occurrence of RAS events during executions, which may or may not take place due to random failure events in the system. In the next section, we measure the impact on application performance as a result of RAS events.

V. Slowdown Analysis

The presence of statistically different groups of executions for same applications as demonstrated in previous section allows us to perform the slowdown analysis herein. The mean of one group is compared with the mean of other group to measure slowdown, i.e., the ratio of average of execution times with events to average of execution times without events. In our analysis, we find significant slowdowns in majority of the

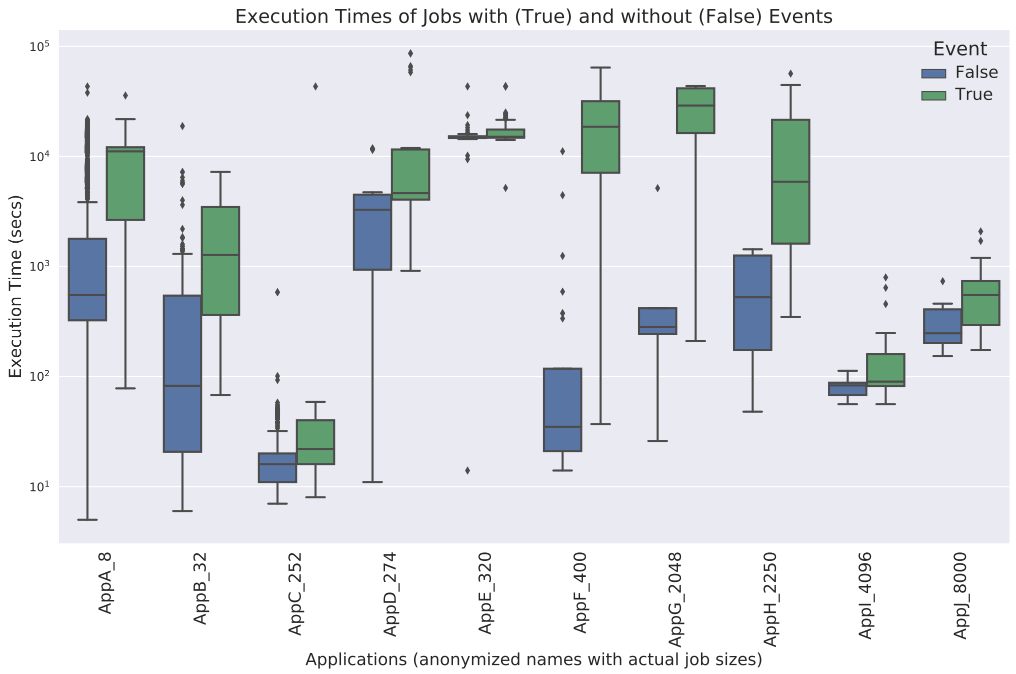


Fig. 4: Plots of selected application runtimes sampled with and without events, showing performance variation and distinct runtimes when RAS events coincide with application runs as compared to the runs with no event. The side-by-side comparison between runtimes shows the expected slowdown as a result of RAS events.

cases, as discussed later on. Based on this finding, we attempt to address the following questions: 1) *do different system components impact application performance differently*, 2) *do larger-scale applications see a greater impact on performance as compared to medium- or small-scale applications*, 3) *do high number of recorded events during an application run cause a greater impact on application performance*, 4) *do occurrence of events across multiple nodes in an application (beyond a single node) cause a greater impact on application performance*. First, we compare performance difference caused due to different system components.

A. Event type and Slowdown

Fig. 5 shows the box plots of slowdowns across different application runs grouped together based on application name and size and then distributed based on the type of RAS events observed in each group. Results indicate that the slowdowns across applications are significant irrespective of the event type, i.e., majority of the slowdown values for different applications lie somewhere between 1 and 10. These results also indicate the performance variation (inconsistent impact on performance) common in extreme-scale systems. Among all classes, the highest variation in performance is due to Lustre events. Additionally, MCEs, segmentation faults, and OOM errors seem to result in a number of outlier cases with performance impacts well over 100X in some cases. Due to the inconsistency, speedups are also observed in some cases (where slowdown is less than 1). One possible explanation of speedups is the early termination of applications due to fatal events. However, we are not able to verify this, as highlighted earlier.

The differences in slowdowns due to different event types and combination of distinct event categories are listed in the results in Table II. It shows the median and average values of the slowdowns across different application runs, and also presents the proportion of cases in which a slowdown is observed (see last column '% Applications'). Among all event classes, highest slowdowns are due to MCEs (both median and average values are high). On the other end of the spectrum, the least impact on application performance is due to interconnect events, which also has the least proportion of runs with slowdowns among all the cases listed.

Overall, a difference in performance impact exists across different system components, e.g., average slowdown due to Lustre events is 9.08 compared to 22.14 for MCEs in the processors. Another interesting observation is the general trend of higher average and median slowdown values when events from distinct categories occur together in the same application run. For example, the median slowdown is as high as 3.04 (average slowdown is 28.06) when MCE, Lustre, and OOM events occur together, as opposed to 1.43 when MCEs occur alone. The proportion of runs with slowdowns are also higher in these cases (over 85% cases). These results show that if an application run coincides with RAS events from two or more distinct system components, it is more likely to encounter a higher slowdown.

An open research question after this analysis is that why do not occurrence of RAS events always cause a slowdown.

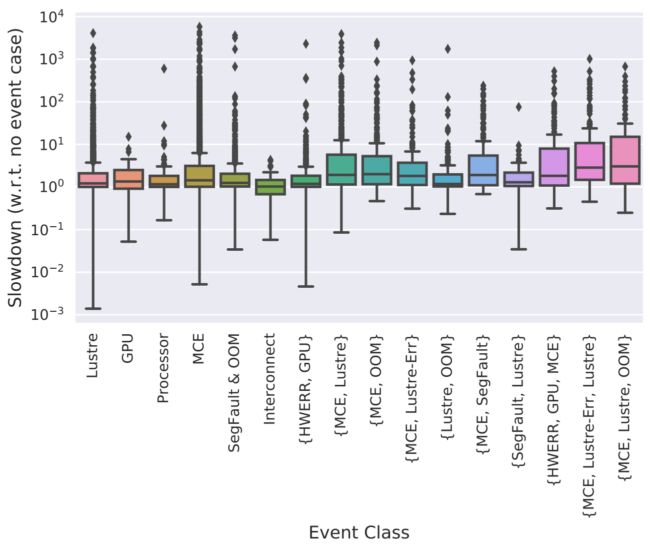


Fig. 5: Application slowdowns w.r.t. corresponding no event executions distributed based on type of recorded events. Event classes are defined in Section III-A. Commonly co-occurring events from multiple classes are also shown.

TABLE II: Median and average values of slowdowns from Fig. 5, and proportion of runs in which a slowdown is observed.

Event Class	**Median**	**Average**	**% Applications**
Parallel File System	1.21	9.08	75.7%
Processor Events	1.16	6.41	76.5%
Machine Check Exceptions	1.43	22.14	78.3%
Graphics Processing Unit	1.34	2.16	72.5%
Seg-Faults & Out-of-Mem.	1.24	12.30	82.2%
Interconnect	1.02	1.23	52.4%
Event Groups – selected, sorted by # of cases			
(MCE, Lustre)	1.92	36.45	88.7%
(HWERR, GPU)	1.18	10.39	76.8%
(MCE, Out-of-Mem.)	2.00	32.77	89.3%
(Lustre, Out-of-Mem.)	1.18	13.08	84.0%
(MCE, Lustre-Err)	1.81	16.33	84.4%
(MCE, Seg-Fault)	1.91	35.96	90.2%
(HWERR, GPU, MCE)	1.82	18.04	84.6%
(MCE, Lustre-Err, Lustre)	2.85	31.87	89.9%
(Seg-Fault, Lustre)	1.29	2.34	85.9%
(MCE, Lustre, Out-of-Mem.)	3.04	28.06	87.8%

Although early termination of applications partly answers this question, we suspect this may not always be the case. Another aspect which needs to be considered is the relationship between resource utilization and the performance overhead once events are recorded due to that particular resource. For example, increased memory utilization of applications has been associated to higher overheads due to MCEs in previous work [6]. Similarly, increased utilization of Lustre I/O bandwidth may be associated to increased chance of slowdown due to subsequent Lustre events. On the other end, occurrence of random Lustre events not associated with increased utilization may not impact application performance. We plan to investigate this question further in future work. Next, we investigate whether using large number of nodes in the system correspond to an increase in slowdown.

B. Application Size and Slowdown

The impact of number of nodes utilized in the system on application slowdown is assessed by grouping the slowdowns of the applications based on size. This set of results shown in Fig. 6 is used to gauge whether expected slowdowns increase as higher number of nodes are used. Comparison of application slowdowns in distinct size bins shows that large-scale applications tend to have higher median slowdown. A clear distinction can be seen between applications of size greater than 3749 and other applications of smaller sizes. We also notice a higher performance variation in small-scale applications, which can be explained by dominance of Lustre events in these applications as observed earlier. A higher median slowdown for large-scale applications as compared to small-scale applications can be explained by dominance of MCE events in the former, which tends to cause higher

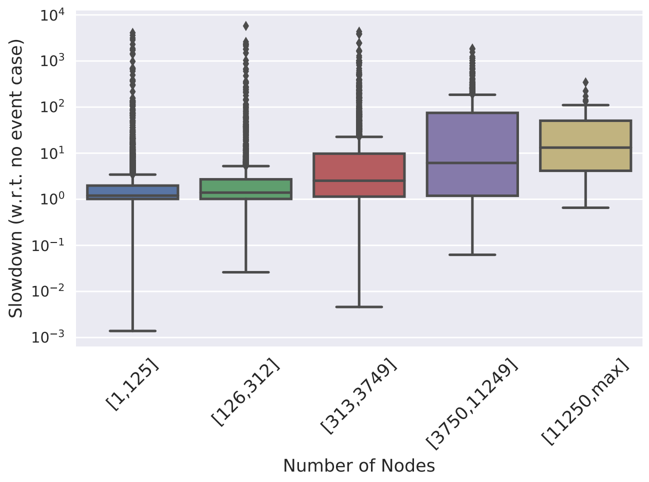


Fig. 6: Application slowdowns w.r.t. corresponding no event executions distributed based on number of nodes.

slowdowns as compared to other components. It should be pointed out that there are less samples for the last two bins (with applications of size greater than 3749) as compared to other bins, since corresponding no event cases occur less often in these bins as observed from the results earlier (see Fig. 2).

C. Number of Events and Slowdown

To investigate the effect of logging infrastructure on application slowdown, we assess how the number of events reported during application runs impact slowdown. In particular, we are interested to find out if increase in number of recorded events cause an increase in application slowdown. This increase might be due to a condition in some component which is triggered over and over again by the application. For example, an application might continuously write to a permanently failed bit in the memory which is corrected on every write using ECC and every time a MCE event is recorded.

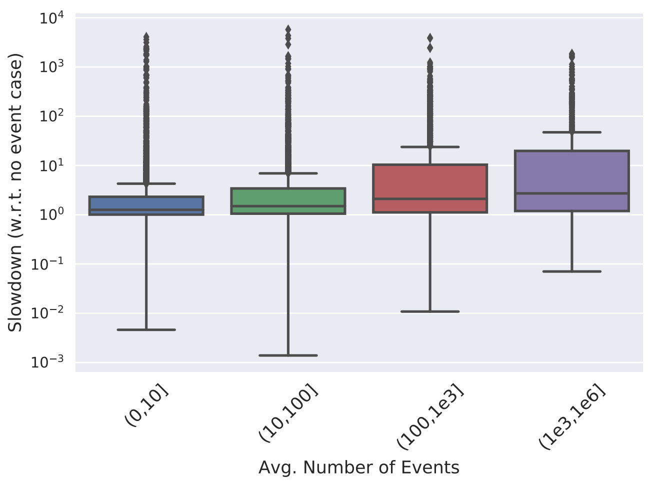


Fig. 7: Application slowdowns w.r.t. corresponding no event executions distributed based on average number of events observed.

Fig. 7 shows the impact of having large number of events recorded during application runs. These results are obtained by averaging number of events recorded in each application group for which the slowdown is being assessed. Most of the times, larger number of events cause more slowdown in applications as compared to applications where less number of events are observed. However, this does not represent the complete picture for all the different event types. This is because some system components tend to generate a large number of events in a short interval as a result of single root-cause, i.e., reporting mechanisms vary across components. For example, Figs. 8a, 8b, and 8c show the disparity in the average number of events recorded in applications as a result of Processor, MCE, and Lustre events, respectively. A separate correlational analysis across applications with events recorded due to different system components shows that higher number of events do not always cause a higher slowdown. Specifically, correlation coefficients (between slowdown and number of events) for MCE and interconnect events are 0.27 and 0.48 respectively, whereas, they are close to zero for other event types. This shows that number of events is not a strong indicator for slowdown across different system components, and the logging infrastructure does not seem to interfere with application performance in most cases. As discussed earlier, multiple reasons may cause slowdown such as utilization of resources, number of nodes utilized, and it is challenging to find an exact root cause of these slowdowns. Next, we investigate how proportion of nodes with events recorded within an application relate to slowdown.

D. Proportion of Nodes with Events and Slowdown

Depending on the cause of RAS events, they may be recorded in a single node or multiple nodes of the application. Fig. 9 shows the histogram of percentage nodes with recorded events out of all the allocated nodes (note the logarithmic scale of the y-axis). The histogram shows a bimodal distribution, whereby either most applications have between 0 to 10% nodes with events or 100% nodes with events. We find that MCEs mostly occur on a fraction of the allocated nodes, i.e., 91.2% of applications with MCEs have between 0 to 5% of the total allocated nodes with events. Similarly, interconnect events also impact a fraction of the allocated nodes. On the other end of the spectrum, Lustre events are recorded on most of the allocated nodes. Other noticeable event types on this end of the spectrum include: segmentation faults and OOM errors, which is plausible. Events which occur on majority of the allocated nodes can be attributed to users in most cases.

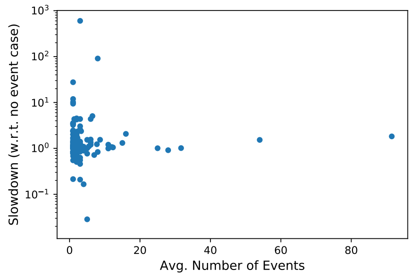
Based on the above, we investigate the relationship between slowdown and percentage of nodes with events. Fig. 10 shows the slowdowns based on percentage nodes with events. We do not observe a strong disparity among cases when percentage nodes with events are different. The median value of slowdown in first bin (0 to 20%) is the highest, which is explainable since MCE is mostly seen in such cases and it causes the highest slowdown among all the event classes, as observed earlier. One of the use of these results is to define a threshold number of nodes with events, which we can use to say that it will cause higher than average slowdown or that the errors are user induced.

E. Discussion

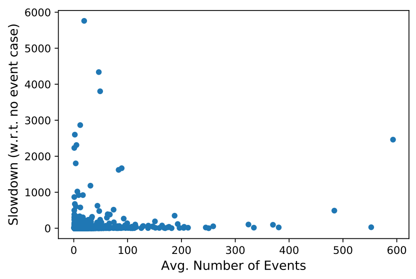
In this section, we found that occurrence of events do correspond to application slowdown in majority of the cases. However, finding the exact root cause is difficult. A limitation of this study is the non-availability of resource utilization information. This study establishes the need to develop fault injection mechanisms to trigger RAS events in different system components irrespective of whether an application failure takes place or not. It will enable systematic understanding of application and system characteristics which leads to application slowdown and performance variation. Although, some of this testing can only be performed while the system is not in use, which is often not feasible. We also plan to use the occurrence of different types of events in applications for classification of applications in extreme-scale systems, which can help system operators to deploy more efficient systems.

VI. Related Work

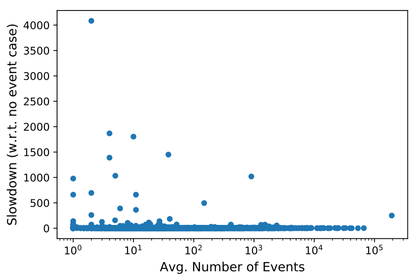
The impact of detectable and correctable memory errors on application performance has been investigated by Gottscho et al. [6]. A proprietary fault injection tool is used to assess the overhead of the memory error reporting stack. The simulated fault injection experiments showed that high workload or increased memory utilization leads to increased performance degradation, which is mostly due to overhead of the firmware/software stack and is not necessarily due to the ECC correction mechanism in hardware. However, their experiments are limited to a single machine with a very specific configuration/setup, and for CPU SPEC2006 benchmarks. A noticeable slowdown of 3746X under peak load is reported for an interactive web-search workload. In this



(a) Processor RAS events



(b) Machine check exceptions



(c) Lustre events

Fig. 8: Application slowdowns plotted versus average number of events recorded in different system components. The disparity in the number of events generated due to different system components can be observed by noticing the maximum values on the x-axes in each case.

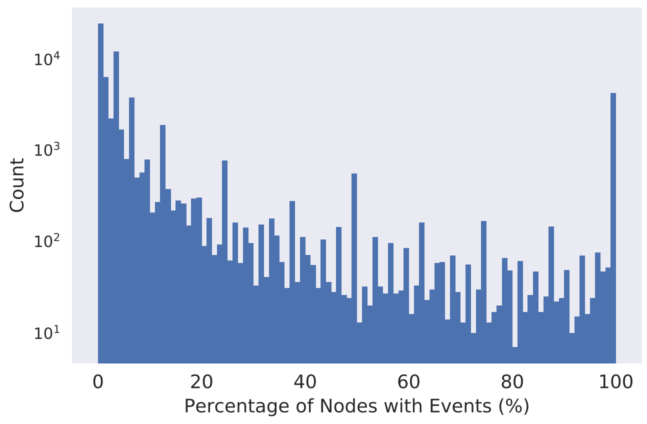


Fig. 9: Histogram of percentage nodes with recorded events out of all allocated nodes for applications with events.

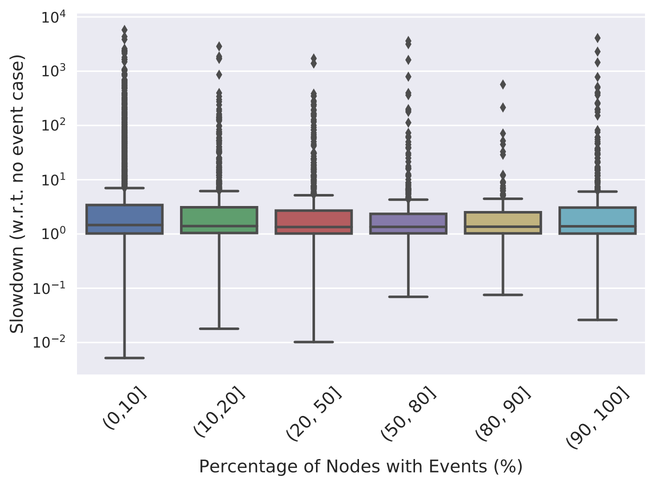


Fig. 10: Application slowdowns w.r.t. corresponding no event executions distributed based on percentage nodes with recorded events out of all allocated nodes.

work, we investigate similar effects on a wide variety of real-world applications and workloads executed on a deployed HPC system. Furthermore, we investigate the performance impact due to other components in a HPC system.

A similar study on a deployed system [7], the Blue Waters supercomputer at the National Center for Supercomputing Applications, analyzes the resiliency of applications to failure events in the system. The efficacy of resiliency mechanisms implemented in applications is also investigated. However, they do not investigate the impact of reliability events on application performance, which is a contribution of this work.

Multiple studies have been conducted on deployed HPC systems in the past to assess failure rates and mechanisms. For example, multiple generations of supercomputers at OLCF [1] have been analyzed. System failure rates are measured and they are found to vary drastically from one time period to another. It is concluded that use of mean-time-to-failure, commonly used for deploying resiliency mechanisms in extreme-scale applications is not adequate. Spatial and temporal analysis of failures in different systems is also done. A similar analysis is performed on the Blue Waters supercomputer by Martino et al. [2]. In case of Blue Waters, it is found that hardware-related failures are not the major cause of system downtime since protection mechanisms such as ECC in memory are effective. In this work, we analyze the impact of such errors, failures and protection mechanisms on the performance of extreme-scale applications.

Previous works have also exclusively studied memory failure data in HPC systems. For instance, DRAM and SRAM failure data for 6000-node and 8500-node systems at Lawrence Berkeley National Laboratory and Los Alamos National Laboratory were analyzed to establish the need to develop effective DRAM protection schemes [3]. Through this work, we establish the need to develop low-overhead protection mechanisms and reporting infrastructure. Similarly, DRAM errors in multiple generations of IBM Blue Gene systems and Google data-centers are analyzed in [12] and [5], respectively. Again, most of these studies are performed independent of application executions and therefore do not consider the impact on application performance.

In this work, we utilize LogSCAN [11], which is a multi-user distributed analytics framework for processing log data. Distributed analytics is performed using Spark and the data is ingested in Cassandra, a distributed database. This framework enables us to perform efficient co-analysis on multiple distinct databases, such as RAS log and application scheduling and placement log, simultaneously. For example, database joins are performed to find which events take place during application executions. Similarly, use of group-by queries enable slow-down analysis, whereby, a set of execution times is compared with a corresponding set of execution times for applications having same binary names and using same number of nodes. Some other failure log analysis tools include: ELSA, used for analysis of failure logs from Blue Waters [13]; LogDiver, used for evaluating resiliency of HPC applications in Blue Waters [14]; DESH, used for predicting lead time to failures in Cray machines with deep learning [15].

VII. Conclusion

A correlational analysis between system events and application executions is conducted in this work. The analysis shows the performance impact on applications when system events coincide with their execution. We quantify this performance impact, and show it as one of the causes of performance variation observed in extreme-scale computing systems. We then investigate the relationship of application slowdown with RAS events from different system components, application size utilized, number of events recorded during application execution, and proportion of allocated nodes with recorded events. We find that some system components cause more slowdown as compared to other components. We also find that some events are caused due to user applications, such as those occurring on all the nodes allocated to an application. Whereas, some event types are truly random events in the system, such as machine check exceptions. It is important to address these cases appropriately and mitigate their performance impact. In some cases, given the current state of large-scale systems, it may be feasible to provide feedback to users, in case a system event coincides with the execution of their application. Such feedback will be useful to limit the number of re-runs for scientists doing performance runs and/or science experiments increasing operational efficiency of HPC systems.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Resilience for Extreme Scale Supercomputing Systems Program, with program manager Lucy Nowell, under contract number DE-AC05-00OR22725. This work was also supported by the Compute and Data Environment for Science (CADES) facility and the Oak Ridge Leadership Computing Facility (OLCF) at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the U.S. Department of Energy (under the contract No. DE-AC05-00OR22725).

References

[1] S. Gupta, T. Patel, C. Engelmann, and D. 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 (SC17)*, 2017, pp. 44:1–44:12.

[2] C. D. Martino, Z. 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 *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2014, pp. 610–621.

[3] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory errors in modern systems: The good, the bad, and the ugly," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS' 15)*, 2015, pp. 297–310.

[4] N. El-Sayed and B. Schroeder, "Reading between the lines of failure logs: Understanding how HPC systems fail," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2013, pp. 1–12.

[5] B. Schroeder, E. Pinheiro, and W.-D. Weber, "DRAM errors in the wild: A large-scale field study," in *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '09, 2009, pp. 193–204.

[6] M. Gottscho, M. Shoaib, S. Govindan, B. Sharma, D. Wang, and P. Gupta, "Measuring the impact of memory errors on application performance," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 51–55, Jan 2017.

[7] C. D. Martino, W. Kramer, Z. Kalbarczyk, and R. Iyer, "Measuring and understanding extreme-scale application resilience: A field study of 5,000,000 HPC application runs," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2015, pp. 25–36.

[8] B. Nie, J. Xue, S. Gupta, T. Patel, C. Engelmann, E. Smirni, and D. 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)*, June 2018, pp. 95–106.

[9] 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 *48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018, Luxembourg City, Luxembourg, June 25-28, 2018*, 2018, pp. 107–114.

[10] 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 Transactions on Dependable and Secure Computing*, pp. 1–1, 2018.

[11] B. H. Park, Y. Hui, S. Boehm, R. A. Ashraf, C. Engelmann, and C. Layton, "A Big Data analytics framework for HPC log data: Three case studies using the Titan supercomputer log," in *Proceedings of the* 19^{th} *IEEE International Conference on Cluster Computing (Cluster) 2018:* 5^{th} *Workshop on Monitoring and Analysis for High Performance Systems Plus Applications (HPCMASPA) 2018*, Belfast, UK, Sep. 10, 2018.

[12] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic rays don't strike twice: Understanding the nature of DRAM errors and the implications for system design," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVII, 2012, pp. 111–122.

[13] A. Gainaru, F. Cappello, and W. Kramer, "Taming of the shrew: Modeling the normal and faulty behaviour of large-scale HPC systems," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, May 2012, pp. 1168–1179.

[14] C. D. Martino, S. Jha, W. Kramer, Z. Kalbarczyk, and R. K. Iyer, "LogDiver: A tool for measuring resilience of extreme-scale systems and applications," in *Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale*, ser. FTXS '15, 2015, pp. 11–18.

[15] A. Das, F. Mueller, C. Siegel, and A. Vishnu, "Desh: Deep learning for system health prediction of lead times to failure in HPC," in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '18, 2018, pp. 40–51.