Analyzing the Interplay of Failures and Workload on a Leadership-Class Supercomputer

Esteban Meneses*, Xiang Ni†, Terry Jones‡, and Don Maxwell§
*Center for Simulation and Modeling
University of Pittsburgh
Email: emeneses@pitt.edu
†Department of Computer Science
University of Illinois at Urbana-Champaign
Email: xiangni2@illinois.edu
‡Computer Science and Mathematics Division
Oak Ridge National Laboratory
Email: trjones@ornl.gov
§Oak Ridge Leadership Computing Facility
Oak Ridge National Laboratory
Email: maxwellde@ornl.gov

***Abstract*—The unprecedented computational power of current supercomputers now makes possible the exploration of complex problems in many scientific fields, from genomic analysis to computational fluid dynamics. Modern machines are powerful because they are massive: they assemble millions of cores and a huge quantity of disks, cards, routers, and other components. But it is precisely the size of these machines that glooms the future of supercomputing. A system that comprises many components has a high chance to fail, and fail often. In order to make the next generation of supercomputers usable, it is imperative to use some type of fault tolerance platform to run applications on large machines. Most fault tolerance strategies can be optimized for the peculiarities of each system and boost efficacy by keeping the system productive. In this paper, we aim to understand how failure characterization can improve resilience in several layers of the software stack: applications, runtime systems, and job schedulers. We examine the Titan supercomputer, one of the fastest systems in the world. We analyze a full year of Titan in production and distill the failure patterns of the machine. By looking into Titan's log files and using the criteria of experts, we provide a detailed description of the types of failures. In addition, we inspect the job submission files and describe how the system is used. Using those two sources, we cross correlate failures in the machine to executing jobs and provide a picture of how failures affect the user experience. We believe such characterization is fundamental in developing appropriate fault tolerance solutions for Cray systems similar to Titan.**

***Keywords*-Failures; Cray; Fault Tolerance; Resilience.**

I. Introduction

The type of scientific and engineering simulations possible today is the result of a relentless increase in the computational power of supercomputers. The last decade brought major breakthroughs in the history of high performance computing; we now live in the era of petascale hybrid supercomputers. This type of system is characterized by a large amount of components all assembled together in a single complex system. Understanding the interaction of all the components in the system is fundamental to both today's complex systems as well as the design of the next generation of exascale machines. In particular, having a profile of failures and a set of principles to build resilient systems are key insights to bringing about a useful exascale system [1]. The first step towards that goal must be studying current leadership-class systems and describing the patterns of both their usage and their failures. Otherwise, fault-tolerance will remain one of the major challenges of extreme-scale computing [2], [3], [4]

In this paper, we investigate the interplay of workload and failures in the Titan supercomputer during 2014. We look at the job scheduler log and describe the workload of the machine during that year. For the same period of time, we look into the failure database constructed by the system administrators. We build a profile of the failures and separate them into the two overarching categories of software and hardware. We then investigate two workload topics: how does job submission frequency correlate with failure rate, and, more importantly, how do failures impact the workload of the supercomputer.

Notice: 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, world-wide 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).

This paper makes the following contributions:

- A characterization of the workload on Titan, a leadership-class supercomputer, for an entire year. We present several descriptors that can be used to model a workload for extreme-scale machines (§ III).
- A profile of failures in Titan during the same period. We highlight the most relevant patterns and show the mean-time-between-failures follows an exponential distribution (§ IV).
- A quantitative analysis of the impact of failures on the workload of Titan (§ V).
- A series of recommendations for understanding the interplay between failures and the workload on large supercomputers (§ VI).

II. Background

A. System Description

Titan is a Cray XK7 supercomputer located at the Oak Ridge Leadership Computing Facility (OLCF). It has a performance of 17.59 petaFLOPS according to the Top500 list [5]. Titan was one of the earliest major supercomputing systems to use a hybrid architecture of CPUs and GPUs. Each node has an AMD 16-core Opteron CPU and an NVIDIA Tesla K20 GPU. There are 18,688 nodes in total for a combined 299,008 Opteron cores. Each node has a total of 32 GB of main memory. Titan features a 3D-torus Gemini interconnect.

Titan uses the MOAB job scheduling and management system. Therefore, every single event related to the workload is recorded by the system. There is a MOAB log file for every day of activity on Titan. The format of a MOAB log file [6] contains a thorough description of each job event. Every time a job is submitted, cancelled, ended, or modified in any way, MOAB records the relevant information. For instance, there is a timestamp associated with every entry. Also recorded are the number of requested nodes, the amount of requested wallclock time, the submit time, and the start and finish time.

Titan is a leadership-class supercomputer, which means the type of applications that run on the system must scale as much as the code permits it, while using the hybrid nature of the architecture. Therefore, there are scheduling policies in place that encourage large job submissions and at the same time enforce a fair share access to the system. This is accomplished by establishing a `batch` queue with several queue classes or bins; each queue bin has its own size properties and wall clock constraints. Table I presents a list of the features in each bin. In particular, bin 5 represents *small jobs* that are meant for testing and debugging code.

Queue	Min Nodes	Max Nodes	Wallclock Limit (hours)
Bin 5	1	125	2
Bin 4	126	312	6
Bin 3	313	3749	12
Bin 2	3750	11,249	24
Bin 1	11,250	18,688	24

Table I: Quality of Service Bins on Titan

In addition to the scheduling system, there is a monitoring system that keeps track of all Titan failure incidents. All failure incidents are captured in a *failure database* [7]. This database is automatically constructed by a program designed by the system administrators. The program uses SEC (Simple Event Correlator) to enforce correlation rules that insert records in the database after examining multiple anomalous outputs in the system. Each entry in the failure database represents an event that can be tracked to a particular component in the machine and that is associated to a job identifier. These events may be related if the root cause of a failure has repercussions on other parts of the system. For example, a hardware malfunction may trigger an event on a monitor program as well as a failure in a user code running on that hardware. The severity of the different events in the failure database varies from warnings to catastrophic incidents. The job scheduler in Titan runs jobs with the assumption that any failure will be *fatal* for the job. Hence, a failure on the hardware assigned to a job will immediately finish the job execution, regardless of the state of the rest of components assigned to the job.

B. Related Work

Joubert and Su [8] presented a study on the application workload of Titan's predecessor at Oak Ridge National Laboratory, namely the Jaguar supercomputer. They analyzed MOAB and ALPS job logs and identified usage patterns of the machine according to different science domains. They listed the most heavily used applications along with their execution characteristics. Although their study focused on the distribution of the workload according to the specific domain of the application, they presented the correlation between general features of the jobs submitted. In particular, they found there is no strong correlation between job size and job execution time. Also, they reported a fraction of less than 15% of the total utilization of the system was due to jobs that executed longer than 12 hours. In this paper, we do not provide a breakdown of the workload according to the science domain of the jobs. Instead, we focus on providing a deeper characterization of the workload itself, the failures during the same period of time and the interaction between the two.

Schroeder and Gibson [9] studied 20 different systems at LANL. They have discovered that failure rates depend mostly on system size rather than hardware work. According to their analysis, there is a strong correlation between the number of failures and system workload. They further found that the time between failures of LANL machines is not modeled very well by an exponential distribution which is different from our findings.

Zheng et al. [10] did a co-analysis of RAS logs and job logs on the BlueGene/P system. They found that jobs that request large node-counts can significantly affect the failure rate. On the other hand, job duration produces a minimal impact upon failure rate.

Heien et al. [11] analyzed 5 years of system log from a production cluster. They found that Weibull and log-normal distributions fit the failure intereval times very well and 91% of the failures only affect one node at a time.

III. Workload Characterization

We built an analysis framework to understand the workload on the Titan supercomputer. The framework consists of multiple scripts that traverse the MOAB logs and distill the fundamental features of jobs that were executed during the 2014 calendar year. The scripts work in two stages. In the first stage, the scripts read every record in each of the MOAB log files and create a record per job. The collection of all job records forms a job database. In the second stage, the scripts sweep the job database and generate different statistics about the workload. In the database, each job's main features are recorded including job identification, submission date, number of requested nodes, amount of requested time, wait time, and execution time.

The analysis framework recorded a total of 375,387 job submissions on Titan during 2014. Out of those job submissions, a total of 324,585 (86.5%) reached the head of the queue and were scheduled for execution. The remaining job submissions were presumably cancelled either by the user or the system. However, some of the jobs that did run were not included in our job database because the MOAB logs were missing both the dispatch and the start time. We call those *incomplete jobs* and they add up to 4657. Therefore, after removing incomplete jobs and other inconsistent cases, the job database contains a total of 319,885 jobs. Those jobs combine for a total of 141,500,811 node service units during 2014. In this paper, we define a *node service unit* as the use of a node on Titan for one hour. Our definition does not correspond with the actual charging factor on Titan (where one hour of allocation on one node represents 30 service units), but simplifies the presentation of the results.

The analysis of submitted jobs presents several salient features with respect to the usage of the system. Figure 1 shows the distributions according to different time scales. The hour distribution (Figure 1a) shows how the system is utilized more heavily during the workday hours, from 9:00am to 6:00pm, with a peak at 2:00pm. According to Figure 1b, weekdays are approximately twice as busy for Titan compared to the weekend. The day with the highest number of job submissions is Monday. The usage of the system was more uniform when the month of the year is considered (Figure 1c), with a notable exception in February when the system saw relatively few job submissions. The reason behind the small amount of submissions during February relates to annual allocations: February is the month that Titan's yearly allocation cycle turns over.

Queue	Jobs	Percentage(%)	Node SUs	Percentage(%)
Bin 5	249,836	78.10	3,636,570	2.57
Bin 4	40,569	12.68	10,570,110	7.47
Bin 3	23,833	7.45	38,290,120	27.06
Bin 2	4861	1.52	65,911,080	46.58
Bin 1	769	0.24	23,092,930	16.32

Table II: Queue Statistics on Titan

The analysis of the set of *complete* jobs displays the fundamental features of the workload. Figure 2 offers a wide spectrum of descriptors aimed at characterizing the way in which Titan was used in 2014. The first feature we address is the distribution of jobs according to their size in number of requested nodes. Figure 2a presents the cumulative fraction of number of jobs when the number of requested nodes varies from 1 to 18,688 (the total number of nodes in Titan). The distribution shows how prevalent *small* jobs (using less than 126 nodes) are in the overall workload. However, small jobs account for an insignificant fraction of the total usage of Titan. Figure 2b offers the cumulative share of node service units executed by the jobs as the number of requested nodes changes. More specifically, small jobs represent 78.10% of all the complete jobs, but they only represent 2.57% of the total number of node service units used on Titan. A detailed description of the use of the different buckets for the quality of service in the `batch` queue is shown in Table II. The category with the highest utilization of Titan are the jobs requesting between 3750 and 11,249 nodes. Since the small jobs do not represent a significant fraction of the utilization of Titan, we excluded them in the rest of the workload analysis.

The features of all non-small jobs are depicted in Figures 2c through 2g. The number of jobs according to the number of requested nodes in Figure 2c shows the skewed distribution discussed above. There are a few popular quantities in the plot. The requested wall clock time in Figure 2d shows less diversity as users tend to request time in coarser units. Figure 2e shows that 87% of the jobs wait no more than 24 hours in the queue. There are a group of no more than 10 jobs that waited weeks in the queue before execution-start. The execution time distribution in Figure 2f shows two groups, one that takes up to 720 minutes (upper bound on Titan if the number of requested nodes do not exceed 3750), and the other that may run up to 1440 minutes (hard limit on Titan for any job). The last distribution in Figure 2g presents the number of node service units used. The highest number of node SUs consumed by any job was approximately 392,000.

Figures 2h through 2l offer a view on the combination of different dimensions. In general, it is not clear from the data that any two variables are correlated. Figure 2h presents the amount of requested wall clock time versus number of

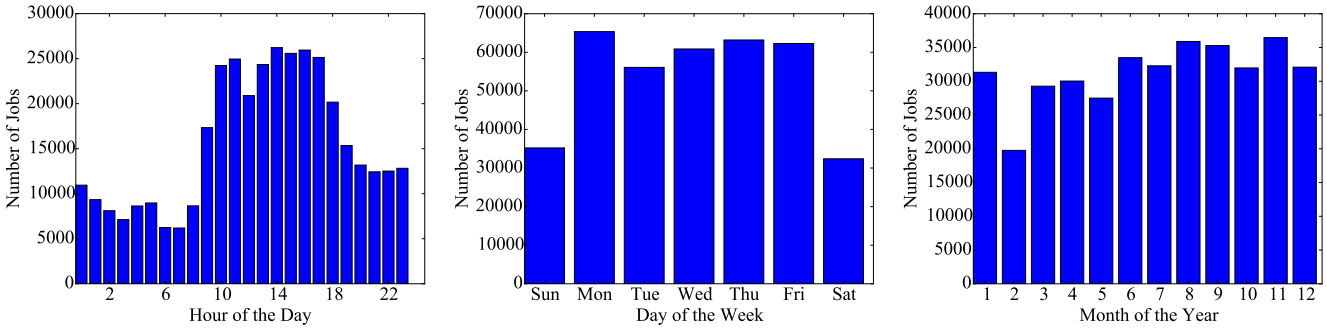

Figure 1: Distribution of Job Submissions during 2014

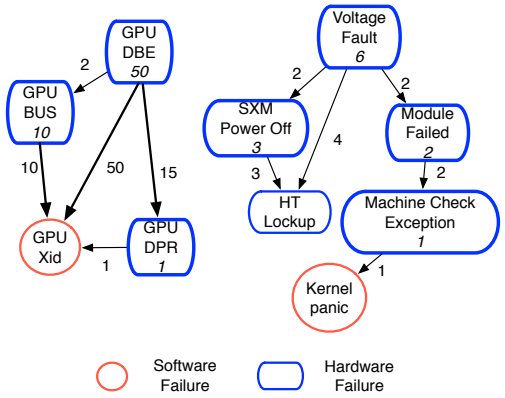

Figure 3: Failure Propagation

requested nodes. There is no clear trend in the data and jobs appear spread out across the space. The wait time associated to the number of requested nodes is shown in Figure 2i, again without indicating that more nodes would be harder (or easier) to obtain. In fact, the data reflects the actual complexity of scheduling policies in current supercomputers where the priority of a job depends on multiple variables; as such, it is hard to predict the wait time based on the sole number of requested nodes. Figure 2j also yields the same scattered data pattern, as do Figures 2k and 2l. To verify the visual signals in the figures, we computed the correlation coefficient between the variables. The results demonstrated how loosely connected the factors are. For example, the correlation coefficient of number of requested nodes versus requested wallclock time is 0.12, while the correlation coefficient of number of requested nodes versus wait time is 0.09.

IV. Failure Characterization

A failure record in the Titan database includes the identification of the job affected by the failure, the time when the failure is reported, the category of the failure (hardware or software), the cause of the failure (system or user code), the failure type and the nodes affected by the failure. There are more than $160,000$ entries in the original failure database of Titan. The massive amount of failure records is due to the redundant information in the database. For example, if a node fails when a job is running, then the node is marked for repair and it is not assigned to any other job. However, before proper repair action is taken, the system keeps reporting the failure of that node.

We first filter out the redundant information in the failure database. If there are many entries regarding the same type of failure related to a node during the execution of a certain job, we only keep one such entry. Sometimes a failure may affect more than one node. For example, a GPU graphics engine fault may bring down other components running the same job or a voltage fault may bring down all the 4 nodes on the same blade. As a result, in our analysis if multiple nodes encounter the same kind of failure in a 60 second duration, we count them as one failure.

Next, we find the root cause of each failure and only keep that entry. For example, a hardware GPU DPR (dynamic page retirement) failure may be followed by a software GPU Xid error on the same node. After confirming with Titan administrators, we mark such cases as a single failure and attribute it to the root cause. Thus we only keep the entry of hardware failure in this case. Figure 3 shows the instances of failure propagation we have observed. The number on an edge indicates the amount of instances that show a connection between the two failures. The number under a failure node indicates the number of instances where that failure is the root cause. For example, 50 of the *GPU Xid* failures, 15 of the *GPU DPR* (dynamic page retirement) failures and 2 of the *GPU Bus* (GPU off the bus) failures are caused by *GPU DBE* (double bit error) failures. In 50 cases, *GPU DBE* failure is the root cause. This is because when *GPU DBE* leads to the failures of *GPU DPR* and *GPU Bus*, *GPU Xid* failures also come along. *Voltage Fault* may also have different subsequences, such as *Module Failed*, *SXM Power Off* and *HT Lockup*. Please note that *HT Lockup*

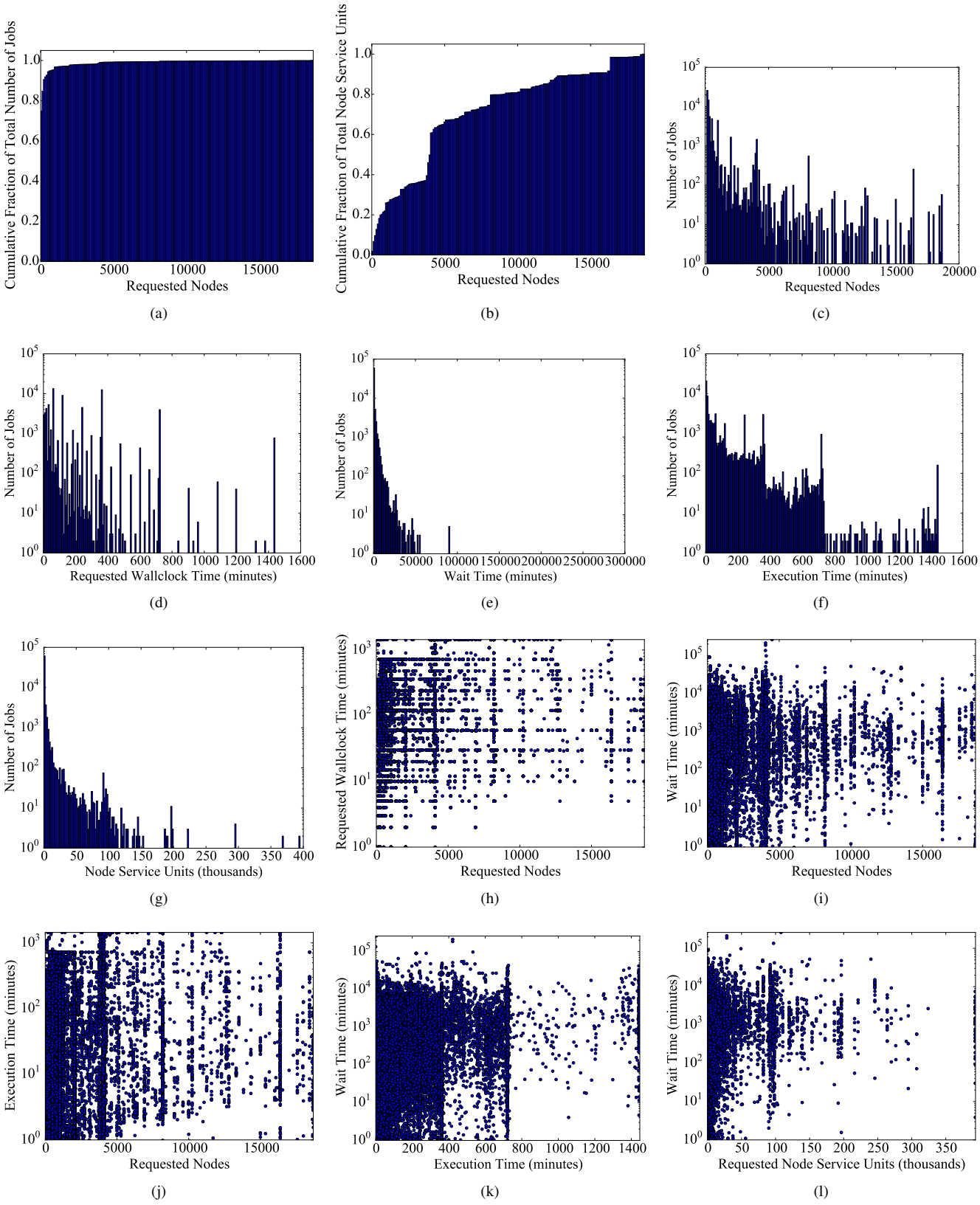

Figure 2: Workload Characteristics on Titan during 2014

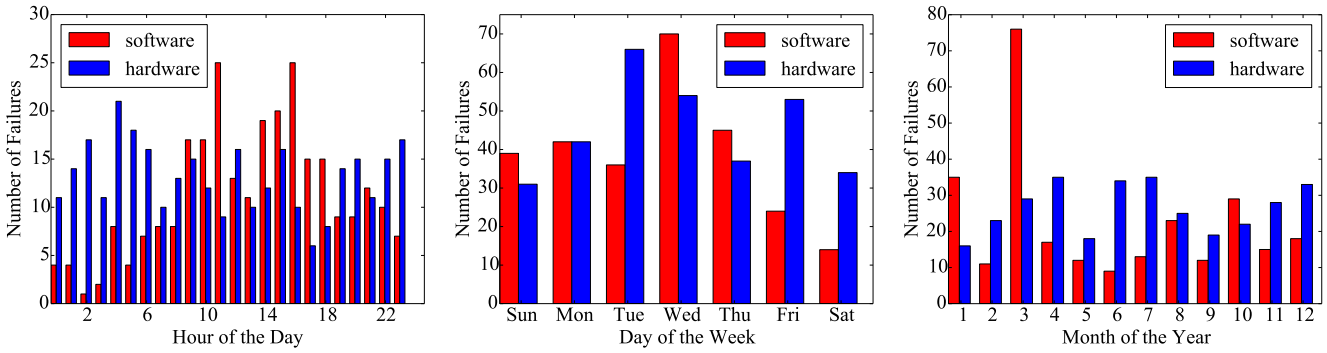

Figure 4: Distribution of Failures during 2014

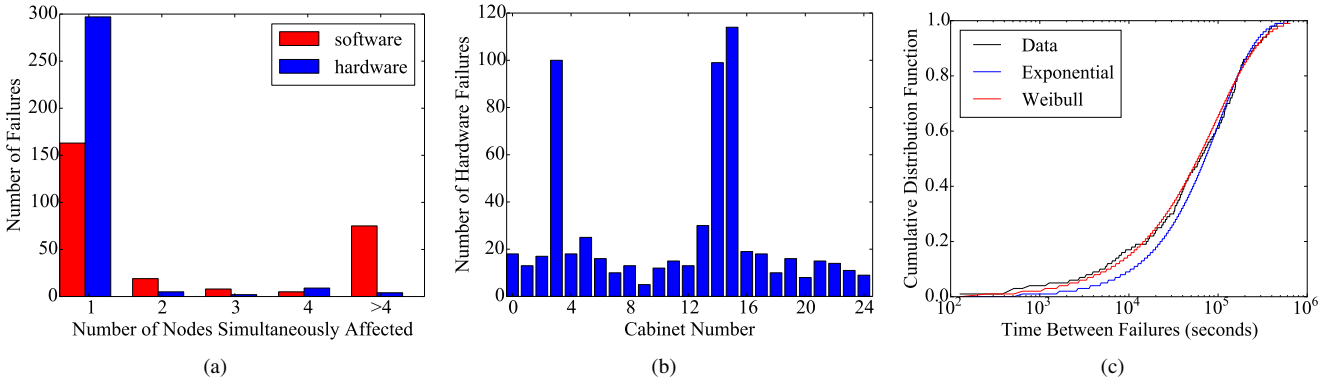

Figure 5: Failure Pattern on Titan during 2014

and *Kernel panic* failures are almost always caused by other failures, thus we exclude them in our later discussion.

Finally, in our study we do not consider failures of certain types. According to Titan administrators, `heartbeat fault` failures are mostly false positive alarms since a network congestion may cause nodes not to respond to the heartbeat within the time limit. Hence all the `heartbeat fault` failures are filtered out. We also filter out all the user code related failures such as `out of memory` failures. There are 3884 user failures in total. Among them, 2615 are `out of memory` failures and 1269 are `GPU memory page fault` failures that are caused by the user code.

Table III shows the break down for hardware failures after our filtering process. There are 317 hardware failures in total. *GPU DBE* is a GPU double bit error. *GPU DPR* refers to GPU dynamic page retirement (the NVIDIA driver will retire a page after it has experienced a single double bit error or two single bit errors). *GPU Bus* indicates that the GPU has fallen off the bus. *SXM* related failures happen when there are GPU problems. *Voltage fault* and *Module failed* are blade level failures. When such failures happen, it usually brings down all 4 nodes on the same blade at a time. As shown in Table III, this type of breakdown results in 45.4% GPU related hardware failures, 9.8% processor related failures, 37.9% memory related failures and 6.9% blade-level related failures.

Failure Category	Failure Type	Count	Percentage
GPU	GPU DBE	51	16.1%
	GPU DPR	66	20.8%
	GPU Bus	11	3.5%
	SXM power off	14	4.4%
	SXM warm temp	2	0.6%
Processor	Machine check exception Bank 0,2,6	31	9.8%
Memory	Machine check exception Bank 4, MCE	120	37.9%
Blade	Voltage fault	12	3.8%
	Module failed	10	3.1%

Table III: Hardware-rooted Failures

Failure Category	Failure Type	Count	Percentage
GPU	GPU Xid	267	98.9%
Lustre (reported client-side only)	LBUG	3	1.1%

Table IV: Software-rooted Failures

Table IV shows the break down for software failures. Note that in the failure logs we obtain, only a portion of Lustre

software failures are reported (i.e., only client side Lustre failures are included). Thus there was likely more Lustre software failures on the server side which were not included in the data we analyzed and are therefore not in the scope of this paper. As can be seen from Table IV, GPU related software failures constitute 98.9% of all the software failures reported and there are only 3 Lustre client side software failures out of 270 software failures. We must emphasize that not all file-system software errors are reported to the failure database. Therefore, this proportion may not represent the actual distribution of software failures on Titan. It does, however, show that GPU-related errors are significant in the system. According to [12], *GPU Xid* indicates a general GPU error; it can be indicative of a hardware problem, an NVIDIA software problem or a user application problem. In our filtering process, when there is a GPU Xid failure along with a hardware failure reported on the same node around the same time, we only keep the entry of the hardware failure. So the causes of the GPU Xid failures shown in Table IV should be either software or user problems if our process has successfully captured all relevant hardware problems and recorded them accordingly.

Figure 4a shows the number of failures by the hour of the day. As shown in Figure 4a, there is no clear correlation between hardware failures and time of day. However, there are more software failures in the daytime compared to at night; this suggests that user behavior plays an important role in software failures. In fact, the pattern in Figure 4a matches the distribution in Figure 1a.

Figure 4b shows the number of failures by day of the week. As shown in Figure 4b, the peak failure rate during the weekdays is twice as high as the failure rate on weekends. Again, the workload is heavier during weekdays as shown in Figure 1b.

Figure 4c shows the number of failures in each month of the year 2014. According to Figure 4c, there is no correlation between the number of hardware failures and the month of a year. However, a peak in software failures was recorded in March. A possible explanation is that there was an increase of GPU workload in March.

Figure 5a shows the number of nodes affected by one failure. According to Figure 5a, more than 93.7% of all hardware failures only affect one node. Out of 317 hardware failures in 2014, 9 failures affected 4 nodes. All of these 9 failure instances are blade-level failures (blade-level failures usually bring down all 4 nodes on the same blade). As for the 270 software failures, 163 failures (60.4%) are single node failures. 29.6% of software failures affect more than 4 nodes.

Figure 5b shows the number of hardware failures in each cabinet. Unlike the previous analysis, if there are multiple nodes affected by the same failure, we count the occurrence on each node as one failure to study the spatial correlation. As can be seen in Figure 5a, cabinet 3, 14 and 15 had the most hardware failures. After a detailed look into the failure logs, we found that `module failed` type failures brought down the entire cabinet at different times for cabinets 3, 14 and 15 . On closer examination, the Titan maintenance log points out that there was an LNET fallout due to a 2-cabinet warm swap and other reasons immediately after cabinet 14 and 15 went down.

We have tried to find the best distribution to describe mean time between hardware failures (MTBF) in Figure 5c. Both exponential distribution and Weibull distribution with shape parameter 0.82 fit the MTBF of Titan very well. The R-squared value for the goodness of exponential distribution is 0.91 while that value for Weibull distribution is 0.93. Using those distributions as a model for Titan's failures, the MTBF of the system is 28.63 and 27.63 hours, respectively, for the exponential and Weibull distribution.

V. Impact of Failures on Workload

To understand the effect of failures on the workload of Titan, we cross-correlated the failure information with the MOAB logs and produced a profile of the impact of failure on the different dimensions of the workload. Our methodology first separates the failures according to their cause, *user* or *system*. This division corresponds to the markedly different profiles of both types. A separation between the two causes allows us to better understand the impact of each cause. In either case, the filtered failures correspond to unique-job incidents. We use the job identifier to look for the features corresponding to the failed job.

The filtered database of *user* failures contains 3859 entries. There are only 10 failures for which the job information was not possible to find with our program. The other 99.74% of the failures correspond to jobs with full records in the MOAB logs. The results show that an overwhelming majority of the jobs that failed in the *user* category are small and short submissions. In fact, 71.4% of the failed jobs are small jobs (requesting at most 125 nodes). In addition, 85.5% of the failed *user* category jobs requested less than 2 hours of wall clock time. These two statistics lead us to conclude that *user* category failures occur mostly during test submissions. The total impact of user failures on the number of node services units is 2187.

There are 564 failures in the filtered database for the *system* type. Except in the case of 7 anomalous samples, when these failures include a particular job identifier they can be associated with entries in the MOAB logs. The problem with the 7 anomalous samples was incomplete information. For instance, an anomalous sample results when a *runaway* job starts execution but is never reported as cancelled, or failed, or ended. In such instances, we were unable to compute all the features of the job. Fortunately, for 98.75% of the filtered failure database, all the information on the jobs is available.

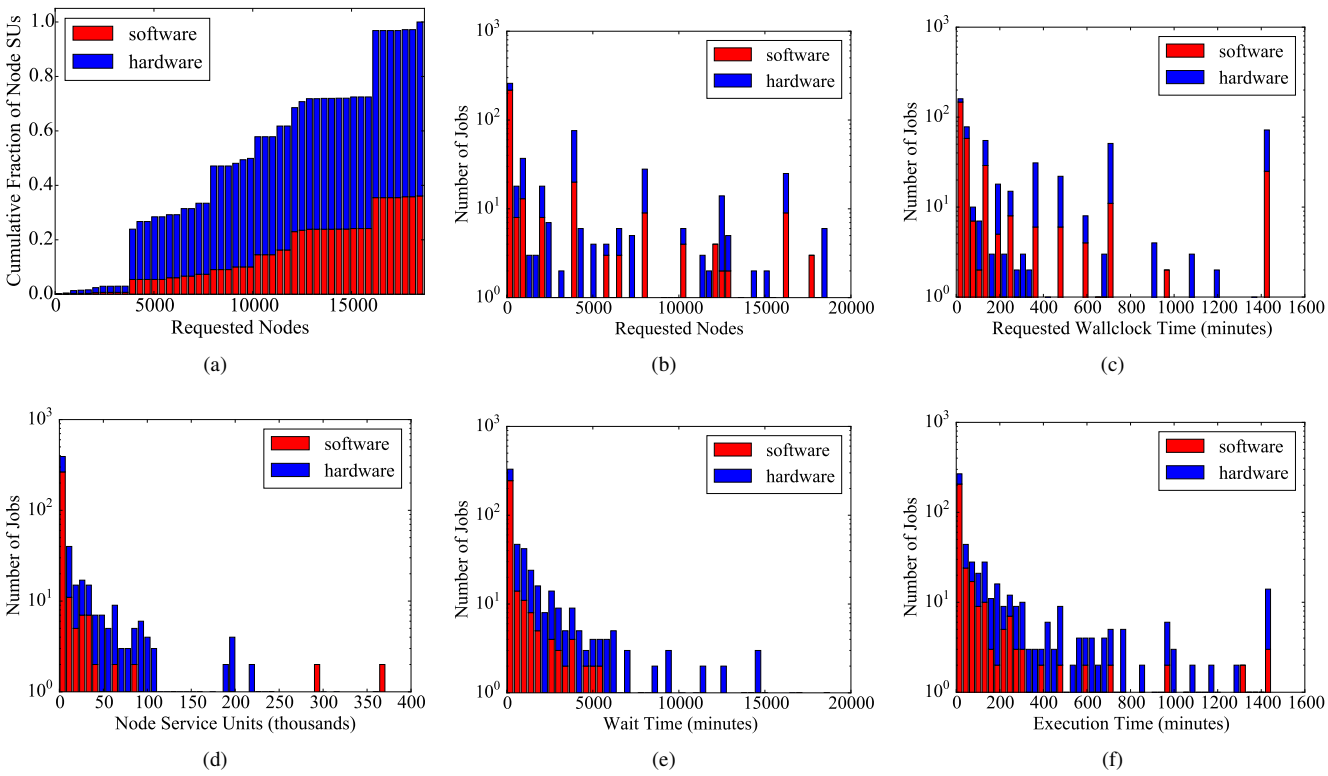

Figure 6: Effect of Failures on the Workload of Titan during 2014

The 557 system failures that can be correlated to job information in the MOAB logs are separated into two categories: software and hardware. We separate the two because each has a different profile as we will see below. We compute the impact of failures on the workload by counting the total resources allocated to a job. For instance, if a hardware failure shuts down the power source of one node allocated to a job running on n nodes, we assume the failure impacts the whole set of n nodes because the job must finish execution. With that perspective, the total number of nodes impacted by software failures is $676,535$, while hardware failures affected $1,258,717$ total nodes. In terms of the execution time, the total execution time of jobs that crashed because of software reasons adds up to $33,406$ minutes, while their hardware counterpart affected $75,706$ minutes of execution. The total wait time of jobs abruptly finished by software failures adds up to $267,495$ minutes, while hardware failures represent $582,545$ minutes in the queue. Finally, the impact of software failures on the total number of node service units is $4,080,000$, while hardware failures impacted $7,222,000$ total node service units. The combined number of node SUs represent 8% of the total number of node SUs executed on Titan during 2014.

Figure 6 shows a collection of descriptors for the set of jobs that failed because of a software or hardware malfunction. Figure 6a presents the cumulative share of total node service units executed by jobs before a crash occurs. The data shows that software failures represent 36% of the total number of affected node SUs, while hardware failures are responsible for the remaining 64%. The reason behind that distribution is offered in Figures 6b and 6c. Both figures show that software failures tend to affect small and short jobs. That is, jobs requesting few nodes and a short wall clock time. The net effect of that combination is also presented in Figure 6d where there are only a few software errors affecting large jobs. The wait time on jobs affected by either type of failure is shown in Figure 6e. The data demonstrates jobs affected by software failures wait less time than their hardware counterparts. However, jobs affected by hardware failures are more uniformly distributed in terms of execution time as presented in Figure 6f.

The fact that software failures tend to be reflected on short and small jobs seems to suggest the user may be somehow connected. Usually, small and short jobs are associated with software testing (programs that attempt to use a new type of strategy to solve a problem). We theorize such jobs may stress the system beyond its normal capacity or inappropriately use a resource, thus ending up in software failures.

VI. Recommendations

Based on the results in the previous sections, we offer a series of recommendations to improve the utilization of extreme-scale systems:

- *Incorporate fault-tolerance mechanisms at the user level*. As applications in different fields reach new scalability milestones, failures become a norm rather than an exception. The results in Section IV demonstrate that this is already the case in systems as big as Titan. The user code can not assume the underlying hardware is failure-free. Rollback-recovery strategies, such as checkpoint/restart must be considered as an option to permit applications to make progress in the presence of failures. The abstraction behind checkpoint/restart is simple enough to be easily adapted to most programs, and powerful enough to run across architectures.
- *Design fault-tolerance strategies that tolerate a single-node failure*. Figure 5a presents a distribution of the failures according to the number of nodes it affects. The majority of the incidents only affect a single node. Fault-tolerance strategies should be optimized for the common case and should perhaps sacrifice generality for performance.
- *Enable automatic restart at the runtime system level*. For certain types of failures where the affected component can be isolated from the rest of the system, the job scheduler may be able to avoid terminating the job execution by letting the runtime system handle the problem. Smart runtime systems are known to deal with this scenarios effectively [13]. Such a feature may prevent the time resubmissions spend in the queue, as presented in Section V. At the same time, runtime systems must develop strategies for running malleable jobs (shrink and expand to use a different number of nodes) and maintaining progress despite the presence of failures.
- *Maintain separate queues for development and testing*. The results in Section V may suggest many software failures are caused by the stress imposed on the system by user codes. Therefore, during testing and debugging, when failures are more prone, special queues would confine the impact of failures. In addition, a special queue for the development of large node-count but short duration testing jobs may prove useful.
- *Simulate extreme-scale workloads and failure rates*. Simulating large-scale systems may provide understanding of the interplay between workload and failures in different environments. This paper contain descriptors of both the way a machine is used and the way a machine fails. Those values can be plugged into a simulator to measure how a particular variable may change the overall utilization of the system.

VII. Conclusion and Future Work

Failures are a common occurrence in large-scale systems. Activities which yield understanding of failure phenomena are essential, especially activities which yield understanding of how workload and failures interplay. This paper explored both aspects in Titan, a leadership-class supercomputer.

The results show that the workload of the system may be behind a significant number of software errors in the system, even when there is no direct proof the user actually caused the failure. The effort of running demanding codes may put the system through excessive stress that inevitably cause failures. On the other hand, the failures of the system may heavily impact running jobs, forcing users to resubmit the failed jobs. Those failures affect approximately 8% of the total execution units of the system. In the worst case, those calculations must be repeated. In addition, a job resubmission must spend time in the queue, potentially delaying new scientific and engineering discoveries.

We believe future extreme-scale systems must be designed with careful consideration given to resilience. In particular, user codes, runtime systems, and job schedulers must be written with the explicit assumption that the underlying hardware may fail.

Acknowledgements

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

References

[1] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, D. Barkai, T. Boku, B. Chapman, X. Chi, A. Choudhary, S. Dosanjh, T. Dunning, R. Fiore, A. Geist, R. Harrison, M. Hereld, M. Heroux, K. Hotta, Y. Ishikawa, Z. Jin, F. Johnson, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, A. Lichnewsky, B. Lucas, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M. Mueller, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, A. V. D. Steen, J. Vetter, P. Williams, R. Wisniewski, and K. Yelick, "The international exascale software project roadmap 1."

[2] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. DeBardeleben, P. C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyffer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen, "Addressing failures in exascale computing," *IJHPCA*, vol. 28, no. 2, pp. 129–173, 2014.

[3] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir, "Toward exascale resilience: 2014 update," *Supercomputing frontiers and innovations*, vol. 1, no. 1, 2014. [Online]. Available: http://superfri.org/superfri/article/view/14

[4] E. N. Elnozahy, R. Bianchini, T. El-Ghazawi, A. Fox, F. Godfrey, A. Hoisie, K. McKinley, R. Melhem, J. S. Plank, P. Ranganathan and J. Simons, "System resilience at extreme scale," Defense Advanced Research Project Agency (DARPA), Tech. Rep., 2008.

[5] "Top500 supercomputing sites," http://top500.org, 2013.

[6] "Moab workload event format," http://docs.adaptivecomputing.com/.

[7] D. M. et al, "Restoring the cpa to cnl," in *Cray User Group Conference*, 2008.

[8] W. Joubert and S. Su, "Application workloads on the jaguar cray xt5 system," in *Cray User Group Conference*, 2012.

[9] B. Schroeder and G. Gibson, "A large scale study of failures in high-performance-computing systems," in *International Symposium on Dependable Systems and Networks (DSN)*, 2006.

[10] Z. Zheng, L. Yu, W. Tang, Z. Lan, R. Gupta, N. Desai, S. Coghlan, and D. Buettner, "Co-analysis of ras log and job log on blue gene/p," in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 2011, pp. 840–851.

[11] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello, "Modeling and tolerating heterogeneous failures in large parallel systems," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*. IEEE, 2011, pp. 1–11.

[12] "Nvidia xid errors," http://docs.nvidia.com/deploy/xid-errors/.

[13] E. Meneses, X. Ni, G. Zheng, C. L. Mendes, and L. V. Kale, "Using migratable objects to enhance fault tolerance schemes in supercomputers," in *IEEE Transactions on Parallel and Distributed Systems*, 2014.