LogDiver: A Tool for Measuring Resilience of Extreme-Scale Systems and Applications

Catello Di Martino, Saurabh Jha, William Kramer, Zbigniew Kalbarczyk, Ravishankar K. Iyer
University of Illinois at Urbana Champaign Urbana, IL, USA
{dimart, sjha8, wtkramer, kalbarcz, rkiyer}@illinois.edu

ABSTRACT

This paper presents *LogDiver*, a tool for the analysis of application-level resiliency in extreme-scale computing systems. The tool has been implemented to handle data generated by system monitoring tools in Blue Waters, the petascale machine in production at the University of Illinois' National Center for Supercomputing Applications. The tool is able: i) to filter, extract, and classify error data from different sources of information, such as system logs, hardware sensors and workload logs; ii) to extract signals from the categorized errors; iii) to consolidate user application data and decode application and job exit status, highlighting the reasons for the application/job exit; and iv) to correlate application failures with errors using a mix of empirical and analytical techniques. To the best of our knowledge, this is the first tool capable of measuring application-level resiliency in extreme-scale machines. We also demonstrate the power of the tool by showing that XK applications are more vulnerable to failures when compared to XE applications.

Keywords

B.8.1 [Performance and Reliability]: Reliability, Testing, and Fault-Tolerance - HPC applications; Log Analysis

1. INTRODUCTION

Although today we understand the main characteristics of failures in supercomputing environments [1–8], the issue of job and application resiliency has been less well-studied. Modern supercomputers are equipped with fault-tolerant infrastructures that are capable of protecting job and application executions from failures due to either hardware or software problems. Hence, important questions are, what are the errors and failures that affect the resiliency of jobs and applications executing on supercomputers? And what factors are important to characterize such errors?

This paper presents *LogDiver*, a tool for the analysis and measurement of system- and application-level resiliency in extreme-scale environments. Unlike past work, *LogDiver* focuses on the analysis of user applications, i.e., the compiled programs launched by user jobs, which can execute across one or more compute nodes. Parallel jobs can spawn several applications at a time, i.e., different programs on one or multiple nodes, that can execute concurrently and/or sequentially. We claim that it is important to analyze the error behavior of applications launched within a job to accurately characterize both system- and application-level resiliency.

Why *LogDiver*? The analysis of application-level resilience requires use of a mix of empirical and analytical techniques: (i) to handle the large amount of textual data; (ii) to decode specific types of system events and application/job exit codes obtained from multiple data sources, e.g., system-generated syslogs, job scheduler logs, and application scheduler logs; (iii) to extract signals of interest (e.g., error rates); and (iv) to measure error propagation and application resiliency. *LogDiver* is a tool set that supports automated error characterization of large-scale systems in a holistic manner. The *LogDiver*-based analysis allows us to examine error propagation patterns, estimate the likelihood of error detection and the frequency of error occurrence, measure error impact(s) at both system and application levels, and assess efficiency of error recovery. Data-driven analysis conducted using *LogDiver* produces highly robust metrics for quantifying the impact of exogenous variables (e.g., the system load, the application scale, and the user expertise) on the application-level resiliency. To the best of our knowledge, this is the first tool that allows the characterization of errors observed at the granularity of user applications (rather than at the batch job level). Such in-depth understanding of application error sensitivity is essential for realistic performance evaluation of current systems and to guide design of resiliency mechanisms.

In this paper, we describe *LogDiver's* workflow and provide examples of the types of analysis the tool supports. Overall the data includes about 6 TB of syslogs, and reflect more than 4 million user applications launched from March 1, 2013 to July 27, 2014 by more than 650,000 jobs run by 919 users, totaling more than 190 million node hours.

2. THE *LOGDIVER* APPROACH

LogDiver is a tool created to measure application-level resiliency of extreme-scale machines in a holistic manner. The *LogDiver*-based analysis allows us to create a unique dataset encapsulating events that are central in i) performing resiliency and performability measurements, iii) support the application of machine learning techniques to create application-level error detectors, and iii) measuring how

FTXS'15, June 15, 2015, Portland, Oregon, USA.

DOI: http://dx.doi.org/10.1145/2751504.2751511.

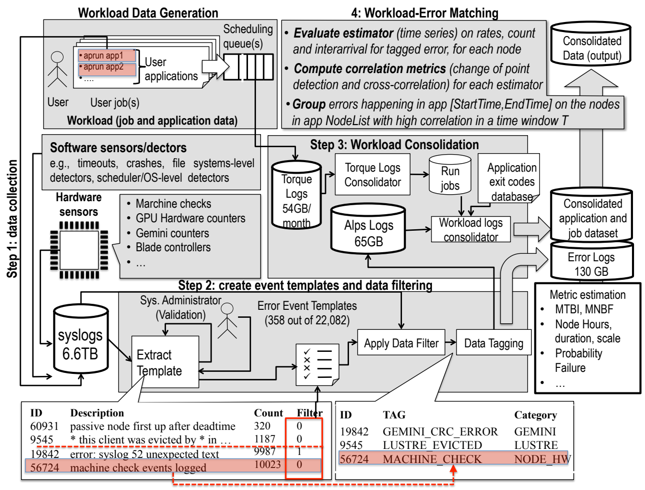


Figure 1: *LogDiver* data processing workflow

multiple factors (e.g., application scale and system errors) impact application. Uniquely, this tool does the following:

- Allow a precise identification of the reasons behind application termination,
- Directly relates system errors and failures (e.g., Gemini ECC errors, GPU MMU errors and Lustre file system failures) to application failures, and
- Provides a unified representation of the workload/error/failure logs, permitting workload-failure analysis and computation of a range of quantitative performance and dependability metrics.

An in-depth characterization of the application failures caused by system-related issues is essential to evaluating the performance of current systems and to guiding the design of resiliency mechanisms.

While the tool was designed with respect to Cray architecture, it can be extended to other type of systems by changing a limited number of components.

To the best of our knowledge, this is the first tool that allows the characterization of the impact and propagation of system errors on running applications. Such in-depth understanding of application error sensitivity is essential for realistic performance evaluation of current systems and to guide design of resiliency mechanisms.

In the following, we briefly describe the workflow enforced in *LogDiver*.

2.1 Workflow

LogDiver operates in 5 main steps, depicted in Figure 1. Each step produces several output files that are fed downstream to the subsequent step. Data in intermediate output files can also be used by external tools (e.g., Matlab and SAS to perform workload characterization) to conduct additional analysis beyond what *LogDiver* supports.

2.2 Step 1: Data Collection

Objective: collecting data from multiple sources. Many sources are redirected to the syslog, including a subset of the data generated by the various hardware sensors deployed in Blue Waters. Data is collected by the system-level logging daemons and periodically moved by *LogDiver* to a workspace where to analysis takes place. [1]

Input: system-generated syslogs and workload logs, including job scheduler logs (i.e., TORQUE logs) and application scheduler logs (i.e., ALPS logs).

Output: data parsed to an internal format that is system-agnostic.

Syslogs include system events logged by the OS and by Hardware Supervisory System (HSS), and entries generated by the Sonexion cluster implementing Blue Waters' LUSTRE file system.[2] Syslogs events include i) the timestamp of the event, ii) the facility, indicating the type of software that generated the messages, iii) a severity level, indicating how severe the logged event is, iv) the identification of the node generating the message, v) the process, including the PID (process identifier) of the process logging the event, and vi) the event description.

TORQUE logs include information on created, canceled, scheduled, and executed jobs in the system. Each entry in the TORQUE logs consists of 45 fields containing time information on all the phases of the job (creation, queue, execution, and termination times), user, group, queue, resources, the type and the list of used nodes, and wall-time used.

ALPS logs include information on node reservation management, job/application launching, periodic monitoring and termination of user applications, and detected internal problems and cleanup operations. ALPS logs are redirected by the system console to the syslogs and merged with other system events.

2.3 Step 2: Event Tagging and Filtering

Objective: to identify, categorize and filter the error events contained in the collected data.

Input: parsed syslogs.

Output: (i) a list of categories containing only events of interest, i.e., the error data, referred as message template; (ii) the filtered dataset, referred as error data.

A message template is a combination of a fixed text and a variable text in each raw log entry. The fixed part is the text that indicates the specific event that generated the syslog entry, e.g. "NODE ID client was evicted by LUSTRE OST NODE ID in PARTITION ID", with "* client was evicted by * in *" being the template (see Figure 1). LUSTRE OST NODE ID, PARTITION ID in the example encodes the information related to the specific event logged, i.e., the id of the node evicted, the evicting Lustre node, and the partition to which the eviction refers. We equipped *LogDiver* with a set of functionalities to identify all the fixed (i.e., the template) and variable items in the logs and to substitute with a wild card symbol (the * in the example in the bottom left corner of Figure 1) the variable items, such as IP, MAC, node id, hardware address, dates, user names, and numbers.

The categorization consists of assigning a specific unique numerical template ID, tag, category and group to each error template. The *tag* is a textual description of the event of interest (e.g., GPU_MMU_ERROR or LUSTRE_EVICT), the *category* refers to the subsystem generating the event (e.g., NVIDIA_GPU or LUSTRE), and the group corresponds to the subsystem involved in the event (e.g., NODE_HW or STORAGE). For instance, the example in the bottom left

[1]We are currently developing a system-level plugin for the ALPS prologue and epilogue in order to collect data before and after each application.

[2]Refer to [8] for a more detailed description of the system.

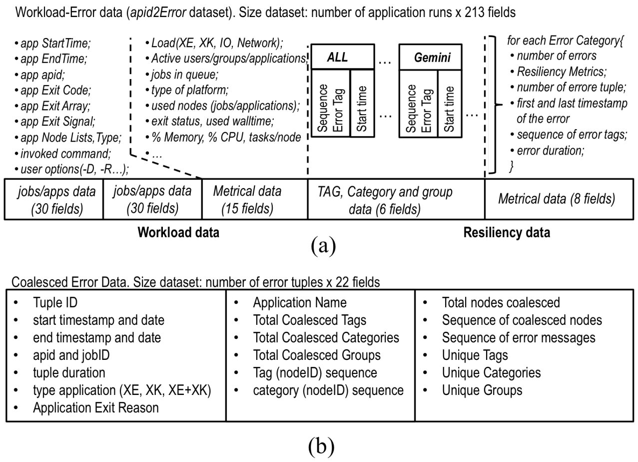


Figure 2: *LogDiver* Main Output: (a) *apid2Error* dataset matching workload data with error data, (b) coalesced error data

corner of Figure 1, we assigned the tag CPU_MACHINE_CHECK and category NODE_HW to the template with ID 56724 1).

That step is the only semi-automated part of the tool; the other steps are fully automated. The categorization of error templates requires frequent interactions with technical personnel for validation purposes. At the end of this step, all the entries matching the obtained templates identified at step 2 are retrieved from the data and tagged according to the tag, category, and group in the template list.

2.4 Step 3: Workload Consolidation

Objective: To create a consolidated dataset that includes information on jobs, application runs, used resources (e.g., type and ID of used nodes), user options (e.g., used resiliency features), in order to enable the matching of workload data with error data, performed in the next step.

Input: the TORQUE and ALPS logs.

Output: an extended data set of user applications (referred to as "application data" in Figure 1), which include 1 entry of 46 fields for each application. Important fields are i) start and end time, ii) reservation ID, job ID, user, group, application name, iii) resources data, e.g., number, ID and type of nodes, memory, and virtual memory, iv) application exit code and job exit code, v) job- and application-required wall time and used wall time, and vi) user command used to launch the application.

The required application data is scattered over several nonconsecutive entries in the ALPS logs and needs to be retrieved and assembled for each user application. Another important operation performed by this step is to extract staff applications from the dataset. Blue Waters staff can execute privileged applications, such as hardware tests or run dedicated benchmarks without passing through the batch queue. As a consequence, many staff applications do not have corresponding job information as it happens for user applications. To this end, *LogDiver* interfaces with the user access list (e.g., the LDAP user and group table) to gather user information and identify system personnel.

2.5 Step 4: Workload-Error Matching

Objective: to match and collate relevant error data with the consolidated workload data.

Input: filtered error events and the consolidated workload traces (application data in Figure 1).

Output: (i) *apid2Error* dataset (Figure 2.(a)) where each each application run by Blue Waters is paired with all the errors that the application experienced while executing; (ii) *Coalesced Errors* dataset, where all the errors occurring with high correlation on nodes executing the same instance of application (including the service nodes serving XE and XK node requests related to the this application) are grouped together to form error tuples.

The error-application association is performed by overlapping the workload data with errors occurring between the start time and end time of the considered application, on one of the nodes executing the application or one of the service/IO nodes serving the application.

In order to correlate application failures with error data, *LogDiver* uses a mix of empirical and analytical techniques that can be classified into 2 categories: (i) correlation analysis methods to separate local from global effects across events generated by different nodes and/or different error categories; and (ii) event coalescence, to group specific errors occurring with high correlation.

Event Correlation. A first step to allow cross-correlation analyses is to model error data in a uniform way. To that end, we model error events as a set of (stochastic) point processes, i.e., a set of individual events generated at random points in time T_i. We represent the point processes using different representations and domain transformations based on both inter-arrival process and sequence of counts. An inter-arrival process is a real-valued random sequence with $I_n = T_n - T_{n-1}$. The sequence of counts, or the count process, is formed by dividing the time axis into equally spaced contiguous intervals of T to produce a sequence of counts $C_k(t)$, where $C_k(T) = N_{k+1}((T) - N_k(T)$ denotes the number of events in the k-th interval. The normalized version of the sequence of counts, the rate process is obtained by dividing $C_k(T)$ by the size of the sampling interval T. The correlation is then performed using Pearson's lagged correlation coefficient computed across different point processes generated by nodes executing the same application. The correlation can be estimated with respect to any error as well as considering only specific tags, categories and groups.

Event Coalescence. *LogDiver* employs different coalescence techniques, making it possible to perform analyses at different levels of detail. Specifically, it can coalesce errors generated by i) the same error category/tag, ii) the same node, iii) nodes allocated to the same job and/or application, and iv) the whole system, considering only console logs. The last type employs hypothesis testing and domain expertise in order to avoid grouping independent events (e.g., two ECC memory errors on two different nodes). We used domain expertise to create an adjacency matrix of signals that can be mutually influenced, e.g., GPU_MMU errors with GPU voltage level. We grouped the events i) temporally when they show high (lagged) cross-correlation values, and ii) spatially (i.e., events generated across different nodes, blades, and cabinets) only when they are generated by nodes executing the same application.

2.6 Step 5: Metrics

The last part of the tool is in charge of estimating various metrics of interest. Table 1 shows an example of a subset of the computation modules *LogDiver* modules and related input and output data. The metrics are computed with respect to i) the whole system, ii) application name, iii)

Input	Output	
Data	Produced Datasets	Field List
Executed jobs and Applications, Node List Database, Node Error Logs, Correlation Matrix	resiliency Metrics	FOR EACH ERROR CATEGORY: numTuples, numTag, tupleStartTime, ALPS_tupleDuration, ALPS_timeFirstError, latency start (time from first error to application exit), latency End (time from last error to application exit), errorRate (rate of the errors the application was exposed to during the execution), % tolerated errors (resiliency), % errors causing failure (%error criticality)
Executed Applications (ALPS DATA)	application node hours	APPNAME, SCIENCE_AREA, TOTAL_RUNS, TOTAL_NODE_HOURS, TOTAL_NODES, AVG_NODE_HOURS, MAX_NODE_HOURS, STDEV_NODE_HOURS, CONF_INT_95_NODE_HOURS, AVG_NUM_NODES, MAX_NUM_NODES, STDEV_NUM_NODES, CONF_INT_95_NUM_NODES, AVG_DURATION_H, MAX_DURATION_H, STDEV_DURATION_H, CONF_INT_95_DURATION
	distributions	node type, scale (single, nano, small, medium, high, full), distribution (duration, nodehours, num. nodes), exit type (success, walltime,system, user, user/system, unknown), total runs, % runs 1st quantile, % runs 2nd quantile... % runs 100th quantile
	summary	(success, walltime,system, user, user/system, unknown), for each application type (XE, XK, XE+XK): count, mean, sd, conf. interval 95th of full, high, med, small, nano, single,
	tree	graphWiz .dot file to create a tree representing the breakdown of applications starting from application type (XE, XK, XE+XK) -> exit type (success, walltime,system, user, user/system, unknown) -> and scale (full, high, med, low, nano, single)
Executed Applications	application metrics	APPNAME, TOTAL_RUNS, TOTAL_USERS, LIFETIME, DURATION, USER_LIFETIME, USER_HOURS, DURATION_XE, USER_LIFETIME_XE, USER_HOURS_XE, DURATION_XK, USER_LIFETIME_XK, USER_HOURS_XK, DURATION_XEXK, USER_LIFETIME_XEXK, USER_HOURS_XEXK, NODE_HOURS, NODE_HOURS_XE, NODE_HOURS_XK, NODE_HOURS_XEXK, FRACTION_XE_NODEHOURS, FRACTION_XK_NODEHOURS, FRACTION_XEXK_NODEHOURS, ***for each exit type (success, walltime, system, user, user/system, unknown) count, MTBI, MNBF of all, XE, XK, XE+XK applications***; NODE_USER_HOURS for all, XE, XK, XE+XK applications
	job metrics	
Filtered Error logs	count tag	tag, count all, count XE nodes, count XK nodes - summary of the template table
	count category	category, count all, count XE nodes, count XK nodes - summary of the template table
	count group	group, count all, count XE nodes, count XK nodes - summary of the template table
Executed jobs and Applications	user/user group MTBF for XE, XK and XE+XK applications (separate files)	appName, userID/groupID , MTBF_H, MTBF_NH, % Success, % system forced exits, % user forced exits, % user/system forced exits, % walltime exits, ... and for each exit type (success, walltime,system exit, user forced exit, user/system forced exit, unknown, unorderly exit): computed hours, max duration, max num. odes, max nodehours, num runs,
	statistics nodehours	for each exit type (success, walltime, system exit, user forced exit, yser/system forced exit, unknown, unorderly exit): prob. Application failure (for different applications, node type, scales, used node hours), probUserFailure, nodeHours, MTBF with respect to node hours, MTBF with respect to computed hours
	statistics runs	
Executed jobs and Applications	commands	start_time, end_time, apid, cmd, 1000xnodeHours, nodeType, user, group, project, Project_theme
	unique commands	nodeHours, AppName, LaunchCount
	app. user statistics	TOTAL_RUNS, TOTAL_NODE_HOURS, NUMBER_DIFFERENT_APPLICATIONS, MAX_NODE_HOURS, AVG_NODE_HOURS_PER_APP, STDEV_NODE_HOURS_PER_APP, CONF_INTERVAL_95%
	app. group statistics	
	app. project statistics	
Executed Applications	number of nodes, duration, node hours for each application type (XE, XK, XE+XK) - multiple files	bucket Number (i.e., group of application within a range of used node, node hours, hours), bucket_lower_bound, bucket_upper_bound, num. Apps,
		bucket_lower_bound, bucket_upper_bound, TotalApids, success_mean, success_sd, success_low_conf95, success_up_conf95, walltime_mean, walltime_sd, walltime_low_conf95, walltime_up_conf95, system_mean, system_sd, system_low_conf95, system_up_conf95, user_mean, user_sd, user_low_conf95, user_up_conf95, user/system_mean, user/system_sd, user/system_low_conf95, user/system_up_conf95

Table 1: Example of *LogDiver*'s output data.

the user ID, iv) the node type (i.e., XE and XK nodes), v) node hours, and vi) the application/job scale. For each of the mentioned metrics, *LogDiver* estimates both empirical distributions (cumulative and density functions) as well as synthetic statistics including mean, standard deviation, and confidence intervals. The metrics provided by the tool are valuable for correlation studies design to quantify the impact of exogenous variables, such as the load, application scale, and expertise of the user, on the application resiliency.

3. BLUE WATERS FIELD DATA

We illustrate our approach by providing examples of the fault/error characterization analytics provided by *LogDiver*. We conducted the analysis of data produced by Blue Waters during the 365 production days (August 1, 2013 to August 1, 2014) to create an initial error categorization. Our dataset includes 2,359,665 user application runs of more than 1,500 code bases, 769,321 jobs and 296,114,457 error events stored in $\tilde{4}$TB of syslogs. During the measured period, we measured an MTBF of 8.8h, and an overall availability of 0.968, computed after excluding scheduled downtimes, system upgrades and programmed maintenance actions.

3.1 Blue Waters Errors

At the end of step 2, *LogDiver* identified 22,082 different templates in the logs, i.e., about 300 million error entries belongs only to about 22,000 different events. Those events were further reduced to 5,127 using a set of 92 keywords identified by Blue Waters staff. These we manually screened and reduced to 398 templates of events potentially affecting system and user operations.

Figure 3 shows the classification and count of the error

G	C	TAG	ENTRIES
NODE/BLADE	RESILIENCY ARCHITECTURE	L1_FIRMWARE	37,789
		ADMINDOWN	264,421
		NODE_HALT	6,088,540
		NODE_BAD_HEALTH	162,333
		NODE_SUSPECT	1,673,973
		NODE_UNAVAILABLE	1,413,764
		EC_NODE_FAILED	2,214
		DVS_HEARTBEAT	84
		DVS_MOUNT	138,533
		DVS_NOT_AVAILABLE	2,041,558
		BLADE_HEARTBEAT	396,015
		CABINET_HEALTH	237,798
		EPO	32
		WARMSWAP_FAILED	40
	BLADE	POWER	14,790
		VRM	185,047
		BLADE_FAILURE	60
		HARDWARE_ERROR	387
		MCE	3,820,645
	GPU	DBE	56
		INVALID_DEVICE	248,789
		MMU_ERROR	218,533
		UNABLE_TO_RESET	125,089
		UNKNOWN_ERROR	1,022
	SYS/USER SOFTWARE	MODULE_MISSING	728
		CRASH	3,729,585
		LDAP_TLS	204
		MPI_USER_EXCEPTIO	313
		OOM	238,354
		SSHD	1,279
		OOPS	31,955
		KERNEL_NULL_POINT	1,190,579
		OS_ERROR	422
		PANIC	49,007
		RSIP	771,597

G	C	TAG	ENTRIES
STORAGE	LUSTRE	EVICT	20,990,344
		INTERRUPTED_SYSCALL	1,740,283
		IO_ERROR	150,457
		LBUG	39,317
		MDS_DEVICE_BUSY	883,526
		MOUNT_TIMEOUT	1,304
		NET_LOOKUP	21,682
		NETWORK_ERROR	17,457,779
		OST_DEVICE_BUSY	1,385,726
		PERMISSION_DENIED	12,344
		FAILOVER_ERROR	2,755
		QUOTA	62
		STALE_NFS_HANDLE	219,670
		TIME_REWIND_BUG	61
		TIMEOUT	3,859,820
		TRANSPORT_SHUTDOWN	15,570,189
		UNKNOWN_NID	2,211
		WAITING_RECOVERY	55,472,681
	SONEXION	FILESYSTEM_FAILURE	167
		IO_ERROR	133,375
		NO_SPACE_LEFT	4,109
		PROCESS_KILLED	807

G	C	TAG	ENTRIES
SCHEDULER	MOAB	CONNECTION	429,124
		INTERNAL_ERROR	310
		JOB_DEPENDENCY	37,092
		JOB_STUCK	3,273
		MOM_MISCOMMUNICATION	6,998
		UNEXPECTED_JOB_STATE	4,106
	ALPS	CANNOT_ALLOCATE_MEM(	1,274
		CONNECTION	60,889
		KILLED_APID	367
		NODE_HEALTH	227,258

G	C	TAG	ENTRIES
NETWORK	GEMINI	BUFFER_OVERFLOW	9,373,707
		CHECKSUM_ERROR	3,238
		DATA_ERROR	204
		ECC_ERROR	95,409
		FMA	36,716
		MACHINE_NOT_ONLINE	101,007
		MISROUTED_PACKET	38,053,828
		PACKET_DROP	364,534
		PACKET_ERROR	1,983,362
		LANE_FAILED	151,866
		LINK_FAILED	236,476
		LANE_RECOVERY_FAILED	1,744
		CHIP_ERROR	2,673
		CONGESTION	204
		CORRUPT_ROUTING_TAB	90
		FAILED_ROUTE	12
		UNCORRECTABLE_ERROR	86
		L0_TIMEOUT	54
		LINK_RECOVERY_FAILED	108
		ROUTING_FAILURE	5,034
	LNET	CONNECTION_FAILED	4,149,469
		CRITICAL_HW_ERROR	409,034
		FATAL_ERROR	659
		HARDWARE_QUIESCE	2,162
		INVALID_MMD	3,030,234
		IO_ERROR	74,411
		NO_ROUTERS	19,855,107
		NODE_DEAD	1,573
		OOM	2,203,423
		PACKET_DROP	22,837,200
		PEER_DOWN	2,156,277
		PROTOCOL_ERROR	838,256
		SINGLE_BIT_ERROR	719,987
		STALE	4,138,421
		TIMEOUT	17,463,278
		TRANSPORT_SHUTDOWN	93
		UNKNOWN_NID	71,882

Tot. Entries
270,166,713

G = GROUP
C = CATEGORY

Figure 3: Extracted error templates.

messages obtained from the considered datasets in Blue Waters. We categorized the 398 templates in 105 error tags (i.e., error types) generated by 10 error categories (relating the error tag to the specific involved subsystem), further categorized into five error groups: i) *Network*, which includes Gemini (e.g., routing or Gemini hardware errors) and LNet errors (e.g., packet dropped or endpoint shutdown), ii) *Scheduling*, which includes errors encountered by Moab/TORQUE (e.g., impossibility to allocate/deallocate resources for a job) and ALPS (e.g., crash of the ALPS processes), iii) *Node/Blade*, which encompasses errors detected by the resiliency mechanisms and hardware/software detectors employed in Blue Waters nodes; events include errors generated by the GPU, memory, and processor (e.g., GPU_MMU errors and node warm-swap failed), as well as system/user software errors; and v) Storage, which includes errors generated by Lustre (e.g., client eviction) and the Sonexion cluster (e.g., I/O errors on the storage nodes).

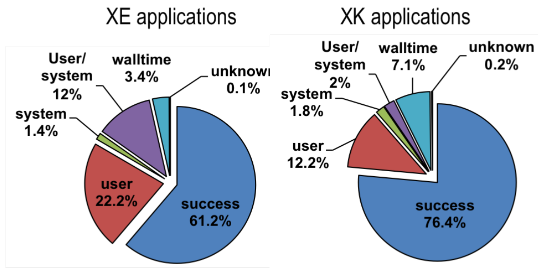
In order to reduce the filtering time, *LogDiver* is equipped with a data-parallelization framework that i) splits the input data set into smaller sets, ii) produces a parallel batch script assigning batches of 32-64 data files per node, iii) orchestrates the submission of a job to the target computing system, and iv) collects and merges the results gathered by the nodes, creating a new single data set. Each node executes a customized filtering script that includes 32-64 concurrent processes, one for each input data file. The number of concurrent processes per node and the total number of nodes used depend on the available memory and number of cores per node in the computing environment.

3.2 Application Exit Status

Table 2 shows the breakdown of the job and applications in our dataset. 64.53% of the total user runs are XE applications (i.e., 1,522,694), 35.46% are XK applications (i.e., 836,971) using CPU and GPU accelerators. To compare the composition of XE and XK applications with respect to ap-

Table 2: Workload across different scales.

scale	xe	xk	%XE jobs	%XE apps	%XK jobs	%XK apps
SINGLE	<=4 (1 blade)		32.81%	53.89%	33.73%	84.86%
NANO	<= 96 (1 cabinet)		52.36%	39.24%	53.84%	14.13%
SMALL	<= 512 (1 row)		11.99%	5.33%	10.08%	0.88%
MEDIUM	≤5896 (25% sys)	≤1056 (25% sys)	2.35%	1.35%	2.03%	0.09%
HIGH	≤11792 (50% sys)	≤2122 (50% sys)	0.40%	0.16%	0.25%	0.03%
FULL	>11792	>2122	0.10%	0.04%	0.07%	0.01%
Total application runs (no staff)			1,492,694		836,972	



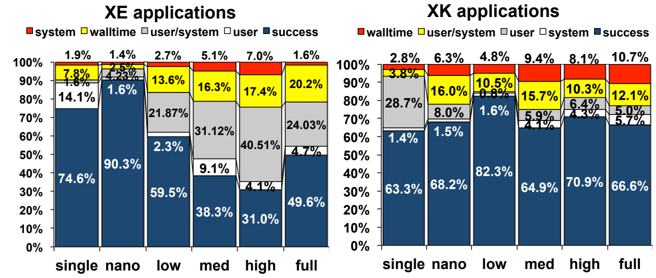
(a) (b)

Figure 4: Breakdown of decoded exit statuses.

plication scale, we subdivided the applications into 6 classes following the rules in Table 2. Blue Waters data include only a limited number of applications that can effectively use full-scale executions. Even when running at full scale, many applications do not execute for a long time, e.g., 75% of the full-scale XE applications in the measured data run for less than 5 h, with a median of 1.2 h. As we shall discuss in the next section, tolerating variety of errors at 'Full' scale is not a trivial task.

Breaking down the application exit status. There are more than 256 possible application exit codes, many of which are ambiguous or application dependent. An example is that of the exit code 143 (i.e., application terminated by issuing a TERM signal), which can be issued when the application is killed either by system errors or by the user. *LogDiver* is able to disambiguate and categorize an application exit reason by matching error data with application exit code data. Exit reasons are classified into the following categories: (i) *Success*, for applications completing successfully, ii) *Walltime*, for applications not completing within the allocated wall clock time, iii) *User*, for abnormal terminations caused by user-related problems including compiler/linking/job script and command errors, missing module/file/directory or wrong permissions, and user-initiated actions such as a control-C signal or termination/kill commands, iv) *System*, when an application is terminated due to system-related issues caused by any of the considered system errors, and v) *User/System*, when an application is terminated for causes that can be related to both user and system events, such as errors detected by the applications (e.g., through assertions) and handled by means of legit exit.

Figure 4 gives the breakdown of the application exit statuses. 61.2% of XE applications (Figure 4.(a)) and 76.4% of XK applications (Figure 4.(b)) exited successfully. The remaining applications failed due to several reasons, including: i) the application execution time exceeds the time limit (3.4% for XE and 7.1% for XK, category 'walltime'); ii) user-related problems (22.2% for XE and 12.2% for XK, category 'user'); iii) system-related problems (1.4% for XE and 1.83% for XK) caused by hardware, software, configuration, or net-



(a) (b)

Figure 5: Breakdown of Blue Waters application exit statuses for XE (a) and XK (b) applications across different scales

work issues at the system or node levels and happen with a MTBI (production hours / total application interrupts) of 15 minutes; and iv) a combination of user- and system-related causes, e.g., exceptions raised because of issues with Gemini rerouting (12% for XE and 2% for XK applications, category 'user/systems'). Compared to earlier Cray systems (for which only job success data are publicly available), Blue Waters shows a lower percentage of application failures [9–11]. For instance, for Franklin, the Cray XT4 100 Teraflops machine at NERSC, 61% of applications complete successfully, whereas 11.5% are terminated because of the walltime limit, 25% are terminated because of user problems, and 2.7% are killed because of system problems. Athena (166TF, 46 XT4 cabinets) shows that 82% of applications finish successfully. The percentage of non-reported application failures is believed to be 3% according to the staff.

XK applications show a higher percentage of application failure because of system errors (1.8%) than XE nodes (1.4%). The difference in the percentage of failed applications is due to the higher number of XE applications failing because of user causes (category "user", 22.24% for XE applications and 12.20% for XK). The reason behind that is (1) XE nodes are often preferred by users to develop and debug their applications before deploying the code, (2) The wait time for accessing XK nodes is typically longer than that for accessing XE nodes.

Figure 5 shows the breakdown application exit reasons for different scales. Key observations are: (i) For XE nodes the number of applications exiting successfully decreases with scale while no clear trend can be observed for XK applications; (ii) The number of application exiting because of 'user/system' problem increases substantially with the scale of XE applications while it remains relatively same for XK applications; (iii) For both XE and XK applications number of system related exit statuses increases with scale. However percentage-wise it is substantially higher for XK applications when compared to XE applications. System problems cause the failure of 5.65% of full scale XK applications (4,224 nodes) against 1.55% for XE applications (22,640 nodes).

Recall that application failing because of 'user/system' problem includes unexpected situations successfully captured built-in error detection mechanism (e.g. assertions). Data in Figure 5 demonstrates that approaching extreme scales the error detection capabilities of XE applications (supported by the XE6 nodes hardware sensors and detectors) are more effective in capturing anomalous situations than as it happens for XK applications. XK7 nodes provide provide limited support for error detection (i.e., GPU monitoring and error detection) and hence cannot handle unexpected situation as

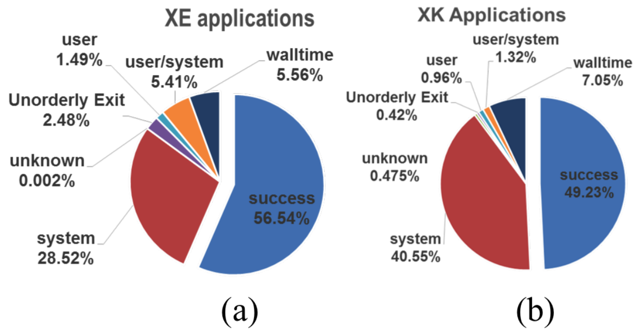


Figure 6: Blue Waters workload exit status for XE (d) and XK (e) applications that experienced at least 1 error during the execution.

effectively as XE6 nodes allow. This finding demonstrates that XK applications might be more prone to undetected errors (e.g. Silent Data Corruption) than XE applications because of the limited error detection capabilities provided by the platform.

4. APPLICATION RESILIENCY

In this section, we use the output produced by *LogDiver* to measure the resiliency of XE and XK applications. Measurements are produced by *LogDiver* with respect to i) different application scales, from single node application up to full scale applications, and ii) sensitivity to different error categories. Recall that the tool performs a matching between the workload run on each node and the 102 error tags identified during the step 2 of the approach (Figure 3).

Application Survivability. Figure 6 illustrates the breakdown of the application exit status for all those applications that experienced at least 1 error during their execution. In particular, Figure 6 shows the joint distribution of how application terminates when operating under error conditions. It is interesting to note that the success rate of both XE and XK application decreases when operating under error. XK applications show little resiliency to error when compared to XE applications. In particular, the success rate of XK application goes down from 76.6% to 49% when the applications operate under error; at the same time, the percentage of application failing because of system problems grows from 1.83% to 40.55%, while the same phenomenon has a more modest yet substantial manifestation for XE nodes, where the success rate goes down from 61.27% to 56.54%. Another interesting observation is percentage of applications exiting with unknown status, only 0.002% of XE applications that suffered error are in this category compared to 0.475% for XK applications. We speculate that this is because of the poor error detection mechanism currently present on the K20X GPUs of XK7 nodes. As it shall be detailed later, XK applications are more sensitive to system errors for a variety of reasons. Recovering from GPU errors without appropriate support from error detection mechanisms is a hard task if not sometime impossible.

4.1 Resiliency to Different Error Categories

As a final demonstration of the potential of *LogDiver*, we used the data produced after step 5 (”evaluation of metrics”) to analyze the impact of different error categories on application resiliency. Figure 7 shows the plot of the application resiliency computed by *LogDiver* when the applications are subject to different error categories. The resiliency is measured as: $\frac{\#applications(successful, error\, in\, category\, C_i)}{\#applications(total\, number, error\, in\, category\, C_i)}$.

Figure 7 shows how applications react on average (i.e., succeed or fail) when subject to errors in the categories in Figure 3 (See Section 3). The error bar represents the 95% confidence interval of the estimated figures.

File system and interconnect are critical for medium to full scale applications. Figure 7.(a) shows the application resiliency across XE applications. An interesting observation is that for applications running at single and nano scale (within single cabinet) the measurements don't change drastically. When applications run on more than one cabinet (>96 nodes), we start to see substantial decrease in resiliency to interconnect and system problems. In particular, 'Interconnect' (Gemini) resiliency goes down from 34.89% for nano to 17.06% for high scale. Similarly, resiliency for ('LUSTURE',' LNET') goes down from (40.8%, 41.34%) for nano scale to (21.38%, 22.32%) at high scale respectively. Slight aberrations in resiliency patterns are observed for applications running at full scale. This is because at full scale application developers generally adopt many resiliency mechanisms to protect application against variety of errors and typically run for less than 5 hours which means that these applications will be exposed to various errors for limited amount of time. It is interesting to note that 'operating system (OS)' related errors (e.g. kernel panic, kernel OOPS) are very critical at any scale. Although, they are not frequent events they can kill the applications almost in all cases. The reason for OS critically is that when OS crashes on a node, it is difficult if not impossible to trigger node health check or recovery procedures like warm swap or application migration on healthy nodes. In these cases, a deeper analysis shows that applications developed using charm++ frameworks are more resilient than other. The charm++ framework maintains two copies of checkpoints in-memory on different nodes and hence chances of recovery from such failures are higher.

This behavior is even more marked on XK applications Figure 7(b). In particular, XK applications running at single and nano scale (i.e within a cabinet, <=96 nodes) show higher error resiliency across different categories compared to other scales. When expanding outside single cabinet, we observe a sharp decrease in resiliency across all error categories. In particular, 'interconnect' (Gemini) problems go down from nano to full scale. When approaching high and full scale, file system ('LUSTURE' and 'LNET') problems are typically unrecoverable for XK applications. Interestingly, resiliency to node/blade error does not follow similar dynamics as the interconnect or file system. We observe limited variance in resiliency for such errors, bounded between 27.7% and 50.5% of resiliency.

Application Resiliency to GPU errors Figure 7 shows that GPU related errors (e.g. GPU_Drivers) are critical to XK workloads. In particular, we look how GPU errors affect the XK workload. Our data revealed that there are 5 different types of GPU error codes that occurred during the measured period. The names and their occurrence count is given in Figure 3 under *GPU* heading. Figure 3 shows that Memory Management Unit (MMU) error is most frequent while Double Bit Exception (DBE) error is the least occurring. Next, we breakdown the occurrence of GPU errors according to scale (Full, High, Medium, Small, Nano, Single) and exit type (Success, System, Unorderly exit, User,

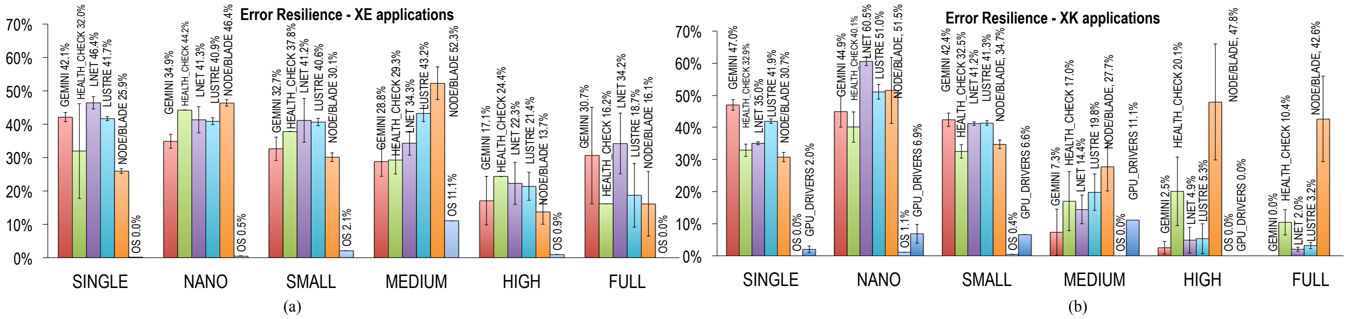


Figure 7: Average resiliency (% of application that tolerated the error) of XE (a) and XK (b) applications to different error categories. The error bars show 95% of confidence intervals.

Exit Category	SU	SY	UE	U	U/S	W	SU	SY	UE	U	U/S	W
SCALE	MMU ERROR						UNABLE TO RESET					
FULL		11.1%			11.1%	77.8%					100%	
HIGH	0.4%	94.6%		4.4%		0.5%		80.0%				20.0%
MED	0.3%	98.8%		0.6%		0.3%	7.7%	53.8%	19.2%		19.2%	
SMALL	1.3%	96.7%		1.2%	0.2%	0.6%	1.6%	85.7%	2.2%	3.8%	0.5%	6.0%
NANO	45.1%	36.2%		1.6%	1.2%	16.0%		1.4%	27.1%	24.3%	47.1%	
SINGLE	12.1%	78.1%	0.2%	0.2%	0.2%	9.3%		95.1%	2.5%	2.5%		
	INVALID DEVICE						DBE					
FULL		0.0%			100%							
HIGH		75.0%				25.0%	50.0%				50.0%	
MED	7.7%	50.0%	23.1%		19.2%			100%				
SMALL	1.7%	86.3%	1.7%	3.4%	0.6%	6.3%		100%				
NANO		57.7%	11.7%	10.4%	20.2%			100%				
SINGLE		94.9%	2.5%	2.5%								
	UNKNOWN ERROR											
FULL	50.0%	50.0%										
HIGH		60.0%			40.0%							
MED	14.3%	71.4%				14.3%						
SMALL	15.4%	46.2%				38.5%						
NANO		50.0%			25.0%	25.0%						
SINGLE												

Legend	
SU	SUCCESS
SY	SYSTEM
UE	UNORDERLY EXIT
U	USER
U/S	USER/ SYSTEM
W	Walltime

Figure 8: Breakdown of GPU errors across all XK applications (a), and Vs. different scales (b).

User/System, Walltime) as shown in Fig. 8. This figure shows the count of applications runs that suffered a particular type of GPU error after the coalescing stage. As can be seen in figure, most of these errors lead to application failing with 'System' exit code. In some cases, the errors can lead to all (or most) application failing either in 'Walltime' such as $MMU\,Error$ at full scale or in 'User/System' such as $Unable\,To\,Reset$ at full scale. The reason being GPU errors are hard to tolerate because of poor detection, logging and recovery techniques limited by hardware itself. The triggered recovery procedure thereafter, takes longer than expected. The application effectively behaves like a hung application and either hits walltime or user decides to manually kill it. Very few application exit successfully despite suffering from GPU related errors and these are concentrated at smaller scales. However, our manual examinations of logs reveal longer recovery times even at these scales. Thanks to the data provided by *LogDiver* tool, we identified that recovering from system errors is more difficult for XK applications when compared to XE applications. We analyzed the traces of few XK application failures because of system errors to validate the data from *LogDiver*. Interestingly, we found that in case of system errors (e.g. 'Gemini' or 'LUSTURE' problems) applications go into recovery are not able to recover within the limits of allocated walltime. A closer look into the trace of few applications allowed us to identify that there is little or no support restore and resume workload running on GPU cores because of lack of appropriate error detection capabilities/API's.

Putting all together: MTBI Vs. different error categories and scales. As discussed in section 2.5, *LogDiver* estimates a variety of metrics across variety of scales and errors. Table 3 shows MTBI number obtained for different scales and error categories. MTBI is computed as the total number of system hours spent while computing at scale x divided by total number of failures occurring because of category c during that time period. An interesting observation is that XE applications at full scale can obtain a higher value of MTBI (8.8 hours) compared to theoretical MTBI value (5.69 hours). The theoretical value for the MTBI is given by $\frac{Single\,Node\,MTBF}{Total\,number\,of\,XE\,nodes}$. This observation implies that resiliency mechanisms at full scale do an excellent job in protecting applications from various errors which is consistent with the results of previous section. For Full scale XK applications, achieved MTBI is 15.1 hours. Unlike XE applications, XK applications at full scale is only able to match theoretical MTBI (of 15.06 hours). Thus, leaving a scope for improvement. If we compute the ratio of achieved MTBI and theoretical(expected) MTBI, we see that at XE Full scale the system is able to improve MTBI by 1.55x. While for XK, it can barely keep up with expected MTBI.

Table 3: MTBI values for XE (a)-(b) and XK (c)-(d) applications when exposed to error of different categories, at different scales.

	MTBI XE applications (hours) - theoretical system level MTBI XE nodes = 5.69 h						
	GEMINI	HEALT CHECK	LNET	LUSTRE	NODE/ BLADE	OS	Overall (hours)
FULL	4	5	20	12	2	12	**8.8**
HIGH	7	8	9	7	3	25	**12.8**
MEDIUM	15	10	3	5	25	38	**52.2**
SMALL	6	9	2	1	5	48	**842.1**
NANO	132	224	42	30	175	1,322	**1,336**
SINGLE	1,872	2,466	452	206	10,078	4,677	**5,209**

	MTBI XK applications (hours) - theoretical system level MTBI XK nodes = 15.06 h							
	GEMINI	HEALT CHECK	LNET	LUSTRE	NODE/ BLADE	OS	GPU DRIVERS	Overall (hours)
FULL	23	21	10	8	10	-	-	**15.1**
HIGH	415	108	131	96	100	142	498	**113.1**
MEDIUM	287	157	105	88	135	515	2,999	-
SMALL	10	10	3	2	5	382	179	**148.3**
NANO	474	371	151	103	1,496	6,043	4,872	**205**
SINGLE	146	144	38	22	61	23,509	3,155	**1,761**

Lustre MTBI goes up as the application scales up from small to full scale. At full scale, the system is obviously loaded with smaller number of applications. We found that there is a correlation between number of active applications and number of Lustre error (correlation value of 0.495). We traced this to the way Lustre handles I/O requests. Requests are handled through Meta Data Server (MDS) and Object Storage Server (OSS). We found that MDS can often be overwhelmed by high rates of I/O requests. Despite of 1000's of threads serving MDS requests, these resources can easily be consumed. This causes long wait times when performing I/O operations that in turn have critical impact on overall application behavior. For example, many applications experiencing lustre errors are actually terminated because of reaching the walltime limits. *LogDiver* is able to spot this problem and categorizes such problems to applications failing because of system problems. This shows that the current lustre architecture suffers from different reliability bottlenecks. Further this shows that there is limited failure containment as the activities of one application can influence the resiliency of other applications.

5. RELATED WORK

Resiliency at extreme-scales comes from detecting and auto-correcting a greater fraction of errors with high impact. Prior research activities have centered on analyzing error logs [1–6] as well as some online analysis for patterns preceding a failure, and evaluated the accuracy and efficacy of anomaly detection and proactive response [12, 13]. They have addressed one or more of the following issues: basic error characteristics [1,2,5], modeling and evaluation [6,14,15], failure prediction and proactive checkpointing [16,17]. There are many challenges in systematically studying large-scale systems using operational data, such as data availability, data collection/mining and fault/failure characterization. In this paper, we present *LogDiver*, our solution to address the issue of the collection, mining and analysis of logs generated by extreme-scale machines.

While many works provide novel filtering approaches, few consider errors that really impact production workload in their analysis. The available characterization studies and techniques for characterizing errors/failures of extreme scale machines do not provide sufficient fidelity of understanding to enable researchers and system architects to determine how applications behave when exposed to errors and assess architectural requirements for future architectures. Blue Waters logs contains more than 100 different types of errors that can impact user applications, while others does not represent a real threat for both system and application operations. *LogDiver* is the first to correlate errors to user applications in extreme-scale environment, providing high-fidelity resiliency measurements.

6. ACKNOWLEDGEMENT

This work is partially supported by the NSF CNS 10-18503 CISE, Air Force Research Lab FA8750-11-2-0084, and an IBM faculty award. We thank Celso Mendes, Gregory Bauer, Jeremy Enos, and Joshi Fullop for providing the raw data and many insightful conversations.

7. CONCLUSIONS

This paper presented *LogDiver*, a tool for the analysis of application-level resilience to system errors in large-scale machines. The tool has been developed with respect to the data produced by Blue Waters but can be applied to other Cray based supercomputer with small effort.

In the future, we will integrate our workflow into a data stream-processing platform in order to gather real-time measurements on the system that can be used to detect major problems at the application level. We will also look on how to take into account additional data generated by the hardware sensors not currently analyzed by *LogDiver*. Finally, we will use the results of the analysis to investigate new data collection mechanisms to support application-aware fault classification, and to derive new metrics to predict the resiliency of the next generation of extreme-scale systems.

While the examples provided in this paper demonstrates that it is possible to provide a detailed characterization of hardware/software errors, we claim that a significant effort is required to create a classification which applies across different platforms, as our data shows with regards to the different error detection capabilities of XE and XK nodes. One of the challenges is the fact that some of sensors used by the system-level error detectors may be available on one platform while absent on another. We will address this challenge by cross validating the considered error categories with data from different systems (e.g., Cray XC30 and/or IBM).

8. REFERENCES

[1] R. K. Sahoo, A. Sivasubramaniam, M. S. Squillante, and Y. Zhang. Failure data analysis of a large-scale heterogeneous server environment. In *DSN '04: Proc. of the 2004 Int. Conference on Dependable Systems and Networks*, pages 772–781, 2004.

[2] Y. Liang, A. Sivasubramaniam, J. Moreira, Y. Zhang, R.K. Sahoo, and M. Jette. Filtering failure logs for a bluegene/l prototype. In *DSN '05: Proc. of the 2005 Int. Conference on Dependable Systems and Networks*, pages 476–485, 2005.

[3] Y. Liang, Y. Zhang, M. Jette, Anand Sivasubramaniam, and R. Sahoo. Bluegene/l failure analysis and prediction models. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 425–434, 2006.

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

[5] A. Oliner and J. Stearley. What supercomputers say: A study of five system logs. *Dependable Systems and Networks, 2007. DSN '07. 37th Annual IEEE/IFIP Int. Conference on*, pages 575–584, June 2007.

[6] C. Di Martino, M. Cinque, and D. Cotroneo. Assessing time coalescence techniques for the analysis of supercomputer logs. In *In Proc. of 42nd Int. Conf. on Dependable Systems and Networks (DSN), 2012*, pages 1–12, 2012.

[7] A. Pecchia, d. Cotroneo, Z. Kalbarczyk, and R. K. Iyer. Improving log-based field failure data analysis of multi-node computing systems. In *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems&Networks*, DSN '11, pages 97–108, Washington, DC, USA, 2011. IEEE Computer Society.

[8] C. Di Martino, F. Baccanico, J. Fullop, W. Kramer, Z. Kalbarczyk, and R. Iyer. Lessons learned from the analysis of system failures at petascale: The case of blue waters. In *Proceedings of the 44th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN), 2014*, 2014.

[9] H.W. Lin, Y He, and Yang W.S. Franklin job completion analysis. CUG 2010, Edinburg, UK, 2010.

[10] Matt Ezell. Collecting application-level job completion statistics. CUG 2010, Edinburg, UK, 2010.

[11] Nicholas P. Cardo. Detecting system problems with application exit codes. CUG 2008, Helsinki, Finland, 2008.

[12] A. Gainaru, F. Cappello, M. Snir, and W. Kramer. Fault prediction under the microscope: A closer look into hpc systems. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–11, 2012.

[13] A. Gainaru, F. Cappello, S. Trausan-Matu, and W. Kramer. Event log mining tool for large scale hpc systems. In *Proceedings of the 17th international conference on Parallel processing - Volume Part I*, Euro-Par'11, pages 52–64, Berlin, Heidelberg, 2011. Springer-Verlag.

[14] E. Heien, D. Kondo, A. Gainaru, A. LaPine, W. Kramer, and F. Cappello. Modeling and tolerating heterogeneous failures in large parallel systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 45:1–45:11, New York, NY, USA, 2011. ACM.

[15] C. Di Martino. One size does not fit all: Clustering supercomputer failures using a multiple time window approach. In JulianMartin Kunkel, Thomas Ludwig, and HansWerner Meuer, editors, *International Supercomputing Conference - Supercomputing*, volume 7905 of *Lecture Notes in Computer Science*, pages 302–316. Springer Berlin Heidelberg, 2013.

[16] X. Chen, C. Lu, and K. Pattabiraman. Predicting job completion times using system logs in supercomputing clusters. In *In Proc. of the 2013 Dependable Systems and Networks Workshop (DSN-W)*, pages 1–8, June 2013.

[17] A. Gainaru, F. Cappello, and W. Kramer. Taming of the shrew: Modeling the normal and faulty behaviour of large-scale hpc systems. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 1168–1179, 2012.