Resiliency of HPC Interconnects: A Case Study of Interconnect Failures and Recovery in Blue Waters

Saurabh Jha[1], Valerio Formicola[1], Catello Di Martino[2], Mark Dalton[3],
William T. Kramer[1], Zbigniew Kalbarczyk[1], Ravishankar K. Iyer[1]
[1]University of Illinois at Urbana-Champaign, [2]Nokia Bell Labs, [3]Cray Inc.
[1]{sjha8, valeform, wtkramer, kalbarcz, rkiyer}@illinois.edu;
[2] lelio.di_martino@nokia-bell-labs.com; [3] mwd@cray.com

Abstract—Availability of the interconnection network in high-performance computing (HPC) systems is fundamental to sustaining the continuous execution of applications at scale. When failures occur, interconnect recovery mechanisms orchestrate complex operations to recover network connectivity between the nodes. As the scale and design complexity of HPC systems increase, so does the system's susceptibility to failures during execution of interconnect-recovery procedures. This study characterizes the recovery procedures of the Gemini interconnect network, the largest Gemini network built by Cray, on Blue Waters, a 13.3 petaflop supercomputer at the National Center for Supercomputing Applications (NCSA). We propose a propagation model that captures interconnect failures and recovery procedures to help understand types of failures and their propagation in both the system and applications during recovery. The measurements show that recovery procedures occur very frequently and that the unsuccessful execution of recovery procedures when additional failures occur during recovery causes system-wide outages (SWOs, 28 out of 101) and application failures (3.4% of all running applications).

Index Terms—Networks, Reliability, Fault tolerance, Fault diagnosis

1 Introduction

The resiliency of high-speed interconnect is fundamental to sustaining continuous execution of applications in any high-performance computing (HPC) environment. Job scheduling, application message exchanges, file system operations, and remote access functionalities rely on the continuous availability of a robust and fast network. To this end, HPC interconnects are equipped with advanced detection and recovery mechanisms to detect and recover from most production failures.

While network recovery procedures are extensively tested offline by system vendors, in practice they often fail, causing partial or total system interruption (a system-wide outage or SWO) and extensive downtime. Those failures are caused by unforeseen situations that result from the extreme scale of current systems and workloads and that can only be measured through production data (i.e., logs and reports) generated by representative systems. Understanding the reasons behind failures of the failover[1] of HPC interconnects is crucial in designing novel resiliency solutions that can keep the entire system available with limited or no negative impact on user workload.

This paper presents an in-depth analysis of the efficacy of the resiliency mechanisms of the world's largest Gemini high-speed network (HSN), deployed by Cray in the 13.3 petaflop Blue Waters supercomputer at the University of Illinois. The study relies on the analysis of information about 27 months of production failures contained in system-generated logs (13 TB reflecting 76 billion events) and system administrator failure reports between January 2013 and March 2015.

To support extensive measurements and analysis of system logs and human-generated reports, we built a comprehensive tool, FLOAT [1], an interconnect FaiLOver Analysis Toolkit, on top of LogDiver [2][2]. FLOAT generates *recovery-sequence clusters*, which are time-ordered sequences of failure and recovery events that occurred in the context of a network-level recovery procedure. Using recovery-sequence clusters, we can evaluate the efficacy of the recovery procedures and their impact on the user workload (e.g., failed applications) and on the system (e.g., system-wide outages).

To the best of our knowledge, this is the first analysis of resiliency mechanisms for a large interconnection network that specifically demonstrates the significance of failures that occur during recovery and the need for data-driven approaches to network resiliency. Failures during recovery procedures have not been well studied in the literature. The three major observations and findings in this paper are as follows:

1. In our measurement period, invocations of network recovery procedures are frequent, accounting for over 253,000 network recoveries. The vast majority of these are lane

1. We use the terms 'failover' and 'recovery' interchangeably.

2. LogDiver is a tool for the analysis of system- and application-level resiliency in extreme-scale environments. It is currently being used by national laboratories including Sandia, Los Alamos, and NERSC.

recoveries, which occurred, on average, once every 4.1 minutes. Lane recoveries succeeded with a probability of 0.99. Link recoveries (which necessitate network-level rerouting) occur every 59.52 hours. Link recoveries completed successfully with a probability of 0.76 and were triggered 318 times in the system. This is important because whenever an automated link recovery operation fails, the system requires manual recovery, which in our measurement succeeds with a probability of 0.64. Manual recovery is invoked every 702.86 hours (approximately once a month). The failure of manual recovery causes an SWO, which requires a full restart of the entire system. Those SWOs accounted for 27.7% of all SWOs encountered during the study period.

2. Ignoring lane recoveries, which are substantially masked, the probability of a recovery being hit by an additional failure is 0.48. The probability that this additional failure causes the recovery procedure itself to fail is 0.5. We show that the probability of the occurrence of additional failures increases with the duration of the recovery, rising to a value of 0.8 when recovery time exceeds 300 seconds. Importantly, our data shows that additional failures that occur during recovery are due to either (1) independently occurring faults with respect to the failure that triggered the recovery initially (these constitute 15.6% of all failures during recovery) and (2) lack of an adequate containment of the failure originally triggering the recovery (these constitute 84.4% of all failures during recovery), as exemplified in section 5. This finding is contrary to the common assumption [3], [4] that failures during recovery are low-probability events to be ignored when addressing resiliency at large scale. In fact, our in-depth analysis in partnership with Cray Inc. revealed that recovery procedures are not aware of the failures occurring during recovery; this is due to the lack of a feedback mechanism. As a result, entities managing the execution of network recovery operations (i.e., blade controller or system management workstation (SMW)) are not able to handle failures occurring during the execution of the recovery procedures. The lack of feedback mechanisms for recovery can be addressed by adopting the some of the methods implemented in FLOAT, e.g., the ability to distinguish between propagated and independently occurring failures during recovery to applications and entities managing recovery.

3. Temporary unavailability of the network during recovery causes errors and failures in the user applications (e.g., application corruption and termination) and system stack (e.g., deadlock, network partitioning, or loss of system services). Quiescing of the Gemini network during a recovery for long periods leads to packet drops, heartbeat failures, and timeouts, which in turn can lead to failure of applications and system services (e.g., job scheduling). Our analyses show that applications can fail with a non-negligible probability even during a successful recovery of the Gemini network (probability of failure of running applications 0.007). While this result is for all applications, those applications that execute on more than 50% of the system nodes are 33.5 times more sensitive to the unavailability of the network during recovery as compared with single-node applications (13.4% and 0.4% probability of failure during recovery for full-scale application and single-node application, respectively). This demonstrates that network is key to ensuring resilient execution of applications and shows that *even a successful recovery can still cause application failures* due to lost of packets and/or read/write failures. Section 6 describes the impact of recoveries in more detail.

This work is significant because we:

- Developed methodology and tools, FaiLover Analysis Toolkit (FLOAT), that
  - formalizes and specifies recovery-sequence clusters capturing failures and their propagation during network recovery procedures
  - helps to distinguish between propagated failures (failures that are related to recovery) and independent failures (failures that are independent of recovery). This tool can assist in the development of failure containment mechanisms using recovery-sequence clusters
- Presented data-driven measurements and analysis to quantify the importance of failures occurring during the network-recovery showing vulnerability of system/applications.

The rest of the paper is organized as follows — Section 2 gives an overview of the system and description of the FLOAT, Section 3 gives a breakdown on frequency of failovers, Section 4 gives a characterizes on the progress of recovery and failures occurring during the recovery, Section 5 describes three case studies showing the failures of the recoveries, Section 6 gives measurements on application and system impact, Section 7 describes limitations of the tools, and finally, Section 8 describes related work.

2 System Overview and Analysis Framework

Blue Waters is a Cray XE system hosted at National Center for Supercomputing Applications (NCSA) in University of Illinois at Urbana - Champaign. It has two types of compute nodes — (a) 22,640 CPU only nodes, or XE nodes and (b) 4228 hybrid CPU + Graphical Processing Unit (GPU) [5] based nodes, or XK nodes. Hybrid nodes were added later to the system on July 2013. Each node has 64 GB of memory. Blue Waters offers 26.4 petabytes of usable storage using Sonexion storage solution [6].

This section describes the architecture and recovery capabilities of the Cray Gemini interconnection network [7] in Blue Waters. The description is based on the limited information available from the Cray maintenance manuals, the Gemini design specifications [3], [7], and consultation with maintenance specialists, system engineers of Blue Waters, and Cray engineers. A detailed discussion of the Gemini-based Cray system is available online at the Argonne National Lab website [8].

In Blue Waters, 4 compute nodes (henceforth, referred to as *nodes*) are packaged on a blade. There are eight blades in a chassis and three chassis in a cabinet, for a total of 96 nodes per cabinet. Each blade is equipped with a mezzanine card. This card contains a pair of Gemini ASICs (application-specific integrated circuits), which serve as network nodes (henceforth, referred as Gemini ASICs), are connected such that each blade provides a 1x4x1 network-nodes of the overall (folded) 3D torus topology [9] (of dimension 24x24x24).

In Blue Waters high-speed network (HSN), 'X' direction provides connectivity between cabinets in a row, 'Y' direction provides connectivity between rows, and 'Z' direction provides connectivity within the cabinet. The 3D torus network and Gemini ASIC are shown in the "System" layer of Fig. 3, where each cube is one Gemini ASIC. Each Gemini ASIC in the torus network is connected to other ASICs by 10 connections, two each in X+, X-, Z+, and Z- and one each in Y+ and Y-. Each connection is composed of four links, and each link is composed of three single-bit, bidirectional lanes. Thus, a connection consists of 12 lanes, and a Gemini ASIC connects to another ASIC on the network via 24 lanes in the X/Z dimension and 12 in the Y dimension. A channel is a logical connection between two link end-points and consists of two virtual channels *VC0* and *VC1*. Further, each Gemini ASIC (refer to "System" layer in Fig. 3) is housing (1) two network interface controllers (NICs), (2) a Gemini router, and (3) a network link block.

NIC. It is a hardware pipeline that has its own $HyperTransport^{TM}3$ [10] (HT3) interface. A compute-node is attached to Gemini ASIC using the HT interface of the NIC. NIC packetizes the compute-node requests (issued via the HT interface) and generates packets on the network. Packets are then routed across the network to a destination NIC.

Gemini Router. This is a 48-port router used for routing the packets across the network. It consists of tiles and routing tables. *Tiles* are the basic building blocks of the Gemini router, each consisting of one input port, one output port, a 8x8 switch, and buffers for holding and moving data in the router. A packet arrives at an input link of the tile. Using the routing table and the switches, the packet is routed to the desired output port. A routing table is used to store mapping of links (and, therefore channels) to dimensions along with input-to-output port mapping; this helps in the directional-order routing. A routing-table-based design allows the system to tolerate failures of network links/nodes and at the same time allows maintenance of the system, such as adding/removing/disabling of compute nodes/blades in a live system (i.e., without shutting the system down). Routing tables are installed and managed by the system-management workstation (SMW) and are first initialized during system boot. The mechanisms for installing and managing routing tables are discussed in next subsection. For packet routing, the interconnect system uses a packet-adaptive, virtual-cut-through-based [11] directional-order routing algorithm [9] across the network links, but it uses wormhole flow-control [12] internally due to buffer size constraints. Directional-order routing does the following:

- Routes the packet in X+/-, Y+, or Z+ until the X dimension is resolved,
- Routes the packet in Y+/- or Z+ until the Y dimension is resolved, and
- Routes the packet in Z+/- until the Z dimension is resolved, at which point the packet must have arrived at its destination.

A multidimensional torus interconnect is susceptible to deadlocks due to possible: (1) dimensional turn dependency cycles, (2) torus dependency cycles, and (3) request/response dependency cycles. Directional-order routing helps to remove dimensional turn-dependency cycles. The two virtual channels (*VC0* and *VC1*) helps to prevent request-response dependency cycle. Finally, to avoid torus dependency cycles, Gemini router chips are divided into two groups (*CG0* and *CG1*). The deadlock avoidance techniques are very similar to Cray T3D system (discussed in detail by Mohapatra et. el. [12]).

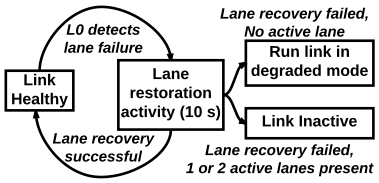


Fig. 1. State-transition diagram for lane recovery

Network Link Block. The network link block connects NICs to Gemini routers and houses a supervisory block (SB) that monitors and reports errors on the Gemini ASIC to blade controller (L0), which relays the information to the SMW. The L0 is connected to a system-management workstation (SMW) through the Cray Hardware Supervisory System (HSS) network (the HSS network is a separate network dedicated to resiliency monitoring and supervision). System responses to failures are orchestrated by the SMW.

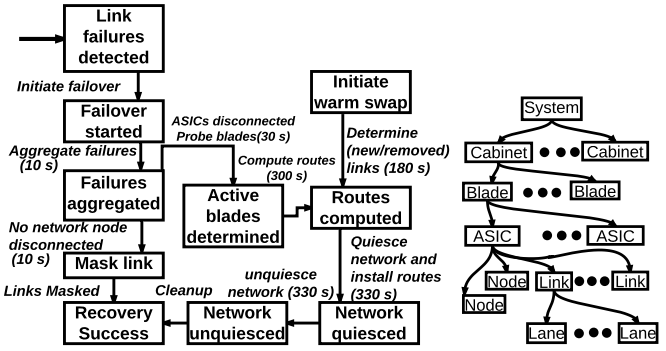
2.1 Fault Tolerance and Resiliency

Gemini provides several levels of protection from errors and failures. Packets are protected through a 16-bit cyclic redundancy check (CRC) and are checked at each Gemini ASIC (and between the transition from NIC to the router). Most of the memory regions are protected with single error correction-double error detection (SEC-DED) except for routing table buffers. For path availability, there are redundant connection in X/Z direction and links in each connection have redundant lanes. Gemini connections and links are capable of running in degraded mode upon failure of lanes and links, respectively, until the failure(s) cause(s) network partitioning. Failure of lanes/links or any other network-component (such as router) are handled by recovery mechanisms.

In this paper, we study three specific recovery mechanisms: lane recovery, link failover, and warm swap. We summarize these recovery procedures (along with system state) using state-transition diagrams (Fig. 1 and Fig. 2(a)) that include the associated timeouts (worst-case as configured in the system configuration files located in SMW) in parenthesis. Next, we describe these procedures in detail.

Lane Recovery, see Fig. 1. The availability of three lanes in each link allows the Gemini network to tolerate up to two lane failures and operate in degraded mode. If all three lanes are permanently disabled/failed, the whole link is marked as inactive by a link-failover procedure (*Link Inactive*). A successful lane recovery restores the lane activity. Lane failures are detected and handled by the blade controller (L0). The L0 attempts lane recovery (via network link block on Gemini ASIC) a certain number of times, as set in the configuration file (for a maximum of 10 seconds), before marking it as disabled.

Link Failover, see Fig. 2(a). Link failover is triggered when a link becomes unavailable for one of the following reasons:



(a) (b)

Fig. 2. (a) State-transition diagram for link failover & warm swap, (b) Fault-tree diagram used for topological ordering and coalescing

(1) all three lanes fail in a link; (2) power is lost in a mezzanine (32 links become unavailable), a blade failure (32 links unavailable), or a cabinet failure (960 links becomes unavailable); (3) faulty cables; or (4) other reasons, such as routing table corruption or software deadlock. Whenever possible, the link-failover procedure masks the failed links, avoiding network interruption. However, when a network node (i.e., Gemini ASIC) cannot communicate with another network nodes following the rules in the routing table (because the rules are not valid due to failures), the failover mechanism quiesces the whole network, i.e., interrupts the network activity in order to install the routes safely. A *successful failover* restores the communication path in the network (i.e., there is no partitioning in the system). A *failed failover* causes the whole network to fail, leaving the system in an unusable state until it is repaired manually. The link failover procedure (shown in Fig. 2(a)) can take two paths, depending upon the type of the failure. When the failure leads to disconnection between ASICs, the failover procedure performs the following sub-procedures: (1) waits 10 seconds to aggregate failures (*aggregate failures*), (2) determines which blade(s) is/are alive (*ASICs disconnected, Probe active blades*), (3) computes and asserts new routes (*compute routes*), (4) quiesces the Gemini network traffic (and also drain the network traffic) for installing the routes (*quiesce network*), and (5) unquiesces the network (*unquiesce network*). If the failure only causes the link to fail without disrupting the communication between Gemini ASICs, the failover procedure masks the link, avoiding steps (3), (4), and (5). The quiescing (coupled with traffic drainage) and unquiescing of the network are important to ensure that no packet is being routed in the system while the new network routes are being established. This avoids any deadlock situation that may occur during routing. Theoretically, the routing tables are proven to be deadlock-free under no error conditions (such as corruptions of routing tables or packets, and soft errors on the Gemini ASIC), however the network can nonetheless deadlock due to the above-mentioned errors.

Warm Swap, see Fig.2(a). Warm swap is the addition or removal (disabling) of the compute blade/cabinet in a live system. It is initiated by the system administrator manually (*initiate warm swap*). The warm swap procedure is similar to the link-failover mechanism with certain exceptions in the stages of the recovery (e.g., initialization of new blades and links). Apart from maintenance reasons, warm swap procedure are also executed to recover from network-issues when automatic recovery from failures do not succeed.

The state transition diagrams described above are the only valid states and transitions in the system. If any sub procedures of the recovery fail, the SMW tries again to recover the failed component. If a sub-procedure does not succeed after retries, it leads to failure of the recovery. Failure of link failover or warm swap procedures may create an SWO scenario, i.e., a situation declared jointly by system administrators and the supercomputer vendor, that informs users that the system is under stress/unavailable and that they might experience application failures or significant performance degradation. SWOs are situations in which the system is not able to perform as expected. An SWO is not a point event; instead, it starts at some time T (declared by system administrators) and ends at T + R, where R is the time needed to restore system-wide functioning. For example, an SWO can be declared when more than a certain number of cabinets fail or when application performance is severely affected due to the high network congestion. A failure in recovering the HSN can lead to an SWO. It is important to note that not all failed failovers are declared SWOs, since system may be restored immediately through manual intervention.

2.2 Analyzed Datasets

The dataset considered in this work was generated by the production system during the 2013-01-01 to 2015-03-31 time frame. All the datasets and their usages in our analyses pipeline are shown in Fig. 3, along with the number of entries (i.e., number of lines as #) and size on the disk. The datasets acquired directly from the system are shown in dotted cylinders. These raw datasets are further processed by LogDiver [2] and by FLOAT for network-resiliency study (shown in colored non-dotted cylinders). More details on the dataset can be found in [2], however for the sake of completeness, we now briefly summarize the raw datasets and datasets generated by LogDiver.

System Logs and System Events of Interest. "System Logs" contain system events logged by the OS and by the Cray HSS. Logs include (1) the time-stamp of the event; (2) the facility, indicating the type of software that generated the messages; (3) a severity level, indicating how severe each logged event is; (4) the identification of the node generating the message; (5) process information, i.e., PID (process identifier) about the process logging the event; and (6) an event description. Those logs are filtered using a dictionary of rules (coded as regular expressions that are specific to Cray HPC systems) by LogDiver to create "System Events of Interest".

Torque Logs are generate by "Torque" [13], the resource manager responsible for reserving system resources for a job. Logs include information on created, canceled, scheduled, and executed jobs in the system. Torque logs are parsed and stored in an intermediate representation by LogDiver. Each entry in this intermediate representation consists of multiple fields describing time information on all phases of the job (creation, queue, execution, and termination times), user, group, queue, resources, type and list of

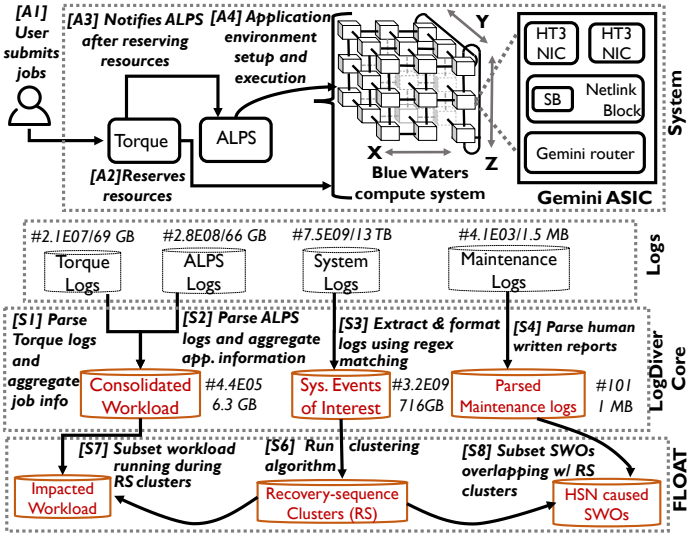


Fig. 3. Overview of system, data sources, and analysis tools

used nodes, and wall time used.

ALPS Logs are generated by "application-level placement scheduler" [14] responsible for application-environment setup and initialization on the reserved nodes. Logs include information related to the executed applications, exit codes, a list of used nodes, exit reasons, the invoked launch script, and the used walltime of each user application and job. LogDiver consumes ALPS logs to produce an intermediate representation in which each entry contains all the information about an application in column format.

Consolidated Workload is a dataset generated by LogDiver that combines parsed Torque logs and ALPS logs, giving end-to-end information about the workloads executed on the system. Such consolidation helps disambiguate user jobs from staff jobs and helps to understand the failure cause of the application. An application can fail due to system-errors, user termination, wrong-input, or bug in the code.

Maintenance Logs* and *Parsed Maintenance Logs are generated by the NCSA maintenance specialists and consolidated by Cray. Events are added to the failure report (1) upon any failure that requires special corrective actions (e.g., a manual reboot or repair of faulty hardware), and (2) upon events that cause system downtime (SWOs). Over 4,000 incidents were reported during this study, and they contained 101 SWOs. The failure reports are parsed into standard column format, and SWOs are retrieved and stored (in "Parsed Maintenance Logs").

2.3 FLOAT: A Tool for Reconstruction and Analysis of Recoveries from Logs

The goal of our work was to recreate network-recovery scenarios in order to characterize recovery mechanisms, understand the reason for their failures, to learn the impact of their failures on applications and the system. To achieve this, we augmented LogDiver [2] with FLOAT (shown in Fig. 3), an interconnect FaiLOver Analysis Tool. FLOAT enables us to increase understanding in this area by (1) decoding Gemini error/failure logs and then clustering errors, failures, and recovery steps (taken to contain and manage failures) into recovery-sequence clusters (*shown in step 6*); (2) retrieving applications that failed for system reasons during network recovery (*shown in step 7* in Fig. 3); and (3) retrieving SWOs that resulted from network-recovery operations (*show in step 8* in Fig. 3). We first define our failure model and recovery model and then discuss the clustering algorithm used in FLOAT to generate "recovery-sequence (RS) clusters".

Failure Model — In our model, a fault (e.g., lane congestion) can lead to fault/error events. An event is represented as – *event name (time, location)*. Errors (e.g., packet drops) can lead to errors/failures, and failures (e.g., link failures) can lead to further failures with respect to interconnect system. Faults/errors/failures propagate from some lower level in the Gemini-interconnect system (the lowest level in our model is a lane) to topologically higher levels (the highest level in our model is the complete system). Thus, our model resembles a fault-tree model with topological ordering as shown in Fig. 2(b). Not all faults, errors, and failures are logged in the system, however we assume that all failures that trigger recoveries are logged in the system. The set of all faults, errors, and failures is given by:

$$E_F = \{\text{e| where e is any fault, error or failure event}\} \quad (1)$$

Recovery Model — Interconnect-related recoveries are launched after the failure or repair of an interconnect-component (e.g., lane/link enable, disable, or failures). The recovery proceeds according to the state-transitions described in Section 2.1, and for each state there is a corresponding recovery event that is logged in the system. In case of failure of the recovery, depending on the recovery type, the system retries the recovery N number of times (e.g., 10 times for lane recovery) before stopping. In cases in which the final completion status of the recovery is missing, it is assumed to be a failed recovery. The set of all recovery events are given by expression 2, and the set of all recovery events that occurred at time t is given by expression 3:

$$E_R = \{\text{e} \mid \text{e} \in \text{interconnect-related recovery events}\} \quad (2)$$

$$E_R(t,l) = \{\text{e| e} \in \text{interconnect-related recovery events at time t and location l of occurrence of event}\} \quad (3)$$

Clustering Model — A recovery-sequence cluster is a combination of faults/errors/failures and recovery events such that all events in the cluster are related to each other (represented by $\overset{P}{\leftrightarrow}$ operator) in time and space. A recovery-sequence cluster $C_{RS}(t,l)$ at any time t and location l is formally given by expression 5. $\overset{P}{\leftrightarrow}$ is a mathematical operator that tests and finds all events related to a given event. An event e_1 is related to another event e_2 if and only if (1) $time(e_1) - time(e_2) \leq$ fixed sliding window time ($T = 120s$), and (2) $location(e_1) = location(e_2)$ or $location(e1)$ is topologically connected to $location(e2)$ (by traversing up or down the tree, but not both) as defined in the failure model. If any two recovery or failure events at two different locations are reachable, then the mathematical model creates multiple copies of the cluster. This happens because for each location a cluster is initialized with $E_R(t,l)$, but when the failure/recovery events propagate and one is reachable from another, each cluster having common events automatically spans the other. Therefore, only unique recovery-sequence

clusters are retained. However, the clustering algorithm can be implemented by maintaining a topological tree and merging the two clusters whenever one overlaps with the other. Further, the sliding window time was empirically found based on the models proposed in [15], [16]. In these studies, authors concluded that the optimum sliding window time is the knee of the curve drawn between the sliding window time and the number of clusters obtained using a clustering algorithm, as this decreases truncation errors (splitting of a cluster into multiple clusters because of a small value of sliding window time) and collision errors (two events not related to each other merged into a single cluster because of a large value of sliding window time). The proposed clustering algorithm did not miss any network-related recovery in the system. This is because whenever there is a recovery event (in the log files) our clustering algorithm initializes a recovery-sequence cluster, as shown in expression 5. The examples of recovery-sequence clusters are shown in Section 5.

$$E = E_R \cup E_F \tag{4}$$

$$C_{RS}(t,l) = \{e \mid e \in E \text{ and } \forall e_{rt} \in E_R(t,l), e \overset{P}{\leftrightarrow} e_{rt}, \text{ where } e \overset{P}{\leftrightarrow} e_{rt} \text{ denotes that } e \text{ and } e_{rt} \text{ are related}\} \tag{5}$$

3 Breakdown of Failover Failures

In the following, we use the described methodology to provide a first set of statistics on the lane recovery, link failover, and warm swap operations. During the considered time frame of 27 months, we measured 253,000 lane failures, 318 link failures, and 559 warm swap initiations. Table 1 shows the total percentage of recovery procedures that completed successfully, calculated separately for lane recovery, link recovery, and warm swap.

TABLE 1
Breakdown of the successful completion of the recovery procedures. Confidence intervals are calculated using Student's t-distribution at alpha = 0.05 aggregated per month

	Before Upgrade (1/1/13-10/31/13)	After Upgrade (1/11/13-3/31/15)	All (1/1/13-3/31/15)
Lane	98.4 ± 0.3 %	99.6 ± 0.7 %	99.1 ± 9.0 %
Link	53.8 ± 7.0 %	91.0 ± 20.2 %	75.8 ± 6.9 %
Warmswap	87.1 ± 2.7 %	96.4 ± 6.2 %	92.1 ± 3.0 %

Lane failover operations were successful 99.1% of the time in masking/restoring the lane failures. Thus, the impact of lane-recovery failures is limited, because a link can typically withstand up to three lane failures (discussed in Section 2). Since the lane recovery procedures run locally on the blades and are managed by the blade controller (L0), they may interfere with other recovery procedures managed through the L0, such as link failovers and warm swaps, as described in case study 3 (Section 5).

Link failovers were successful in only 75.8% of the cases. Examining the recovery-sequence clusters, we observe that the link-failover procedures failed because of additional failures (e.g., of a node, blade, mezzanine, or link) during failover and timeouts. A failed recovery could lead to a situation in which the system must be recovered manually. If the manual recovery is unsuccessful, an SWO is declared. More details are provided in Section 6.2.

Warm swaps were successful 92.1% of the time. Warm swap operations are performed by system administrators for system maintenance reasons, or when the network becomes unrouteable. Warm swaps are launched carefully by experts at fixed time intervals (e.g., twice a week) or during system emergencies. Manual orchestration helps deal with any additional failures that might occur during recovery. Warm swaps are still susceptible to failures occurring naturally in the system. *Out of 559 warm swap procedures, 77 were launched exclusively to manually recover the network (after a failure of the automatic recovery)*, of those only 63.6% succeeded. A thorough analysis of the cause of failures of failovers is discussed in the next section.

3.1 Time Between Recoveries

To understand the rate of recovery procedures, and failures of these recoveries, we calculated (a) Mean Node-hours Between Recovery procedures (MNBR) which takes system scale (in terms of number of nodes) into account, and Mean Time Between Recoveries (MTBR) which is a traditional resiliency metric.

Mean Node-hours Between Recovery MNBR is calculated according to equation 6, where one node-hour equals to one node's up-time (either computing or idle) for one hour. Our measurements take into account the actual node hours (i.e., does not account for hours when the node was unavailable due to failure or maintenance) for each Blue Waters node in the 2 year and 3 month time interval considered in this study.

$$MNBR = \frac{\text{Total node hours}}{\text{Count of recovery events}}, where \tag{6}$$

$$\text{Total node hours} = \sum_{i=1}^{i=\#\text{Nodes}} \text{node-hour of node i} \tag{7}$$

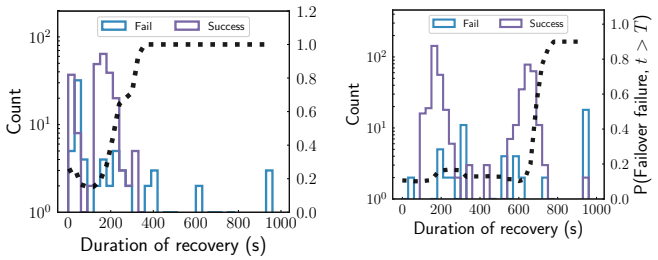
Table 2 summarizes the MNBR for lanes, links, and warm swaps for failed recovery procedures as well as the total number of launched recovery procedures. Warm swap procedures have lower MNBR, as these procedures are initiated multiple times on fixed days (twice a week) as part of maintenance. A blade that is going to be warm swapped is drained of any running job before the procedure is initiated.

TABLE 2
Mean Node-Hour Between Recovery (MNBR) events

Recovery Procedure Type	MNBR of failed recoveries	MNBR across recoveries
	(hours)	(hours)
Lane recovery	211,813	1,934
Link failover	6,354,375	1,538,638
Warm swap	11,120,157	553,608

Mean Time Between Recovery

MTBR is calculated according to equation 8 where system up-time is obtained by subtracting maintenance hours and duration of all SWOs from operational hours. The measurements show that lane recoveries, link recoveries, and warm swap procedures are launched every 4.1 min, 59.52 h, and 33.86 h (across all warm swaps), respectively. Note, if we consider only those warm swaps that were launched to handle failures of automatic recovery, the MTBR comes to be about 703 hours.



(a) Link failovers (b) Warm swaps

Fig. 4. Histograms of the completion times of single recovery procedures for (a) link failovers and (b) warm swaps. Hatched black lines show the failover failure probability after time $t > T$ seconds

$$MTBR = \frac{\text{System up-time}}{\text{Count of recovery events}} \quad (8)$$

3.2 Impact of software updates

The recovery software is a core part of the network design. Thus, it is imperative to understand the effects of system software updates on the successful completion of recovery procedures. Table 1 gives the breakdown of the completion status of recovery procedures before and after a major software update (on November 1, 2013) in the SMW (a patch for the recovery control software fixing design/code bugs). The percentages of success of all of these procedures improved significantly after the SMW software upgrade. **The ratio of failures of procedures to the total number of launched procedures decreased by 4.0x for lane recovery, 5.1x for link failover, and 3.6x for warm swap procedures**. This shows the importance of validating the network resiliency software and protecting it against its own faults/failures.

4 Recovery Characterization

In this section, we characterize the recovery procedures to understand factors causing their failures. In the remainder of this paper, we focus only on link failovers and warm swaps, given the high success rate of lane recoveries and their negligible impact on the system.

4.1 Failover Completion Time Versus Probability of Failure

The duration of a recovery procedure depends on the failure type, the path taken by the procedure (refer to the recovery state transition diagrams shown in Figs. 1 and 2(a)) during the recovery, and the time taken to complete each stage during the procedure. As a result, the durations of recoveries are highly variable, ranging from $<$ 1 s to $\sim$1,000 s. Fig. 4(a) and Fig. 4(b) show the histograms of the completion times of the recovery procedures and the probabilities that the procedures will fail after $t > T$ seconds for link failovers and warm swaps, respectively.

The histogram of the link failover procedures (see Fig. 4(a)) has four parts: region **A** (0 to 60 s), region **B** (60 to 240 s), region **C** (240 to 420 s) and region **D** (420 to 1,000s). To explain the characteristics of this distribution, we analyzed the completion time of the procedures based on the data logs. Link failover procedures can finish in $<$ 60 s (region A) when the procedure (1) does not involve route recomputation, e.g., when the recovery procedure transparently disables the faulty link (as discussed in Section 2) or (2) fails very quickly (i.e., one of the early stages in the failover path fails). When the network is partitioned, it takes longer to compute, assert, and install new routes (around 180 s, region B). In region B, the probability of success is higher than the probability of failure of the recovery procedures. For example, at time 240 s from the start of recovery, the success probability is around $\sim$0.8, hence the failure probability is 0.2 (calculated as 1.0 $-$ success probability). Region C and D capture recovery procedures that have long completion times (barely meeting timeouts) because of additional problems in recovering the system (e.g., cascading failures or hangs of recovery procedures).

Fig. 4(b) shows a histogram of the completion times of the warm swap procedures. There are three distinct peaks (at $\sim$200 s, $\sim$600 s, and $\sim$1,000 s) in the plot. The first peak corresponds to cases in which the warm swap procedures disable blades/cabinets or fail quickly (because of timeout of the first stage of the procedure, see Fig. 2(a)). The second peak corresponds to cases in which the warm swap procedure either adds or removes new nodes, blades, or cabinets. The third peak corresponds to situations in which warm swap procedures never complete (except for one case in which it completes after 983 seconds); we set the upper bound at 1,000 seconds, after which we mark the recovery procedures as failed. The types of additional failures and their impact on the ongoing recovery is explained in the next subsection.

Across both type of recoveries (link failovers and warm swaps), the probability that the recovery fails increases with time, as shown by hatched black lines. For link failovers, the failure probability starts increasing after 180 seconds. The probability that the link failover fails after 240 seconds is 0.7, and after 390 seconds it is $\sim$1.0 (i.e., *link failover always failed in our dataset after 390 seconds*). This shows that the link failovers are highly susceptible to the recovery duration. Comparing the corresponding values for warm swap procedures, we observe that warm swaps complete successfully until 600 seconds. After that time, however, the failure probability for warm swap procedures rises steeply.

There is a design trade-off in selecting the optimum timeout values of the stages (i.e., sub-procedures) in recovery procedures. The smaller the timeout values, the lower the chance of additional failures during recovery. However, it also means there is less time to complete all the required functions in the sub-procedures before the timeout, causing failures of the recovery procedures. On the other hand, it is clear that the currently set of large timeout values exposes the recovery procedures to additional failures. The situation would be worse for future large-scale systems, as they would require more time for calculating, verifying, and installing routes, so the probability of additional failures during recovery would be higher. *Thus, it is important to choose recovery procedure timeouts carefully and to optimize their values for individual systems (depending on the scale)*. Recent techniques, such as hierarchical callback-based timeouts [17], would allow for sub-second failure detection, thus reducing timeouts to smaller values whenever possible.

4.2 Characterization of Failures during Recovery

To facilitate characterization of the impact of additional failures that occur during the recovery and their impact on the system (in terms of propagation), we have divided the space of all possible failure events during recovery into *propagated* (84.4% of all failures during recovery) and *independent* failure events (15.6% of all failures during recovery) with respect to the ongoing recovery. A failure event type could be a result of propagation (i.e., failure event has a *relation* $\overset{P}{\leftrightarrow}$ with the recovery) or independent occurrence (i.e., the failure event has *no relation* to the recovery). However, a failure event can be placed into one of the two categories through the analysis of recovery-sequence clusters (described in section 2.3). Consider the case studies 1 and 2 discussed later in Section 5. The case study 1 shows that the failure of the blade occurred due to failure propagation (i.e., failure of the pump gasket lead to the failure of mezzanines which caused blades to become unreachable) whereas case study 2 shows another example of blade failure which was caused by faulty voltage regulator module (*VRM Fault*) and hence, occurred independently of ongoing network recovery. Formally, propagated failure events (E_{FP}) and independent failure (E_{FI}) events are defined by Expression 9 and Expression 10, respectively. For the following analysis, we consider only those independent failures that overlap in time with recoveries.

$$E_{FP} = \underset{t,l}{\cup} C_{RS}(t,l) - E_R \tag{9}$$

$$E_{FI} = E_F - E_{FP} \tag{10}$$

Not all events from the set of independently occurring failures or propagated failure events prevent the successful completion of the recovery. The events that adversely impact the successful completion of the recovery are classified as *critical events*. These events were obtained by evaluating the conditional probability of the failure of the recovery procedures given the occurrence of a failure event E_f from the set of all events and use of domain knowledge. Table 3 lists the top 10 critical events observed in the system.

Fig. 5 shows the probability distribution of occurrences of additional failures of types "independent," "propagated," and "critical" over time. For the independent failures shown in Fig. 5(a), the occurrence probability is much higher for the first 180 seconds than for the rest of the observed period. The reasons are that the bulk of the errors occur in the first 200 seconds and that most of those errors are machine-check exceptions (MCE). On further analysis, we found that MCEs occur frequently in the system and that during quiescence due to suspension of all CPU-related activity, fewer or no MCEs are observed. Similar trend was observed for propagated failures (shown in Fig. 5(b)). The peak in the the errors is due to high packet and Lustre file system [18] client-related errors occurring before the network is quiesced successfully. Hence, the system is more vulnerable to propagated errors in the first few seconds. *In summary, the probability of occurrence of a failure during the recovery is nonzero, and additional failures can occur at any time during the recovery.*

Distribution of critical failures — Fig. 5(c) shows the critical event occurrence probability distribution during recovery procedures. Because recovery procedures suffer very few critical failures during their executions, we show the probability that a recovery procedure suffered at least $k \geq N$ failures (where N = 1, 4, 7) for failed and successful recovery procedures in Fig. 6(a) and Fig. 6(b), respectively. The probability that a recovery procedure suffers from $k \geq 1$ failure(s) increases at a higher rate for failed recovery procedures than for successful recovery procedures. Similar trends are observed for $k \geq 4$ and $k \geq 7$ failures during the recovery procedures. The increase is linear for failed recovery procedures, but near zero in the case of successful recovery procedures. By 500 seconds, all recovery procedures have encountered at least one failure. Not all failures are equally critical. The recovery procedure's failure depends on (1) the stage in which the recovery is executing when the failure occurs, (2) the type of the failure, and (3) the number of failures. The probability of the failure of a recovery procedure given the occurrence of critical events is summarized in Table 3 for the top 10 event types. Failure events such as "routing table corruption," "ASIC fatal error," and "ASIC chip errors," "fan fault," and SDB (system database, used for maintaining the state of the system) timeout always lead to recovery procedure failure. "Route computation failure" could be due to additional link failure(s) or timeout of the route computation phase. For all the other events mentioned in Table 3, the conditional probability of recovery procedure failure is less than 0.85, and those events lead to the failure of the recovery under specific conditions (i.e., in combination of other failure events not mentioned in the table). For example, auto throttling (for which the the recovery procedure failure probability is 0.25), which is a network congestion protection mechanism, can lead to the failure of a warm swap during quiescence.

TABLE 3
Summary of critical events threatening successful completion of recovery

Critical event	CRFP[1]	MNBE[2]	AOT[3] detection time	
		(hours)	mean(s)	median(s)
Routing table corruption	1.00	4932	272.10	10
Routing fault	1.00	1031577	108.40	165
ASIC fatal error	1.00	6189346	686.14	800
Route computation failure	1.00	12378693	1.20	1
ASIC chip error	1.00	285750	32	31
SDB time out	1.00	44209618	Hang	Hang
KRSIP	0.83	814	14.77	12
Link failures	0.85	11071	90.31	13.50
BMC time out	0.25	6877051	170.72	146
Auto-throttle	0.25	390	487.51	562

Terms — [1]CRFP: Conditional recovery failure probability given the occurrence of critical event, [2]MNBE: Mean node hours between events, and [3]AOT: Ahead of time detection

The events listed in Table 3 can be used as early indicators for ahead of time (AOT) detection of the failure of a recovery. The AOT detection values vary from as low as one second to as high as 686 seconds. The ability to detect failure of recovery procedures could potentially allow us to terminate an ongoing recovery and restart from a recovery phase instead of restarting from scratch after the current recovery attempt fails. Terminate and restart (e.g., micro-reboot [19]) techniques have been successfully deployed in Linux-HA systems to orchestrate successful recovery with minimal intervention from other failures. We plan to

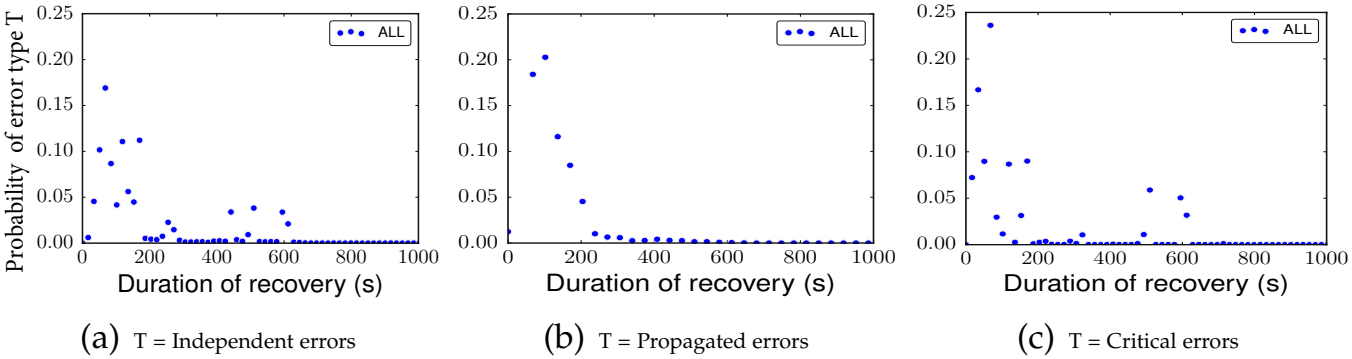


Fig. 5. Probability of additional errors/failures during recovery

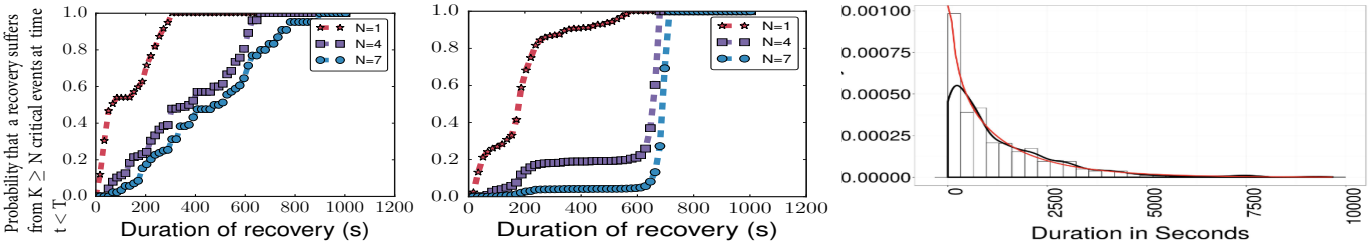


Fig. 6. Probability that a recovery procedure will suffer from $k \geq N$ critical errors/failures at time $t < T$ for (a) failed recovery, and (b) successful recovery. (c) Recovery-sequence cluster duration characterization.

extend this analysis in the future by using machine learning methods to develop algorithms for runtime prediction of recovery procedure failures.

In summary, *the system is unprotected from other failures that occur while it is executing recovery procedures*. Additional failures in the system can occur during the execution of recovery procedures either in Gemini components or in other parts of the system.

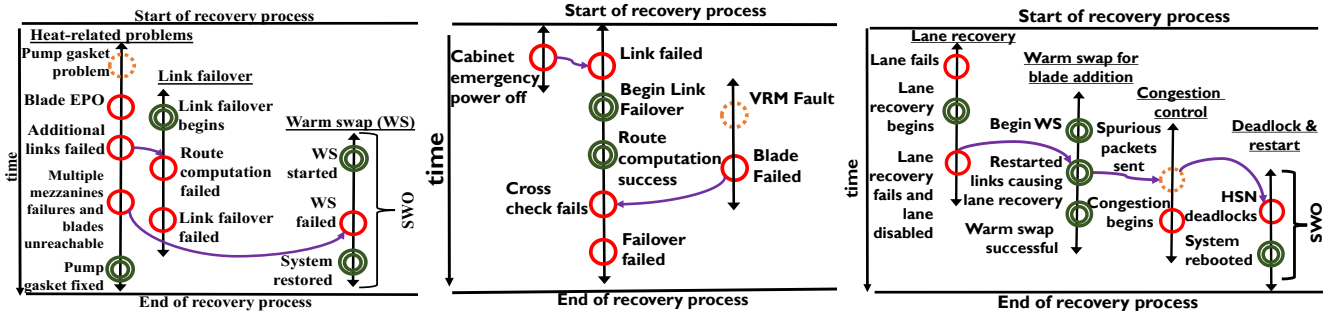
4.3 Duration of Recovery-Sequence Clusters

A characterization of duration of recovery-sequence clusters using the probability density function plot (refer to Fig. 6(c)) indicates that the ***duration of the recovery-sequence cluster is influenced by overlapping recovery procedures and failures***. This distribution shown in Fig. 6(c) can be modeled as a gamma distribution with parameters (scale = 0.73 s; and shape = 1,766.53 s). In this figure, 81% of the recovery-sequence clusters have duration of less than 41 minutes, 50% have duration less than 15 min, and 25% have duration less than 3 min. The tail of the distribution includes 14 samples (not shown in figure for visual clarity) all of which corresponded to SWOs. In this distribution, most of the recovery-sequence cluster duration are small, and the main contributions to the sample mean or variance come from the rarely occurring long recovery-sequence clusters (i.e., clusters with long duration). A good fit (the p-value of goodness-of-fit was found to be 0.076) between the modeled gamma distribution and observed data confirms our observation that the time to completion of the recovery sequence depends on the duration of the cluster. Gamma process [20] is ideally suited to model gradual deterioration that monotonically accumulates over time, such as wear, and aging, which are common causes of failure of engineering components (empirically confirmed from Fig, 4 for recovery procedures). Thus, the longer the lifetime of a component (in this case recovery procedure), the higher the chances of its failing. Due to the self-restarting nature of the recovery upon additional failure, the duration of the recovery-sequence cluster (which is composed of one or more recoveries and their retries) gets elongated with increasing recovery time, i.e., there is a component of auto-correlation [20]. In a large-scale system like Blue Waters, the time to complete the recovery can take up to 1200 seconds in some cases (as discussed in Section 2.1). Using the analyses from Fig 6, we argue that **in a large-scale system, the recovery procedures should be able to tolerate the occurrence of additional failures/errors during recovery**. Therefore, to avert system-wide failures, the health supervisory systems (HSS) of future large-scale production systems should be capable of tracking failure events during the recovery which can help to execute a recovery successfully. Our results show that it is possible to track failures during recovery and take proactive actions, as discussed earlier in this section.

5 Case Studies

In this section, we describe three case studies showing examples of recovery failure scenarios due to independent and propagated failures. We also present a scenario in which the system deadlocked despite successful recovery. Those examples demonstrate the effectiveness of our tool and analyses.

Figs. 7(a), 7(b), and 7(c) show examples of recovery operations that failed due to the occurrence of failures during their operations. We use a recovery-sequence cluster diagram to show faults, failures, and associated recovery actions initiated to restore the system. Each vertical line represents a sequence of related activities (such as recovery procedures or sequences of faults/failures related to the same root cause) consisting of one or more events. Dotted



(a) Case Study 1 (b) Case study 2 (c) Case study 3

Fig. 7. Recovery-sequence cluster of studied case studies. Each vertical line represents a sequence of related activities. Dotted circles, single circles, and double circles represent faults, failures, and recovery events, respectively. These events are temporally ordered from top to bottom and left to right.

circles, single circles, and double circles represent faults, failures, and recovery events, respectively. These events are temporally ordered from top to bottom and left to right. An SWO window in these figures shows the time window in which system administrators declared an SWO.

Case study 1: Failure of an external cooling subsystem that triggers an emergency power-off of a blade and subsequent widespread fault propagation. In this case (shown in Fig. 7(a)), the failure of a pump gasket (*pump gasket problem*) caused the temperature to rise in a cabinet. The overheating triggered emergency power-off (EPO) of the blade (*blade emergency power off*), a protection mechanism to guard the blades from permanent damage. This event caused links to go down, hence triggering link failover (*link-failover begins*). As the link failover progressed, additional failures occurred on other links in the same cabinet, caused by ASIC failures (*additional links failed*). These failures induced further failures in the route computations (*routing computing failed*); indeed, the topology experienced changes from the time the routes were calculated to the time the routes were asserted. The failure of the route computation (and its retries) to establish an alternative path for nodes to communicate with other nodes in the system led to the failure of the link failover (*link-failover failed*). At that point, the system administrator tried a manual recovery of the system. However, as the overheating effect increased and propagated to nearby cabinets and blades, additional mezzanines failed (*multiple mezzanines failures and blades unreachable*). This caused the manual recovery attempt to fail as well. To solve the problem, an SWO was declared for several hours until the pump gasket problem was detected and fixed. In this scenario, the multiple network mezzanine/blade failures ("multiple mezzanines failures and blades unreachable") were the result of failure propagation from the pump gasket failure ("pump gasket problem"). However, the SMW knows only about the failures and knows neither about the propagation nor the cause of this propagation. The failure of recovery operations was due to the inability of the recovery procedures to recover from multiple failures occurring in close proximity in time, i.e., *the time between the failures was less than the time required to orchestrate a successful recovery operation.*

Case Study 2: An independent failure during a failover. Fig. 7(b) shows an example of a failure recovery operation due to an independent failure of a component ("Blade Failed") during recovery. Unlike the previous case, in this scenario a blade failure occurred independently of the failure and recovery events, i.e., independently of "cabinet emergency power off," "link failed," and recovery events marked in green in the figure. The failure of the blade was caused by "VRM fault" (voltage regulator module), which is responsible for providing electric power to the blade. However, for a SMW charged with management of recovery of network components, a blade failure caused by propagation (as in previous case) and one occurring independently are identical. The SMW cannot distinguish between the two types of blade failure and thus initiates exactly same recovery sequence in either case. In this case, the system activity is interrupted after the failure of the failover ("Failover failed.") A warm-swap procedure was executed manually by sysadmin to immediately restore the system functionality. Hence, this incident was not declared as SWO.

To counter the problems studied in case study 1 and case study 2, we argue for using the monitoring of data to dissect relationships between failures and for understanding the cause of failures in real time to help manage failures more accurately. For example, in case study 1, SMW could have sent alerts to sysadmin for repairing the pump gasket before initiating recovery or could have completely removed/killed the offending cabinet from the network topology using a technique called "STONITH" (i.e., Shoot The Other Node In The Head). For situations like case study 2, the network-recovery can retry the recovery with slightly higher delays between retries (exponential backoff) with some upper-limit threshold. Monitoring and dissection of the cause of the failure could help decide between strategies (e.g., between STONITH, exponential backoff, or predictive mitigation (discussed elsewhere)).

Case Study 3: An example showing that a successful failover does not necessarily lead to a consistent system state. This example also captures failure propagation in the system. Fig. 7(c) shows a successful warm swap operation that triggered a latent design bug (in the initialization software), causing network deadlock and congestion. Initially, a single lane failure triggered the lane-recovery procedure. The lane recovery failed, causing the lane to be disabled permanently (*lane recovery fails and lane disabled*). Soon after the lane recovery failure event, the system administrator (as part of an unrelated maintenance procedure) launched a warm-swap procedure to add blades (*begin warm swap*) to the system. During the warm swap, one of the sub-procedures checked and initialized newly added links.

Because of a design bug in the initialization software, that sub-procedure, in addition to initializing newly added links, also reinitialized other alive links in the system, causing the disabled lanes to be re-enabled (*lane recovery fails*, *lane disabled permanently*). Although the warm-swap procedure eventually succeeded, it forced a resend of old packets stored in the card buffers of the disabled lanes. Since the addition of blades changed the routing information on the nodes, the spurious packets set with the old routing information were rerouted by Gemini using the new routing information. This caused a deadlock of the HSN (*HSN deadlocks*). Although the Gemini interconnect has built-in provisions to avoid such dependency cycles in routing paths of packets, a deadlock can still occur because of corrupt routing information in the ASICs (*routing table corruption*) or the sending of packets with incorrect/stale header information, as in this case. The deadlock further led to congestion (*congestion begins*) due to blocked paths in the network and, consequently, activated congestion-avoidance mechanisms (i.e., network throttling). At that point, all applications were stopped and a full system restart was required to restore the system operations, forcing a termination of the jobs.

The foregoing scenario revealed a bug in the warm-swap procedure, which Cray engineers fixed with a patch. The fix avoided the reactivation of previously disabled links by initializing only the links connected to the newly added blades. This scenario illustrates one possible reason for an invalid system-state at the end of a successful recovery, and one possible cause of application failures during a successful recovery. These examples demonstrate that tracking of the failures in a system and maintenance of a consistent global system state with respect to errors and failures, is essential to enhancing robustness of failover procedures. For example, the ability of system state tracking could be useful for helping a scheduler avoid placement of jobs during recovery, and for helping a file system coordinate its own recovery with the network recovery.

6 Impact of Failovers

In this section, we describe the impact of recovery procedures in terms of error propagation, system-wide impact, and application failures.

6.1 Error Propagation in the System

Failovers, irrespective of their exit status, cause additional errors in the system. During recovery procedures errors do propagate and frequently manifest as timeouts (e.g., timeout of client running on compute node which is trying to contact the file system) and corruptions (e.g., misrouted or dropped packets in the network).

For instance, Fig. 8(a) shows the average number of packet errors (dropped packets, misrouted packets, or corrupted packets) across all interconnect recoveries; Fig. 8(b) shows a similar propagation for other kinds of errors (file-system errors, scheduler errors and system services). The reason for such propagation can be adduced to the recovery mechanisms in other parts of the system, which rely on the network recovery to succeed quickly. For example, the MOAB job scheduler can be negatively impacted by the quiescence phase of the recovery procedure. During the quiescence MOAB may fail to contact the "application level placement scheduler (ALPS)" client running on nodes to remove applications and reset the nodes. Similarly, during the quiescence, applications can drop an ongoing connection to the file-system ("client eviction"), as shown in the example in Fig. 8(c). The issue of overlapping network quiescence with the file-system recovery has been partially addressed in Lustre through the use of imperative recovery [21]. Imperative recovery accelerates reconnection between application (running on compute node) and storage via notification and callback mechanism instead of using timeouts. A detailed discussion on imperative recovery can be found in [21]

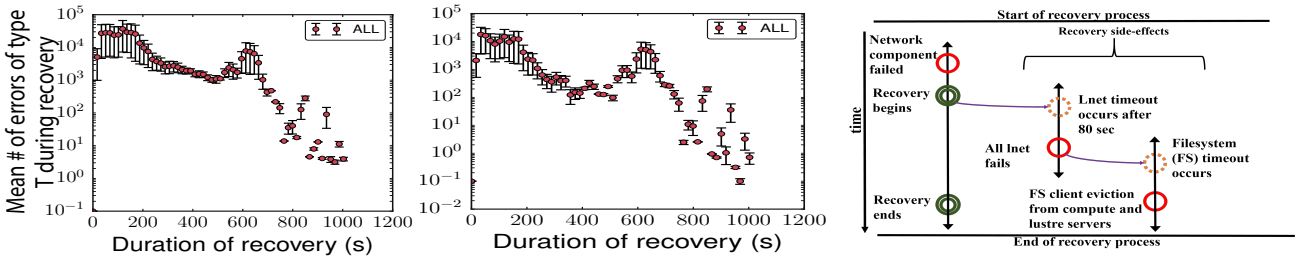
The number of packet-related errors during recovery procedures is initially very high due to unavailability of the network components (that have failed). The system is designed to stop the injection of packets during quiescence. Hence, packet-related errors decrease sharply from 4 to 5 orders of magnitude in the initial phase of the recovery procedures to few hundred seconds in the latter phases. However, our in-depth analyses suggest that the quiescence and CPU stalling is not perfect (as there is non-zero delay in quiescing the whole network) and there is a chance of packets leaking or getting lost during the quiescence phase of the recovery procedures. This loss of packets negatively impacts MPI-based applications which assume reliable packet delivery.

Similar trends were observed for other errors (i.e., file system, scheduler and system services) during the recovery procedures. The initial peek in the errors (refer to Fig. 8(b)) is due to Lustre connection problems (message drops) and client failures. As time passes, the number of errors decreases, but more critical events are triggered because of timeouts. For example, Lustre network routers (LNet) fail after 40 seconds (due to timeout) of the quiescence. Forty seconds is the default timeout value for the routers before they initiate their recovery. Fig. 8(c) shows a typical failure propagation scenario in the file system due to interconnect-related recovery. The problem starts with a network component failure, and a corresponding recovery is triggered to restore the system. However, during this recovery all the LNet routers (which are used to communicate between the compute cluster and storage cluster) connected using Gemini ports become unavailable due to the quiescence of the network as mentioned earlier. Unavailability of the LNet router leads to the failure of the Lustre clients that are communicating (after a timeout expiration), as servers and clients cannot ping each other due to loss of LNet routers. Cray and NCSA have deployed imperative recovery [21] to protect from some of these cases. Our dataset does not contain the recovery procedures after the deployment of this new feature, and thus we are unable to investigate its usefulness.

6.2 Impact on System Infrastructure

In this subsection, we dissect the reasons for the 28 SWOs caused by interconnect-related recoveries. The breakdown is shown in Table 4 and described below.

Handshake issues. In those cases, SWOs occurred because of improper/no handshake (e.g., throttling during quiescence) and timeout issues (e.g., blade controller not



(a) T = Packet errors, caused by recovery procedures

(b) T = File system, scheduler and system services errors, caused by recovery procedures

(c) Propagation of errors in the file system

Fig. 8. Propagated errors characterization for packets (a) and other components (b); an example case study for propagation of errors in file system due to network errors

TABLE 4
Breakdown of SWOs reasons and their occurrence count

SWO Reason	#Count
Handshake issues	16
Network deadlock	4
Concurrent recoveries in the file-system and network	5
Occurrence of independent failures negatively impacting the recovery	3

responding during route computation phase of the recovery and timing out). As pointed out earlier, the interconnect recovery procedures consist of multiple phases, each with its own timeout value. A timeout can expire due to either (1) failure in coordinating with other services/components (e.g., L0s, or system database) or (2) inability to complete some computations (e.g., the new routes computations or pings) within a fixed time window. Those timeouts are defined in system configuration files. Some of the timeout values depend on system-scale (such as routing time out value), whereas others are fixed (such as lane masking) and cannot be configured because of hardware/technology requirements and limitations.

Network deadlock. SWOs were due to HSN deadlock caused by soft errors in Gemini ASIC (such as routing table corruption and Gemini ASIC logic failure due to soft errors). In three cases, the deadlock was due to routing table corruption, whereas in one case deadlock was caused by ASIC logic failure (e.g., due to soft errors). Although those errors are transient, in a live system, they can lead to deadlocks. Routing table corruption happens due to the corruption in the buffer that stores the routing tables. Routing table buffers are not protected by error-correcting codes (ECC), and hence routing table corruption (e.g., due to soft errors and/or bad data being written when populating the routing table) is a real possibility. To prevent such errors, Cray installed a patch (in early August 2013) in the system to create back-up copies of routing tables, to be used for periodic checking of routing tables on the Gemni ASICs. For safely correcting the routing tables (i.e., replacing the corrupted routing table with the backup), the network activity on the corrupted router is interrupted throughout the duration of correction. Most of those interruptions are treated gracefully. However, in some cases, by the time new routing tables are calculated, there might already be network packets sent out with wrong routing headers. These spurious packets create cyclic paths, leading to a network deadlock.

Concurrent recoveries in file-system and network. Interconnect-related recoveries overlapped with Lustre file-system recovery procedures, which led to SWOs. In one case, an interconnect recovery operation led to failures of several Lustre file-system clients (because of the quiescence phase of interconnect recovery operation) that triggered Lustre recovery procedures. Although, the interconnect recovery was successful, the Lustre recovery procedure was unsuccessful. This is shown in Fig. 8(c) and described in previous subsection.

Occurrence of independent failures negatively impacting the recovery. Those SWOs were due to independent failures (i.e., failures having no relation to the recovery) that occurred during a recovery procedure. Section 4 discusses these additional failure types in detail. Such failures are hard to handle unless a global view of the system state is built to synchronize recovery activities and to eliminate or protect from unexpected interference among these activities.

6.3 Impact on User Applications

We characterized the impact of recovery operations on the system workload by computing the number of applications failing due to network-related failures and recoveries on the nodes involved in the recovery. The recovery intervals considered are the time windows that include successful network recoveries, failed network recoveries, and system-wide outages (caused by network recoveries). In total, we had 438,796 applications running in the time window of the network recoveries. Table 5 shows a breakdown of number of application (app) runs at different scales (in terms of number of compute nodes) and percentage of application failure at that scale. From Table 5, it can be inferred that *full-scale applications (i.e., applications using more than 50% of the available nodes) are 33.5 times more sensitive to the unavailability of the network during recovery than are single-node applications.*

TABLE 5
Breakdown of applications (apps) runs at different scale (i.e., number of compute nodes) and % failure in that scale

Scale	#XE nodes	#XK nodes	#Apps	Failures (%)
Single	$\leq$ 4 (1 blade)		305472	0.40
Nano	$\leq$ 96 (1 cabinet)		117271	1.70
Low	$\leq$ 512 (1 row)		12609	2.00
Med	$\leq$ 5896 (25% sys)	$\leq$ 1056 (25% sys)	3047	4.17
High	$\leq$ 11792 (50% sys)	$\leq$ 2122 (50% sys)	240	4.58
Full	$>$ 11792	$>$ 2122	157	13.38

Further, we compared the percentage of application failures during successful and failed recoveries. In the studied period, 3.4% of running applications failed during unsuccessful recoveries compared to only 0.7% of application failures during successful recoveries. *Thus, there is non-negligible*

chance of application failures during network recovery even in cases when the recovery completes successfully.

Next, we compared the percentage of application failures during the Gemini recovery operations with the application failures during normal system operation. The total up-time of the system (i.e., operational hours) from January 2013 to March 2015 was 18,926.6 hours, and the total time spent for network recovery was 91.56 hours. The total number of applications that failed was 50,874, out of which 3,591 failed during network recovery. The mean time between application failures for reasons other than Gemini is 18,926.6 / 48,443 = 0.39 h, hence the failure rate is 2.6 failures per hour. The mean time between application failures for network recovery is 91.56 / 3,591 = 0.025 h / failure, hence the failure rate is 40 failures per hour.

Finally, we sampled 25 days of Blue Waters workload that specifically had no system software or component failures. The percentage of application failures during those 25 days is close to zero regardless of the application scale. Through those analyses, we show that *failover procedures (regardless of whether they are successful) impact applications and frequently lead to application failures.*

The failure cause of an application and the events leading to this failure (i.e., fault-to-failure propagation path) can differ significantly for different applications depending on runtime-framework (such as MPI [22], charm++ [23], PGAS [24]) and system-state. Due to the limited logging/monitoring capabilities of the system and to the diversity of application frameworks (and their network usage profiles), deciphering the complete fault-propagation path leading up to the failure of the application is intractable or very hard. However, our study allowed us to make the following three general observations, which we elaborate through examples.

• We found charm++ applications to be tolerant towards recovery procedures. When we looked at three large-scale charm++ applications [25] — NAMD, ChaNGa, and Episimdemics — only ChaNGA was seen to have a higher failure probability (0.08), whereas other charm++ applications only failed once. Charm++ employs different collaborative checkpoint/restart mechanisms (disk and in-RAM checkpoint implemented as part of the FTCharm++ library [26]), all relying on error detection performed by means of heartbeat. There are two phases in the checkpoint support. In the first phase, after reaching a global synchronization point, each node stores its checkpoint in the main memory. After every node safely stores the newest copy of the checkpoint, the normal execution resumes. The Charm++ runtime system is responsible for creating a backup copy of each checkpoint on another node. When a processor crashes, the restart protocol is automatically invoked to recover all objects using the last checkpoints. This technique can successfully recover from one or more failures at a time, as long as all the nodes storing backup checkpoints are alive.

• GPU (Graphical Processing Unit) applications (running on Cray XK nodes) are most vulnerable to network-related failures and recovery procedures. For example, the failure probability of "AMBER" [27] was seen to be as high as 0.20. In general, for all GPU applications running on more than 100 nodes, we observed high failure probability during recoveries and network-related failures. GPU applications can fail when using GPUDirect [28] due to PCIe-related errors (as GPUs and PCIe interfaces cannot be quiesced) unlike network interface cards, which are quiesced during network recovery.

• Finally, PGAS (Partitioned Global Address Space) applications are vulnerable to network failures and recoveries. PGAS applications require ordered message delivery capabilities and atomic memory operations. Some of the atomic transactions (such as "atomic-add-one") cannot be replayed or retried safely in case of failures due to the limitations of the transaction commit protocol. For example, if an application process on compute node A sends an atomic memory operation to compute node B (say "atomic-add-One" to some memory location), that "happens" in hardware. During link-failures and recoveries, requests to remote node B or response from remote node B can get lost. In case the response is lost and the network simply replays the transactions, it would make the final result increment by two instead of one. Such scenarios can be handled either using two-phase commit protocol in the network layer or by application-specific exception handling in the code. However, PGAS jobs tend to rely on a lot of small transactions and such an overhead would slow down the application. Hence, PGAS-based applications, in general, do not tolerate lost transactions and trade resiliency for higher performance. This leads to failure of PGAS-based applications during such scenarios. We measured failure probability of PGAS-based applications across many different application types[3], and found it to be 0.4 during network-recoveries.

7 Limitations

An empirical study like ours has limitations when it comes to applying the findings to other systems and domains. Here, we comment on the threats to the validity of our study under three axes: *construct validity, internal validity, and external validity*.

Construct validity implies that variables associated with the study are measured correctly, i.e., that the measurements are constructed in accordance with theoretical foundations in the area. In this study, we analyzed field-failure data from over 27 months of operational time for calculating various resiliency metrics, such as MNBR, in order to ensure statistical significance of those metrics. However, the measurements discussed in Section 4.2 can be subjected to internal validity, which is discussed next.

Internal validity implies that there are no systematic errors and biases. In any distributed system, the validity of time stamps across machines can be questionable due to clock drift and skew. Since our measurements involve measuring probabilities of failure with time (at seconds-to-minutes intervals), the metrics presented in subsection 4.2 are potentially susceptible to small variations. However, we believe the sliding window time (T = 120s) chosen in this study is greater than the expected clock drift in the system. Further, to eliminate any event-selection bias (the filtering stage of the tool) and to ensure the correctness of our tool (FLOAT), we took the following two additional steps:

3. No single PGAS-application was found to be running statistically number of time during failover.

- Worked with the vendor and system engineers to choose events for filtering.
- Conducted our own fault injection experiments to test the validity of chosen events and the capability of the clustering algorithm to capture network-related events [29].

External validity concerns the extent of generalization of a study to other systems. The logs generated and collected in this study are very specific to Cray Gemini-based systems. However, the tools and methodology presented in this paper can be adapted to newer a generation of systems. In particular, we have tested our tool on the Cray Aries (DragonFly network) system (Edision) [30] at NERSC.

8 Related Work

Previous HPC failure studies have focused on the measurements of mean time between failures (MTBF), availability, and reliability of the system [31]–[34] and on the impact of failures on running workloads [35], [36]. The studies on interconnect networks for HPC systems are mainly focused on performance, network protocol, hardware reliability, and congestion control. There is a lack of empirical study on the efficacy of recovery mechanisms and the impact of recovery operations on the user workload.

HPC reliability: Achieving exascale performance within the decade is essential for progress in science and technology [37]. Exascale computing without resilience is not possible. Many exascale technical studies [38]–[40] have argued that the key challenges for scaling to exascale are energy, memory, concurrency, and resiliency. In [31], [41], the authors show that faults, errors, and failures are widespread in supercomputers and that is only going to increase in future systems.

Network reliability: The interconnection network is one of the three pillars of an HPC system (the others being the computing and file system subsystems). Ensuring interconnect reliability is fundamental to computing, as it serves as the major data-path for the communication between application processes and file systems. In [9], Dally and Towels have summarized the existing failure modes for building resilient networks. The research in interconnect reliability has mainly focused on building reliable routers [42], [43], deadlock avoidance [44], [45], data transmission reliability [46], [47], and congestion control [48], [49]. There is limited research on the effects of network recovery on applications (in execution) for massively parallel systems. Recent techniques, such as Immunet [50] proposes low cost fault-tolerant switching mechanisms using hardware reconfiguration techniques, but such techniques generally limited to certain class of applications (such as MPI [22] applications) and may not be fit for low-latency PGAS applications.

This is the first study characterizing the recovery techniques of a fault-tolerant interconnection network of a large-scale system. This study outlines the effects of network-related failures and recoveries on system and applications.

9 Conclusions

Using field-failure data, we studied the efficacy of the recovery mechanisms of the Gemini interconnect network deployed on Blue Waters. Our analysis model and characterization help to understand the cause of failure of network-recovery mechanisms and their impact on applications and the system. We show that the most important factors in determining whether a recovery will complete successfully are (1) failures that occur during the recovery, whether they were caused by the same faults that triggered the recovery or by faults that occurred independently of it; (2) the type of the recovery; and (3) the duration of the recovery. Our analysis suggests that data-driven resiliency mechanisms can identify critical events that can cause the failure of the recovery and hence can be useful in detecting the failure of the recovery in advance. Our future work will focus on building detectors and predictors of recovery failure so that appropriate mitigating action at system or application level can be taken.

Acknowledgment

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Number 2015-02674. This work is partially supported by NSF CNS 13-14891, Air Force Research Lab FA8750-11-2-0084, an IBM faculty award, and an unrestricted gift from Infosys Ltd. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Application. We thank Celso Mendes, Gregory Bauer, and Jeremy Enos from NCSA for providing raw data and many insightful conversations. We thank Larry Kaplan for providing Cray-specific information.

References

[1] S. Jha, V. Formicola, Z. Kalbarczyk, C. Di Martino, W. T. Kramer, and R. K. Iyer, "Analysis of gemini interconnect recovery mechanisms: Methods and observations," in *CUG 2016 Conference*, pp. 8–12, Cray User Group, 2016.

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

[3] "Network resiliency of cray xe systems." http://docs.cray.com/books/S-2393-4003//S-2393-4003.pdf, 2013.

[4] D. Jewett, "Integrity s2: A fault-tolerant unix platform," in *Fault-Tolerant Computing, 1991. FTCS-21. Digest of Papers., Twenty-First International Symposium*, pp. 512–519, IEEE, 1991.

[5] M. Harris and D. Luebke, "Gpgpu: General-purpose computation on graphics hardware," in *International Conference on Computer Graphics and Interactive Techniques: ACM SIGGRAPH 2005 Courses: Los Angeles, California*, vol. 2005, 2005.

[6] "http://www.cray.com/Products/Storage/Sonexion/Specifications.aspx."

[7] R. Alverson, D. Roweth, and L. Kaplan, "The gemini system interconnect," in *2010 18th IEEE Symposium on High Performance Interconnects*, pp. 83–87, IEEE, 2010.

[8] "https://wiki.alcf.anl.gov/parts/images/2/2c/Gemini-whitepaper.pdf."

[9] W. J. Dally and B. P. Towles, *Principles and practices of interconnection networks*. Elsevier, 2004.

[10] "https://www.hypertransport.org/."

[11] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Computer Networks (1976)*, vol. 3, no. 4, pp. 267–286, 1979.

[12] P. Mohapatra, "Wormhole routing techniques for directly connected multicomputer systems," *ACM Computing Surveys (CSUR)*, vol. 30, no. 3, pp. 374–410, 1998.

[13] G. Staples, "Torque resource manager," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, p. 8, ACM, 2006.

[14] M. Karo, R. Lagerstrom, M. Kohnke, and C. Albing, "The application level placement scheduler," in *Cray User Group - CUG*, 2008.

[15] C. Di Martino, "One size does not fit all: Clustering supercomputer failures using a multiple time window approach," in *International Supercomputing Conference*, vol. 7905 of *Lecture Notes in Computer Science*, pp. 302–316, Springer Berlin Heidelberg, 2013.

[16] T.-T. Lin and D. Siewiorek, "Error log analysis: statistical modeling and heuristic trend analysis," *Reliability, IEEE Transactions on*, vol. 39, no. 4, pp. 419–432, 1990.

[17] J. B. Leners, H. Wu, W.-L. Hung, M. K. Aguilera, and M. Walfish, "Detecting failures in distributed systems with the falcon spy network," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 279–294, ACM, 2011.

[18] P. J. Braam, "File systems for clusters from a protocol perspective," in *Proceedings of the Second Extreme Linux Topics Workshop, Monterey, CA*, 1999.

[19] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, "Microreboot-a technique for cheap recovery.," in *OSDI*, vol. 4, pp. 31–44, 2004.

[20] K. S. Trivedi, *Probability and statistics with reliability, queuing and computer science applications*. Chichester, UK: John Wiley and Sons Ltd., 2nd edition ed., 2002.

[21] "Imperative recovery." https://wiki.hpdd.intel.com/display/PUB/Imperative+Recovery. Accessed: 2016-10-28.

[22] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the mpi message passing interface standard," *Parallel computing*, vol. 22, no. 6, pp. 789–828, 1996.

[23] L. V. Kale and S. Krishnan, "Charm++: a portable concurrent object oriented system based on c++," in *ACM Sigplan Notices*, vol. 28, pp. 91–108, ACM, 1993.

[24] G. Almasi, "Pgas (partitioned global address space) languages," in *Encyclopedia of Parallel Computing*, pp. 1539–1545, Springer, 2011.

[25] "http://charmplusplus.org/applications/."

[26] G. Zheng, L. Shi, and L. V. Kalé, "Ftc-charm++: an in-memory checkpoint-based fault tolerant runtime for charm++ and mpi," in *Cluster Computing, 2004 IEEE International Conference on*, pp. 93–103, IEEE, 2004.

[27] "Amber." http://ambermd.org/. Accessed: 2016-10-28.

[28] D. Rossetti and S. C. Team, "Gpudirect: Integrating the gpu with a network interface," in *GPU Technology Conference*, 2015.

[29] V. Formicola, et al., "Understanding fault scenarios and impacts through fault injection experiments in cielo," *Cray User Group*, pp. 8–12, 2017.

[30] "http://www.nersc.gov/users/computational-systems/edison/."

[31] 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 *Proc. of 44th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, 2014.

[32] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *Dependable and Secure Computing, IEEE Transactions on*, vol. 7, no. 4, pp. 337–350, 2010.

[33] Y. Liang, A. Sivasubramaniam, J. Moreira, Y. Zhang, R. 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*, pp. 476–485, 2005.

[34] 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*, pp. 575–584, June 2007.

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

[36] E. Meneses, X. Ni, T. Jones, and D. Maxwell, "Analyzing the interplay of failures and workload on a leadership-class supercomputer," *computing*, vol. 2, no. 3, p. 4, 2015.

[37] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward exascale resilience," *International Journal of High Performance Computing Applications*, 2009.

[38] J. Dongarra, et al., "The international exascale software project roadmap," *International Journal of High Performance Computing Applications*, p. 1094342010391989, 2011.

[39] A. Geist and R. Lucas, "Major computer science challenges at exascale," *International Journal of High Performance Computing Applications*, 2009.

[40] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, vol. 13, 2008.

[41] F. Cappello, "Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities," *International Journal of High Performance Computing Applications*, vol. 23, no. 3, pp. 212–226, 2009.

[42] W. J. Dally, L. R. Dennison, D. Harris, K. Kan, and T. Xanthopoulos, "The reliable router: A reliable and high-performance communication substrate for parallel computers," in *International Workshop on Parallel Computer Routing and Communication*, pp. 241–255, Springer, 1994.

[43] W. J. Dally and C. L. Seitz, "Torus routing chip," June 12 1990. US Patent 4,933,933.

[44] D. H. Linder and J. C. Harden, "An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes," *IEEE Transactions on Computers*, vol. 40, no. 1, pp. 2–12, 1991.

[45] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on computers*, vol. 100, no. 5, pp. 547–553, 1987.

[46] R. E. Blahut, *Algebraic codes for data transmission*. Cambridge university press, 2003.

[47] A. A. Chien and J. H. Kim, *Planar-adaptive routing: Low-cost adaptive networks for multiprocessors*, vol. 20. ACM, 1992.

[48] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN systems*, vol. 17, no. 1, pp. 1–14, 1989.

[49] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM computer communication review*, vol. 18, pp. 314–329, ACM, 1988.

[50] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide, "Immunet: A cheap and robust fault-tolerant packet routing mechanism," in *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pp. 198–209, IEEE, 2004.

Saurabh Jha is a Ph.D. student in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He obtained his master's degree in computer science from University of Illinois at Urbana-Champaign in 2016 and bachelor's degree in computer science & engineering from VIT University in 2014.

Valerio Formicola is Research Associate at the University of Illinois at Urbana-Champaign. He earned his master's degree in Telecommunication Engineering in 2008 and a Ph.D. in Information Engineering at the University of Naples, Italy, in 2014. His research interests are the reliability and security of large-scale systems.

Catello Di Martino is Research Member of Technical Staff at Nokia Bell Labs, working on the creation of resilient communication platforms and networks. Across different institutions, he has contributed to research projects related to resiliency in the areas of HPC, cloud computing, SDNs, and Sensor Networks.

Mark Dalton is a system engineer at Cray.

William T. Kramer is the director of deputy project manager for the sustained-petascale Blue Waters project at Illinois' National Center for Supercomputing Applications (NCSA) and the director of the NCSA @Scale Program Office. He was named one of HPCWire's "People to Watch" in 2005 and 2012 and chaired SC05. At NASA Ames, he put the world's first UNIX supercomputer into production.

Zbigniew Kalbarczyk is a Research Professor at the Electrical and Computer Engineering and the Coordinated Science Laboratory of the University of Illinois at Urbana-Champaign. Dr. Kalbarczyks research interests are in the area of design and validation of reliable and secure computing systems.

Ravishankar K. Iyer is the George and Ann Fisher Distinguished Professor of Engineering at the University of Illinois at Urbana-Champaign. He holds appointments in the Department of Electrical and Computer Engineering, the Coordinated Science Laboratory (CSL), and the Department of Computer Science, serves as Chief Scientist of the Information Trust Institute, and is affiliate faculty of the National Center for Supercomputing Applications (NCSA).