Analysis of Gemini Interconnect Recovery Mechanisms: Methods and Observations

Saurabh Jha, Valerio Formicola, Catello Di Martino,
Zbigniew Kalbarczyk, William T. Kramer, Ravishankar K. Iyer
{sjha8, valeform, dimart, kalbarcz, wtkramer, rkiyer }@illinois.edu
University of Illinois, Urbana-Champaign

Abstract—**This paper presents methodology and tools to understand and characterize the recovery mechanisms of the Gemini interconnect system from raw system logs. The tools can assess the impact of these recovery mechanisms on the system and user workloads. The methodology is based on the topology-aware state-machine based clustering algorithm to coalesce the Gemini-related events (i.e., errors, failure and recovery events) into groups. The presented methodology has been used to analyze more than two years of logs from Blue Waters, the 13.1-petaflop Cray hybrid supercomputer at the University of Illinois - National Center for Supercomputing Applications (NCSA).**

Keywords—Networks , Reliability, Fault Tolerance, Fault diagnosis

I. INTRODUCTION

This work is motivated by the failures of the recovery procedures in the Gemini interconnect system during the first two years of operational hours of Blue Waters, the 13.1-petaflop Cray hybrid supercomputer at the University of Illinois, managed by the National Center for Supercomputing Applications (NCSA). The objective is to bridge the gap between theoretical and practical understanding of the failures of the recovery procedures. We have developed models to assist in the analysis of real-case scenarios of recovery procedures, and to assess their impact on the applications and on the system. Such a gap in understanding can be traced back to the underlying assumptions about the failures in the systems. Recovery procedures are built with the assumption of handling only one failure at a time; this works well for small-scale systems, where the chances of concurrent failures is negligible compared to large-scale cases. Indeed, for a large-scale system, the number of observed failures is much higher. This fact increases the chances of (1) multiple recovery procedures running concurrently in the system due to simultaneous failures, (2) failures during recovery procedures, which can lead to unsuccessful recovery and system outage. Even in cases when the recovery operation is successful, applications can fail due to corrupt application states. Currently, recovery-management software cannot ensure survivability/correctness of the applications during the recovery procedures; hence, applications can fail at any stage during the execution of recovery operations.

In this work we present a methodology and tools to facilitate an in-depth analysis and characterization of Gemini interconnect recovery procedures. Our study is based on mining failure data logs, application logs, and human-written failure reports collected from Blue Waters over three years, from January 2013 to March 2015. In particular, our contributions include:

• Development of techniques to extract, track, and cluster recovery procedure events from system logs, and correlate these clusters with application logs and manual failure reports. The implemented data-analysis pipeline extends the LogDiver tool previously presented in [1], [2], and includes: a filter that is able to extract and decode events generated during Gemini recovery procedures; a novel state-aware coalescing algorithm. This algorithm coalesces events based on (1) the system hierarchy, a tree-like structure which captures topological dependencies among system components (e.g., cabinet $\rightarrow$ blade $\rightarrow$ node/ASIC$\rightarrow$ link) and helps track propagation of events across the levels of the system hierarchy, (2) a state machine, which captures sequences of activities (or actions) corresponding to recovery procedures. The coalescing algorithm is able to reconstruct recovery sequences from filtered events. In addition, the analysis pipeline determines the termination status of recovery procedures (successful/fail), matches recovery sequences with system-wide outage (SWO) events from failure reports, and evaluates the impact of recovery procedures and related-SWOs on the applications executed on the compute nodes involved in the recovery operations.

• Demonstration of the proposed methodology in characterizing the recovery procedures of the Gemini interconnect system. Specifically, we provide initial results on (1) distribution of failures/successes of recovery procedures caused by lane, link, and warm swap failures; (2) trends in the rates of recovery procedure failures; and (3) impact of interconnect recovery procedures on applications.

Finally, our methodology and tools can be used to create models and analyze logs different from Gemini, with a reasonable adaptation effort. This makes the proposed approach valuable for any researchers and practitioners that aim at generating system-level models to analyze field data.

II. Related Work

Cray XT and Cray XE6-based systems, such as Blue Waters, use the 3-D torus interconnect technology, as descried in [3], [4]. The recent advancements/changes and a description of the Gemini interconnect technology is provided in [5]. The 3-D torus interconnect design has been chosen by Cray based on performance simulations ([6], [3]) and resiliency requirements at that time. The newer generation of Cray systems, such as the XC40, is installed with the Aries (Dragonfly) network [7]. The exact routing techniques are not publicly avaiable for Cray XE systems; a high-level overview of the routing algorithms and techniques can be found in [4], [5], [8]. *Principles and Practices of Interconnection Networks* [9] contains a detailed study and overview on the various different models and techniques for the interconnect systems.

Most of the work in interconnect system focuses on the design and scalability of the networks such as in [10], [11], [12], [6]. In [13], Ezell presents micro-benchmarks to diagnose problems in HPC systems with the Gemini interconnect, using performance registers. That analysis is performed in unloaded network scenarios. In contrast, we have developed tools to investigate the impact of failover mechanisms on the applications from the real logs of production hours.

III. Overview of Gemini Interconnect

In this section, we describe the main characteristics of the Gemini interconnect system of Blue Waters and the related recovery features. In order to keep in the section comprehensive description of the interconnect system, we have combined some of the exploratory results gained from the analysis of field data, with the information contained in Cray official documentation on the Gemini interconnect resiliency [14] mechanisms. An overview of the Gemini interconnect is provided in [5].

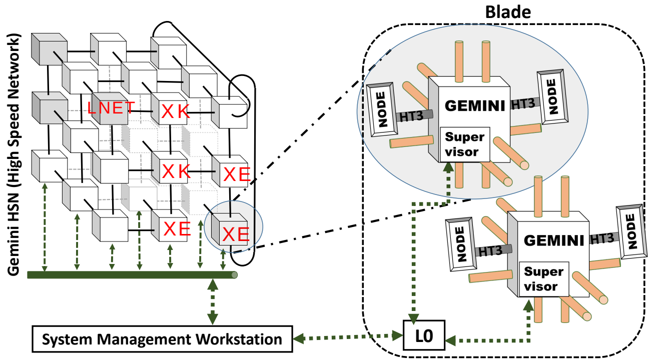

Fig. 1. Blue Waters 3-D Torus Gemini interconnect layout. XE and XK are CPU and GPU nodes respectively where as LNET is Lustre NETwork routers running on top of the Gemini interconnect

A. Architecture Overview

The Blue Waters' high-speed network consists of an anisotropic 3-D torus using Cray Gemini router (see Figure 1) to connect all the nodes in the system. Each blade includes two Gemini application specific integrated circuits (ASICs), each housing two network interface controllers (NICs) and a 48-port router. A NIC is attached to one node using a $HyperTransport^{TM}3$ host interface. Each ASIC is connected to the network by means of 10 torus connections, two each in X+, X-, Z+, Z- and one each in Y+ and Y-. An ASIC also connects internally two nodes using NICs. Each connection is composed of four links and each link is composed of 3 single-bit bidirectional lanes. Thus, each connection consists of 12 lanes, and an ASIC connects to the other ASICs on the network via 24 lanes in the X/Z and 12 in the Y dimension. A channel is a logical connection between two link end-points. Further, each channel comprises two virtual channels to prevent request-response dependency cycle. 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. To avoid these dependencies, the Gemini interconnect system uses packet adaptive virtual cut-through directional ordering routing algorithm. In Gemini, each channel supports two virtual channels (VC0 and VC1). However, in order to avoid both request-response and torus dependency cycle, each channel would need four virtual channels. To address this, Gemini router chips are divided into two groups (CG0 and CG1). These two groups along with two available virtual channels make up for the need for four virtual channels required to avoid request-response dependency cycle (via VC0 and VC1) and torus dependency cycles (via CG0 and CG1).

B. Fault Tolerance and Resiliency

Gemini provides several levels of protection from errors and failures. Packets are protected through 16 bit cyclic redundancy check (CRC) and are checked at each Gemini ASIC (and between the transition from NIC to router as well). Gemini ensures reliable delivery of packets using sliding window protocol. Most of the memory regions are protected via single error correction double error detection (SEC-DED); however, the buffer storing the router tables are not protected by an error-correcting code.

In terms of path availability for successful delivery of packets, there are two redundant connections between any two Gemini ASICs in the X and Z directions whereas only one connection connecting Gemini ASICs in the Y direction. Each of these connections has two redundant links, and each link has three redundant lanes. Gemini interconnect is capable of running in degraded mode as long as there is at least one active link with a minimum of one lane functioning properly. In the general literature, software and hardware designers have used links and channels interchangeably for different purposes with significant differences in meaning. For the purpose of consistency and readability with respect to Cray system logs, we refer to a channel as a logical connection between link end points, and, therefore, use channel and link interchangeably. Thus, in this work, a link down does not necessarily mean that there is no active communication path between two ASICs.

C. Fault Detection and Recovery

Fault detection and recovery of the network is managed through the supervisor block that connects Gemini to an embedded control processor (L0) on the blade. The L0 is connected to system management workstation (SMW) through the Cray Hardware Supervisory System (HSS) network. This is shown in a block diagram in Figure 1. The L0 blade controller detects failed links and power loss to Gemini mezzanine cards (mezzanine is a board on which two ASICs are situated) using the "gmnwd" daemon. System responses to failures are logged (via the "xthwerrlogd" daemon) and orchestrated (via the "xtnlrd" daemon) by the SMW. In this work, we study three specific recovery mechanisms: (1) lane recovery (2) link failover, and (3) warm swap.

1) Lane Recovery: The availability of 3 lanes in each link allows the network to tolerate up to two lane failures and operate in a degraded mode. When all the three lanes fail in a link, the link is marked as inactive and a link failover is triggered. Each time a lane goes down, an error is written in the logs and a lane recovery is triggered by the L0. The controller attempts to recover the lane a certain number of times (as configured by the system administrator) before marking the lane as a permanent failed. No lane recovery is triggered for an inactive link.

A state-transition diagram for the lane recovery is depicted in Figure 2. The two outgoing transitions form the "All Lanes Healthy" state report a lane failure (one or two lanes unavailable) or the whole link failure (three lanes unavailable). Lane mask represents a three-digit bitmap (0 indicates a lane down; 1 indicates a lane healthy). A lane mask 7 means that the lane recovery is successful. Any other values indicate the position of active lanes in a bit mask (e.g., a lane mask of 5 means lanes 1 and 3 are active, whereas lane 2 is inactive).

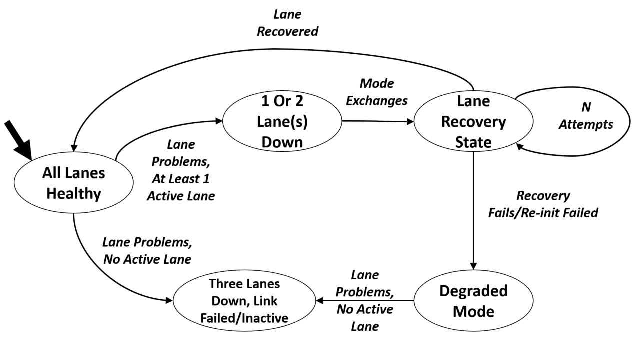
Lane Recovered
1 Or 2 Lane(s) Down
Mode Exchanges
Lane Recovery State
N Attempts
All Lanes Healthy
Lane Problems, At Least 1 Active Lane
Recovery Fails/Re-init Failed
Lane Problems, No Active Lane
Three Lanes Down, Link Failed/Inactive
Lane Problems, No Active Lane
Degraded Mode

Fig. 2. State transition diagram for lane recovery procedure. 3 lane failures (in the same link) result in an inactive link.

2) Link Failover: Figure 3 shows the state-transition diagram for the link failover and warm swap operation of the Gemini interconnect. The failover procedure consists of (1) waiting 10 seconds to aggregate failures, (2) determining which blade(s) is/are alive, (3) quiescing the Gemini network traffic, (4) asserting a new route in the Gemini chips (performed by the SMW), and (5) cleaning up and resuming Gemini. The total time to execute the procedure varies from 30 to 600 seconds. Links can

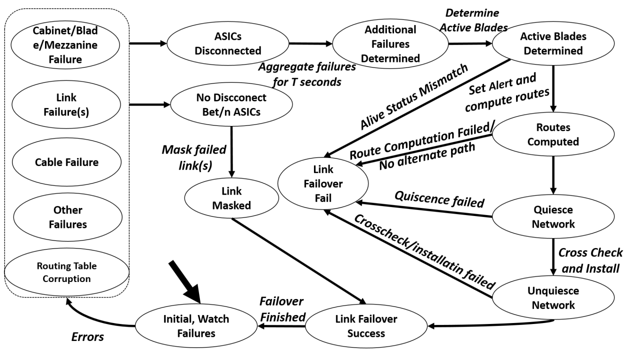

Fig. 3. State transition diagram of Gemini link failover operations.

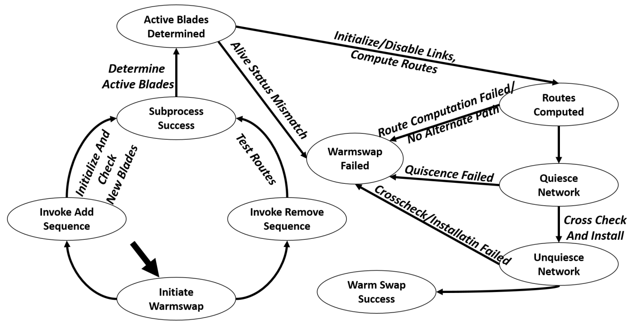

Fig. 4. State transition diagram of Gemini warm swap operation (which is always invoked by a system administrator).

become unavailable due to one of the following reasons:

- All three lanes failed in the link
- Power loss in a mezzanine, blade, or cabinet
- Faulty cable
- Other reasons such as routing table corruption, software deadlocks, etc.

A faulty cable causes 32 link endpoints to become unavailable. Power loss on a mezzanine, blade, and cabinet causes 32, 32, and 960 end points to fail, respectively. Link failover is triggered whenever a link becomes unavailable. The link failover operation masks failed links whenever possible without causing interruption of the network. However, when there is a complete disruption between the communication of two ASICs or a node/blade/cabinet becomes unavailable, the failover mechanism has to quiesce the whole network to install the routes safely. A successful failover restores the communication path in the network and the functioning of the system, whereas a failed failover causes the whole network to completely fail, and leads to system-wide outage. In Figure 7, a state-transition diagram shows the various steps and conditions leading to a successful or failed link failover.

3) Warm swap: Warm swap is the addition or removal (disabling) of compute blade/cabinet in a running system. This operation cannot be performed on service blade/cabinets. A warm swap is invoked by human administrators by logging into the SMW and calling warm swap procedures; hence this procedure is highly controlled. Warm swap procedure is similar to link failover mechanism with certain exceptions. Details are shown in the state-transition diagram for warm swap in Figure 4.

IV. Methodology and Tools

In order to understand and analyze the recovery procedures of the Gemini interconnect, we augmented LogDiver [1] with additional capabilities to implement a recovery analysis workflow. The augmented LogDiver filters, coalesces, and correlates Gemini-related errors with application failures and human-written system failure reports managed by system administrators. This approach helps to determine the recovery type -lane recovery, link failover, warm swap category- and recovery exit status as successful or failed. Coalescence is the crucial step in our work flow and is responsible for:

- recomposing the events from the logs to create recovery-sequence clusters. The clusters contain information about the events that triggered the recovery, steps taken to mitigate failures and anomalies in this sequence. Coalescing is described in detail later in this section. Note that the triggering events do not necessary correspond to the root cause of the failure; rather they are the closest events (in time), since logged immediately prior to the recovery procedure invocation.
- giving a consolidated list of all the components involved in the recovery.
- giving the timing details of the recovery sequence.

In addition to the characterization of the recovery procedures, the recovery-procedure analysis workflow is used to evaluate the impact of recovery (successful/-failed) on user applications and on the system availability.

Our tool takes three inputs (see Table I): (1) syslogs, (2) consolidated application workload (obtained from ALPS and Torque logs), and (3) manual failure reports (created by administrators). The consolidated application workload logs are filtered in order to contain only user applications (i.e., all debug and benchmark jobs run by system administrators are removed from this analysis).

TABLE I. Summary of data sources

Data time span: January 2013-March 2015		
Datasource	***Count***	***Dataset Size***
*Raw syslogs**	75,760,682,632	13 TB
Manual failure reports	4,184	1.4 MB
Coalesced Workload	20,600,030	8 GB

Figure 5 shows the block diagram of the recovery-procedure analysis workflow of our tool. It consists of five operations, indicated by diamond shapes (in Figure. 5): *Filter*, *Coalesce*, *Merge*, *Workload Consolidation*, and *Gather*. *Filter* and *Coalesce* are implemented using the C++ language. The *Filter* takes approximately 15-25 minutes to process 13 TB of syslogs on 815 Blue Waters nodes depending on the I/O and network utilization of Blue Waters. *Coalesce* is currently serial in nature and takes approximately 8 hours to run on a single node. We expect that a parallel version of *Coalesce* runs much faster; we consider the parallel implementation for future work. *Merge* and *Gather* are implemented (in serial) using the Python programming language, and take approximately 1 minute and 20 minutes to execute on a single node, respectively. *Workload Consolidation* was previously implemented in LogDiver and we refer the readers to [1] for more details on this analysis.

In the following, we describe all the operations of the recovery-procedure analysis workflow in details, that have been added to LogDiver (from this point forward, the additional tools for the workflow + LogDiver will be called augmented LogDiver).

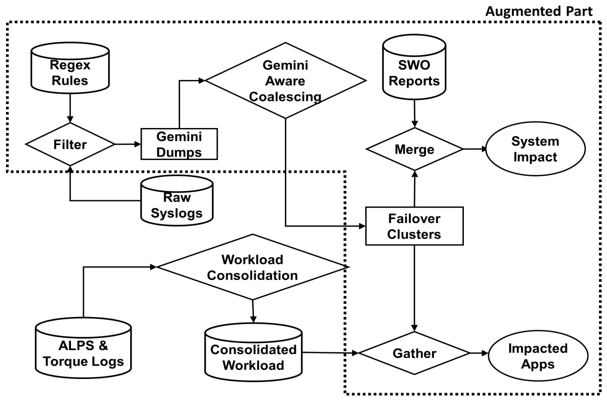

Fig. 5. Block diagram of the pipeline for analyzing Gemini recovery procedures on system and user applications.

A. Filter

Filter operation applies regular expression (regex) rules onto the raw system logs line by line, to transform and tag (i.e., label) these logs into a set of features, represented by $nple < Time, Location, Tag, Info >$. The $Time$ field is the timestamp of the logged event; $Location$ is the component that either suffered from errors/failures, or where the recovery procedure was running; Tag is an identifier of the matched regex rule; $Info$ is used for storing other important information, such as completion time or error exit reason. $Location$ of the log message is determined either through the location field in the logging protocol [15] or from the *Tag* and the body of the message itself. It is important to extract the exact location to evaluate the impact. As an example, below we show a message taken from syslog, which is written by the *xtlnlrd* daemon on the SMW, upon a failure of a Gemini ASIC module.

```
1368343836 local3 5
2013-05-12T02:30:36.909109-05:00
smw xtnlrd 15324 p0-20130503t234552
[hss_nlrd@34] 2013-05-12 02:30:36 smw
15325 cb_hw_error: failed_component
c17-10c1s6g1, type 21, error_code
0x0d10, error_category 0x0002
```

In this example, filter operation determined the $< Time >$ as the unix time - "1368343836", $< Location >$ as "c17-10c1s6g1", which is the failed component, $< TAG >$ as "ASIC_FAILED", and $< Info >$ as " error_code=0x0d10". In this case, the identification label of the failed component is "c17-10c1s6g1", and contains the $< Location >$ of the Gemini ASIC failure event, which is not directly indicated in the syslog header. However, this is more of an exception as in many cases the reporting component can directly be used as $< Location >$.

TABLE II. An example line from rules database

Tag ID	Regex Expression	Tag
2020	Link recovery operation was successful	LINK_RECOVERY_SUCCESS

All the events that indicate Gemini interconnect-related errors, failures, and recovery procedure stages are stored as rules in our database. These events were selected based on our understanding of Gemini recovery procedures, as discussed in section III, and each event is matched to only one rule (in which case it is dropped).

There are over 122 tags in our regex rules database. An example line from our rules database is shown in Table II. These tags are divided into 4 categories: *Trigger-Latent*, *Trigger-Immediate*, *Recovery Transition*, and *Recovery Finish*. Table III gives representative examples of tags for each of the four broader categories.

These four tag categories are created based on the general understanding of the *failure manifestation and propagation* in the system. The problem starts with faults, which manifest as error in the system. Errors lead to failure(s) either immediately or some time later. A failure invokes a recovery mechanism, which can either success or fail. The details of these categories are as follows —

• *Trigger-Latent* —These tags represent errors which can lead to failure, or are indicative of failures that have not been detected yet. Since these tags do not cause immediate failure of the link/lane, they do not trigger any recovery procedures. However, some of these events result in disabling of lanes/links, which in turn generates new events (belonging to *Trigger-Immediate* category). For example, when a single lane has more errors than its companion lanes, it is deactivated leading to 'Lane Down 'event in the system.

• *Trigger-Immediate*—These tags indicate an activity in the system that modifies or affects the network topology, i.e., link addition, link disable, link unavailability, thus causing the invocation of recovery procedures to handle these changes. For example, "ASIC_FAILED" indicates a failure of Gemini ASIC on a blade, causing the links to be unavailable and making the network unroutable. In almost all cases, when events from this category are observed, recovery procedures are expected to handle these events.

• *Recovery start and Transition Step* —These tags indicate the starting point and the intermediate steps of the recovery procedures, and, hence help to keep track of the system actions taken for failure mitigation. Also, this data helps to understand if any other failures in the system interfere with the recovery procedure. For example, "LINK_AGG_FAILURES" tag indicates that the recovery procedure is waiting and collecting any additional failures for T seconds before calculating new routes for the network.

• *Recovery Final* —These tags include all the states that indicate the final state (success/fail/missing) of the recovery procedure. For example, "LINK_FAILOVER_SUCCESS" indicates that the link failover operation finished successfully.

TABLE III. TAG CATEGORIES WITH REPRESENTATIVE TAG EXAMPLES AND RELATED COUNTS, AS OBSERVED IN THE LOGS

TRIGGER-LATENT		**RECOVERY TRANSITION**	
BLADE_ELECTRICAL_ISSUE	2.24E+07	BLADE_RECOVERY	2.79E+02
CABINET_HEARTBEAT_FAILED	5.37E+05	BLADE_RECOVERY_SUCCESS	2.79E+02
GEMINI_BUFFER_OVERFLOW	4.90E+07	CABINET_READDED	7.11E+06
GEMINI_CHECKSUM_ERROR	8.64E+03	FINISHED_LINK_RECOVERY	3.72E+02
GEMINI_ECC_ERROR	1.16E+06	GEMINI_TIMEOUT	4.50E+01
GEMINI_MISROUTED_PACKET	5.06E+08	HSN_NETWORK_QUISCED	4.67E+02
GEMINI_PROTOCOL_ERROR	8.19E+08	HSN_NETWORK_UNIQUISCE	9.33E+02
HWERR_B2B	2.68E+09	HSN_WAIT_NETWORK_DRAIN	4.67E+02
HWERR_CMD_MISMATCH	1.00E+03	HW_ERR_LINKF_IDENTIFIER	3.04E+02
HWERR_MISROUTE_PACKET	4.05E+05	INIT_NEW_BLADES	2.78E+02
HWERR_NIF_SQUASHED_REQ	7.17E+07	INIT_NEW_LINKS	4.04E+02
SSID_RESP_PROT_ERROR	2.58E+07	LINK_FAILED_HANDLED	4.50E+05
LINK_INACTIVE	4.39E+05	LINK_AGG_FAILURES	4.24E+02
ONE_LANE_DOWN	3.50E+07	NETWORK_QUISCE	9.19E+02
RX_VC_DESC_INV	7.22E+05	NETWORK_UNQUISCE	9.15E+02
SSID_UNEXPECTEDRSPSSID	2.14E+05	REROUTE_SUCCESS	1.68E+03
TWO_LANE_DOWN	5.69E+05	ROUTE_COMPUTE	1.05E+03
		ROUTING_RETRY	1.09E+02
TRIGGER-IMMEDIATE		STARTED_LINK_RECOVERY	2.51E+05
ASIC_FAILED	1.85E+04	WARM_SWAP_FINISH_TIME	7.02E+02
BLADE_DOWN_DETECTED	3.73E+02	WARM_SWAP_STARTED	7.04E+02
EPO_FAULT	1.60E+03	**RECOVERY FINISH**	
FAN_FAULT	1.27E+03	LINK_RECOVERY_FAILED	9.60E+01
LINK_FAILED	2.51E+05	LANE_RECOVERY_FAILED	2.85E+03
MEZZANINE_POWER_FAILED	5.18E+04	LINK_RECOVERY_FAILED	9.60E+01
NODE_DOWN	1.95E+06	LINK_RECOVERY_SUCCESS	3.22E+02
ROUTING_TABLE_CORRUPTION	1.17E+02	WARM_SWAP_DELAYED	4.00E+00
THREE_LANE_DOWN	5.76E+05	WARM_SWAP_FAILED	5.10E+01
NODE_UP	3.93E+05	WARM_SWAP_SUCCESS	6.51E+02

In a complex system like Blue Waters, it is hard to build (and then track) deterministic model of the state transitions (described in section III) during recovery due to the following reasons —

• *Failure during recovery procedure:* A failure during a recovery procedure can alter the state, and, hence the state transition of the recovery path, depending on the type of failure. This failure acts as interference, and thus, alters the normal recovery path. It is impossible to know all the transition paths in advance due to - (1) unavailability of the underlying code, and (2) non-determinism in the system.

• *Issue of bias versus variance in modelling*: Explicitly coding all the recovery paths into the model increases the model complexity, and hence, can lead to a high bias in the model. This hinders the discovery of true causes of the failures.

• *Logging issues*: The timestamps stored in the logs represent the logging time of an event. Sometimes, there can be incoherency in the logging as the different events are logged by different subsystems, and writes (to the log file) may not follow a strict order. An event can be completely missed when the system is heavily stressed, e.g., the logging service itself is down, the memory is corrupted, or the network is unavailable.

To cope with these problems and represent all recovery procedures using a general model (i.e. a common state-transition diagram) of *failure manifestation and* *propagation* in the system, the state-transition diagrams described in section III are mapped injectively onto a reduced state-transition diagram depicted in Figure 7. This abstraction does not only helps us cope with the problems discussed above, but also reduces the time to do our analysis as there are many fewer states to track.

B. Coalesce

The coalescing algorithm is executed on the output of *Filter*. The algorithm shown in *Algorithm 1* creates clusters of events (corresponding to messages logged) to form Gemini recovery-sequences. The algorithm coalesces tags based on the fixed sliding window algorithm proposed in [16], [17]. The algorithm starts by initializing an empty tree. The tree is essentially based on a fault tree model (see Figure 6), which captures the topology of the system and, hence, potential error propagation paths in the Gemini interconnect. The idea of using this model lies in the fact that an event at higher level in the topology affects all the sub-levels, and hence, all events that occurred in the lower levels during same time window (as the high-level event) need to be merged. Similarly, an event at lower level can affect components located at higher levels in this tree. Thus, the fault tree model effectively captures Gemini recovery events, as well as it provides an effective way to index the clusters (generated by the coalescing algorithm) and components of the system. Such indexing speeds up the processing time, by at least an order of magnitude.

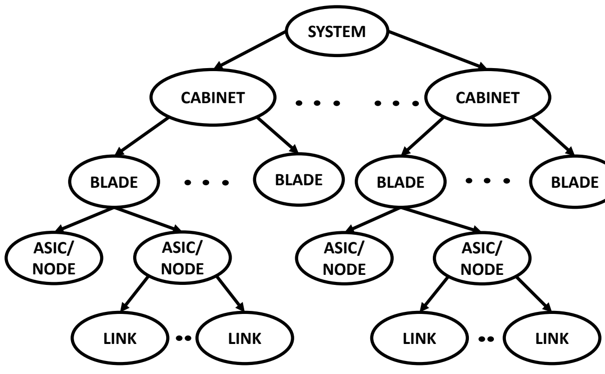

Fig. 6. Gemini topology-aware fault model. The model helps track the effects of events in sub-components.

The algorithm first tries to find the index where log_line (refer to Algorithm 1) can be inserted in the cluster tree. If no index is found, a new leaf is created at the appropriate level by traversing the tree. If the log_line is inserted at a level that has a sub-tree, all the clusters in the sub-tree are merged with the cluster of the current level, as required by our fault tree model described earlier. For example, if the log_line belongs to the blade-4 level, all the clusters (if any) below this level are merged with this cluster. When the cluster is complete, the final state of the sequence is decided using the state-transition diagram shown in Figure 7. The algorithmic complexity of the outlined model is *O(n log(m))*, where *n* is the number of lines and *m* is the system size determined by the total number of unique components (e.g., links, ASICs, blades, cabinets, etc).

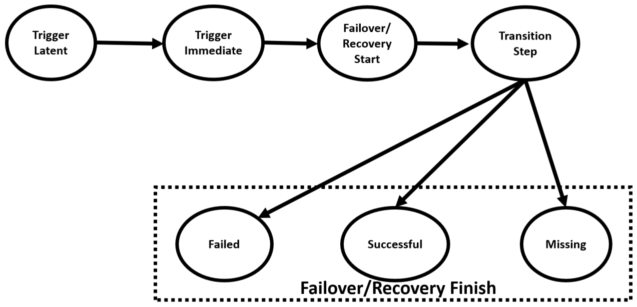

Fig. 7. Reduced recovery-sequence state-transition diagram. All state-transition diagrams described in Section III are mapped to this reduced state-transition diagram.

Algorithm 1 Coalescence of Gemini recovery procedures related events.

```
1:  $Tree = initialize\_cluster\_tree()$
2:  for $log\_line$ in filtered logs do
3:      $current\_cluster = find\_cluster\_index(Tree, log\_line.location)$
4:      if $((current\_cluster.timestamp - log\_line.timestamp) \leq slide\_time)$ && $(current\_cluster \neq NULL)$ then
5:          $current\_cluster.timestamp = log\_line.timestamp$
6:          $current\_cluster.add(log\_line)$
7:          $merge\_all\_clustrs\_in\_subtree(current\_cluster)$
8:      else
9:          if $(current\_cluster \neq NULL)$ then
10:             $current\_cluster.states = determine\_cluster\_states(current\_cluster)$
11:             $write(current\_cluster)$
12:         end if
13:         $start\_new\_cluster(Tree, log\_line.location)$
                ▷ Start new cluster with log_line being it's first member at the appropriate location in tree
14:     end if
15: end for
16: write_active_clusters(Tree)                ▷ Write all active clusters
```

Table IV provides an example output of *coalescing*. This example shows a successful link failover operation, as indicated by *Recovery Status*. The problem starts with the corruption of routing tables in one of the ASICs as indicated *First Tag*. This results in failure of links, and hence, triggering of the failover. The *End Tag* indicates the unquiesce of the node. Although the failover ran for 120.09 seconds (indicated by *Additional Info*), the time,

TABLE IV. EXAMPLE OF RECOVERY SEQUENCE CLUSTER(OUTPUT OF COALESCING ALGORITHM)

Start Time [Unix Time]	1431583171
End Time [Unix Time]	1431583401
Locations	blade_c11-8c1s1, blade_c14-10c0s3, smw1, blade_c9-8c1s3, blade_c14-3c0s1, blade_c13-6c2s0, blade_c19-0c2s3, blade_c1-9c2s5, blade_c12-7c0s4, blade_c16-1c2s7, asic_c11-8c1s1g0, blade_c21-0c0s4
First Tag	CORRUPT_ROUTING_TABLES
End Tag	LINK_RECOVERY_SUCCESS
Tag Counts	ROUTING_TABLE_CORRUPTION[1], ASIC_FAILED[1], CABINET_READDED[6912], ROUTE_COMPUTE[1], NETWORK_QUIESCE[1], NETWORK_UNQUIESCE[1], FINISHED_LINK_RECOVERY[1], LINK_FAILOVER_SUCCESS[1],
Total Events	6919
Duration [Seconds]	230
Recovery Status	SUCCESS
Additional Info	RECOVERY_HANDLING_TIME= 120.09

from the beginning of the first event to the last event, in the cluster was 230 seconds. When the $< Location >$ tag contains $smw1$ in the output, it means that the whole system is affected. In particular, the time between quiescence to unquiescence stops the traffic in whole system.

C. Merge

The *Merge* operation processes SWO failure reports filed by the technical staff of Blue Waters/Cray, and correlates them with *recovery-sequence clusters* obtained from *coalescing*. This operation looks at the overlapping time between recovery-sequence clusters and SWO duration mentioned in the reports. This merging does not necessarily indicates causation. For causality inference, we manually verified the merged output. This was a feasible task due to the small number of SWOs (approximately ~ 101 in this study) that needed to be analyzed manually.

D. Gather

The *Gather* operation is related to the evaluation of applications affected by successful and failed recovery procedures. It finds applications -from the database of consolidated workload- that overlap in time and location with any of the recovery-sequence clusters. The final output of this operation is a statistical characterization of affected applications. It provides the number of applications running during the time window of the recovery-sequence clusters and applications terminated during this time window; further it extracts the the number of applications terminated with success, and the number of applications terminated with failing status, both system-wide and on the nodes involved in the recovery operations. Finally, the tool provides the scale of the applications executed on the nodes.

V. RESULTS

This section shows our preliminary results from the analysis of the outputs obtained from the methodology and tools, as described in section IV. We discuss completion status and failure rates of recovery procedures, and their impact on the system and applications.

A. Completion Status of Recovery Procedures

Figure 9 shows the breakdown in percentage of successful lane recovery, link failover, and warm swap procedures. The Gemini interconnect is highly resilient to lane failures. Specifically, 99.1% of lane failures are successfully recovered. Moreover, the impact of lane recovery failures is very limited since a disabled lane does not cause link failure, as explained in section III (except for three lanes down). Since lane recovery operations are managed by the L0, they may potentially interfere with other recovery procedures managed by the SMW through the L0, such as link failovers and warm swaps. For example, after analyzing recovery-sequence cluster summaries generated by our tool, we found out that in one case, the failure of the lane recovery negatively affected a link failover procedure running in the system. This resulted in a high-speed network deadlock and SWO. There are other cases where such synchronization issues lead to failure of Gemini recovery procedures.

Link failovers are successful only in 75.8% of the cases. Although warm-swap operations are highly controlled (since they are triggered by system administrators), around 7.9% of warm swaps fail. Apart from maintenance reasons, warm swaps are initiated when the network becomes unroutable. By analyzing recovery-procedure clusters, we observe that most failures in the link failovers and warm swaps are caused by other simultaneous failures in the system. These simultaneous failures could not be managed by the SMW and, in the end, resulted in a SWO.

B. Recovery Procedure Failure Rate

Figure 8 shows the total count of failures of recovery procedures per month in Blue Waters for (a) lane, (b) link, and (c) warm swap. Failures of recovery procedures

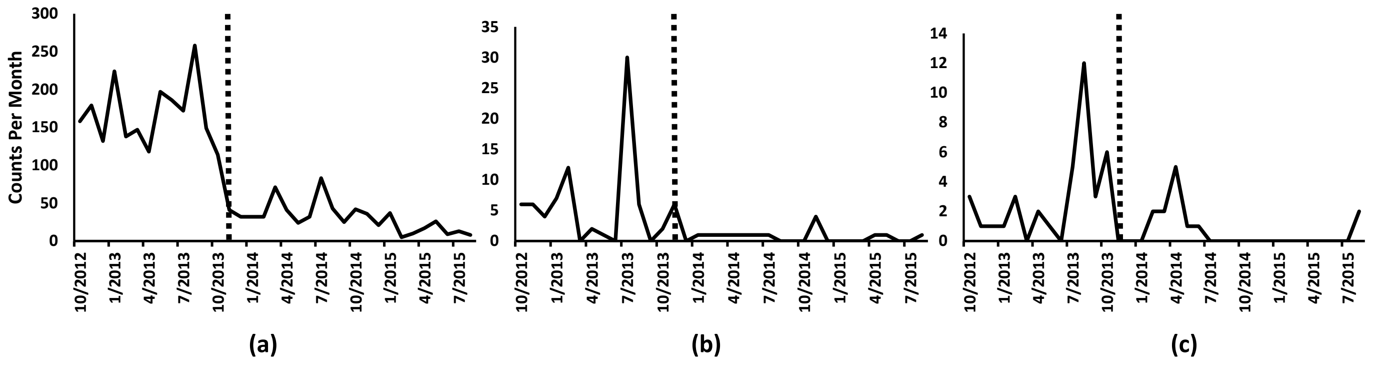

Fig. 8. Count of total failure of recovery procedures for (a) lane (b) link (c) warm swap per month. The vertical line shows a major software upgrade fixing bugs in the Gemini interconnect resiliency management code.

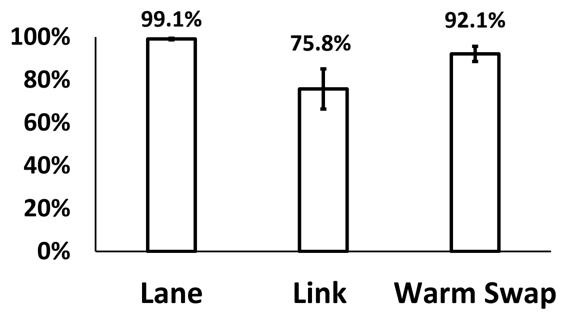

Fig. 9. Recovery completion status for lane, link, and warm swap.

have decreased over time. There is a sharp decrease in failures of recovery procedures after November 1, 2013. On that day, Blue Waters SMW software was updated and further patched (a day later) to fix major software bugs related to handling failures in the Gemini interconnect. Beyond this point, the recovery procedures failure had been continuously decreasing over time, thus showing the increase in stability of the system.

C. System-Wide Outages

Of 101 SWOs observed in human-written reports filed by NCSA Blue Waters staff, the *Merge* operation (see Figure 5) identified 28 SWOs related to Gemini interconnect recovery procedures. Of these 28 cases, 14 of these are strictly due to failure of the recovery procedures in the Gemini interconnect.

D. Application Impact

Using the methodology in section IV, we have analyzed the impact of interconnect-related recovery procedures on running applications. The impact was assessed in terms of percentage of applications that failed during the recovery-sequence clusters. We calculated this metric for the 28 SWOs and found that 20.13% of the applications failed during these SWOs. Additionally, we found that 0.20% of the applications failed during successful recoveries. This shows that successful recovery can also lead to failure of applications due to the delay in handling the failures. A successful recovery can last between 60 to 1000 of seconds.

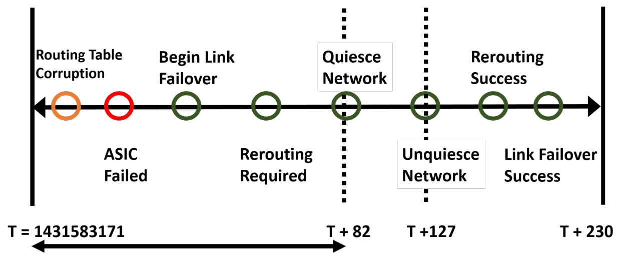

Fig. 10. An example of a successful link failover sequence (taken from the output of the coalescing algorithm). Orange circle represents faults, red circle represents failures, and green circles represents recovery operations.

In Figure 10, we show an example of a link failover sequence that ended successfully. A fault (Routing Table Corruption) occurred at time T. A short time later, the failure (ASIC Failed) occurred, and the failover operation (Begin Link Failover) was triggered. The failover procedure determined the need for calculating new routes to handle this failure (Rerouting required), and calculated new routes to be installed on the routers. Finally, at time $T+82$ seconds, the network was quiesced, and the routes were installed. Once the new routes were installed and asserted, the network operation was fully restored. In this scenario, in the time interval from T to $T+82$ seconds, the network was still active (quiesce had not yet begun), and hence, there was a chance (in this 82 seconds) for the failure to propagate across the system, and amplify the impact of the initial fault (the Routing Table Corruption) on the system and applications. At $T + 82$ seconds, network was quiesced to install the calculated routes. However, quiescence does not guarantee protection from application failures and hence, the application can fail due to - (1) lag in propagation of quiescence in the network, and (2) packet loss. From $T + 127$ seconds to $T + 132$ seconds, applications that do not employ work-load redistribution or check-pointing could potentially fail due to unavailability of the nodes, blades, or cabinets. As shown through this example, the failure containment by recovery procedures does not always hold in practice. Hence, applications can fail even during a successful recovery for different reasons, as discussed above.

E. Validation on Mutrino Dataset

The methodology and tools presented in this paper works out of the box for Cray Aries interconnect. We ran the augmented LogDiver on 'Mutrino dataset '[18] for testing and validating our models. Mutrino is a Cray XC40 based testbed system consisting of 100 nodes connected through Aries interconnect for testing readiness of applications on Trinity Supercomputer. The dataset consists of 100 days of logs. We were able to extract and report the Aries errors from this dataset.

VI. Conclusion and Future Work

In this work, we presented a study to understand the Gemini interconnect operations, and, in particular, the steps of the recovery procedures. Based on this understanding, we proposed and implemented methodology and tools to extract and reconstruct recovery sequences in order to measure system and application resiliency.

In the future, we plan to take steps to delve deeper into understanding the root cause of the failure of the recovery procedures, and have an elaborate discussion on the impact of these recovery procedures on applications and systems. We also plan to compare the resiliency of the Gemini and Aries interconnects in two large-scale systems using the developed methodologies.

VII. Acknowledgement

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, 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 Mark Dalton (Cray), Larry Kaplan (Cray), Gregory Bauer (NCSA), Timothy Bouvet (NCSA), Jeremy Enos (NCSA), Joseph Fullop (NCSA), and Sharif Islam (NCSA) for providing us the raw data and having many insightful conversations.

References

[1] Catello Di Martino, Saurabh Jha, William Kramer, Zbigniew Kalbarczyk, and Ravishankar 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*, pages 11–18. ACM, 2015.

[2] 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*, pages 25–36, June 2015.

[3] Richard E Kessler and James L Schwarzmeier. Cray t3d: A new dimension for cray research. In *Compcon Spring'93, Digest of Papers.*, pages 176–182. IEEE, 1993.

[4] Steven L Scott et al. The cray t3e network: adaptive routing in a high performance 3d torus. 1996.

[5] Robert Alverson, Duncan Roweth, and Larry Kaplan. The gemini system interconnect. In *2010 18th IEEE Symposium on High Performance Interconnects*, pages 83–87. IEEE, 2010.

[6] William J Dally. Performance analysis of k-ary n-cube interconnection networks. *Computers, IEEE Transactions on*, 39(6):775–785, 1990.

[7] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, James Reinhard, et al. Cray cascade: a scalable hpc system based on a dragonfly network. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 103. IEEE Computer Society Press, 2012.

[8] Parviz Kermani and Leonard Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks (1976)*, 3(4):267–286, 1979.

[9] William James Dally and Brian Patrick Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[10] Narasimha R Adiga, Matthias A Blumrich, Dong Chen, Paul Coteus, et al. Blue gene/l torus interconnection network. *IBM Journal of Research and Development*, 49(2/3):265, 2005.

[11] Ron Brightwell, Kevin Pedretti, and Keith D Underwood. Initial performance evaluation of the cray seastar interconnect. In *High Performance Interconnects, 2005. Proceedings. 13th Symposium on*, pages 51–57. IEEE, 2005.

[12] M Blumrich, Dong Chen, Paul Coteus, Alan Gara, Mark Giampapa, Philip Heidelberger, Sarabjeet Singh, B Steinmacher-Burow, Todd Takken, and P Vranas. Design and analysis of

the bluegene/l torus interconnection network. Technical report, IBM Research Report RC23025 (W0312-022), 2003.

[13] Matt Ezell. Understanding the impact of interconnect failures on system operation. In *Proceedings of Cray User Group Conference (CUG 2013)*, 2013.

[14] *Network Resiliency of Cray XE Systems*. Cray.

[15] Rainer Gerhards. The syslog protocol. 2009.

[16] R.K. Iyer, L.T. Young, and P.V.K. Iyer. Automatic recognition of intermittent failures: an experimental study of field data. *Computers, IEEE Transactions on*, 39(4):525–537, 1990.

[17] J. P. Hansen and Siewiorek D. P. Models for time coalescence in event logs. In *Proc. of 1992 Fault Tolerant Computing FTCS 92*, pages 221–227, 1992.

[18] A. Gentile J. Brandt and J. Repik. Mutrino Dataset 2/15-5/15. SAND2016-2449 O. 2016. *Accessed: 2016-03-23*.