ANALYSIS OF GEMINI INTERCONNECT RECOVERY MECHANISMS: METHODS AND OBSERVATIONS

BY

SAURABH JHA

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Adviser:

Professor Ravishankar K. Iyer

ABSTRACT

This thesis focuses on the resilience of network components, and recovery capabilities of extreme-scale high-performance computing (HPC) systems, specifically petaflop-level supercomputers, aimed at solving complex science, engineering, and business problems that require high bandwidth, enhanced networking, and high compute capabilities. The resilience of the network is critical for ensuring successful execution of the applications and overall system availability. Failure of interconnect components such as links, routers, power supply, etc. pose a threat to the resilience of the interconnect network, causing application failures and, in the worst case, system-wide failure. An extreme-scale system is designed to manage these failures and automatically recover from such failures to ensure successful application execution and avoid system-wide failure. Thus, in this thesis, we characterize the success probability of the recovery procedures as well as the impact of the recovery procedures on the applications. We developed an interconnect recovery mechanisms analysis tool (I-RAT), a plugin built on top of LogDiver [1] to characterize and assess the impact of recovery mechanisms. The tool was used to analyze more than two years of network/system logs from Blue Waters, a supercomputer operated by the NCSA at the University of Illinois. Our analyses show that recovery mechanisms are frequently triggered (in as little as 36 hours for link failovers) that can fail with relatively high probability (as much as 0.25 for link failover). Furthermore, the analyses show that system resilience does not equate to application resilience since executing applications can fail with non-negligible probability during (or just after) a successful recovery. Our analyses show that interconnect recovery mechanisms are frequently triggered (the mean time between triggers is as short as 36 hours for link failovers), and the initiated recovery fails with relatively high probability (as much as 0.25 for link failover). We also show that as many as 20% of the executing applications fail during the recovery phase.

To my family and friends, for their love and support.

ACKNOWLEDGMENTS

Foremost, I would like to thank my adviser Prof. Ravishankar K. Iyer and Prof. Zbigniew Kalbarczyk for their guidance and input throughout the completion of this work. It is their confidence that kept me motivated and determined to complete this thesis. My special thanks to Dr. Catello Di Martino and Dr. Valerio Formicola for their contribution in this research, for developing my research skills, and for their continued support beyond the scope of this research. This work would not have been possible without the help and support from Prof.William Kramer (Director of the Blue Waters Project and NCSAs @Scale Programs and professor at UIUC), Mark Dalton (Cray), Joshi Fullop (NCSA), Jeremy Enos (NCSA), Gregory Bauer (NCSA), Timothy Bouvet (NCSA), Sharif Islam (NCSA) and Larry Kaplan (Cray). I would like to thank my peers in DEPEND lab for their insightful and useful discussions.

This work is partially supported by NSF CNS 13-14891, NSF award number 1513051, DOE award number 2015-02674, 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.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ALPS	Application Level Placement Scheduler
ASIC	Application Specific Integrated Circuit
BER	Bit Error Rate
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CUDA	Compute Unified Device Architecture
ECC	Error-correcting Codes
GPU	Graphics Processing Unit
HPC	High-Performance Computing
HSN	High Speed Network
MNBF	Mean Node-hours Between Failure
MTBE	Mean Time Between Error
MTBF	Mean Time Between Failure
MTTR	Mean Time To Repair
NCSA	National Center for Supercomputing Applications
SDB	System Database
SECDED	Single Error Correction Double Error Detection
SWO	System-wide Outage

CHAPTER 1

INTRODUCTION

1.1 Motivation

This work was motivated by the failures of the recovery procedures observed in the high-performance interconnection network 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). A high-performance interconnection network is a key infrastructure providing communication paths for interprocess communication and file accessibility in a supercomputer. A high-performance computing (HPC) interconnect must meet the following needs – (1) high performance (low latency and high bandwidth to ensure application scalability [3, 4]), and (2) resilience (ability to continuously provide services without disruption in the face of failures via quick recovery and failure containment [5]). The research on interconnects has primarily focused on the performance and reliable delivery of messages/data transmitted over the network. However, the research on the resiliency of the network infrastructure itself, specifically building recovery procedures for large-scale systems is limited to handling one failure at a time disregarding concurrent failures and other system activities occurring in the system during the recovery time. In the past, the resilience of network infrastructure was improved through methods such as link redundancy, data protection (checksum, ECC), rerouting and link retransmission. This approach works well for a relatively small-scale system as the time to recover/switch to redundant components is negligible. However, with the growing system scale, the mean time between failures (MTBF) of components has decreased leading to the increase in triggers of recovery procedures and the manifestation of complex recovery scenarios (e.g., concurrent execution of multiple recovery procedures). Such

behavior has resulted in new challenges such as 1) the increase in duration of the recovery (due to increase in convergence time of the routing algorithm and additional checking to ensure correctness of the system state) and 2) arbitration and management of concurrently executing recovery procedures that can interfere with each other. These new challenges limit interconnect resiliency scaling and potentially can affect the resilience of future extreme-scale systems [6]. However, to understand the severity of these challenges for future systems, we need to study and characterize the impact of interconnect failures and recoveries on system and applications for current systems and model these failures for future generations of supercomputers. We developed interconnect recovery mechanisms analysis tool (I-RAT), a plugin built on top of LogDiver[1] to characterize and assess the impact of recovery mechanisms. The tool was used to analyze more than two years of network/system logs from Blue Waters a supercomputer operated by NCSA. Our study is based on mining failure data logs, application logs, and human- written failure reports collected from Blue Waters over two years, from January 2013 to March 2015.

1.2 Contribution and Results

Key contributions of this thesis are:

- 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. I-RAT extends the LogDiver tool previously presented in [1, 7], and includes a filter that is able to extract and decode events generated during Gemini recovery procedures, and a novel state-aware coalescing algorithm. This algorithm coalesces events based on: (1) the system hierarchy, a tree-like structure that captures topological dependencies among system components (e.g., cabinet $\rightarrow$ blade $\rightarrow$ node/ASIC $\rightarrow$ link) and helps track event propagation across the levels of the system hierarchy; and (2) a state machine, which captures sequences of activities (or actions) corresponding to recovery procedures. The coalescing algorithm reconstructs recovery sequences from filtered events. In addition, the analysis pipeline: (1) determines the termination status of recovery procedures (successful/fail), (2) matches

recovery sequences with system-wide outage (SWO) events from failure reports, and (3) 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 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) the impact of interconnect recovery procedures on applications.

- This thesis provides the following crucial findings :

    1. Showcasing that failure of recovery mechanisms lead to a significant number of SWOs and application failures. Despite having various levels of hardware redundancy and error-protection mechanisms, Blue Waters suered from 28 SWOs due to interconnect failures during two years of production. In our study, we found that interconnect-related problems caused SWOs only once due to unrouteable topology configuration.

    2. Our analyses provide evidence that applications can fail during successful execution of recovery. Statistically, on Blue Waters 20.13% of applications failed during failed execution of recovery and 0.20% of the applications failed during successful execution of recovery.

    3. The analyses further show that the mean time between trigger of recovery events (and hence, MTBF) for interconnect-network components are either comparable or much lower than that for other system components such as nodes, memory, etc. For example, Martino et al. [8] showed the MTBF of node failures to be approximately 6.7 hours compared with 2.8 minutes for lane recovery, 36.2 hours for link recovery, and 20.6 hours for warm swap. A failure of a node only affects the application running on the system. However, failure of a link and its recovery affects the whole system. This is a serious issue for overall reliability of the

system due to the low success probability associated with the recovery of interconnect-network failures. For example, the success probability of link recovery is only 75.8%.

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 researchers and practitioners that aim to generate system-level models to analyze field data.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 provides the details of the Blue Waters system. Chapter 3 provides background on interconnects and a detailed overview of the Gemini interconnect. Chapter 4 defines a list of technical words used in this thesis. Chapter 5 describes models, methodology, and tools developed to analyze and understand the impact of recoveries on applications and systems using field data. Chapter 6 describes the results obtained from the analysis of field data. Chapter 7 describes the related work. Finally, chapter 8 presents the conclusions drawn from the thesis and discusses the future directions of this work.

1.4 Related publications

Some of the work presented in this dissertation has previously been published in various venues. The relevant publications are:

FTXS15 Martino, Catello Di, 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, pp. 11-18. ACM, 2015.

CUG16 Jha, Saurabh, Valerio Formicola, Z. Kalbarczyk, C. Di Martino, William T. Kramer, and Ravishankar K. Iyer. "Analysis of Gemini Interconnect Recovery Mechanisms: Methods and Observations." Cray User Group: 8-12.

CHAPTER 2

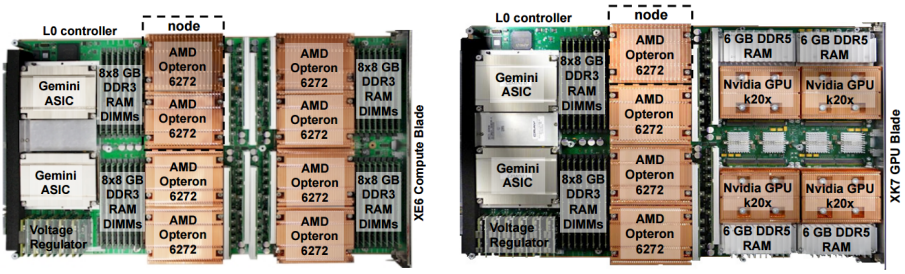
BLUE WATERS SYSTEM DETAILS

In this chapter, we will describe the Blue Waters system architecture, job submission and scheduling, and error logging details. All our insights and data are based on the data collected from Blue Waters.

2.1 Blue Waters System Architecture

Blue Waters is a sustained petaflop system capable of delivering approximately 13.34 petaflops (at peak) for a range of real-world scientific and engineering applications. The system is equipped with —

- 288 Cray liquid-cooled cabinets hosting 26,496 nodes and 1.66 PB of RAM. Each cabinet consists of an L1 cabinet controller, several fan trays, power conversion electronics, breakers, a blower and chiller, and related piping. Each cabinet is organized in 3 chassis, and each chassis hosts 8 blades
- 26,496 compute nodes (based on AMD Opteron processors) with a total of 362,240 cores
- 4,228 GPU hybrid nodes equipped with Nvidia K20X GPU accelerators and AMD Opteron processors with 33,792 cores
- 672 service nodes
- The high-speed Cray Gemini network, to provide node connectivity
- The online storage system, consisting of 198 Cray Sonexion 1600 storage units equipped with 20,196 disks, and 396 SSDs (used to store file system metadata) that provide access to 26 petabytes (36 raw) of usable storage over a Lustre distributed file system



(a) Blue Waters XE Blade (b) Blue Waters XK Blade

Figure 2.1: Blue print of Blue Waters blades

- 300 petabytes (380 raw) of usable near-line tape storage.

Compute node hardware Compute nodes are hosted in 5,660 Cray XE6 blades (see Figure 2.1a), 4 nodes per blade. A compute node consists of 2 16-core AMD Opteron 6276 processors [9] at 2.6 GHz. Each Opteron includes 8 dual-core AMD Bulldozer modules. Each compute node is equipped with 64 GB of DDR3 RAM in 8 GB DIMMs. System memory is protected with x8 Chipkill [9, 10] code. XE node specifications are summarized in Table 2.1.

AMD 6276 Interlagos Processors	2
Bulldozer Cores	16
Integer Scheduling Units	32
Memory / Bulldozer Core	4 GB
Total Node Memory	64 GB
Peak Performance	313.6 GF
Memory Bandwidth	102.4 GB/s

Table 2.1: XE Node Specs

GPU node hardware GPU nodes are hosted in 768 Cray XK7 blades, 4 nodes per blade (see Figure 2.1b). A GPU node consists of a 16-cores Opteron 6272 processor equipped with 32 GB of DDR3 RAM in 8 GB DIMMs and a Nvidia K20X accelerator. The accelerators are equipped with 2,880 single-precision CUDA cores and 6 GB of DDR5 RAM memory with the latter protected with ECC. XK node specifications are summarized in Table 2.2.

AMD 6276 Interlagos Processors	1
Bulldozer Cores	8
Integer Scheduling Units	16
Memory / Bulldozer Core	4 GB
Node System Memory	32 GB
GPU Memory	6 GB
Peak CPU Performance	156.8 GF
CPU Memory Bandwidth	51.2 GB/s
CUDA cores	2688
Peak GPU Performance (DP)	1.31 TF
GPU Memory Bandwidth (ECC off)***	250 GB/s

Table 2.2: XK7 GPU Node Specs

Service node hardware Service nodes are hosted on 166 Cray XIO blades and 30 XE6 blades, 4 nodes per blade. Each XIO service node consists of a 6-core AMD Opteron 2435 working at 2.3 GHz and equipped with 16 GB of DDR2 memory in 4 GB DIMMs protected by x4 Chipkill (with single symbol error correction and dual-symbol error detection capabilities). Service nodes host special PCI-Express cards such as Infiniband and fiber-channel cards. A service node can be configured as 1) a boot node to orchestrate system-wide reboots, 2) a system database node to collect event logs, 3) a MOM node for scheduling jobs, 4) a network node to bridge external networks through Infiniband QDR IB cards, or 5) as an Lnet (Lustre filesystem network) node to handle metadata (via Lustre metadata servers, or MDSes, to keep track of the location of the files in the storage servers) and file I/O data (via Lustre Object Storage Servers, or OSSes, to store the data stripes across the storage modules) for file system servers and clients.

Inteconnect Blue Waters high-speed network consists of a Cray Gemini System Interconnect. A detailed description of the interconnect is given in chapter 3.

File System. All blades are diskless and use the shared parallel file system for IO operations. Blue Waters hosts the largest Lustre installation to date. It consists of parallel file system used to manage data stored in Cray Sonexion 1600 [11] storage modules. Each Sonexion module has 1) 2 SSD of 2 TB in a RAID 1 configuration for journaling and logging, 2) 22 disks of 2 TB

for metadata storage, and 3) 80 disks of 2 TB for data storage, organized in units of 8 disks in RAID 6. All disks are connected to two redundant RAID controllers. In each unit, two additional disks serve as hot spares, which automatically provide failover for a failed drive. Data are accessed transparently from the nodes via the Lustre service nodes (Lnet), which interface with the Sonexion storage modules. Blue Waters includes three file systems, i.e., project, scratch, and home, and provides up to 26 PB of usable storage over 36 PB of raw disk space.

Running Jobs on Blue Waters Blue Waters users can submit their jobs using *aprun* command which submits the job to Application Level Placement Scheduler (ALPS) [12]. ALPS then launches applications on compute nodes. For submitting batch jobs or interactive jobs, users can use qsub command. Blue Waters is configured to use Moab/Torque [13] for launching batch jobs. Figure 2.2 shows the transition of job state from its start to end. For each of these states, Torque writes an event message in the torque logs which can then be used to track the status of the job in the system. In [14, 15] studied the job exit status for Blue Waters system.

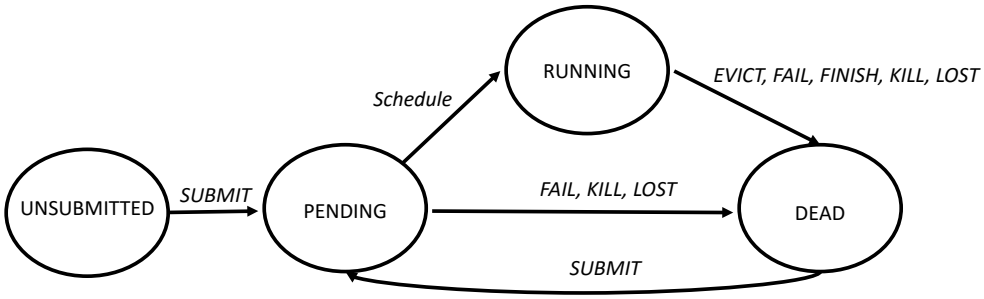


Figure 2.2: State transition diagram of the job from start to end.

CHAPTER 3

INTERCONNECTS

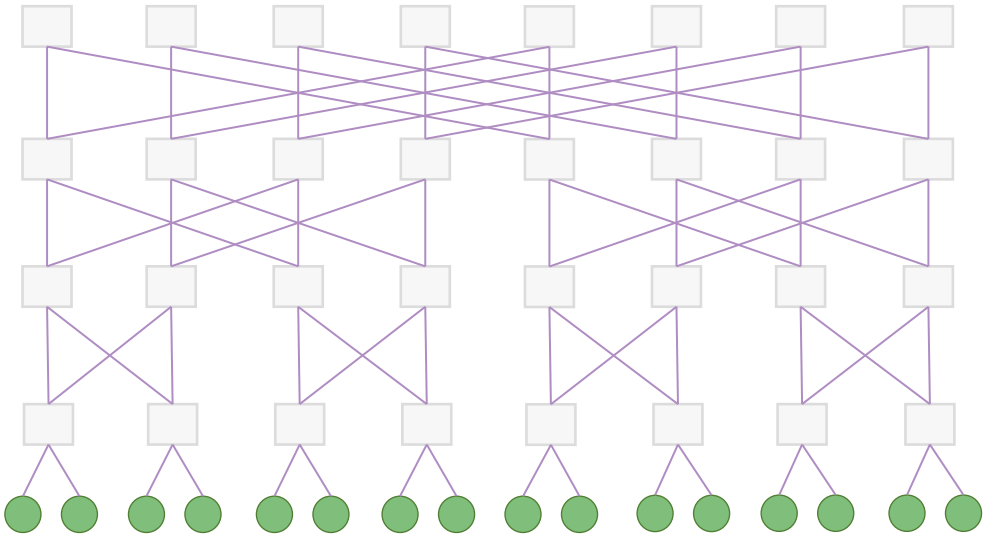
In this chapter, we will first briefly give an overview of interconnects and dive deeper into the details of torus networks. Then, we will describe the architecture and system design of the Cray Gemini interconnects. Blue Waters uses the Cray Gemini interconnect, which is a torus network.

3.1 Interconnection Networks

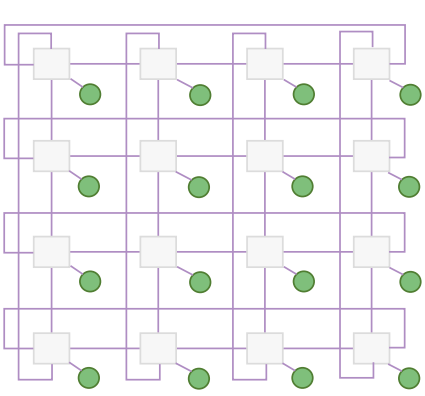
An interconnection network is a programmable system that transports data between terminals. The main design aspects of interconnection networks are (1) topology, (2) routing, (3) flow control, and (4) recovery. Topology determines the connection between compute nodes and network nodes (routers, switches, etc.). Routing, flow control, and recovery heavily depend on the topology of the interconnection system. The most widely used topologies in high-performance computing (HPC) are (1) fat tree (e.g. Roadrunner HPC system [16]), (2) dragonfly (e.g., Trinity [17]), and (3) torus (e.g., Blue Waters [18]). At the time of writing of this thesis, 40% of top 10 HPC systems are using torus, another 40% dragonfly and 20% use fat-tree topologies in their interconnect technologies. Fig. 5.4 shows the arrangement of nodes in these three topologies.

3.1.1 Torus Networks

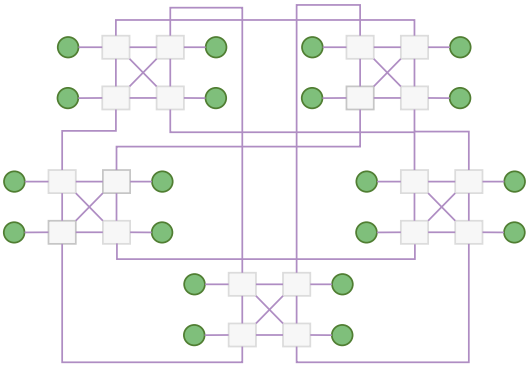
Torus networks can support $N = k^n$ nodes which are arranged in a *k-ary n-cube* grid (i.e., nodes are arranged in regular n-dimensional grid with k nodes in each dimension). In the case of Blue Waters, $n = 3$. In torus networks, each node serves simultaneously as an input terminal, output terminal and switching node of the network. Torus networks are regular (i.e.,



(a) Binary 4-tree fat tree Topology



(b) 2-D Torus Topology



(c) 1,4,1 Dragon fly Topology

Figure 3.1: Interconnect Topologies (a) Binary 4-tree fat tree Topology, (b) 2-D Torus Topology, (c) 1,4,1 Dragon fly Topology - arrangement of network nodes (shown as boxes) and compute nodes (shown as circles)

all nodes have the same degree) and are also edge-symmetric (useful for load-balancing). Torus networks are very popular for exploiting physical locality between communicating nodes, providing low latency and high throughput. However, the average hop count to route packets to a random node is high compared with that of logarithmic networks. On the other hand, extra hop counts provide path diversity, which is required for building fault-tolerant architecture.

Path diversity is given by the total number of minimum routes from a to b that are possible in a topology. For a torus network, it is given by equation 3.1

$$|R_{ab}| = \frac{(\sum_{i=0}^{n-1} \delta_i)!}{\prod_{i=0}^{n-1} \delta_i!} \tag{3.1}$$

where δ_i is the number of hops between the ith dimension of a and b. The number of unique paths increases rapidly with dimension and distance. This makes interconnects more tolerant to single failures. However, situations exist in which the topology becomes unrouteable due to multiple failures. Unrouteable topology configurations can be enumerated for any given n and k.

Maximum throughput in a torus network [19, 20] is achieved when equation 3.2 is satisfied -

$$nk \leq \frac{N * W_n}{W_s} \tag{3.2}$$

where W_n is the channel width limit (highest attainable frequency on the channel) due to node pin-out (which determines the the ability to send bits in parallel) and W_s is the channel width limit due to bisection (highest attainable frequency on the channel across the bisection).

Latency in a network is a function of serialization latency and hop count. Serialization latency is the time required for a packet of length L to cross a channel with bandwidth b. On the other hand minimum hop count in a torus network is determined by equation 3.3.

$$H_{min,T} = \begin{cases} \frac{nk}{4}, & \text{if k even} \\ n(\frac{k}{4} - \frac{1}{4k}), & \text{if k odd} \end{cases} \tag{3.3}$$

Routing involves selection of the path from the source node (src) to destination node (dst) among many possible paths in a given topology. In torus networks, routing is done through the dimensional-order routing algorithm. In dimensional-order routing, each packet travels in the network by resolving each dimension one at a time. Once the algorithm resolves a dimension, it does not account for the solution to reenter the resolved dimensions. For example, in a 3-D torus, the router first resolves the X dimension, then the Y dimension, and finally the Z dimension. Note that packets in torus networks can travel in a clockwise or counterclockwise direction. Thus, the first step in routing is to determine the direction of routing. Routing direction can be determined by equation 3.4.

$$D_i = \begin{cases} 0, & \text{if } |\delta_i| = k/2 \\ sign(\delta_i), & \text{otherwise} \end{cases} \tag{3.4}$$

where δ_i is given by

$$\delta_i = m_i - \begin{cases} 0, & \text{if } m_i \leq \frac{k}{2} \\ k, & \text{otherwise} \end{cases} \tag{3.5}$$

and m_i is given by

$$m_i = (dst_i - src_i) \mod k \tag{3.6}$$

3.1.2 Errors, Failures, and Recovery

The first step in building a resilient network is to identify the nature and types of errors and failures and then build abstract models that can help to understand and prevent errors and failures. In [2], Dally and Towels have summarized the existing failure modes. We adapt their table in Table 3.1 and give a brief summary of these failure modes. We advise readers to read chapter 21 from "Principles and Practices of Interconnection Networks" [2].

Failure Modes	**Fault Model**
Gaussian noise on a channel	Transient bit error
Alpha-particle strikes on memory (per chip)	Soft error
Alpha-particle strikes on logic (per chip)	Transient bit error
Electomigration of a conductor	Stuck-at fault
Threshold shift of a device	Stuck-at fault
Connector corrosion open	Stuck-at fault
Cold solder joint	Stuck-at fault
Power supply failure	Fail-stop
Operator removes good modules	Fail-stop
Software failures	Fail-stop or Byzantine

Table 3.1: Failure modes for a typical interconnection network (adapted from [2])

Transient failures cause one or more bits to flip, causing temporary errors. These can be easily detected and corrected using checksums and error-correcting codes (ECCs) or a retry. Transient errors are typically harmless, and the problem dies on its own over time. An example of transient errors includes Gaussian noise on the channel and alpha-particle strikes on logic gates. Transient errors are measured in bit-error rate (BER). Soft errors, on the other hand, change the values of data, which continue to affect the system for a long period, thus have a lasting eect. An example of such a case includes alpha-particle strikes on memory, such as routing tables. These errors are handled by using memory scrubbers, or simple reboots can be detected using ECCs. Soft errors are measured by soft-error rate (or SER). The stuck-at fault permanently damages the component where the logic is either always high or low. Fail-stop failures are permanent failures in the interconnection system, such as link failures, router failures, etc. These failures are measured in mean time between failures (MTBF) or failures in time (FIT).

It is surprising that models do not exist for failure modes of recovery mechanisms for interconnection networks and the behavior of the system during the execution of these recovery mechanisms.

3.1.3 Unrouteable Topology Configurations

Depending on the state of the system and manual recovery/replacement strategy of failed components, a series of failures can lead to an unrouteable topology configuration. An ideal interconnect-recovery mechanism would fail only when it encounters the case of unrouteable topology configurations. An unrouteable configuration can be defined as a state under which the packets cannot be routed from source node (src) to destination node (dst) using the specified routing rules. In Fig. 3.2, we show such a configuration for a 2-D torus. The failure of two network nodes (indicated in red) blocks all traffic to the network node (indicated in blue) that tries to send packets using the loop in the Y direction on which the network node is located. For example, in this configuration packets from node (0,1) cannot be sent to node (1,2).

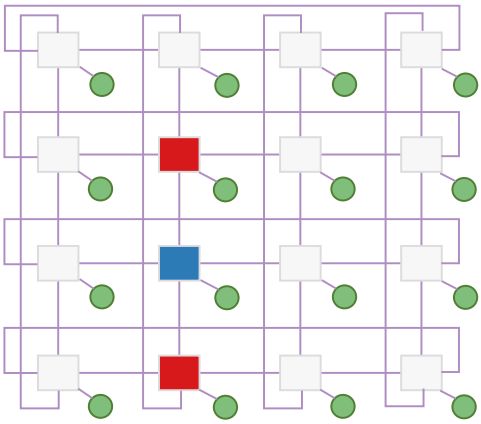


Figure 3.2: Unrouteable topology configuration for 2-D torus. Red boxes are the failed network nodes and disabling the blue-colored network node will restore the connectivity of the network

3.1.4 Cray Gemini Architecture

To keep the section description comprehensive, 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 mechanisms [21]. An overview of the Gemini interconnect is provided in [22]. Evolution of interconnect technologies for Cray systems can be tracked using [23, 24, 25] The Blue Waters' high-speed network consists of an anisotropic *3-D torus* using Cray Gemini router (see Figure 3.3) to connect

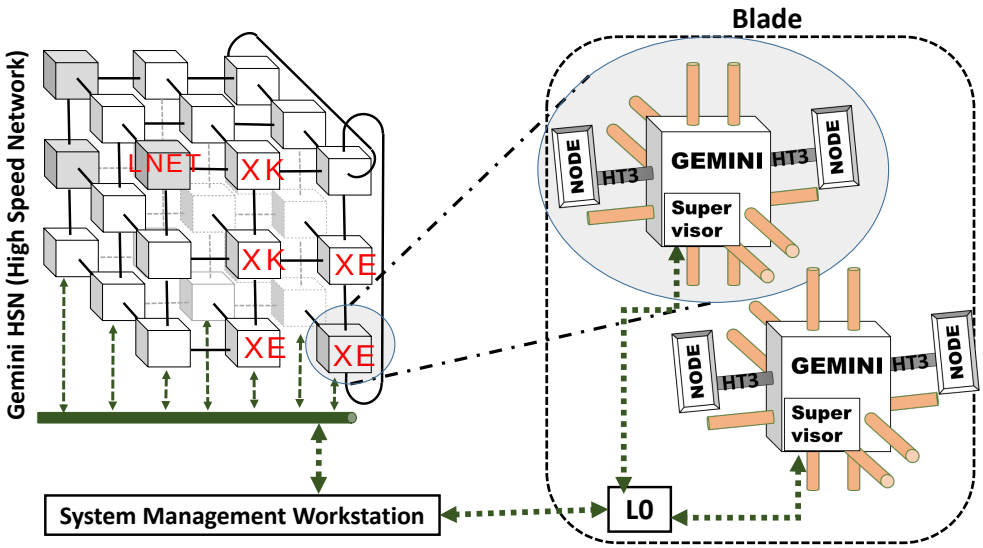


Figure 3.3: Blue Waters 3-D Torus Gemini interconnect layout. XE and XK are CPU and GPU nodes respectively where as LNET is Lustre NETwork routers running on top of the Gemini interconnect

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).

3.1.5 Fault Tolerance and Resiliency in Gemini

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.

3.1.6 Fault Detection and Recovery in Gemini

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 3.3. 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.

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 3.4. 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).

Link Failover

Figure 3.5 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 ~ 1000 seconds. Links can become unavailable due to one of the following reasons:

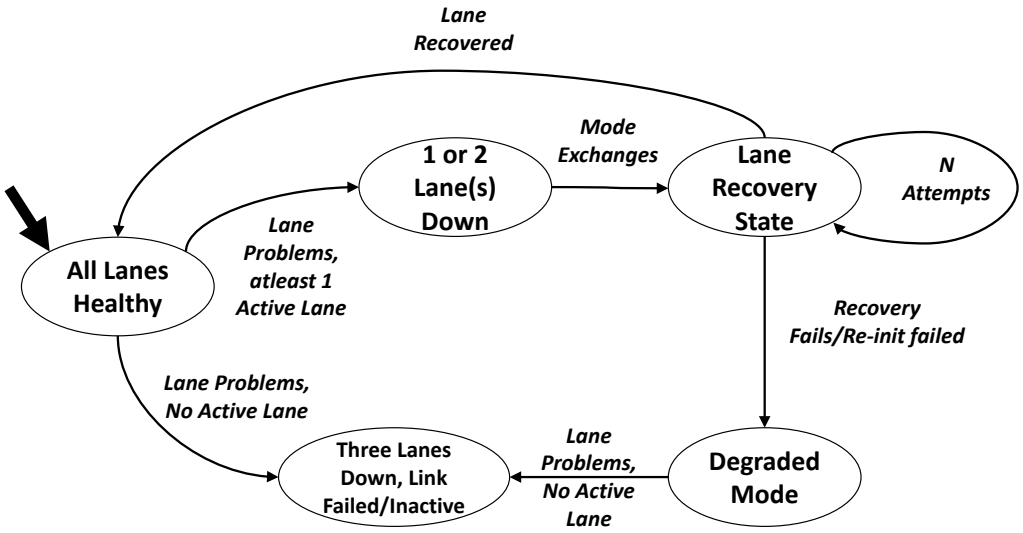


Figure 3.4: State transition diagram for lane recovery procedure. 3 lane failures (in the same link) result in an inactive link.

- 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 5.3, a state-transition diagram shows the various steps and conditions leading to a successful or failed link failover. Appendix B shows a real trace of successful and failed link failover scenario obtained from Blue Waters.

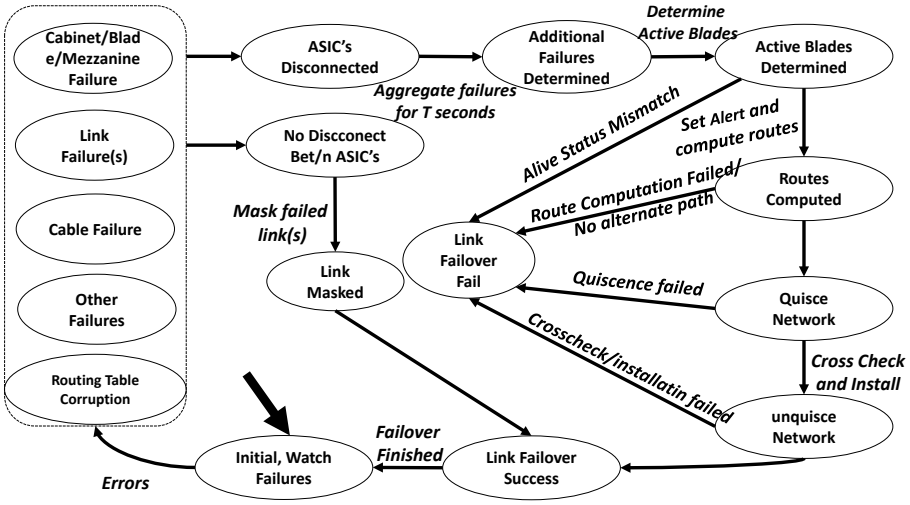


Figure 3.5: State transition diagram of Gemini link failover operations.

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 3.6.

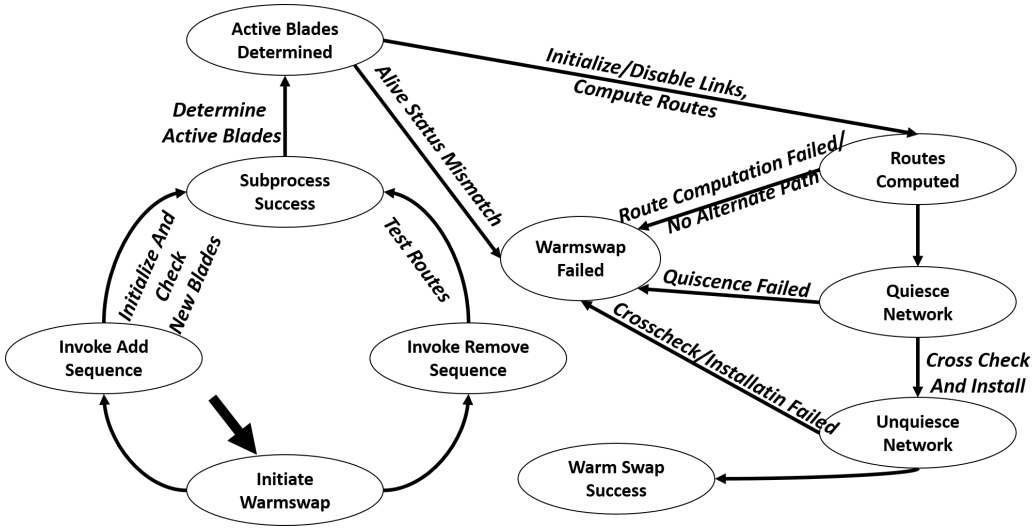


Figure 3.6: State transition diagram of Gemini warm swap operation (which is always invoked by a system administrator).

CHAPTER 4

DEFINITIONS

In this chapter, we have defined the terms used in this thesis.

Availability — Availability is the ratio of time a system or component is functional to the total time it is required or expected to function.

Bisection bandwidth — It is the highest bandwidth that can be achieved across the smallest cut that divides network into two equal halves (partitions).

Fault — Fault is the hypothesized cause of error(s).

Dependability — Dependability is the ability of a system to deliver a specified service.

Error — Error is the observed incorrect result of an operation after the computation is finished.

MTBF — MTBF is defined as mean time between failures. MTBF of event X is given by :

$$\text{MTBF(X)} = \frac{\text{Total Production Hours}}{\text{Total \# Recovery Event X of type Y}} \tag{4.1}$$

MNBR — MNBR is defined as mean node-hours between recovery operations. MNBR of event X of type Y for a fixed time period is defined as:

$$\text{MNBR(X,Y)} = \frac{\text{Total Production Hours}}{\text{Total \# Recovery Event X of type Y}} \tag{4.2}$$

Where, the total number of production node hours is calculated as total system availability hours (i.e., hours the system is available) * total number of available nodes, X is the event and Y is the event type. For, example X can be one of success, failed, or total recovery procedures and Y can be one of lane, link, warm swap.

Failure — Failure is an event that transitions the state of the system from good (healthy) to bad (unhealthy).

Failover — Failover is a method of switching to a standby or redundant component when the primary fails.

Interconnect — An interconnect is a programmable network that provides communication paths and protocols for interaction between terminals.

Recovery — Recovery is a method of restoring the system operation(s) (i.e. returning to desired functionality). Note that the system-state may change during recovery, however, the system is able to produce outputs when presented with a valid input after recovery.

Recovery Procedure — A recovery procedure consists of an isolated recovery operation.

Recovery Process — A recovery process is a combination of recovery procedures and represents either concurrent or sequential execution of multiple recovery procedures (e.g., multiple failover operations to recover the system).

Reliability — Reliability is defined as the probability that a device will perform its required function under stated conditions for a specific period of time.

Resiliency — Resilience is the ability to provide and maintain an acceptable level of service in the face of faults and challenges to normal operation

System-wide Outage — In scientific literature, system-wide outage is a loosely defined term and is site-specific. In the context of NCSA/Blue Waters, it is used to declare situations in which the system is not able to perform as expected.

CHAPTER 5

METHODOLOGY AND TOOLS

5.1 Approach

To understand and analyze the recovery procedures of the Gemini interconnect, we augmented LogDiver[1], a tool for measuring application resiliency in HPC systems, with additional capabilities to implement an interconnect-recovery analysis workflow. Interconnect-recovery analysis workflow tool (I-RAT) in that sense is a plugin built on top of LogDiver.

I-RAT filters, coalesces, and correlates Gemini-related errors with application failures, and human-written system failure reports managed by system administrators. I-RAT uses a state-machine model to determine the recovery type — lane recovery, link failover, warm swap category- and recovery exit status as successful or failed from logs.

In addition to the characterization of the recovery procedures, I-RAT is used to evaluate the impact of recovery (successful/failed) on user applications and the system availability by using LogDiver utilities.

We briefly summarize the steps of our approach here before diving deeper into the details. These steps are —

- **Data collection and filtering** — In this step, we created a set of rules that can be used to extract interconnect-specific event logs. Raw logs from Blue Waters do not directly indicate the source of these logs and relationship between these logs, thus making it most time-consuming step. We used a series of strategies to learn about these rules such as reading the technical manuals from Cray describing the logging mechanism, interaction with Blue Waters system admins/Cray engineers, automatic discovery, and extraction of interconnect-related logs during interconnect-related failures found in system-wide outage reports. The relevant event logs are shown in Table A.1 and the relationship

between these logs is described in section 3.1.6. The filters obtained at the end of this step are used in filtering stage of LogDiver in addition to the filters that were used for extracting application event logs (described later in this section). Other datasets (see 5.1) used in this study were readily available from Blue Waters. The formats of these logs are described in section 5.2.

- **Clustering/coalescing the event logs** — In this step, we cluster the interconnect-related events according to the failure recovery model described in this section. This is a clear addition to LogDiver and forms the core part of I-RAT. Differently from LogDiver clustering algorithm which tries to cluster system errors to find the cause of application failures, I-RAT traces the failure-recovery path to understand the cause of the failure of the recovery as well as the conditions that triggered those failures in the first place. This required building a custom coalescing engine which could cluster the events to enable this study.

- **Analyzing the impact on application and system** — In this step, we take the output of LogDiver application parser and correlate the application failures with the recovery-sequence clusters obtained in the previous step. In addition, we correlate the failure reports maintained by the system admins of Blue Waters to correlate the system-wide outages to recovery clusters. Currently, I-RAT/LogDiver produces the following output for studying interconnect-related recovery mechanisms —

  - Recovery-sequence clusters and the final state of the recovery
  - Number of applications impacted by the recovery and a boolean variable indicating if it caused SWO or executed during recovery.
  - Measure of mean-time between recovery, mean-time between failures, etc.

5.2 Data Sources

Our tool takes three inputs (see Table 5.1): (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 so as to contain only user applications (i.e., all debug and benchmark jobs run by system administrators are removed from this analysis).

Table 5.1: 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

5.3 Measuring Application Resiliency : LogDiver

DiMartino et. al. implemented LogDiver to measure the resiliency of applications in HPC systems. LogDiver produces measurements for 1) measuring resiliency of different application scales, from single node application up to full-scale applications, and 2) analyzing the sensitivity of applications to different errors. In [15], authors measured the resiliency of the application for 5 million application runs to system and application errors. In [1], authors discussed the general architecture of LogDiver and further showed the resiliency of application to GPU errors on XK nodes of Blue Waters. We briefly describe the general architecture of LogDiver here to help understand the interaction of I-RAT with LogDiver.

LogDiver operates in four main steps, depicted in Fig. 5.1. Each step produces several output files that are fed downstream to the subsequent steps.

- **Data Collection** — In this step, data is collected from multiple sources such as syslogs, torque logs, and alps logs. To ease the porting of this tool to different systems, data are parsed to an internal format that is system-agnostic.
- **Parsing and Filtering Data** — In this step, syslogs are parsed through filters (that were created manually) to convert the raw logs to events using standard error templates. The error template consists

of specific unique numerical template ID, tag, category, and group to each error template. The tag is a textual description of the event of interest (e.g., GPU DOUBLE BIT EXCEPTION or LUSTRE CLIENT EVICT), the category refers to the subsystem generating the event (i.e., NVIDIA GPU or LUSTRE), and the group corresponds to the type of the subsystem involved in the event (e.g., NODE HW or STORAGE). At the time of writing this thesis, LogDiver has over 700 error templates.

- **Workload Consolidation** — The required application data (i.e., executed aprun commands, see Figure 1) are scattered over several non-consecutive entries in the ALPS logs and need to be retrieved and assembled for each user application. The output of the step is an extended data set of user applications (referred to as application data in Figure 3.(c)), which contains one entry for each application. The entry includes i) start and end time, ii) reservation ID, job ID, user, group, and application name, iii) resources data, e.g., number, ID, and type of nodes, memory, and virtual memory, iv) application exit code and job exit code, v) job- and application-required wall time and used wall time, and vi) the command used to launch the application.

- **Workload-Error Matching** — In this step, error data is merged with the workload data generated in the previous step. Error data is first coalesced using a sliding window algorithm as described in [26, 27] to approximate the ground truth occurrence of events.

- **Metric Estimation** — LogDiver estimates various metrics of interest with respect to applications that fail because of system related issues. Metrics used in this study include:

  - Mean Node Hours Between Failures (MNBF) computed as a ratio of the total number of production node hours to the total number of application failures
  - Mean Time Between Interrupt (MTBI) computed as a ratio of the total number of production hours to the total number of application failures
  - Probability of an application failure

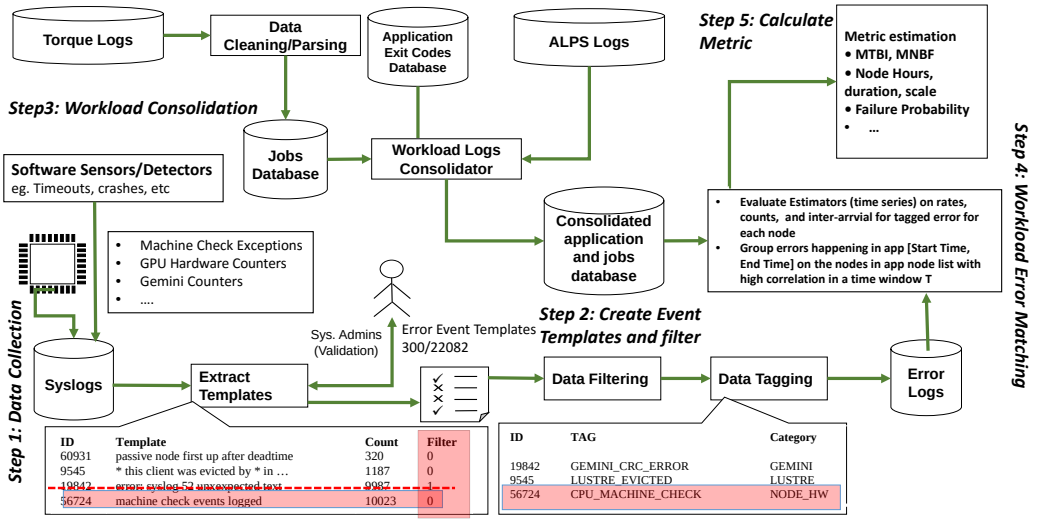


Figure 5.1: LogDiver : An HPC log data analysis toolkit

5.4 Interconnect Recovery Analysis Workflow Tool : I-RAT

Figure 5.2 shows the block diagram of the I-RAT plugin built on top of LogDiver. I-RAT consists of six operations, indicated by diamond shapes (in Figure. 5.2): *Filter Gemini logs*, *Gemini aware coalescing*, *Job impact analyzer*, and *System impact analyzer*. In order to support I-RAT on LogDiver, some of the steps of LogDiver was modified. These steps are indicated as *Modified* in 5.2. Plugin was developed in C++14.

In the following, we describe all the operations of the I-RAR in detail, that have been added to LogDiver.

5.4.1 Filter Gemini Logs

Date filtering and tagging operation from LogDiver applies regular expression (regex) rules onto the raw system logs line by line to extract interconnect-related logs from syslogs. We added over 100 regex rules to the existing LogDiver database for extracting interconnect-related events. These regex rules were selected based on our understanding of Gemini recovery procedures, as discussed in section 3.1.4 and each event is matched to only one rule. All these rules are listed in Appendix A. These extracted logs are then filtered and transformed into a set of features, represented by *nple*

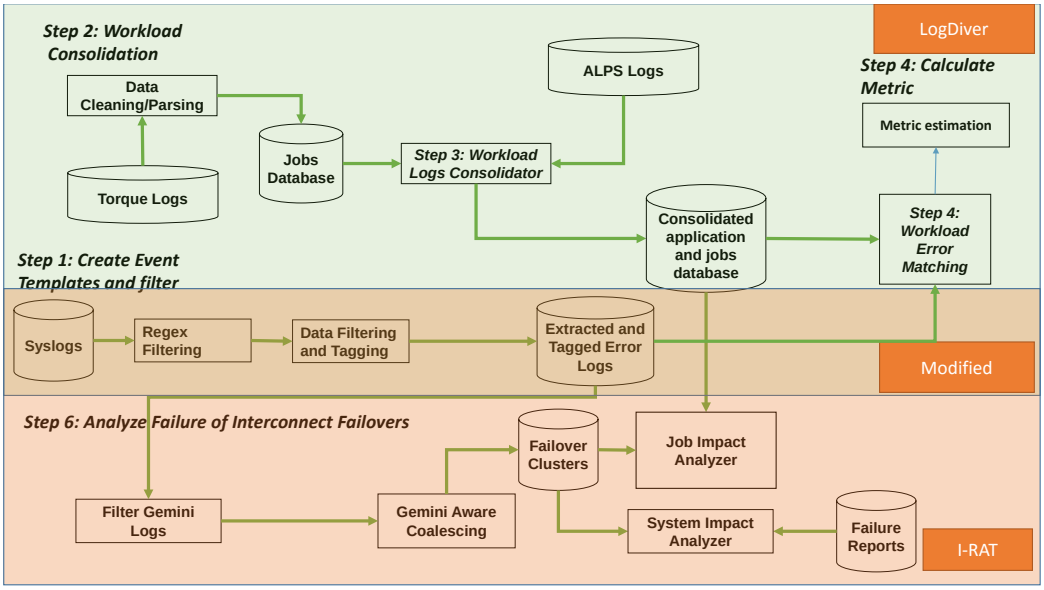


Figure 5.2: Block diagram of the pipeline for analyzing Gemini recovery procedures (I-RAT) built on top of LogDiver.

$< Time, Location, Tag, Info >$. 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 [28] or from the Tag and the body of the message itself. It is important to extract the exact location to evaluate the impact. For this reason, we modified the LogDiver *data filtering and tagging* operation. 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

Table 5.2: An example log from Blue Waters indicating Gemini ASIC failure.

In the log example shown in Table 5.2, 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 $< Location >$.

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

Table 5.3: An example line from rules database

An example line from our rules database is shown in Table 5.3. These tags are divided into 4 categories: *faults/errors*, *failures*, *recovery transition*, and *recovery finish*. Table 5.4 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* (as shown in Fig. 5.3) in the system. The problem starts with faults, which manifest as an error in the system. Errors lead to failure(s) either immediately or some time later. A failure invokes a recovery mechanism, which can either succeed or fail. The details of these categories are as follows —

- *Faults/errors* —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 *Failure* 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. We refer to these errors as *trigger latent* as these events are indicative of interconnect-component failures. For examples, an increased number of misrouted packet can indicate routing issues. Similarly, increased number of one lane down event can indicate link issues and may lead to failure of the link itself.

- *Failures*—These tags indicate activity in the system that modifies or affects the network topology directly, 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 unrouteable. In almost all cases, when events from this category are observed, recovery procedures are expected to handle these events. Since these events trigger immediate recovery action in the system to restore the topology and functioning of interconnect, these events are also called as *trigger-immediate*.

- *Recovery start and transition steps* —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.

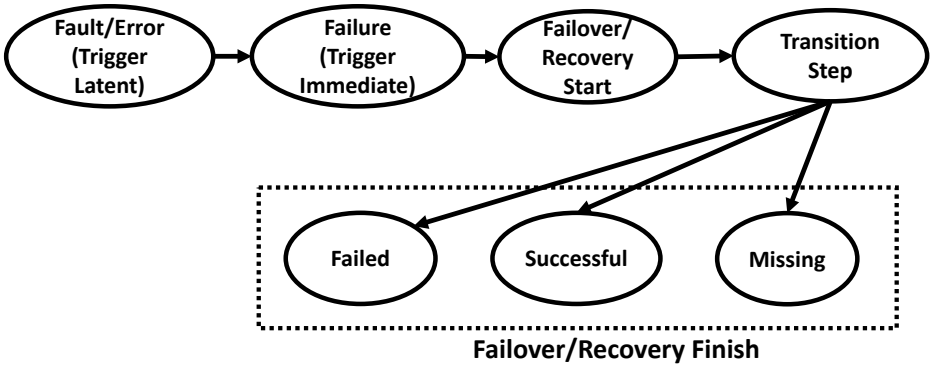


Figure 5.3: Reduced recovery-sequence state-transition diagram. All state-transition diagrams described in Section 3.1.4 are mapped to this reduced state-transition diagram.

FAULTS/ERRORS		RECOVERY START & TRANSITION	
BLADE_ELECTRICAL_ISSUE	2.24E+07	BLADE_RECOVERY	2.79E+02
CABINET_HEARTBEAT_FAILED	5.37E+05	BLADE_RECOVERY_SUCCESS	2.79E+02
BUFFER_OVERFLOW	4.90E+07	CABINET_READDED	7.11E+06
CHECKSUM_ERROR	8.64E+03	FINISHED_LINK_RECOVERY	3.72E+02
ECC_ERROR	1.16E+06	ROUTING_TIMEOUT	4.50E+01
MISROUTED_PACKET	5.06E+08	HSN_NETWORK_QUISCED	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
FAILURES		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

Table 5.4: Tag categories with representative tag examples and related counts, as observed in the logs

In a complex system like Blue Waters, it is hard to build (and then track) deterministic model of the state transitions (described in section 3.1.4) 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 3.1.4 are mapped injectively onto a reduced state-transition diagram depicted in Figure 5.3. This abstraction does not only help us cope with the problems discussed above, but also reduces the time to do our analysis as there are many fewer states to track.

5.4.2 Gemini Aware Coalescing

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 [29, 30]. The algorithm starts by initializing an empty tree. The tree is essentially based on a

fault tree model (see Figure 5.4), 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 a higher level in the topology affects all the sub-levels, and hence, all events that occurred in the lower levels during the same time window (as the high-level event) need to be merged. Similarly, an event at a lower level can only affect components located at a higher level in this tree. In this way, 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.

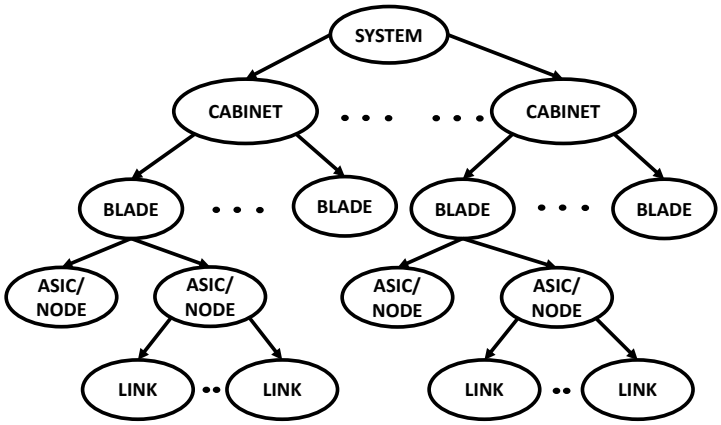


Figure 5.4: Gemini topology-aware fault model. The model helps track the effects of events in sub-components.

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 5.3. 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.).

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) ≤ slide_time)
   && (current_cluster ≠ NULL) then
5:         current_cluster.timestamp = log_line.timestamp
6:         current_cluster.add(log_line)
7:         merge_all_clustrs_in_subtree(current_cluster)
8:         current_cluster.transitions.add(determine_cluster_states(current_cluster)
9:     else
10:        if (current_cluster ≠ NULL) then
11:            current_cluster.final_state =
   determine_cluster_states(current_cluster)
12:            write(current_cluster)
13:        end if
14:        start_new_cluster(Tree, log_line.location)
              ▷ Start new cluster with log_line being it's first member at the
   appropriate location in tree
15:    end if
16: end for
17: write_active_clusters(Tree)                    ▷ Write all active clusters
```

Table 5.5: 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

Table 5.5 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, 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 the whole system.

5.4.3 System Impact Analyzer

The *System impact analyzer* processes SWO failure reports filed by the technical staff of Blue Waters/Cray, and correlates them with *recovery-sequence clusters* obtained from *Gemini aware coalescing* step. This operation looks at the overlapping time between recovery-sequence clusters and SWO dura-

tion mentioned in the reports. This merging does not necessarily indicate 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.

5.4.4 Job Impact Analyzer

The *Job impact analyzer* evaluates the impact of successful and failed recovery procedures on applications. 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 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.

CHAPTER 6

RESULTS

This section shows our preliminary results from the analysis of the outputs obtained from the methodology and tools, as described in section 5. We discuss completion status and failure rates of recovery procedures, and their impact on the system and applications.

6.1 Measuring Occurrence and Success of Recovery Procedures

In this section, we measure the following —

- Completion status
- Failure Rate
- Mean-node hours between recovery

6.1.1 Completion Status of Recovery Procedures

Figure 6.1 shows the breakdown in the 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 3.1.4 (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 an interconnect network deadlock that led to congestion and ultimately system-wide outage. There are other cases where such synchronization issues lead to failure of Gemini recovery procedures.

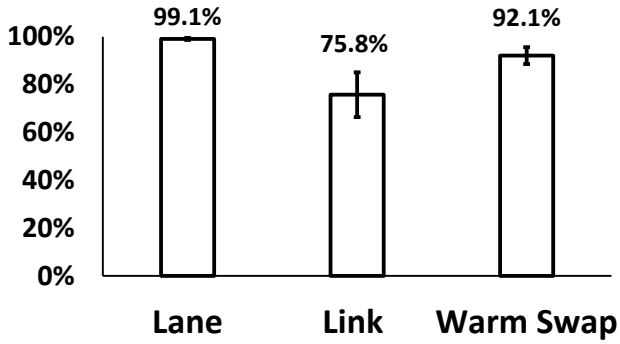


Figure 6.1: Recovery completion status for lane, link, and warm swap.

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 unrouteable. 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.

6.1.2 Recovery Procedure Failure Rate

Figure 6.2 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 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.

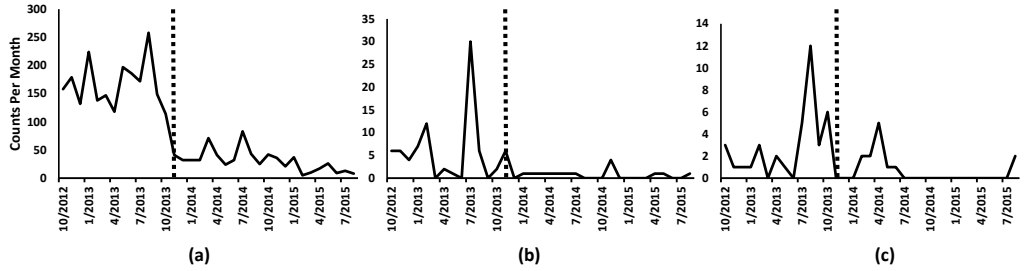


Figure 6.2: Count of total failure of recovery procedures for (a) lane (b) link (c) warm swap per month. The vertical line shows a major software upgrade fixing bugs in the Gemini interconnect resiliency management code.

6.1.3 Mean Node-Hours Between Recovery Procedures

Next, we calculated mean node-hours between recovery procedures (MNBR) to understand the frequency of recovery operations with respect to scale.

In this study, we define two metrics: (1) mean-node hours between launched recovery operations, i.e., X is launched recovery operations of type Y, and (2) mean-node hours between failed recovery operations (i.e., X is failed recovery operations of type Y). Y can be lane recovery, link failover, and warm swap.

Recovery Procedure Name	**Failed Recoveries [h]**	**Launched Recoveries [h]**
Lane Recovery	133,968.5	1223.2
Link Failover	4,019,056.3	973,167.7
Warm Swap	7,033,348.5	533,608.8

Table 6.1: Mean Node-Hour Between Recovery Events

Table 6.1 summarizes the MNBR for lanes, links, and warm swaps for failed recovery procedures and the total number of launched recovery procedures. Lane recovery, link recovery, and warm-swap procedures are launched every 2.8 minutes, 36.2 hours, and 20.6 hours, respectively. **Given the high probability of failure of recovery procedures, a low mean-time between trigger of recovery procedures poses a significant threat to system availability and reliability. Our results show that mean time between recovery events of interconnect is significantly lower than the other components present in the system.** For example, Martino et. al. reported an MTBF of 6.7 hours for the nodes in the same system.

6.2 Analyzing impact of recovery procedures

In this section, we analyzed the impact of a recovery procedure for successful and failed recovery procedures on system and applications. System impact is quantified by counting system-wide outages that were caused by interconnect recovery procedures.

6.2.1 System-Wide Outages

Of 101 SWOs observed in human-written reports filed by NCSA Blue Waters staff, the *system-impact analyzer* (see Figure 5.2) identified 28 SWOs related to Gemini interconnect recovery procedures. Of these 28 cases, 14 of these are strictly due to the failure of the recovery procedures in the Gemini interconnect. A deeper analysis of the recovery-sequence clusters associated with interconnect-related SWOs shows that only one instance of recovery-procedure failed due to unrouteable topology configuration. **a significant number of system-wide outages are caused by the failure of interconnect recovery procedures. Hence, failure of recovery procedures decreases the overall MTBF of the system.**

6.2.2 Application Impact

Using the methodology in chapter 5, 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 the failure of applications due to the delay in handling the failures. A successful recovery can last anywhere between 60 to 1000 of seconds depending on failure type and invoked recovery procedure.

In Figure 6.3, we show an example of a link failover sequence that ended successfully but lead to the failure of two applications. Through this example, we describe the all the potential phases due to which the applications can fail. A fault (Routing Table Corruption) occurred at time T. A short time

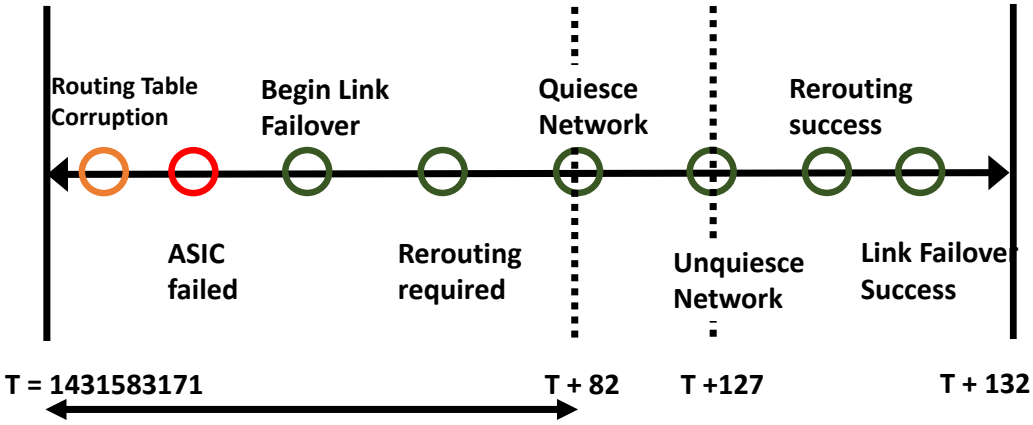


Figure 6.3: An example of a successful link failover sequence (taken from the output of the coalescing algorithm). Orange circle represents faults, red circle represents failures, and green circles represents recovery operations.

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 the propagation of quiescence in the network, and (2) packet loss. From $T + 127$ seconds to $T + 132$ seconds, applications that do not employ workload redistribution, or checkpointing could potentially fail due to unavailability of the nodes, blades, or cabinets. In this particular example, the two applications failed between T to $T + 82$. **This example concretely shows that 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.**

6.2.3 Validation on Mutrino Dataset

The methodology and tools presented in this thesis, work out of the box for Cray Aries interconnect. We ran I-RAT on 'Mutrino dataset '[31] for testing and validating our models. Mutrino is a Cray XC40 based testbed system consisting of 100 nodes connected through Aries interconnect for testing the 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. **Thus, our approach to analyze failure data from a supercomputer interconnect can be suitably extended to other interconnect technologies for studying and characterizing interconnect recovery capabilities of the system.**

CHAPTER 7

RELATED WORK

There are several studies on analyzing the behavior and resiliency of large-scale systems such as [32, 33, 34, 35, 36, 37, 38]. These studies have focused on overall system behavior and system resilience for understanding failure causes and predicting failures from logs. [39, 40, 41, 42], etc. focused on one of the HPC components such as file system, interconnect, compute hardware, etc. However, there is no study detailing the failure of the recovery mechanisms in HPC systems. This is the first work that describes the failure of interconnect-recovery mechanisms in large-scale HPC system such as Blue Waters.

Most of the work in interconnect system focuses on the design and scalability of the networks such as in [43, 44, 45, 19, 2]. With respect to resiliency, the scientific community has focused on failures and recovery of data-transmission [46], network-links [47, 48] and network-devices [49, 50]. The MIT Reliable Router [49] describes techniques for link monitoring, link-level retry, link shutdown and fault-masking techniques. Error control codes for data transmission is given [46]. Adaptive routing algorithms are useful for routing around faulty links (fail-stop model) and is first described in [48]. In [41], 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 to existing literature on interconnect-resiliency, we describe an abstract recovery model to understand the causes for the failure of the recovery mechanisms in interconnection networks. In this work, we primarily proposed the model and then measured the impact of failures of recovery-mechanisms on system and applications.

CHAPTER 8

CONCLUSION AND FUTURE WORK

In this thesis, we presented a study to understand the Gemini interconnect recovery operations. Based on this understanding, we proposed and implemented I-RAT as a plugin on top of LogDiver, a tool to extract and reconstruct recovery sequences to measure system and application resiliency to interconnect recovery operations.

Through our detailed analysis, we showed that the failure of interconnect recovery procedures in a large-scale system posses a real threat to availability and reliability of the overall system. We further analyzed the impact of interconnect-failures on system and applications. Out of 101 SWOs, 28 were shown to be caused by interconnect-related. As many as $\sim$ 20% of the executing applications fail during the recovery procedures. We also show that the applications can fail even during a successful recovery.

We found only one instance of unroutable topology configuration, however as many 28 SWO's were due to interconnect-related failures. This calls for revisiting recovery models and mechanisms for HPC interconnects.

8.1 Future Directions

In the future, we plan to take steps to delve deeper into understanding the cause of the failure of the recovery procedures and have an elaborate discussion on the need for building abstract recovery models for interconnects. We also plan to implement and validate our abstract models through fault injection experiments.

We also plan to compare the performance and resiliency of the Gemini and Aries interconnect in two large-scale systems using the developed methodologies.

The goal of this thesis is to enable data-driven resiliency mechanisms. Our

work provides a strong foundation to understand and build data-driven approaches for building monitoring and resiliency mechanisms for future large-scale systems.

REFERENCES

[1] 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*. ACM, 2015, pp. 11–18.

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

[3] R. F. Barrett, C. T. Vaughan, S. D. Hammond, and D. Roweth, "Application explorations for future interconnects," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE, 2013, pp. 1717–1724.

[4] P. Yebenes, J. Escudero-Sahuquillo, P. J. Garcia, and F. J. Quiles, "Towards modeling interconnection networks of exascale systems with omnet++," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE, 2013, pp. 203–207.

[5] J. Chung, I. Lee, M. Sullivan, J. H. Ryoo, D. W. Kim, D. H. Yoon, L. Kaplan, and M. Erez, "Containment domains: A scalable, efficient and flexible resilience scheme for exascale systems," *Scientific Programming*, vol. 21, no. 3-4, pp. 197–212, 2013.

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

[7] 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*, June 2015, pp. 25–36.

[8] C. Di Martino, , G. Goel, S. Sarkar, R. Ganesan, Z. Kalbarczyk, and R. Iyer, "Characterization of operational failures from a business data processing saas platform," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE Companion 2014. New York, NY, USA: ACM, 2014. [Online]. Available: http://doi.acm.org/10.1145/2591062.2591172 pp. 195–204.

[9] A. Inc., "Bios and kernel developers guide, for amd family 16th."

[10] T. D. I. M. Division, "A white paper on the benefits of chipkill-correct ecc for pc server main memory, 1997."

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

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

[13] "http://www.adaptivecomputing.com/products/hpc-products/moab-hpc-suite-enterprise-edition."

[14] M. Ezell, "Collecting application-level job completion statistics," ser. CUG 2010, Edinburg, UK, 2010.

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

[16] K. Koch, "The new roadrunner supercomputer: What, when, and how," *Presentation at SC06*, 2006.

[17] N. Wichmann, C. Nuss, P. Carrier, R. Olson, S. Anderson, M. Davis, R. Baker, E. W. Draeger, S. Domino, A. Agelastos et al., "Performance on trinity (a cray xc40) with acceptance-applications and benchmarks," *Memory*, vol. 2, p. 4, 2016.

[18] B. Bode, M. Butler, T. Dunning, W. Gropp, T. Hoe-fler, W.-m. Hwu, and W. Kramer, "The blue waters super-system for super-science. contemporary hpc architectures, jeffery vetter editor," 2012.

[19] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *Computers, IEEE Transactions on*, vol. 39, no. 6, pp. 775–785, 1990.

[20] A. Agarwal, "Limits on interconnection network performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398–412, 1991.

[21] *Network Resiliency of Cray XE Systems.* Cray. [Online]. Available: https://fs.hlrs.de/projects/craydoc/docs/books/S-0032-3101/html-S-0032-3101/S-0032-3101-toc.html#toc

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

[23] G. Faanes, A. Bataineh, D. Roweth, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, J. 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.* IEEE Computer Society Press, 2012, p. 103.

[24] S. L. Scott et al., "The cray t3e network: adaptive routing in a high performance 3d torus," 1996.

[25] R. E. Kessler and J. L. Schwarzmeier, "Cray t3d: A new dimension for cray research," in *Compcon Spring'93, Digest of Papers.* IEEE, 1993, pp. 176–182.

[26] J. P. Hansen and D. P. Siewiorek, "Models for time coalescence in event logs," in *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on.* IEEE, 1992, pp. 221–227.

[27] R. K. Iyer, L. T. Young, and P. V. K. Iyer, "Automatic recognition of intermittent failures: An experimental study of field data," *IEEE Transactions on Computers*, vol. 39, no. 4, pp. 525–537, 1990.

[28] R. Gerhards, "The syslog protocol," 2009.

[29] R. Iyer, L. Young, and P. Iyer, "Automatic recognition of intermittent failures: an experimental study of field data," *Computers, IEEE Transactions on*, vol. 39, no. 4, pp. 525–537, 1990.

[30] J. P. Hansen and S. D. P., "Models for time coalescence in event logs," in *Proc. of 1992 Fault Tolerant Computing FTCS 92*, 1992, pp. 221–227.

[31] A. G. J. Brandt and J. Repik, "Mutrino Dataset 2/15-5/15." SAND2016-2449 O. 2016, *Accessed: 2016-03-23.* [Online]. Available: http://portal.nersc.gov/project/m888/resilience/datasets/mutrino/

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

[33] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Fault prediction under the microscope: A closer look into hpc systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 2012, p. 77.

[34] Y. Liang, Y. Zhang, M. Jette, A. Sivasubramaniam, and R. Sahoo, "Bluegene/l failure analysis and prediction models," in *Dependable Systems and Networks, 2006. DSN 2006. Int. Conference on*, 2006, pp. 425–434.

[35] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *Proceedings of the International Conference on Dependable Systems and Networks*, ser. DSN '06. Washington, DC, USA: IEEE Computer Society, 2006. [Online]. Available: http://dx.doi.org/10.1109/DSN.2006.5 pp. 249–258.

[36] 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.

[37] X. Chen, C. Lu, and K. Pattabiraman, "Predicting job completion times using system logs in supercomputing clusters," in *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*, June 2013, pp. 1–8.

[38] J. M. Brandt, F. X. Chen, V. De Sapio, A. C. Gentile, J. Mayo, P. P. Pebay, D. C. Roe, D. Thompson, and M. H. Wong, "Quantifying failure prediction in large scale hpc systems: A case study." Sandia National Laboratories Albuquerque, NM; Sandia National Laboratories (SNL-CA), Livermore, CA (United States), Tech. Rep., 2009.

[39] B. Schroeder and G. A. Gibson, "Disk failures in the real world: what does an mttf of 1,000,000 hours mean to you?" in *Proceedings of the 5th USENIX conference on File and Storage Technologies*, ser. FAST '07. Berkeley, CA, USA: USENIX Association, 2007. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267903.1267904

[40] G. Shipman, D. Dillow, S. Oral, F. Wang, D. Fuller, J. Hill, and Z. Zhang, "Lessons learned in deploying the worlds largest scale lustre file system," in *The 52nd Cray user group conference*. Citeseer, 2010.

[41] M. Ezell, "Understanding the impact of interconnect failures on system operation," in *Proceedings of Cray User Group Conference (CUG 2013)*, 2013.

[42] J. R. Mayo, F. X. Chen, P. P. Pebay, M. H. Wong, D. Thompson, A. C. Gentile, D. C. Roe, V. De Sapio, and J. M. Brandt, "Understanding large scale hpc systems through scalable monitoring and analysis." Sandia National Laboratories, Tech. Rep., 2010.

[43] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus et al., "Blue gene/l torus interconnection network," *IBM Journal of Research and Development*, vol. 49, no. 2/3, p. 265, 2005.

[44] R. Brightwell, K. Pedretti, and K. D. Underwood, "Initial performance evaluation of the cray seastar interconnect," in *High Performance Interconnects, 2005. Proceedings. 13th Symposium on.* IEEE, 2005, pp. 51–57.

[45] M. Blumrich, D. Chen, P. Coteus, A. Gara, M. Giampapa, P. Heidelberger, S. Singh, B. Steinmacher-Burow, T. Takken, and P. Vranas, "Design and analysis of the bluegene/l torus interconnection network," IBM Research Report RC23025 (W0312-022), Tech. Rep., 2003.

[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.* ACM, 1992, vol. 20, no. 2.

[48] 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.

[49] 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.* Springer, 1994, pp. 241–255.

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

APPENDIX A

INTERCONNECT RELATED EVENTS

Table A.1 contains a list of all the rules used to filter the raw systems logs obtained from Blue Waters. The table contains three columns – (1) template ID (unique numerical identifier used internally by the tools), (2) regex rules containing regular expressions, and (3) tag (uniquely identifying the error).

Table A.1: Regular expressions used for filtering Gemini failure/recovery related events.

Template ID	**Regex Rule**	**Tag**
1	ERROR\s*powerSlotDown:	blade_emergency_-power_off
5	Response\s*Protocol\s*Error	protocol_error
12	ERROR:\s*Routing\s*fault	routing_failure
133	ERROR\s*\w+:\s* cab\s*health\s*fault\s*detected.	cabinet_degraded_-health
142	\s*table\s*full\s*dropping\s*packet.	packet_drop_on_-buffer_overflow
165	SSID\s*Detected\s*Misrouted\s*Packet	misrouted_packet
103	SSID\s*Correctable\s*Memory\s*Error	ecc_error
104	ipogif_rx_irq:Received \s*packet\s*checksum\s*error	checksum_error
180	ERROR:\s*route \s*command\s* for\s*partition.*timed\s*out	routing_timeout
238	Gemini\s*Error:\s*lost\s* \d+\s*messages?\s*due\s*to\s*full	message_lost_on_-buffer_overflow
2001	HSN\sASIC\sLCB\slane.*\sreinit\sfailed	lane_recovery_failed
2002	TX\slanemask=[356]+	one_lane_down
2003	TX\slanemask=[124]+	two_lane_down

Table A.1: (cont.)

Template ID	**Regex Rule**	**Tag**
2004	TX\slanemask=0	three_lane_down
2005	cb_hw_error: failed_component\s.*\stype\s23	link_failed
2006	cb_hw_error:\sfailed_component\s.*\stype\s21	asic_failed
2007	alert\—Mezzanine\sVoltage\sFault	mezzanine_power_failed
2008	generic_rsp_timeout:\sERROR.*L0	blade_power_failed
2009	dispatch:\scurrent_state warm_swap	warm_swap_started
2010	bwsmw1\s[\d]+\sWarm\s swap\soperation\swas\ssuccessful	warm_swap_success
2011	WARNING:\sTimed\sout\s waiting\sfor\sreply\s to\sSDB\supdate\sevent\sfrom\sboot\s node	warm_swap_delayed
2012	routing\stable\scorruption\sfound	routing_table_corruption
2013	;\s*reroute\srequired	routing_reroute_triggerred
2014	All\scomponents\sreturned\ssucces	reroute_success
2015	do_throttle:\sTelling\sall\sblades\s to\sunthrottle\snetwork\sbandwidth	throttle_sys
2016	do_node_quiesce	stop_network_traffic
2017	do_node_unquiesce	start_networ_traffic
2018	cb_hw_error:\shandling\sfailed\slink	started_link_recovery
2019	done\shandling\sfailed\slinks\s in\s[\d+.\d+]*\sseconds	finished_link_recovery
2020	bwsmw1\s[\d]+\sLink\s recovery\soperation\swas successful	link_recovery_success
2021	ALERT:\sEmergency\sPower\sOff\sFault.	epo_fault
2022	ALERT:\sFan\sRPM\sFault	fan_fault
2023	done\shandling\swarm\sswap\sin	warm_swap_finish_time

Table A.1: (cont.)

Template ID	**Regex Rule**	**Tag**
2024	set_warm_swap_err	set_warm_swap_err
2025	do_throttle:\sTelling\sall\sblades \sto\s\throttle \sGemini	do_network_throttle
2026	(\d\d:\d\d:\d\d).*(handling\s corrupted\srouting\stable\serror)	handling_corrupt_routing_table
2027	HSN\sBoot\sfailed	hsn_node_boot_failed
2028	dispatch.*(current_state\squiesce\s)	network_quisce
2029	dispatch.*(current_state\sunquiesce)	network_unquisce
2030	(\d\d:\d\d:\d\d).*\s(c\d+-\d+c\d+s\d+).*\sauto-throttled	blade_throttled
2031	Link\srecovery\soperation\sfailed	link_recovery_failed
2032	Warm\sswap\soperation\sfailed	warm_swap_failed
2033	HWERR.*0x0(00([1-9]—a)—30[1-4]—4(0e—0f—10—11)—50([1-9]—a—b—c)—60[1-3]—70([1-8]—c—d)—80(6—7—c—d—e—f)—81[0-5]—90[1-6]—b(0—2)4—b3([3-8]—a)—c0[1-6]—d0(6—c—d—e—f))	asic_logic_failure
2034	HWERR.*(BTE\s\wX\sDescriptor\s Table\sIndex\s\d\sSBE):Info	bte_error
2036	L0_T_BAX_NODE0_VRM_VDD:\s\d+	l0_vdd_error
2037	send_slot_down_rsp:\sbladefailedlist	blade_down_detected
2038	dispatch:\scurrent_state route_compute	route_compute
2039	Retry\s\d\s-\srouting\swith\s[XYZ]+\srouting\sorder	routing_retry
2040	ERROR:\sroute\scommand\sfor\s partition\sp0\stimed\sout	routing_timeout
2041	add_link_to_list:\sadding\slink	link_failed_handled
2042	l0sys_healthmon:\sL0SYS_HEALTHMON\sfailed	blade_electrical_issue
2043	Warm\sswap\sbeginning	warm_swap_begin

Table A.1: (cont.)

Template ID	**Regex Rule**	**Tag**
2044	Attempting\sto\sadd\sblades	warm_swap_transition
2045	Appending\sL0\sc\d+-\d+c\ds\d\sstate\sready	cabinet_readded
2046	build_modulelist:\ssetting\sblade\sc\d+-\d+c\ds\d\sto\sdisabled\smodule\slist	blade_recovery_gemini
2047	Clearing\sblade\swarnings	blade_recovery_success
2048	Clearing\sblade\salerts	blade_recovery
2049	Installing\snew\sroutes	rerouting
2050	Links\sthat\swill\sbe\s unused\sare\snow\s turned\soff	disable_links
2051	Finished\sinstalling\snew\sroutes	reroute_complete
2052	Unquiescing\sthe\shigh\-speed\snetwork	hsn_network_uniquisce
2053	Computing\snew\sroutes	reroute_compute
2054	Waiting\sfor\sthe\shigh-speed\snetwork\straffic\sto\sdrain	hsn_wait_network_-drain
2055	Finished\squiescing\sthe\shigh-speed\snetwork	hsn_network_quisced
2056	Test\sreroute\sproceeding	test_reroute
2057	Turning\slink\smonitoring\s on\sfor\sblades\sthat\swere\sadded	warmswap_blade_on
2058	Finished\supdating\sSDB\sdatabase	sdb_update_success
2059	cb_node_unavailable:\snode\sc\d+-\d+c\ds\dn\d\sfound\s in\sunavailable\sevent	node_down
2060	cb_node_available:\sfound\snode\sc\d+-\d+c\ds\dn\d	node_up
2061	Cabinet\sReceived\sUnexpected\s Blade\sController\sHeartbeat	cabinet_heartbeat_-failed
2062	Link\sInactive	link_inactive
2063	HWERR.*0x0801	hwerr_misroute_packet

Table A.1: (cont.)

Template ID	**Regex Rule**	**Tag**
2064	HWERR.*0x0802	hwerr_request_with_-no_entry_orb
2065	HWERR.*0x0803	hwerr_command_mis-match
2066	HWERR.*0x0816	hwerr_dword_flit_mis-match
2067	HWERR.*0x0b2b	hwerr_b2b
2068	handling\slack \sof\sforward \sprogress\serror\s.*for \s node\s(c\d+-\d+c \d+s \d+n \d+)	lack_of_forward_-progress
2069	set_warm_swap_err	warm_swap_err_info
2071	HWERR.*0x0b2e	hwerr_ssid_response_-protocol_error
2080	HWERR.*0x0d08	hwerr_nif_bad_req_-packet
2081	HWERR.*0x0d09	hwerr_nif_bad_re-sponse_packet_hsn_-bug_1
2088	HWERR.*0x0814	hwerr_flow_ctl_orb_to_-nl_hsn_bug_2
2094	HWERR.*0x0(401—b15—b1d—b30—b2d)	hwerr_ssid
2095	HWERR.*0x0d0(4—5)	hwerr_nif_squashed_req
2142	HWERR.*0x080d	illegal_r_flag
2143	HWERR.*0x080e	illegal_pid_flag
2144	HWERR.*0x080f	illegal_pt_flag
2150	HWERR.*0x0815	orb_tail_on_head_flag
2154	HWERR.*0x0904	overflow_flit_vc1
2158	HWERR.*0x0b24	resp_malformedpacket
2172	HWERR.*0x0d06	buffer_overflow
2173	HWERR.*0x0d0c	nic0_req_bubble
2186	HWERR.*0x0403	malformed

Table A.1: (cont.)

Template ID	**Regex Rule**	**Tag**
2187	HWERR.*0x0404	npes_err
2188	HWERR.*0x0405	already_ssid_alloc
2189	HWERR.*0x040c	ht bad nonposted request
2194	HWERR.*0x0a03	rx_vc_desc_inv
2195	HWERR.*0x0b05	lreq_uncortranserror
2196	HWERR.*0x0b0e	lreq_npes_violation
2197	HWERR.*0x0b11	rreq_misroutedpacket
2199	HWERR.*0x0b19	rreq_membounderror
2200	HWERR.*0x0b1a	rreq_writepermissionerror
2207	HWERR.*0x0b39	ssid_unexpectedrspssid
2208	dispatch.*\scurrent_state\saggregate_failures\s*****	link_failover_agg_failures
2209	dispatch.*\scurrent_state\shw_error.*****	hw_err_linkf_identifier
2210	dispatch.*\scurrent_state\sfinish	failover_finish
2211	CMD:\s*****\salive\s*****	cmd_alive
2212	CMD:\s*****\sunload\s*****	cmd_unload
2213	CMD:\s*****\sdownload\s******	cmd_download
2214	CMD:\s*****\snode_down\s*****	cmd_node_down
2215	CMD:\s*****\smodule_down\s*****	cmd_module_down
2216	CMD:\s*****\smodule_up\s*****	cmd_module_up
2217	CMD:\s*****\snode_up\s*****	cmd_node_up
2218	CMD:\s*****\scoldstart\s*****	cmd_cold_start
2219	dispatch:\scurrent_state\sdown_unused_links	down_unsused_links

Table A.1: (cont.)

Template ID	**Regex Rule**	**Tag**
2220	dispatch:\scurrent_state\sinit_new_links	init_new_links
2221	dispatch:\scurrent_state\scheck_init_new_blades	init_new_blades
2222	***ERROR***\sMMR\s or\sMemory\sRead\sFailed	mmr_memory_read_failed
2223	*****\sdispatch:\scurrent_state\sinitial\s*****	failover_initial
2224	TX\slanemask=7	lane_recovery_success
2225	_do_throttle:\ssending\sthrottle\smask	unthrottle_sys
2226	at\stop\sof\scongestion\slist\sfor	app_kill
2227	add_component_to_node_fault_list	nlink_victim_node_added
2228	cb_aggregate_timeout:\sonly\scritical\s node\serrors;\sskipping\s alive\s steps	nlink_id
2229	send_node_nmi_req:\snumber\sof\snodes\sfailed	nlink_node_disabling
2230	send_node_nmi_req:\ssending\sHALT\sNMI\sto\snode	nlink_node_disabled
2231	no\sreroute\srequired	no_reroute_required

APPENDIX B

CASE STUDY - LINK FAILOVER

This appendix summarizes the trace of successful and failed link failover procedures. Successful failover trace is given in Table B.1 and failed failover trace is given in Table B.2

Table B.1: Trace of a successful link failover from Blue Waters.

Time	**Action**	**Action Information**
2015-11-02 16:51:39	dispatch:	current_state: aggregate_failures
2015-11-02 16:51:49	dispatch:	current_state: hw_error
2015-11-02 16:51:50	dispatch:	current_state: alive
2015-11-02 16:52:23	dispatch:	current_state: check_alive
2015-11-02 16:52:23	dispatch:	current_state: interrupt_bounce
2015-11-02 16:52:24	dispatch:	current_state: check_interrupt_bounce
2015-11-02 16:52:24	dispatch:	current_state: alive2
2015-11-02 16:52:40	dispatch:	current_state: check_alive2
2015-11-02 16:52:40	dispatch:	current_state: set_alerts
2015-11-02 16:52:47	dispatch:	current_state: route_compute

Table B.1: (cont.)

Time	**Action**	**Action Information**
2015-11-02 16:52:47	Executing Command	/opt/cray/hss/default/bin/rtr -S
2015-11-02 16:54:41	dispatch:	current_state: check_route_compute
2015-11-02 16:54:41	dispatch:	current_state: quiesce
2015-11-02 16:54:48	dispatch:	current_state: check_quiesce
2015-11-02 16:54:48	dispatch:	current_state: quiesce_drain
2015-11-02 16:54:52	dispatch:	current_state: switch_netwatch
2015-11-02 16:54:56	dispatch:	current_state: check_switch_netwatch
2015-11-02 16:54:56	dispatch:	current_state: down_unused_links
2015-11-02 16:55:32	CMD:	get_class
2015-11-02 16:55:32	CMD:	get_nids
2015-11-02 16:55:32	CMD:	gather_partition_info
2015-11-02 16:55:32	CMD:	check_partition_info
2015-11-02 16:55:32	CMD:	gather_user_components
2015-11-02 16:55:32	CMD:	gather_partition_components
2015-11-02 16:55:32	CMD:	cross_check
2015-11-02 16:55:32	CMD:	interrupt_sysd

Table B.1: (cont.)

Time	Action	Action Information
2015-11-02 16:55:32	CMD:	alive
2015-11-02 16:55:32	CMD:	gather_mezz_up
2015-11-02 16:55:32	CMD:	link_down_unused
2015-11-02 16:55:33	dispatch:	current_state: check_down_unused_-links
2015-11-02 16:55:33	dispatch:	current_state: down_drain
2015-11-02 16:55:36	dispatch:	current_state: route_install
2015-11-02 16:55:36	Executing Command	/opt/cray/hss/default/bin/rtr -P
2015-11-02 16:55:42	dispatch:	current_state: check_route_install
2015-11-02 16:55:42	dispatch:	current_state: unquiesce
2015-11-02 16:55:49	dispatch:	current_state: check_unquiesce
2015-11-02 16:55:49	dispatch:	current_state: finish
2015-11-02 16:55:49	switch_warm-swap_led:	switching warmswap LED blink for blade C9-7c2s1
2015-11-02 16:55:50	set_warm_swap_-err:	appending warm swap text: Link recovery operation was successful

Table B.2: Trace of a failed link failover from Blue Waters.

Time	Action	Action Information
2014-11-08T02:21:50	INFO:	c20-8c0s7g0l52 ***ERROR*** HSN ASIC LCB lanes(s) reinit failed

Table B.2: (cont.)

Time	**Action**	**Action Information**
2014-11-08T12:52:41	dispatch:	current_state: aggregated_failures
2014-11-08 12:52:51	dispatch:	current_state: hw_err
2014-11-08 12:53:28	dispatch:	current_state: finish
2014-11-08 12:53:28	INFO:	handling: failed links in 37.76 seconds
2014-11-08 12:53:28	***ERROR***:	recovery operation failed; error 5
2014-11-08 12:53:28	dispatch:	initial
2014-11-08 12:53:28	dispatch:	aggregate_failures
2014-11-08 12:53:39	dispatch:	hw_error
2014-11-08 12:54:09	dispatch:	finish
2014-11-08 12:54:09	INFO:	handling failed links in 30.19 seconds
2014-11-08 12:54:09	***ERROR***:	recovery operation failed; error 5
2014-11-08 12:54:09	dispatch:	current_state: initial
2014-11-08 12:54:09	dispatch:	current_state: aggregated_failures
2014-11-08 12:54:19	dispatch:	current_state: hw_err
2014-11-08 12:54:49	dispatch:	current_state: finish
2014-11-08 12:54:49	INFO:	handling failed links in 30.04 seconds

Table B.2: (cont.)

Time	**Action**	**Action Information**
2014-11-08 12:54:49	***ERROR***:	recovery operation failed; error 5
2014-11-08T12:54:49	dispatch:	current_state: initial
2014-11-08 12:54:49	dispatch:	current_state: aggregated_failures
2014-11-08 12:54:59	dispatch:	current_state: hw_err
2014-11-08 12:55:17	dispatch:	current_state: alive
2014-11-08 12:55:50	dispatch:	current_state: check_alive
2014-11-08 12:55:51	dispatch:	current_state: alive2
2014-11-08 12:55:54	dispatch:	current_state: check_alive2
2014-11-08 12:55:54	dispatch:	current_state: set_alerts
2014-11-08 12:55:54	dispatch:	current_state: finish
2014-11-08 12:55:54	INFO:	handling failed links in 54.5 seconds
2014-11-08 12:55:54	***ERROR***:	recovery operation failed; error 11
2014-11-08 12:55:54	dispatch:	current_state: initial