Study of Interconnect Errors, Network Congestion, and Applications Characteristics for Throttle Prediction on a Large Scale HPC System[☆]

Mohit Kumar[a,*], Saurabh Gupta[b], Tirthak Patel[c], Michael Wilder[d], Weisong Shi[e], Song Fu[f], Christian Engelmann[a], Devesh Tiwari[c]

[a]*Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, TN 37831-6164, USA*
[b]*Intel Labs, Bangalore, India*
[c]*Northeastern University, 360 Huntington Avenue,Boston, Massachusetts 02115, USA*
[d]*Leidos, Oak Ridge, TN 37830, USA*
[e]*Wayne State University, 5057 Woodward Ave, Room 2218, Detroit, MI 48202, USA*
[f]*University of North Texas, Denton, Texas 76203, USA*

Abstract

Today's High Performance Computing (HPC) systems contain thousand of nodes which work together to provide performance in the order of petaflops. The performance of these systems depends on various components like processors, memory, and interconnect. Among all, interconnect plays a major role as it glues together all the hardware components in an HPC system. A slow interconnect can impact a scientific application running on multiple processes severely as they rely on fast network messages to communicate and synchronize frequently. Unfortunately, the HPC community lacks a study that explores different interconnect errors, congestion events and applications characteristics on a large-scale HPC system. In our previous work, we process and analyze interconnect data of the Titan supercomputer to develop a thorough understanding of interconnects faults, errors, and congestion events. In this work, we first show how congestion events can impact application performance. We then investigate application characteristics interaction with interconnect errors and network congestion to predict applications encountering congestion with more

[☆]This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

[*]Corresponding author

Email addresses: `kumarm1@ornl.gov` (Mohit Kumar), `saurabg@gmail.com` (Saurabh Gupta), `patel.ti@husky.neu.edu` (Tirthak Patel), `michaelallenwilder@gmail.com` (Michael Wilder), `weisong@wayne.edu` (Weisong Shi), `Song.Fu@unt.edu` (Song Fu), `engelmannc@ornl.gov` (Christian Engelmann), `devesh.dtiwari@gmail.com` (Devesh Tiwari)

than 90% accuracy.

Keywords: Cray, Gemini, Interconnect, Titan, Errors

1. Introduction

High Performance Computing (HPC) systems consist of tens of thousand nodes that have multiple Central Processing Unit (CPU) and Graphical Processing Unit (GPU), shared or distributed memory, and back-end storage. All these components utilize an advance interconnect network to communicate with each other for running scientific parallel applications at petaflops speed. An advance interconnect supports minimum system latency and maximum throughput and scalability.

A scientific parallel application runs on multiple processors on different nodes that communicate continuously to share data with each other. A slow speed interconnect can significantly increase an application execution time as it can cause a processor to sit idle and wait for data from other processors. A low resilient interconnect can cause an application to crash due to interconnect errors. Therefore, interconnect can play a major role in application performance. Performance of an interconnect depends on network topology, routing methods, flow-control algorithm, resilience mechanism, congestion reaction mechanism, and communication pattern of applications.

Unfortunately, the HPC community lacks a study that details different interconnect errors, congestion events, and applications characteristics on a large-scale HPC system. In previous work [1], we study the interconnect resilience, congestion events and application characteristics on Titan, one of the fastest open-science supercomputer in the world. We used daemon services on Titan to collect useful interconnect resilience, congestion events and applications characteristics data for over a year. We process and examine this data to develop a thorough understanding of interconnect faults, errors, congestion events, and application characteristics. In this work, we further explore the application characteristics and their interaction with interconnect errors and network congestion events. Our analysis addresses the following concerns:

- How network congestion impacts application performance?
- What are the major interconnect faults and errors?
- What are the key characteristics of different interconnect errors and network congestion events?
- What is the interaction between interconnect errors, network congestion, and application characteristics?
- What are the application characteristics for congestion causing and bandwidth-heavy applications?

- Can network and application characteristics predict the applications encountering network congestion events?

This study exploits various daemons, such as netwatch and Network Link Recovery Daemon (nlrd) that use Memory Mode Register (MMR), for collecting and logging interconnect related events. However, analysis of this data presents several challenges. First, the collected data is highly noisy and hence, needs to be filtered discreetly for accurate analysis. The noise is in form of redundant, missing, and disorder values. Second, the log patterns differ for the same type of events across different logging mechanisms as different daemons have different logging formats. This requires the development of unified format types for different events. Finally, as data is distributed across several nodes and storage locations, it requires performing multi-source analytics to ensure consistency and accuracy. The following are the highlights of our analysis:

- **Application Performance:** We compare multiple executions of applications with and without network congestion. Network congestion results in high application execution time.
- **Interconnect Errors:** The magnitude of interconnect errors is very high. These errors are distributed unevenly across different types of links within and across cabinets.
- **Spatial Correlation:** Some interconnect errors have a strong spatial correlation among them. On the other hand, some errors show counter-intuitive patterns.
- **Congestion Events:** Network congestion events are highly frequent and bursty. These events are not homogeneously distributed across blades.
- **Application Characteristics:** We analyze application characteristics of the top five bandwidth applications. Same application runs show different processes per node count, CPUs count, execution time and user id.
- **Throttle Prediction:** We extract network and application events which has a relation to network congestion or throttling. Using these events, we predict whether an application will encounter network congestion events with more than 90% accuracy.

In the following sections, we provide our previous and new analysis of the interconnect, network congestion and application data to explore the above and previous insights in detail. Given the lack of field data and analysis on interconnects, we believe our study addresses an important topic and would be useful for current and future HPC systems.

2. Background

This study primarily studies data from the Titan supercomputer; however, it's insights are applicable to other supercomputers as well. Titan is a 27.1

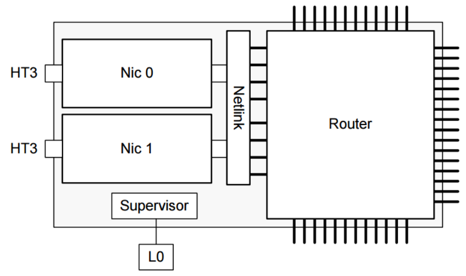


Figure 1: ASIC block diagram

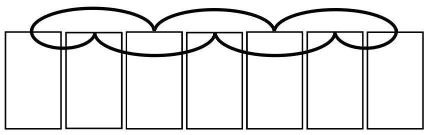


Figure 2: Practical folded torus-implementation of 3D-torus network topology.

petaflop supercomputer consisting of 18,688 compute nodes, each with a 16-Core AMD Opteron CPU and an NVIDIA Tesla K20x GPU. It has a total system memory of 710 TB. The supercomputer is divided into 200 cabinets in 25 rows and 8 columns. Each cabinet consists of three cages and each cage has eight blades. Each blade consists of two application specific integrated circuits (ASICs). Each ASIC has two network interface controllers (NICs) and a 48-port router. Each NIC within an ASIC is attached to one node using a HyperTransportTM 3 link [2].

The block diagram of an ASIC is shown in Fig. 1. The Netlink block connects the two NICs to the Router. The Netlink distributes the traffic and handles changes in the bandwidth between the two NICs and the Router [2]. The supervisor block connects ASIC to an embedded control processor(L0) which is connected to the System Management Workstation(SMW) through the Cray Hardware Supervisory System(HSS) network.

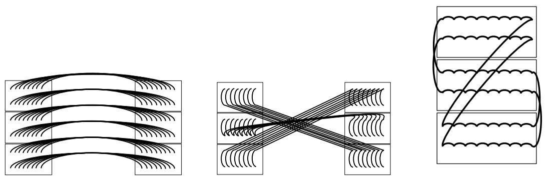


Figure 3: Link connections in X, Y, and Z direction.

2.1. Titan Network Architecture

Titan follows a 3D torus topology using the Cray Gemini Interconnect in which each ASIC is connected to six of its nearest neighbors in X+, X-, Y+, Y-, Z+, and Z- dimensions. The X, Y, and Z dimensions track the rows, columns, and blades, respectively [3]. Nodes that are close physically may not be close topologically as Cray follows a "folded torus" architecture to minimize the maximum cable length (as shown in Fig. 2). In the X and Y directions, every other cabinet is directly connected together with "loopback" cables. In the Z dimension, the uppermost chassis is connected to the lowermost chassis (Fig. 3).

In a 3D torus design, each ASIC is connected to the network using 10 torus connections, two each in X+, X-, Z+, Z-, and one each in Y+ and Y- [2]. Each torus connection has four links where each link is composed of 3 lanes. Therefore, each connection consists of 12 lanes, providing 24 lanes in the X and Z dimensions, and 12 lanes in the Y dimension. A lane provides bi-directional communication between two ports.

2.2. Interconnect Resilience

The Gemini Interconnect is tolerant to various types of failures and errors [3]. It supports 16-bit packet Cycle Redundancy Checks (CDCs) to protect packets at each ASIC it passes through before reaching the final ASIC, packets on the receiving ASIC and packets transitioning from the router to the NIC. Link control Blocks (LCBs) on ASICs implement a sliding window protocol to provide reliable delivery of packets. Memory on each ASIC is protected using Error-Correcting Codes (ECCs). ASICs can withstand lane failures as long as there is at least one functional lane in a link. Whenever a lane fails, it is deactivated and the traffic is balanced over the remaining lanes. In such situations, the network operates in a degraded mode. The interconnect tries to reinstate the failed lane to restore the full bandwidth within a user-specified time limit.

The lanemask value determines the current state of the lanes in a link. It is a three-bit number corresponding to the three lanes in a link. A lanemask value of 7 means that all lanes are working. A lanemask value of 3, 5 and 6 means that one lane has failed. A lanemask value of 1, 2 or 4 means that two lanes have failed. A lanemask value of 0 means that all lanes have failed. When all three lanes in a link fail and the lanes are not recovered in the configured number of attempts, the link is marked as inactive and the link failover or warm swap protocol (Replace or remove faulty components without turning the system off) is triggered [4]. When these protocols are executed, the Cray nlrd on the SMW quiesces the network traffic, computes new routing tables, and assigns them to each ASIC.

2.3. Network Congestion

A network becomes congested when there is more data in the network than it can accommodate for. The HSS software manages the network congestion into the network whenever necessary. Two daemons: one on the SMW (xtnlrd), and one on the blade controller (bcbwtd), can handle network congestion by

Table 1: Summary of Netwatch events

Events	**Count**	**Percent**
All	9367031.0	100.0
Mode Exchanges	5065536.0	54.08
RX	2146693.0	22.91
TX	2144221.0	22.89
Link Inactive	7280.0	0.08
Bad Send EOP Error	2548.0	0.03
Send Packet Length Error	366.0	0.004
Routing Table Corruption Error	200.0	0.002
HSN ASIC LCB lane(s) reinit failed Error	187.0	0.002

limiting the aggregate injection bandwidth across all compute nodes to less than the ejection bandwidth of a single node (also known as throttling).

2.4. Dataset

The collected dataset consists of the network logs from January 2014 to January 2015. The interconnect metadata is collected by two daemons: xtnetwatch and xtnlrd. The xtnetwatch daemon logs the system High-Speed Network (HSN) faults for LCBs and router errors. These logs include details about the transmitting packets, receiving packets, mode exchanges, lanemask, link inactive and different interconnect failures data for particular nodes, along with a timestamp. The xtnetwatch data is summarized in Table 1. Events detail are presented during analysis in the later sections.

When the percentage of time that traffic tries to enter the network is stalled more than a high water mark threshold, the xtnlrd daemon produces log files that include various collection information. It also collects a list of the top 10 applications sorted by the aggregate ejection bandwidth whenever a congestion protection event occurs. Moreover, it estimates the top 10 most congested nodes sorted by ejection flit counts whenever a congestion protection event occurs. In both cases, it includes the job characteristics of the applications running on those nodes, including APID, number of nodes, the user ID, and the application name.

We also gathered application data to extract different application characteristics. This data include application start time, stop time, application id, user id, node list, and number of CPUs.

3. Impact of Network Congestion

One can argue the impact of congestion events on application execution time. We conduct a study to find out how congestion events can impact application

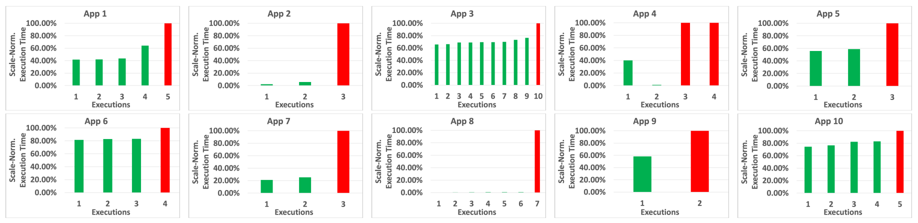


Figure 4: Applications execution times while encountering and not encountering congestion events.

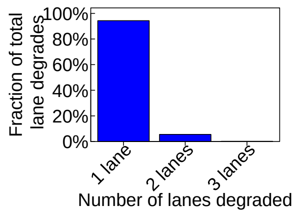
execution events. We start by finding applications that were executed on the same nodes. For each application running on specific nodes, we check the nodes for congestion events for the execution time of the application. We then filter out those applications that encounter congestion events. Finally, we compare the execution time of the applications which encounter and don't encounter congestion events in Fig. 4. The x-axis denotes the different executions of the same application. The y-axis denotes the scale-normalized execution time of different executions in percentage. The application names are denoted using numbers as they can be business-sensitive. The red color shows the application execution time when the applications encounter congestion events. The green color shows the application execution time without any congestion events. The application execution time for App 8 while encountering congestion events were up to 99 times more than the normal execution. This shows that the congestion events can have a significant impact on an application execution time while running on a large-scale HPC system. There is a possibility that other factors can affect the execution time as congestion events can be the result of hardware or software errors. A more in-depth study considering other factors can provide more insights into how congestion affects application execution time.

Takeaway 1: *Congestion events have significant impact on application execution time.*

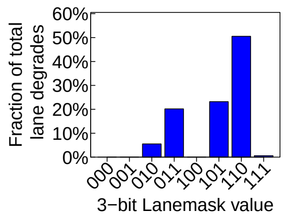
4. Analysis of Interconnect Errors

In this Section, we characterize and analyze different types of interconnect faults and errors. First, we quantify and characterize lane degrade events. A lane degrade event is triggered when any one of the three lanes in a link goes down. This has a negative impact on the application performance and may cause network congestion. Unfortunately, these events occur with a very high frequency. We observed that one lane degrade event take place at a high rate of one event per minute. Despite the high frequency and negative consequences of these events, the characterization of these events in an HPC system is not available to researchers, users, and system operators.

Fig. 5a shows the frequency of different types of lane degrades. We observe that in more than 90% cases only one lane in a link is degraded. Two lanes are degraded in less than 10% of the cases. Three lanes are degraded relatively



(a)



(b)

Figure 5: Frequency distribution of lane degrade (a) and lanemask values (b) over one year.

less frequently (<1%). When all three lanes are in a degraded state, the link is declared inactive (or failed), and an alternate route is computed for packets. While link inactive or failed events happen relatively less frequently, they do occur about 28 times per day on average, and cause more disruption than single or double lane degrades.

Takeaway 2: *Even when three lane degrades happened very infrequently as compared to one lane degrade, link inactive occurs 28 times per day on average.*

Fig. 5b shows the frequency of lanemask values for every instance of lane degrade events. We note that a lanemask bit value of 0 indicates that the corresponding lane is degraded. For example, a lanemask value of 5 (binary value 101) indicates that the middle lane is degraded. We observe that the frequency of lanemask values indicates that even single lane failures vary significantly. Lanemask value 110 is two times more frequent than lanemask values 101 and 011. Interestingly, for two lane failures, the corner lanes failing together (010) is more likely than adjacent lanes failing together (001 and 100).

In the absence of per-lane and per-link based utilization data, we hypothesize that lane failure location indicates the utilization and load pattern on links. Given this, our results indicate that the load among lanes within a link may vary significantly. This finding should encourage designers to balance the load more homogeneously and not overload the rightmost lane. This insight could also be exploited for power optimization in interconnect links where rightmost lanes need not to be switched-on at all times.

Takeaway 3: *Load among lanes within a link is not homogeneous.*

Next, we plot the relative frequency distribution of lane degrades over time in Fig. 6. We make two important observations. First, lane degrades are not limited to a specific time period, instead they happen continuously over time. Second, one may expect that the high single-lane degrade events will lead to an increase in the count of two-lane degrades and link failures. However, our field data suggests that this hypothesis is not necessarily true. For example, peaks in two-lane degrades are not necessarily during the high intensity of one-lane degrades. Later, we also investigate deeper to understand the correlation between network congestion and the period of high intensity of lane degrades.

Takeaway 4: *Lane degrades happened continuously temporally where different lane degrades doesn't impact each other.*

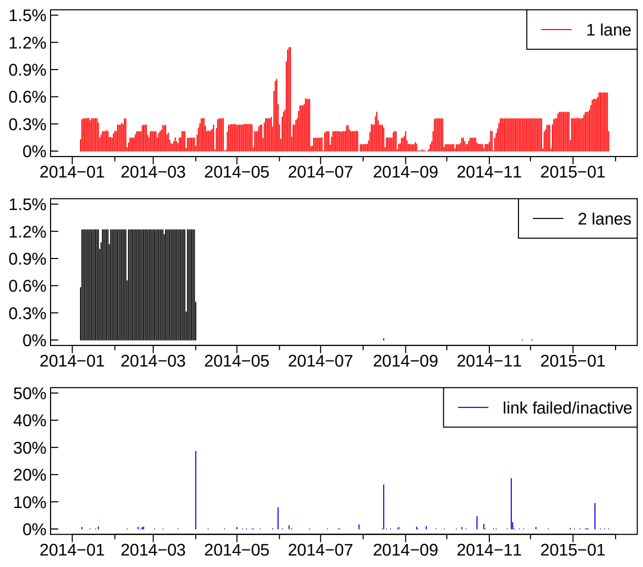
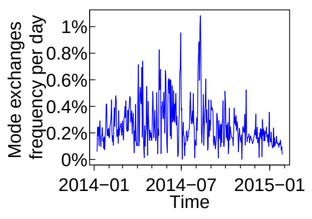


Figure 6: Frequency of different types of lane degrades over one year.

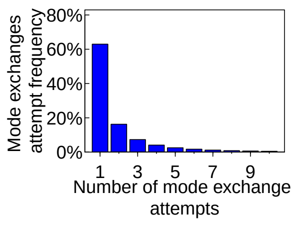
When a lane goes down, the network resiliency mechanism attempts to bring the lane back up via multiple repair events, called mode exchanges. Fig. 7a shows the frequency of mode exchange events over time and Fig. 7b shows the number of mode exchange attempts before a lane is brought up successfully. As expected, the frequency of mode exchange events over time is similar to that of lane degrades. System operators of Titan have set the threshold for the number of attempts allowed to restart a lane to 256. Interestingly, our result shows that more than 85% of the lanes can be restored in three or fewer attempts. Furthermore, more than 99% of the lanes can be restored within 10 attempts.

Takeaway 5: *Temporal frequency of mode exchanges is similar to that of lane degrades.*

Next, we attempt to understand how lane degrade and link inactive/failed events are distributed across the system spatially. Fig. 8 shows the lane degrade events count for links in and across cabinets. First, we observe that several hot spots exist for lane degrade events in the system. We conduct the Kolmogorov-Smirnov test (K-S test) to test whether our sample of spatial distribution of lane degrades per cabinet has a uniform distribution [5]. The test results show D-statistic = 1, p-value = 2.2e-16. For our sample size of 200 cabinets, the critical D-value for a 0.05 level of significance is 0.0960, and therefore the null hypothesis (i.e., the sample is taken from a uniform distribution) can be rejected. This shows that the spatial distribution of lane degrades per cabinet is significantly different than uniform. This behavior can be a combination of factors like HW instance variation, external transient effects, overloading of links, uneven usage, and complex interaction between applications and interconnect network,



(a)



(b)

Figure 7: Daily frequency of mode exchanges to repair lane degrades (a) and number of mode exchange attempts before successful recovery of the lane (b) over one year.

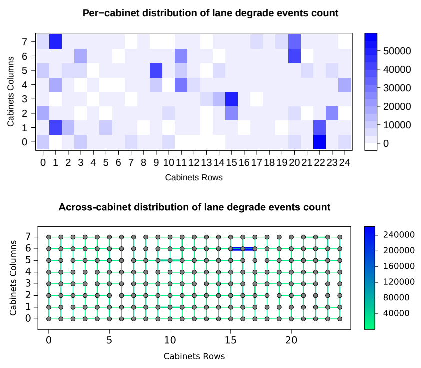


Figure 8: Spatial distribution of lane degrades count inside and across cabinets over one year. Due to the folded 3D-torus design, cross-cabinet links connect to alternate cabinets.

although accurate root-cause analysis is not possible.

Takeaway 6: *Spatial distribution of lane degrade events is not uniform.*

Interestingly, we note that the hot spots for links contained within the cabinet are not the same as the hot spots for links crossing cabinet boundaries. Second, when we compare lane degrade hot spots with the hot spots of link inactive errors (Fig. 8 vs. Fig. 9), we find that they do not necessarily match. This also explains why their high intensity periods do not match (Fig. 6). This indicates that it is not possible to determine the location of link inactive/failed errors by only observing the time and location of lane degrade events.

Takeaway 7: *Spatial and temporal distribution hot spots of link inactive/ failed and lane degrades events are not same.*

Next, we investigate other interconnect errors: Bad Send EOP error, Send Packet Length error, Routing Table Corruption error, and HSN ASIC LCB lane reinit failed error.

Bad Send EOP error: Each packet in Gemini Interconnect has a single phit end-of-packet that contains the status bits for error handling [2]. If a packet is corrupted, the end-of-packet is marked as bad and the packet will be discarded at its destination.

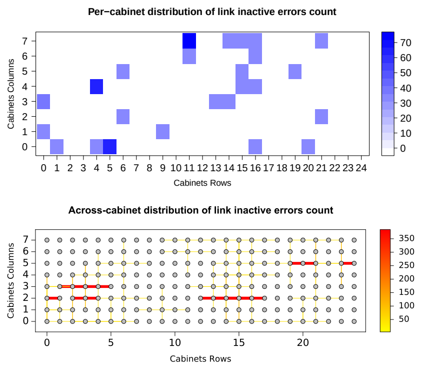


Figure 9: Spatial distribution of link inactive errors count inside and across cabinets over one year. Due to the folded 3D-torus design, cross-cabinet links connect to alternate cabinets.

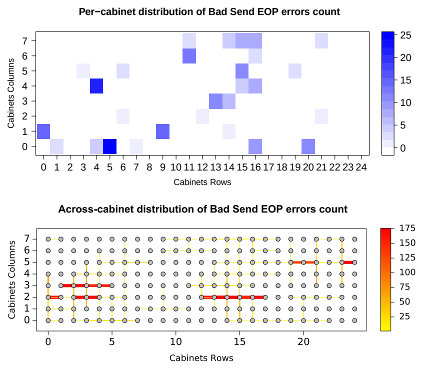


Figure 10: Spatial distribution of Bad Send EOP errors count inside and across cabinets over one year.

Send Packet Length error: Send Packet Length error occurs when the length of a packet does not match with the expected length value at destination.

Routing Table Corruption error: A routing table is a data table stored in a router that contains the information necessary to forward a packet along the best path toward its destination. When a packet is received, a network device examines the packet and matches it to the routing table entry providing the match for its destination. The table then helps in guiding the packet to the next hop on its route across the network. A routing table corruption results in link failure and eventually causes network congestion [4]. Each packet in Gemini Interconnect contains information about the originating node and the destination node. Each packet uses a 18-bit address to uniquely identify a node in Titan. This 18-bit address has 16-bit ASIC identifier and 2-bit for specifying NICs and node.

HSN ASIC LCB lanes reinit failed error: This error occurs when all the 256 attempts to bring up a downgraded lane are exhausted.

Fig. 10-13 show the frequency distribution of these errors in- and across-

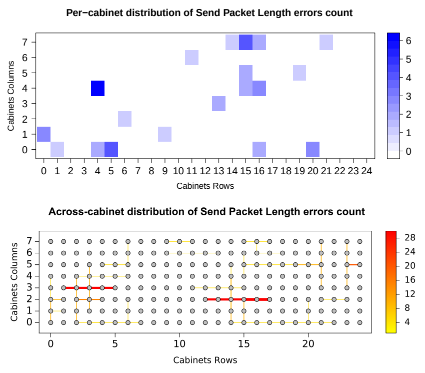


Figure 11: Spatial distribution of Send Packet Length errors count inside and across cabinets over one year.

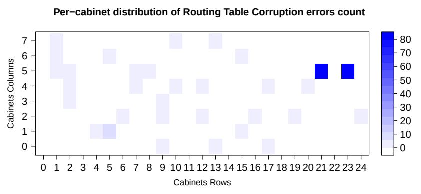


Figure 12: Spatial distribution of Routing Table Corruption errors count inside the cabinets over one year.

cabinets. First, we observe that these errors also show hot spots in- and across-cabinets, although we found that the magnitude of these errors is relatively small. For example, Routing Table Corruption error occurs only 200 times while HSN ASIC LCB lane(s) reinit failed error happens only 187 times throughout the entire observation period.

On deeper investigation, we found that most of these errors are highly correlated with link inactive/failed errors. Table 2 shows the correlation of these interconnect errors with link inactive errors. This indicates that link inactive errors can be used to predict other interconnect errors. We also found that more than 80% of link failed errors lead to Bad Send EOP, Send Packet Length, and Routing Table Corruption errors. We also found that HSN ASIC LCB lane(s) reinit failed error has a weak correlation with link failed errors. This can be

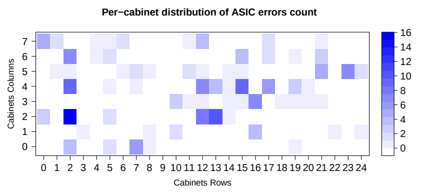


Figure 13: Spatial distribution of ASIC errors count inside the cabinets over one year.

Table 2: Correlation factor between link inactive and other interconnect errors.

Errors	**Link Inactive**
Bad Send EOP Error	0.99
Send Packet Length Error	0.96
Routing Table Corruption Error	0.84
ASIC Error	0.04

explained by our previous findings where it showed that lane degrades and link failed errors are not correlated and ASIC errors are an outcome of failed repair attempts of lane degrades.

Takeaway 8: *Interconnect errors shows a high correlation with link inactive/failed events.*

5. Analysis of Network Congestion

Understanding network congestion in conjunction with interconnect errors is important since it is likely that one may cause the other. A daemon on the compute cluster monitors the percentage of time that network tiles [3] are stalled due to increased traffic or other reasons. When these values cross a set threshold, the daemon communicates this data (network throttle events) to the xtnlrd daemon running on the SMW. After the congestion subsides, the daemon again passes this information to the SMW.

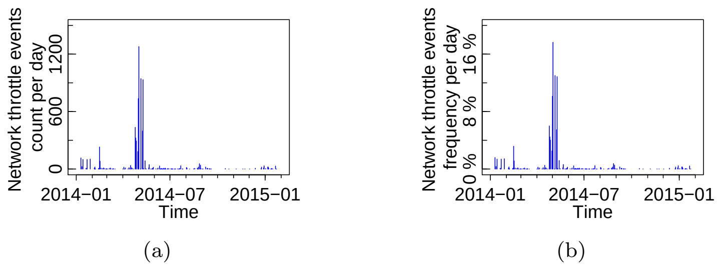
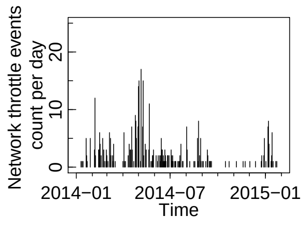
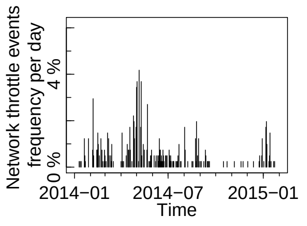


Figure 14: Count of network throttle events (a), and relative frequency of throttle events (b) over one year.

In this section, we first attempt to understand the characteristics of network throttle events. Fig. 14a plots the network throttle events over time. We note that a large fraction of throttle events occur in a short period of time. We also note that each throttle event is typically 20-30 seconds, but it can also last up to a few minutes depending on the magnitude of the congestion observed. As shown in Fig. 14b, network throttle events can be quite bursty. An application that is causing network congestion can induce multiple throttles in a very



(a)



(b)

Figure 15: Network throttle events with 1-hour filtering (a), and relative frequency of throttle events with 1-hour filter (b) over one year.

short amount of time (< 20 mins). Fig. 15 shows the network throttle events over time, counting only one event at maximum per hour. We experimented with multiple time windows and found that a 1-hour time window removes the skewness. However, this type of time window filtering cannot completely remove the skewness since a long-running communication-intensive application may cause multiple throttle events over multiple hours. For example, Fig. 16 shows that without 1-hour filtering the mean time between throttle events is less then 1 minute, with 90% of the events occurring within the first hour of the preceding throttle event. Even when we apply 1-hour filtering, the meantime between throttle events is approximately 22 hours. Therefore, to better understand the characteristics of network throttle events we analyze our subsequent results without any time window based filtering and with 1-hour time window based filtering. We try to find out whether the number of jobs running per day have any correlation to throttle events. However, as we can see in Fig. 17, the throttle events relative frequency Fig. 14b is not correlated to the number of jobs running per day relative frequency. The Spearman correlation coefficient was found to be very weak (-0.01). In the second half of the year, we see bursts in the number of jobs which may be a result of users running the same applications multiple times to achieve the best performance in ACM Gordon Bell prize [6].

Takeaway 9: *Network throttle events are very bursty and are not dependent on number of jobs running at a specific day.*

From nlrd data, we calculated mean time between link recovery and warm swap (Fig. 18). It indicates that link recovery and warm swap procedures take place at a high rate. Naturally, one may hypothesize that the lane degrades/failures may induce the network throttle events or vice versa. Therefore, we investigate the possibility of temporal correlation between the time series of throttle events and interconnect errors, in particular lane degrades and link failed events. We found the Spearman correlation coefficient to be very weak (0.03). This result indicates that lane degrades/failures alone cannot be used to predict throttling events. While lane degrades/failures can cause the performance degradation and variability, lane degrades/failures do not always immediately cause significantly high network congestion to trigger network throttling.

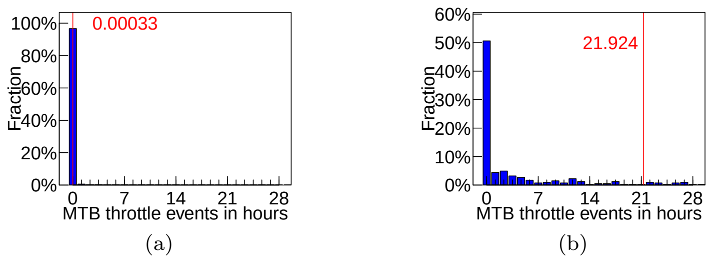


Figure 16: Mean time between network throttling events without filter (a) and with 1-hr filter (b) over one year.

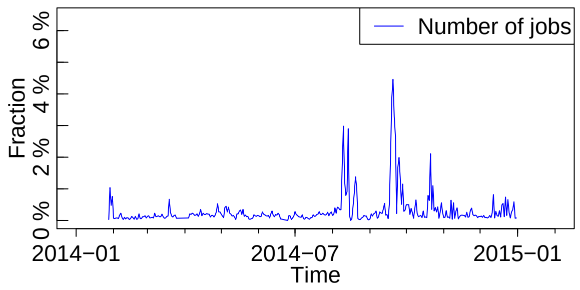


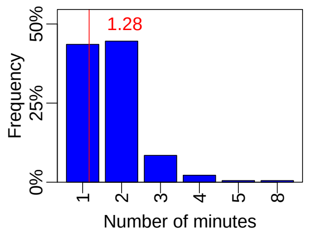
Figure 17: Number of jobs running per day relative frequency over one year.

Takeaway 10: *Temporally, lane degrades and link failed events shows very low correlation with throttle events.*

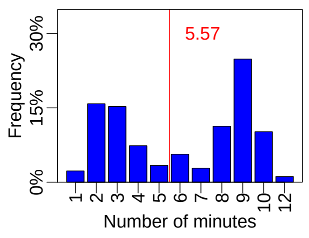
Next, we plot the heatmap of compute blades that were throttled due to these network throttle events. Fig. 19 shows the heatmap without any filtering and Fig. 20 shows the same heatmap with 1-hour filtering. As expected, we observe that not all blades are throttled equally over the period of observation. Interestingly, hot spots remain similar even after applying filter. We again conduct the K-S test to test whether our sample of the spatial distribution of blades throttle events per cabinet without and with one hour filter has a uniform distribution. The test results show D-statistic = 1, p-value = 2.2e-16 in both cases. For our sample size of 200 cabinets, the critical D-value for a 0.05 level of significance is 0.0960, and therefore the null hypothesis (i.e., the sample is taken from a uniform distribution) can be rejected. This shows that the spatial distribution of blade throttled events per cabinet is significantly different than uniform. As a next step in our analysis, we want to investigate the role that congestion information at the node and application levels can play in improving our understanding of congestion behavior at the blade-level.

Takeaway 11: *Spatial distribution of blade throttle events per cabinet is not uniform.*

Node-level congestion data provides information about the nodes which are heavily congested at the time of throttling. It lists the top 10 heavily conges-



(a)



(b)

Figure 18: Mean time between link recovery (a) and warm swap (b) over one year.

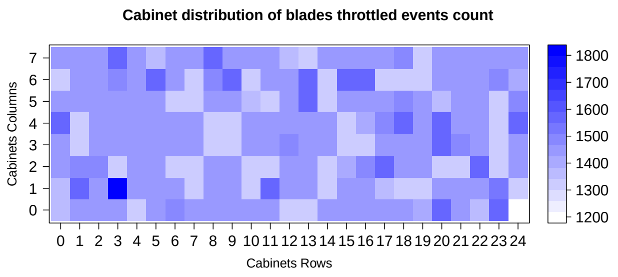


Figure 19: Spatial distribution of throttled blades count without filter over one year.

tion node based on ejection bandwidth rate for every throttle event. Along with node ID, several other information is included such as the application running on that node, user ID of the application, number of nodes allocated to the application, ejection bandwidth at the node level. Fig. 21 and 22 show the spatial distribution of congested nodes in the Titan supercomputer for the no filter and 1-hour filter cases, respectively. We make two observations. First, some nodes are much more congested than others as shown by the uneven distribution of congested nodes. This is because the applications creating significant network traffic may be repeatedly getting scheduled on the same nodes. Second, apply-

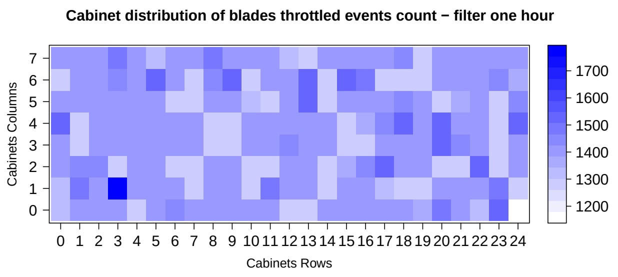


Figure 20: Spatial distribution of throttled blades count with 1-hr filter over one year.

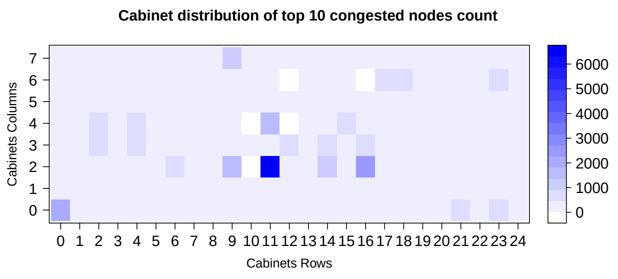


Figure 21: Spatial distribution of congested nodes count without filter over one year. Note that the top 10 congested nodes are calculated for each throttle event. This plot is aggregated over all throttle events.

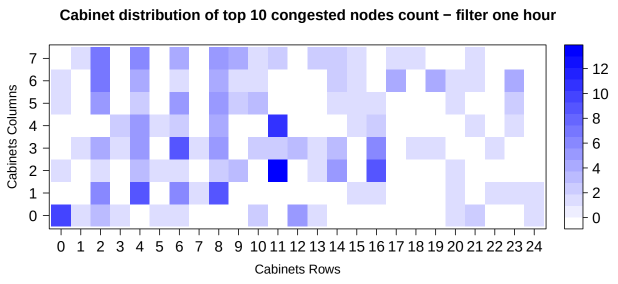
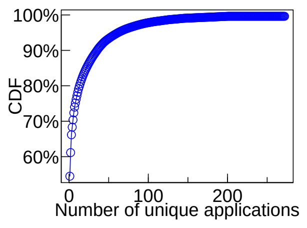


Figure 22: Spatial distribution of congested nodes count with 1-hr filter over one year.

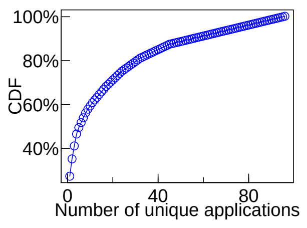
ing 1-hour filtering shows that the spatial distribution evens out compared to Fig. 21. However, interestingly it continues to show uneven distribution; skewed toward the left part of the supercomputer. This indicates that communication-intensive applications are not scheduled evenly across the cabinet. Applications scheduled on the left part are likely to see more performance impact due to network congestion. Therefore, applications whose performance is sensitive to interconnect-latency could potentially benefit from getting scheduled on the right part of the cluster.

Takeaway 12: *Spatial distribution of congested nodes shows skewness toward the left part of the supercomputer.*

We also calculated the correlation coefficient between the spatial distribution of congested nodes and lane degrades/failures. The Spearman correlation coefficient was close to zero (0.01); the nodes with high ejection bandwidth are not strongly correlated with the interconnect errors. This result was expected since we found the correlation between throttle events and interconnect errors were not high. However, surprisingly, there is a low correlation between the heatmap of congested nodes and throttled blades (Spearman correlation 0.01). This is because congested node information is collected after the throttle command has been issued, so it may not capture the nodes which actually caused the congestion and induced throttling. This also indicates that the aggregate

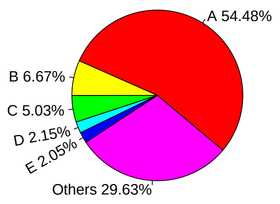


(a)

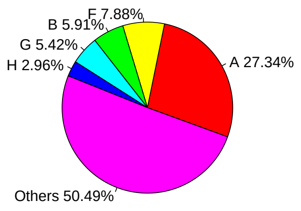


(b)

Figure 23: Cumulative distribution of unique applications over congested node events without filter (a), and with 1-hr filter (b) over one year.



(a)



(b)

Figure 24: Fraction of top 5 congestion-causing and other applications without filter (a), and with 1-hour filter (b) over one year.

network traffic at the blade can be potentially different than individual node-level traffic. Future network performance tools should focus on building more accurate and fine-grained tools that can detect the root cause in real time.

Takeaway 13: *Spatially, lane degrades events shows very low correlation with throttle events.*

Next, we analyze the characteristics of applications running on all congested nodes. First, we plot the frequency of unique applications that were running on congested nodes when the throttling events occurred. Fig. 23 shows that only a few applications tend to dominate. We refer to these applications as congestion-causing applications in our discussion; however, we note that these applications may not be necessarily responsible for increasing the congestion that eventually resulted in the network throttling event. For example, 5 applications alone appear in more than 70% of congested node reporting events (Fig 24), while more than 250 unique applications are logged in total across all congested node reporting events. Interestingly, when 1-hour filtering is applied, the number of unique applications decreases significantly. The top 5 most frequently occurring applications appear only in approx. 50% of congested node reporting events (Fig 24). Total number of unique applications go down from 250 to 90. Reduction in the number of unique applications clearly indicates that when multiple throttle events occur in the small time period, they are not because of the same application. In fact, it turns out that within a 1-hour time window, multiple unique applications can cause nodes to be highly congested. We also see the same results on a per-user basis with and without filtering (Fig. 25). These results confirm that applications and users can work as proxies for each other.

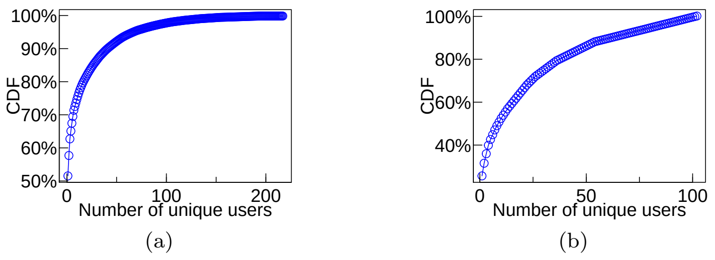


Figure 25: Cumulative distribution of unique users over congested node reporting events without filtering (a), and 1-hour filtering (b) over one year.

Takeaway 14: *Same applications and users don't cause multiple throttle events over a small time period.*

Next, we analyze the job size of these applications. We limit our discussion to the top 10 most frequent congestion-causing applications. As application names can be business-sensitive, we identify them with English letters. Fig. 26 shows the job size distribution of top 10 applications that appear most frequently on congested nodes. We make several interesting observations. First, most of the applications tend to run on the same number of nodes every time they appear in the congested node reporting events. For example, applications A, B, and C run on 4000, 2, and 45 nodes, respectively, for more than 90% of the time that they appear in the congested node reporting events.

Moreover, counter-intuitively, the job sizes of these applications are relatively small. For instance, 7 out of the 10 applications most frequently have a job size of less than 512 nodes. In fact, 5 applications have the most frequent job size of less than 128 nodes. In such cases, the many-to-few communication pattern can be responsible for congesting the nodes (high ejection bandwidth). Therefore, only focusing on large scale jobs for identifying culprit applications is an ineffective strategy. Our results show that node congestion is caused by small-scale applications in real-world scenarios.

Takeaway 15: *Congestion causing applications run on same number of nodes and are generally small in job size.*

Next, we want to extend our understanding of communication-intensive applications and their job size distributions. On every throttle event, the nlrd daemon collects the bandwidth data of all applications running on the system and lists the top 10 of these application sorted by their network bandwidth consumption (total flits/s aggregated over all nodes). Note that these bandwidth-heavy applications are different than the ones running on the top 10 heavily congested nodes.

Fig. 27 shows that a few applications tend to be heavy-hitters. For example, 5 applications alone appear in approximately 57% of the top bandwidth

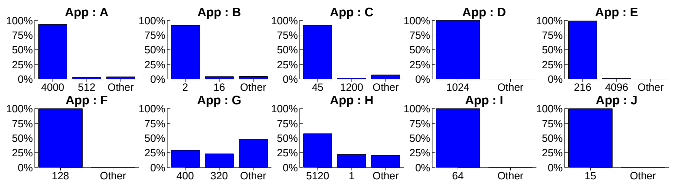


Figure 26: Job size distribution of top 10 congestion-causing applications over one year. X-axis denotes different job size. Y-axis denotes relative frequency of different job size.

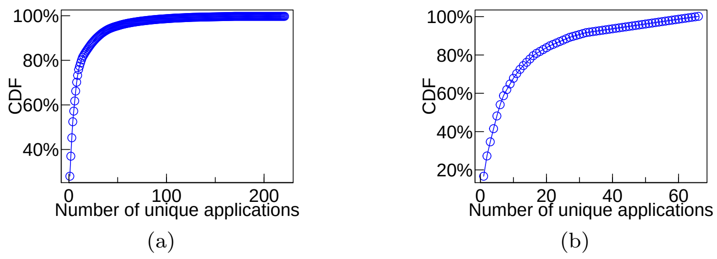


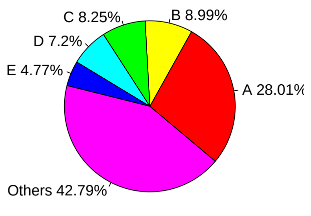
Figure 27: Cumulative distribution of unique applications over top bandwidth-heavy application events without filter (a), and with 1-hr filter (b) over one year.

application reporting events (Fig. 28), while more than 200 unique applications show up in total across all top bandwidth application reporting events. Interestingly, when 1-hour filtering is applied, the number of unique applications decreases significantly. However, the top 5 most frequently occurring applications constitute 50% of top bandwidth application reporting events (Fig. 28). The total number of unique applications reduces dramatically to 60. These results indicate that focusing on the top 5-10 applications can cover 50% of the communication-intensive applications space. We also observe the same results on a per-user basis with and without filtering (Fig. 29). These results again confirm that application and user can work as a proxy for each other, even for top bandwidth application reporting events.

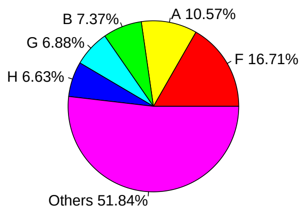
Takeaway 16: *Same applications and users don't cause communication intensive operations.*

Next, we want to answer two questions: (1) are these top bandwidth applications the same as the top congestion-causing applications running on congested nodes?, and (2) is the job size distribution of the top bandwidth applications different than that of the top congestion-causing applications?

Fig. 30 shows the job size distribution and anonymized application names of the top 10 bandwidth-heavy applications that appear most frequently in the top bandwidth application reporting events. We observe that most of the applications tend to run on the same number of nodes every time they appear in the top bandwidth application reporting events. However, interestingly, these applications are not the same as the top congestion-causing applications running on congested nodes. Only three applications are common between these

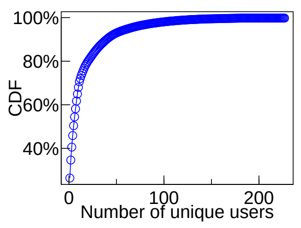


(a)

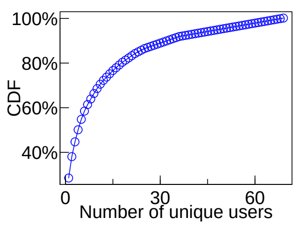


(b)

Figure 28: Fraction of top 5 bandwidth-heavy and other applications without filter (a), and with 1-hour filter (b) over one year.



(a)



(b)

Figure 29: Cumulative distribution of unique users over top bandwidth application reporting events without filtering (a), and 1-hour filtering (b) over one year.

two sets (Fig. 30 vs. Fig. 26). They are situated at positions 1, 4, and 7, in the figures. This indicates that bandwidth-heavy applications are not necessarily the ones that cause congestion or run on congested nodes. These bandwidth-heavy applications are producing a significant amount of traffic, and are likely to be spread over a large number of nodes or have a many-to-many communication pattern. We notice that only 3 applications have a job size larger than 4000 nodes, indicating that even bandwidth-heavy applications are not necessarily large in size. The communication pattern seems to be playing a critical role. As an example, App K which runs mostly on 32 and 128 nodes appears second in the bandwidth-heavy applications list, but does not appear in the congestion-heavy applications list. This could be because this particular application does not intensively exhibit a many-to-one communication pattern. Many-to-few and many-to-one communication patterns can result in high congestion due to the concentration of messages over a few nodes. Many-to-few or many-to-one communication patterns results in over subscription of outgoing link of specific nodes. This over subscription results in network congestion [7, 8, 9]. In summary, bandwidth-heavy applications' job sizes are similar to that of congestion-causing applications', but there is no significant overlap between these two sets and they may differ in their communication patterns.

Takeaway 17: *Top bandwidth-heavy applications are not same as top congestion-causing applications.*

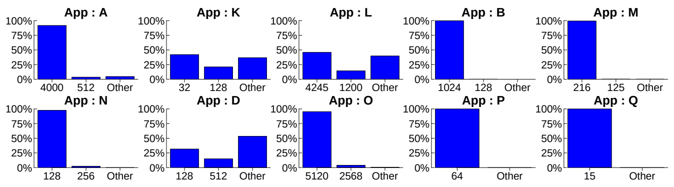


Figure 30: Job size distribution of top 10 bandwidth-heavy applications over one year. X-axis denotes different job size. Y-axis denotes different job size relative frequency.

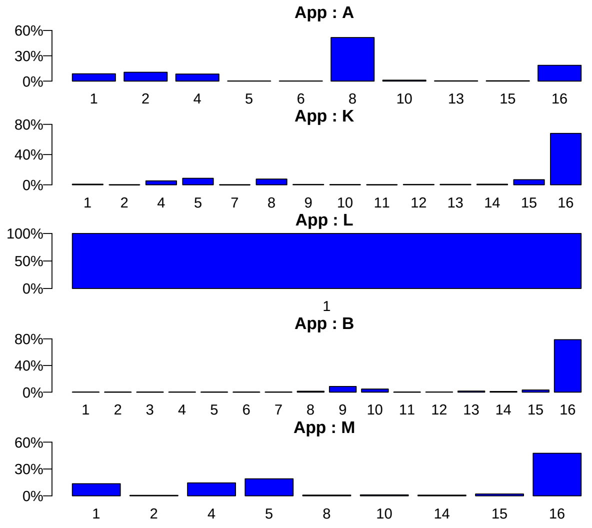


Figure 31: Processes per node count relative frequency of top 5 bandwidth-heavy applications over one year. X-axis denotes different processes per node count. Y-axis denotes different processes per node count relative frequency.

As network congestion can significantly affect an application execution, we next explored different application characteristics for the top five bandwidth-heavy applications: (1) processes per node count (2) CPUs count, (3) execution time, and (4) users. Fig. 31-34 show the frequency distribution of these application characteristics. Most of the applications run have a higher processes per node, CPUs and users count. For the third application, as there are less number of users, processes per node and CPU count have lower variation, however, the execution time doesn't show the same pattern. For all applications, different execution time and skewness toward shorter execution time can be a result of congestion. To confirm it, we compared the number of runs encountering congestion events with the total number of runs for the top five bandwidth-heavy applications. As expected, all the applications encounter the congestion events multiple times (Fig. 35).

Takeaway 18: *Top-bandwidth applications have multiple processes per node, CPUs and users.*

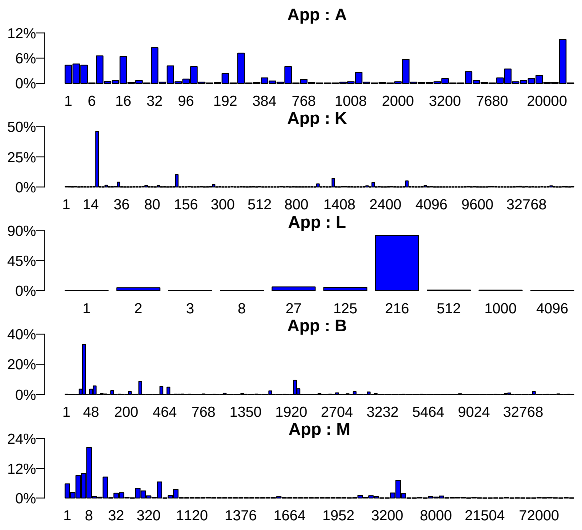


Figure 32: CPUs count relative frequency of top 5 bandwidth-heavy applications over one year. X-axis denotes different CPU count. Y-axis denotes different CPU count relative frequency.

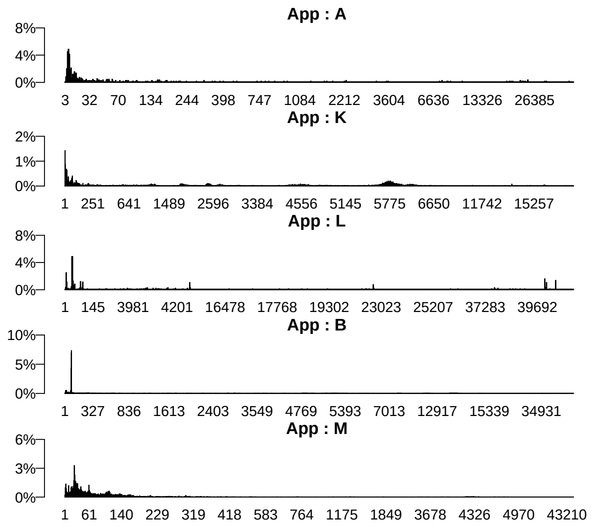


Figure 33: Execution time (s) relative frequency of top 5 bandwidth-heavy applications over one year. X-axis denotes different execution times in seconds. Y-axis denotes the relative frequency of different execution times.

Takeaway 19: *Top-bandwidth applications show skewness in execution time as they observe congestion continuously over time.*

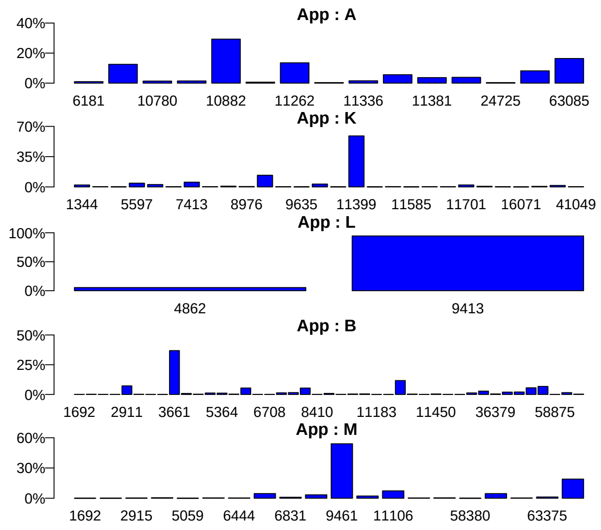


Figure 34: Users id count of top 5 bandwidth-heavy applications over one year. X-axis denotes the different users id. Y-axis denotes the relative frequency of different users.

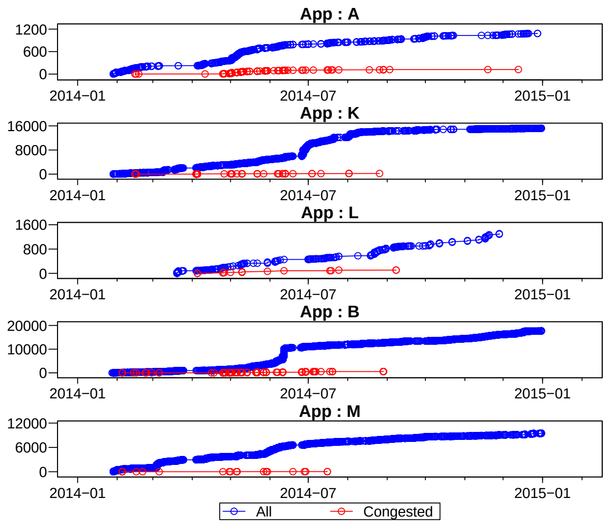


Figure 35: Congestion count for top 5 bandwidth-heavy applications over one year. X-axis denotes the time over one year. Y-axis denotes the congestion event count.

6. Throttle prediction

Network congestion shows weaker correlation with lane degrade, and link failed events in the previous section. However, we have different application characteristics and network events that we can leverage to predict whether an application is going to encounter throttling (congestion results in throttling) or not. The goal is to find a high accuracy machine learning model that can predict the applications encountering network congestion based on the available features. This model will be a proof-of-concept and requires rigorous testing before using in a real-world scenario. The users can use such a prediction model to determine the congestion causing applications without even looking at throttle events. A prediction model will not only be helpful for the applications found in our study but also for new applications as the application set running on the Titan changes over time. Even for the same applications, such a model will be beneficial as an application can result in different characteristics over time. The users can run this model before the application execution or during the application execution to take pro-active and dynamic actions for congestion causing applications. Pro-active actions include detecting congestion causing applications and not scheduling them simultaneously (temporally or locally). Dynamic actions include detecting an application experiencing congestion and applying resilience techniques like restructure of interconnect to avoid congestion. These pro-active and dynamic actions will improve both the congestion causing applications performance and network reliability. The process of finding such a model involves three steps:

1: Feature selection. Explore and finalize the features for the machine learning model.

2: Model selection. Evaluate different machine learning models to find the best model for the selected features.

3: Model analysis. Analyze all the models on a temporal dataset to determine the best model. Find out the best features set and prediction quality for the best model.

6.1. Feature Selection

We have three different type of data which we can use to finalize the features: xtnetwatch, xtnlrd, and application. xtnetwatch has transmitting packets, receiving packets, mode exchanges, lane mask, link inactive and different interconnect failures data. xtnlrd has link failed, application encountering congestion, and the job characteristics data of the applications running during congestion, including APID, number of nodes, the user ID, and the application name. Application data has application start time, stop time, application ID, user ID, node list, and number of CPUs. We use xtnlrd and application data to determine whether an application encounter congestion or not. From each of these data, we extract features which can be extracted dynamically or known beforehand and have a direct impact on throttle events. The features include application execution time, transmitted packets, received packets, mode exchanges, link inactive, link failed, application encountering congestion, number of nodes and number of CPUs. Even though mode exchanges, link inactive, and link failed shows a low correlation to throttle events, they helped in achieving better accuracy while running the prediction model.

Table 3: Machine Learning Models Prediction Accuracy

Model	**Up-Sample**		**Down-Sample**	
	Precision	**Recall**	**Precision**	**Recall**
LR	82.60	93.91	81.57	95.38
SVM	98.66	99.54	80.95	96.58
SGD	81.04	96.95	81.26	96.39
DT	98.66	99.54	92.63	93.01
GBC	84.80	95.12	85.29	94.89
NB	81.61	97.69	82.20	95.54
KN	98.13	99.28	90.13	93.99
RF	83.46	87.07	82.61	86.85

Table 4: Training Time for Machine Learning Models

Model	**LR**	**SVM**	**SGD**	**DT**	**GBC**	**NB**	**KN**	**RF**
Time	31.98 s	71.40 s	31.41 s	31.60 s	32.50 s	34.92 s	31.43 s	31.82 s

6.2. Model Selection

We run various machine learning models for prediction to determine the best model. The first step in this process is to collect the features data periodically. Then we select 60% of this dataset randomly and used it to train the machine learning model. After that, the remaining 40% dataset is used to predict the accuracy of the machine learning model. One of the problems with the collected dataset was that it was highly imbalanced. Only 1% of the application run encounter throttle events. This type of dataset can be handled in two ways. The first one is to up-sample the minority data. The second way is to down-sample the majority data. For our case, we go with the second approach as we want to know the best prediction accuracy in the worst case. We also normalize the dataset as link failed and link inactive has a lot of zeros.

Next, we run our training dataset on various machine learning models like Decision Tree (**DT**), Gradient Boosting Classifier (**GBC**), Stochastic Gradient Descent Classifier (**SGD**), KNeighbours (**KN**), Logistic Regression (**LR**), Naive Bayes (**NB**), Random Forest (**RF**) and Support Vector Machine (**SVM**). We use Python module *Scikit-learn* [10] for the machine learning models. For each model, we calculate the feature ranking. We remove the features with lower feature ranking however it results in lower accuracy for each model. Therefore, we include all the features for our experiments. In later subsection, we analyze the features sets in more detail. Table 3 shows the prediction effectiveness for all machine learning models. Precision indicates the percentage of correct

Table 5: Machine Learning Models F1 Score for Each Month Down-Sampled Dataset

Model	**Jan**	**Feb**	**Mar**	**Apr**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Oct**	**Nov**	**Dec**
LR	0.00	82.75	79.05	76.05	82.20	82.33	80.08	91.94	91.17	91.29	75.71	88.44
SVM	0.00	82.43	78.52	75.73	82.72	81.89	80.46	91.57	90.16	86.96	72.25	84.42
SGD	0.00	83.10	84.26	77.90	82.31	82.37	80.49	92.04	96.03	90.24	0.00	33.47
DT	0.00	91.48	90.26	87.73	92.44	95.45	91.82	97.24	95.33	96.89	94.12	93.95
GBC	0.00	90.93	91.38	85.26	86.99	91.95	90.71	96.91	96.34	96.92	92.11	94.71
NB	0.00	84.95	86.37	72.78	81.20	85.10	81.66	93.21	95.94	94.47	87.72	93.27
KN	0.00	89.68	89.69	86.48	92.24	94.47	90.53	96.25	95.51	92.24	79.75	91.13
RF	0.00	83.31	86.31	72.29	81.69	86.15	84.68	93.88	97.37	93.10	79.74	91.58

predictions in all predictions, defined as:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{1}$$

while recall reveals the ratio of identified samples to the ground truth, defined as:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{2}$$

Decision tree performs the best in both up-sampling and down-sampling. As expected up-sampling results in better accuracy, however, we go with down-sampling to handle the worst-case scenario. KNeighbours performs the second best with a slighter higher recall and lower precision. It means KNeighbours fetch higher relevant instances among the total relevant instances. We do further analysis in the next subsection to determine which is the best model for diverse datasets.

Training overhead can play a significant role in determining the best model for prediction. Therefore, we evaluate the training time overhead for all the models. We conduct all the experiments on Intel(R) Xeon(R) E3-1275 v5 server with 32 GB memory. Table 4 shows the training time of all the machine learning models. Stochastic Gradient Descent Classifier takes least amount of time, followed by KNeighbours and Decision tree. However, the percent difference among them is less than 1%, which means choosing KNeighbours and Decision tree over Stochastic Gradient Descent Classifier will not affect the training time much. SVM took the most amount of time, which was expected as the training time taken is proportional to the third power of the size of the dataset.

6.3. Model Analysis

In this step, we evaluate the machine learning models on diverse datasets. Before presenting the results, we explain the datasets and evaluation metrics.

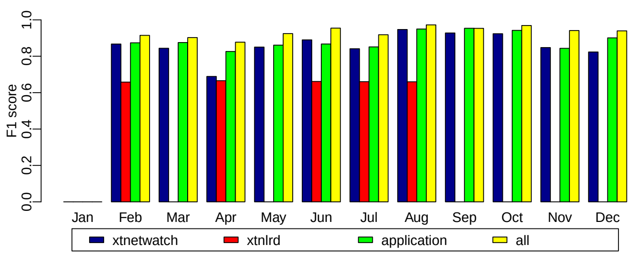


Figure 36: F1 Score of different feature set using Decision Tree.

6.3.1. Dataset Description and Evaluation Metrics

We first collect dataset over the entire sampling period (from January to December, 2014). We then divide this data into twelve sub-datasets (each month data). We then perform downsampling for each sub-dataset, normalize it and fed it to machine learning models. We use *F1 Score* [11] to measure the prediction accuracy as it conveys a balance between precision and recall. It is defined as the harmonic average between precision and recall.

$$F1\ Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{3}$$

6.3.2. Model comparison

We measure the *F1 Score* for each machine learning model for each sub-datasets. Table 5 shows these *F1 Score*s. The scores for the January month is zero as it represents the data for only last four days of the month and no throttling event happen in that period. For the remaining eleven months, KNeighbours perform slightly better than Decision Tree for only the month of September. Therefore, we conclude that the Decision Tree is the best model for throttle events prediction.

6.3.3. Feature Analysis

Selection of feature can severely impact the accuracy of any machine learning model. To find the best features set, we evaluate different features set for the best machine learning model Decision Tree. We split the available features into four categories: xtnetwatch, xtnlrd, application and all. Each category is discussed in subsection 6.1. For xtnetwatch, we use transmitting packets, receiving packets, mode exchanges, and link inactive data. For xtnlrd, we use link failed data. For application, we use execution time, number of nodes, and number of CPUs. All category combines all the features in xtnetwatch, xtnlrd, and application. Fig 36 shows the *F1 Score* of Decision Tree model for different features selection. As we can see, all features combination dataset perform better than any other features set for each month. xtnlrd performs the worst for two reasons: (1) it has only one feature linked failed, and (2) linked failed has a lower correlation to network congestion.

Table 6: Throttle Prediction for Short, Medium and Long Running Applications using Decision Tree

Application	**Precison**	**Recall**	**F1 Score**
Short	0.00	0.00	0.00
Medium	92.71	95.70	94.18
Long	86.18	86.76	86.47

6.3.4. Prediction Analysis

In this section, we check how application execution time affects the prediction quality of the best machine learning model Decision Tree. We categorize applications into three groups: short-running, medium-running and long-running. Short-running are those applications which have a runtime that falls below 25 percentile. Long-running are those applications which have a runtime in top 25 percentile. Medium-running includes all the remaining applications. Table 6 shows the prediction quality of Decision Tree model for each application type. Short-running applications never encounter the throttle events. Medium running applications encounter less than 1% of the total throttle events. Long-running applications encounter more than 99% of the total throttle events. Both medium and long-running applications show higher prediction quality.

Takeaway 20: *Network and application events can predict congestion in an application at a high accuracy.*

7. Related Work

Interconnect networks have been a vital part of computer systems. With thousands of nodes in HPC systems, execution time is more dependent on communication time than the calculation time [12]. Various HPC interconnect networks are proposed for improving HPC systems performance - QsNET [13], SeaStar [14], Tofu [15], Blue Gene/Q [16], Aries [17], TH Express-2 [18] and others - which use different types of topology like k-Ary n-Cube, fat-tree/Clos, and dragonfly. Several studies are performed to understand [19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 4, 30] and improve [31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41] interconnect failures in HPC systems.

Interconnect studies in [19, 20, 22] describe and evaluate specific interconnect networks in Cray T3E multiprocessor and BlueGene/L. Interconnect networks detail and concepts like topology, routing, flow control, and router architecture are discussed in [21, 23, 25, 28]. Blue Waters workload and error/failure logs are investigated to determine system errors and failures impact on user applications in [26]. The analysis in this work was limited to system errors and failures logs. Blue Waters Gemini interconnect architecture is described in detail in [27, 4]. These studies focus on characterization of recovery mechanisms and interconnect failures using raw systems logs. These studies are limited to interconnect failures like lane recovery, link failure, and warm swap only. Blue Waters Gemini interconnect congestion events impact on two benchmark applications and network congestion characteristics are presented in [29]. Our work provides a better understanding of impact of congestion events as we analyze real-word applications with focus on job characteristics. Fault injection tool is presented in [30] to understand impact of failures on network links, nodes, and blades in HPC systems. Data centers network failure analysis with focus on network workload characteristics, failures of network links and devices, and effectiveness of network redundancy in masking failures is presented in [24]. The limitation of this work is that they only characterize link and device failures, instead of focusing on all type of errors and failures in network logs.

Interconnect networks type like fat-trees and k-ary n-cube performance evaluation is presented in [31, 32, 33]. Methods for automatically routing faults, recursive torus shifting, adaptive bubble router, fail-in place network, task mapping, and communication intensive applications optimizations for improving interconnect performance is discussed in [34, 35, 36, 37, 38, 39]. A functional network simulator, Damselfly, is presented in [40] to study the effects of job placement, parallel workloads and network configurations on network health of dragonfly-based supercomputers. A network fault influence domain analysis tool, FIDA, is presented in [41] to study the impact of network faults on a system.

Supervised learning algorithms such as forests of extremely randomized trees and gradient boosted regression trees are used to ascertain the causes and mechanisms of network congestion in [42]. They created a regression model using communication data and application execution time to predict the execution time of communication heavy applications. In this study, they also determine the hardware components that play a major role in network congestion. A machine learning approach to predict total communication time of parallel applications is proposed in [43]. A machine learning framework to automatically detect compute nodes with performance anomalies and to diagnose performance anomalies is presented in [44]. This framework leverages easy-to-compute statistical approach to reduce data required for performance anomaly detection.

Titan is the successor of Cray X-series which use the XK7 system and 3D Gemini interconnect. The Gemini system interconnect architecture is explained in [2] and evaluated in [45, 3] using micro-benchmarks. Cray's latest XC series is implemented using the Aries interconnect which supports better bandwidth, latency, message rate and scalability [46]. Our previous work [1] differs from all these studies and evaluations as none of these works evaluate how different interconnect errors and congestion events occur on a large-scale HPC system. Our current work differs as none of the work above utilize interconnect errors, congestion events and applications characteristics data to predict throttle events. Our field data and analysis is unique and provides useful insights that can be used by users, system architects, and operators to improve the overall efficiency of HPC systems.

8. Conclusion & Future work

Overall, we show how congestion events can impact application performance. We discussed many interesting insights derived from our analysis. Interconnect faults like lane degrades are continuous and vary significantly among lanes. Link inactive errors do not have a temporal or a spatial correlation with lane degrades, while interconnect errors have a high correlation with link inactive/failed errors. We showed that these characteristics can be exploited for different purposes. We also demonstrated that multiple applications can cause multiple congestion events within a short period of time. Furthermore, these applications can be, surprisingly, small in job size, not scheduled evenly across the cabinet and have a many-to-few communication pattern. Our analysis can be used in identifying such applications and users to minimize the performance impact on other applications. In end, we evaluate different machine learning models to predict applications encountering throttle events.

Given limited literature on field data and analysis on interconnects error, we hope our study addresses an important topic and would be useful for current and future systems. A real-world test of the model on a future HPC system and looking into the impact of other system attributes on the model in the future can further strengthen the model to detect network congestion. Furthermore, the researchers can investigate system and application attributes to detect a specific application causing congestion while multiple applications are running at the same time in an HPC system.

Acknowledgments

The work was supported by in part through NSF Grants (#1563728, #1561216 and #1563750), Northeastern University and by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, program manager Lucy Nowell. This work also used in part the resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at ORNL, which is managed by UT Battelle, LLC for the U.S. DOE under contract number DE-AC05-00OR22725. We thank Robinson Pino and Lucy Nowell for their support.

References

[1] M. Kumar, S. Gupta, T. Patel, M. Wilder, W. Shi, S. Fu, C. Engelmann, D. Tiwari, Understanding and analyzing interconnect errors and network congestion on a large scale hpc system, in: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, 2018, pp. 107–114.

[2] R. Alverson, D. Roweth, L. Kaplan, The gemini system interconnect, in: High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on, IEEE, 2010, pp. 83–87.

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

[4] S. Jha, V. Formicola, C. Di Martino, M. Dalton, W. T. Kramer, Z. Kalbarczyk, R. K. Iyer, Resiliency of hpc interconnects: A case study of interconnect failures and recovery in blue waters, IEEE Transactions on Dependable and Secure Computing 15 (6) (2017) 915–930.

[5] F. J. Massey Jr, The kolmogorov-smirnov test for goodness of fit, Journal of the American statistical Association 46 (253) (1951) 68–78.

[6] ACM, Gordon bell prize, URL https://awards.acm.org/bell/.

[7] T.-c. Tam, Performance studies of high-speed communication on commodity cluster, (2001) 1–0.

[8] HP, Measuring network performance to better manage it, URL https://awards.acm.org/bell/.

[9] S. Jha, A. Patke, J. Brandt, A. Gentile, B. Lim, M. Showerman, G. Bauer, L. Kaplan, Z. Kalbarczyk, W. Kramer, et al., Measuring congestion in high-performance datacenter interconnects, in: 17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20), 2020, pp. 37–57.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[11] D. M. Powers, Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation, arXiv preprint arXiv:2010.16061.

[12] A. Grama, Introduction to parallel computing, Pearson Education, 2003.

[13] F. Petrini, E. Frachtenberg, A. Hoisie, S. Coll, Performance evaluation of the quadrics interconnection network, Cluster Computing 6 (2) (2003) 125–142.

[14] R. Brightwell, K. T. Pedretti, K. D. Underwood, T. Hudson, Seastar interconnect: Balanced bandwidth for scalable performance, IEEE Micro 26 (3) (2006) 41–57.

[15] Y. Ajima, S. Sumimoto, T. Shimizu, Tofu: A 6d mesh/torus interconnect for exascale computers, Computer 42 (11) (2009) 0036–41.

[16] D. Chen, N. A. Eisley, P. Heidelberger, R. M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. L. Satterfield, B. Steinmacher-Burow, J. J. Parker, The ibm blue gene/q interconnection network and message unit, in: High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for, IEEE, 2011, pp. 1–10.

[17] 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.

[18] Z. Pang, M. Xie, J. Zhang, Y. Zheng, G. Wang, D. Dong, G. Suo, The th express high performance interconnect networks, Frontiers of Computer Science 8 (3) (2014) 357–366.

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

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

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

[22] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, et al., Blue gene/l torus interconnection network, IBM Journal of Research and Development 49 (2.3) (2005) 265–276.

[23] J. Duato, S. Yalamanchili, L. M. Ni, Interconnection networks: an engineering approach, Morgan Kaufmann, 2003.

[24] P. Gill, N. Jain, N. Nagappan, Understanding network failures in data centers: measurement, analysis, and implications, in: ACM SIGCOMM Computer Communication Review, Vol. 41, ACM, 2011, pp. 350–361.

[25] D. Abts, B. Felderman, A guided tour of data-center networking, Communications of the ACM 55 (6) (2012) 44–51.

[26] C. Di Martino, W. Kramer, Z. Kalbarczyk, 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, IEEE, 2015, pp. 25–36.

[27] S. Jha, V. Formicola, Z. Kalbarczyk, C. Di Martino, W. T. Kramer, R. K. Iyer, Analysis of gemini interconnect recovery mechanisms: Methods and observations, Cray User Group (2016) 8–12.

[28] R. Trobec, R. Vasiljević, M. Tomašević, V. Milutinović, R. Beivide, M. Valero, Interconnection networks in petascale computer systems: A survey, ACM Computing Surveys (CSUR) 49 (3) (2016) 44.

[29] S. Jha, J. Brandt, A. Gentile, Z. Kalbarczyk, R. Iyer, Characterizing supercomputer traffic networks through link-level analysis, in: 2018 IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2018, pp. 562–570.

[30] V. Formicola, S. Jha, D. Chen, F. Deng, A. Bonnie, M. Mason, J. Brandt, A. Gentile, L. Kaplan, J. Repik, et al., Understanding fault scenarios and impacts through fault injection experiments in cielo, Urbana 51 (2017) 61801.

[31] C. E. Leiserson, Fat-trees: universal networks for hardware-efficient supercomputing, IEEE transactions on Computers 100 (10) (1985) 892–901.

[32] W. J. Dally, Performance analysis of k-ary n-cube interconnection networks, IEEE transactions on Computers 39 (6) (1990) 775–785.

[33] W. J. Dally, Express cubes: Improving the performance of k-ary n-cube interconnection networks, IEEE Transactions on Computers 40 (9) (1991) 1016–1023.

[34] D. W. Mackenthun, Method and apparatus for automatically routing around faults within an interconnect system, uS Patent 5,450,578 (Sep. 12 1995).

[35] Y. Inoguchi, S. Horiguchi, Shifted recursive torus interconnection for high performance computing, in: High Performance Computing on the Information Superhighway, 1997. HPC Asia'97, IEEE, 1997, pp. 61–66.

[36] V. Puente, R. Beivide, J. A. Gregorio, J. Prellezo, J. Duato, C. Izu, Adaptive bubble router: a design to improve performance in torus networks, in: Parallel Processing, 1999. Proceedings. 1999 International Conference on, IEEE, 1999, pp. 58–67.

[37] J. Domke, T. Hoefler, S. Matsuoka, Fail-in-place network design: interaction between topology, routing algorithm and failures, in: High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for, IEEE, 2014, pp. 597–608.

[38] A. Bhatele, Task mapping on complex computer network topologies for improved performance, Lawrence Livermore Nat. Laboratory.

[39] N. Jain, Optimization of communication intensive applications on hpc networks, Ph.D. thesis, University of Illinois at Urbana-Champaign (2016).

[40] A. Bhatele, N. Jain, Y. Livnat, V. Pascucci, P.-T. Bremer, Analyzing network health and congestion in dragonfly-based supercomputers, in: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2016, pp. 93–102.

[41] Z. Wu, K. Lu, X. Wang, W. Chi, Topology-aware network fault influence domain analysis, Computers & Electrical Engineering 57 (2017) 266–280.

[42] A. Bhatele, A. R. Titus, J. J. Thiagarajan, N. Jain, T. Gamblin, P.-T. Bremer, M. Schulz, L. V. Kale, Identifying the culprits behind network congestion, in: Proceedings of the IEEE International Parallel & Distributed Processing Symposium, IPDPS '15, IEEE Computer Society, 2015, lLNL-CONF-663150.
URL `http://doi.ieeecomputersociety.org/10.1109/IPDPS.2015.92`

[43] N. Papadopoulou, G. Goumas, N. Koziris, A machine-learning approach for communication prediction of large-scale applications, in: Cluster Computing (CLUSTER), 2015 IEEE International Conference on, IEEE, 2015, pp. 120–123.

[44] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, A. K. Coskun, Diagnosing performance variations in hpc applications using machine learning, in: International Supercomputing Conference, Springer, 2017, pp. 355–373.

[45] A. Vishnu, M. ten Bruggencate, R. Olson, Evaluating the potential of cray gemini interconnect for pgas communication runtime systems, in: High Performance Interconnects (HOTI), 2011 IEEE 19th Annual Symposium on, IEEE, 2011, pp. 70–77.

[46] B. Alverson, E. Froese, L. Kaplan, D. Roweth, Cray xc series network, Cray Inc., White Paper WP-Aries01-1112.

Mohit Kumar is a Postdoctoral research associate at Oak Ridge National Laboratory. His research interests include energy efficiency, software engineering, and high performance computing. He received his B.Tech. from Maharshi Dayanand University, MS and Ph.D. from Wayne State University.

Saurabh Gupta is a CPU research scientist in the Mircoarchitecture Research Lab (MRL) within Intel Labs. His general research interest lies in the area of high performance computer architecture. Prior to Intel, he pursued his post-doctoral research at Oak Ridge National Laboratory (ORNL). He obtained his Ph.D. in Computer Engineering from North Carolina State University. He completed his Bachelors and Masters in Electrical Engineering from Indian Institute of Technology Kanpur in 2009.

Tirthak Patel is a Ph.D. student in Computer Engineering at Northeastern University. His research interests include computer systems and architecture, high performance computing, fault-tolerance and super computing. Prior to joining Northeastern University, he received his Honors Bachelor of Applied Science degree in Electrical Engineering from the University of Toronto.

Michael Wilder is a junior-level backend programmer. He has a demonstrated history of writing testable, maintainable, and efficient code. He received his BS in Computer Engineering from the University of Tennessee, Knoxville.

Weisong Shi is a professor of Computer Science at Wayne State University, IEEE Fellow. His research interests include computer systems, mobile and cloud computing, wireless health. He received his BS from Xidian University in 1995, and Ph.D. from Chinese Academy of Sciences in 2000, both in Computer Engineering. He is a recipient of National Outstanding Ph.D. dissertation award of China and the NSF CAREER award.

Song Fu is an Associate Professor in the Department of Computer Science and Engineering at the University of North Texas. He was an Assistant Professor in the Department of Computer Science and Engineering at New Mexico Institute of Mining and Technology from 2008 to 2010. He obtained the Ph.D. degree in Computer Engineering from Wayne State University, Detroit Michigan, in 2008. His research interest is primarily in high-performance computing, distributed and cloud systems, including architecture, resilience, dependability assurance, resource management, and security.

Christian Engelmann is an Senior R&D Staff Scientist in the Computer Science Research Group at Oak Ridge National Laboratory. He has 17 years experience in software research and development for extreme-scale high-performance computing (HPC) systems with a strong funding and publication record. In collaboration with other laboratories and universities, Dr. Engelmanns research solves computer science challenges in HPC software, such as scalability, dependability, energy efficiency, and portability.

Devesh Tiwari is an Assistant Professor of Computer Science in Northeastern University. His research focuses on developing novel computational analytical tools and models toward designing resilient, sustainable, and scalable computing systems. Before joining Northeastern, Devesh was a staff scientist at the Oak Ridge National Laboratory. Devesh earned his Ph.D. in Electrical and Computer Engineering from North Carolina State University and B.S. degree in Computer Science and Engineering from Indian Institute of Technology (IIT) Kanpur in India.