Measuring Congestion in High-Performance Datacenter Interconnects

Saurabh Jha and Archit Patke, *University of Illinois at Urbana-Champaign;* Jim Brandt and Ann Gentile, *Sandia National Lab;* Benjamin Lim, *University of Illinois at Urbana-Champaign;* Mike Showerman and Greg Bauer, *National Center for Supercomputing Applications;* Larry Kaplan, *Cray Inc.;* Zbigniew Kalbarczyk, *University of Illinois at Urbana-Champaign;* William Kramer, *University of Illinois at Urbana-Champaign and National Center for Supercomputing Applications;* Ravi Iyer, *University of Illinois at Urbana-Champaign*

https://www.usenix.org/conference/nsdi20/presentation/jha

This paper is included in the Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)

February 25–27, 2020 • Santa Clara, CA, USA

978-1-939133-13-7

Open access to the Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20) is sponsored by

Measuring Congestion in High-Performance Datacenter Interconnects

Saurabh Jha[1], Archit Patke[1], Jim Brandt[2], Ann Gentile[2], *Benjamin Lim*[1],
Mike Showerman[3], Greg Bauer[3], Larry Kaplan[4], Zbigniew Kalbarczyk[1], William Kramer[1,3], Ravi Iyer[1]

[1]*University of Illinois at Urbana-Champaign,* [2]*Sandia National Lab,*
[3]*National Center for Supercomputing Applications,* [4]*Cray Inc.*

Abstract

While it is widely acknowledged that network congestion in High Performance Computing (HPC) systems can significantly degrade application performance, there has been little to no quantification of congestion on credit-based interconnect networks. We present a methodology for detecting, extracting, and characterizing regions of congestion in networks. We have implemented the methodology in a deployable tool, *Monet*, which can provide such analysis and feedback at runtime. Using *Monet*, we characterize and diagnose congestion in the world's largest 3D torus network of *Blue Waters*, a 13.3-petaflop supercomputer at the National Center for Supercomputing Applications. Our study deepens the understanding of production congestion at a scale that has never been evaluated before.

1 Introduction

High-speed interconnect networks (HSN), e.g., Infiniband [48] and Cray Aries [42]), which uses credit-based flow control algorithms [32, 61], are increasingly being used in high-performance datacenters (HPC [11] and clouds [5, 6, 8, 80]) to support the low-latency communication primitives required by extreme-scale applications (e.g., scientific and deep-learning applications). Despite the network support for low-latency communication primitives and advanced congestion mitigation and protection mechanisms, significant performance variation has been observed in production systems running real-world workloads. While it is widely acknowledged that network congestion can significantly degrade application performance [24, 26, 45, 71, 81], there has been little to no quantification of congestion on such interconnect networks to understand, diagnose and mitigate congestion problems at the application or system-level. In particular, tools and techniques to perform runtime measurement and characterization and provide runtime feedback to system software (e.g., schedulers) or users (e.g., application developers or system managers) are generally not available on production systems. This would require continuous system-wide, data collection on the state of network performance and associated complex analysis which may be difficult to perform at runtime.

The core contributions of this paper are (a) a methodology, including algorithms, for quantitative characterization of congestion of high-speed interconnect networks; (b) introduction of a deployable toolset, *Monet* [7], that employs our congestion characterization methodology; and (c) use of the the methodology for characterization of congestion using 5 months of operational data from a 3D torus-based interconnect network of *Blue Waters* [1, 27, 60], a 13.3-petaflop Cray supercomputer at the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign. The novelty of our approach is its ability to use *percent time stalled* (P_{Ts})[1] metric to detect and quantitatively characterize congestion hotspots, also referred to as *congestion regions (CRs)*, which are group of links with similar levels of congestion.

The *Monet* tool has been experimentally used on NCSA's *Blue Waters*. *Blue Waters* uses a Cray Gemini [21] 3D torus interconnect, the largest known 3D torus in existence, that connects 27,648 compute nodes, henceforth referred to as *nodes*. The proposed tool is not specific to Cray Gemini and *Blue Waters*; it can be deployed on other k-dimensional mesh or toroidal networks, such as TPU clouds [3], Fujitsu TOFU network-based [18, 20] K supercomputer [70] and upcoming post-K supercomputer [10][2]. The key components of our methodology and the *Monet* toolset are as follows:

Data collection tools: On *Blue Waters*, we use vendor-provided tools (e.g., `gpcdr` [35]), along with the Lightweight Distributed Metric Service (LDMS) monitoring framework [17]. Together these tools collect data on (a) the network (e.g., transferred/received bytes, congestion metrics, and link failure events); (b) the file system traffic (e.g., read/write bytes); and (c) the applications (e.g., start/end time). We are released raw network data obtained from *Blue Waters* [57] as well as the associated code for generating *CRs* as artifacts with this paper [7]. To the best of our knowledge, this is the first

[1] P_{Ts}, defined formally in Section 2, approximately represents the intensity of congestion on a link, quantified between 0% and 100%.

[2] The first post-K supercomputer is scheduled to be deployed in 2021.

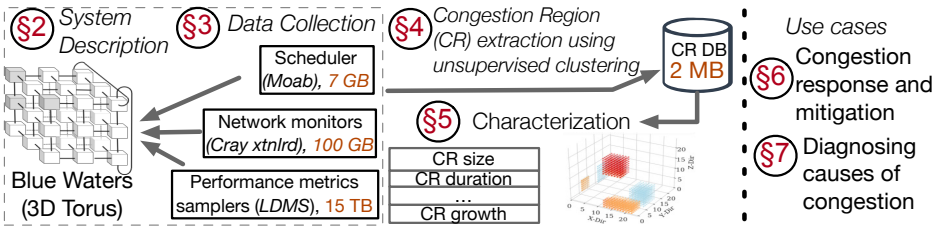


Figure 1: Characterization and diagnosis workflow for interconnection-networks.

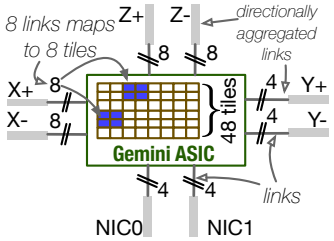


Figure 2: Cray Gemini 48-port switch.

such large-scale network data release for an HPC high-speed interconnect network that uses credit-based flow control.

A network hotspot extraction and characterization tool, which extracts *CRs* at runtime; it does so by using an unsupervised region-growth clustering algorithm. The clustering method requires specification of congestion metrics (e.g., percent time stalled (P_{Ts}) or stall-to-flit ratios) and a network topology graph to extract regions of congestion that can be used for runtime or long-term network congestion characterization.

A diagnosis tool, which determines the cause of congestion (e.g., link failures or excessive file system traffic from applications) by combining system and application execution information with the CR characterizations. This tool leverages outlier-detection algorithms combined with domain-driven knowledge to flag anomalies in the data that can be correlated with the occurrence of CRs.

To produce the findings discussed in this paper, we used 5 months of operational data on *Blue Waters* representing more than 815,006 unique application runs that injected more than 70 PB of data into the network. Our key findings are as follows:

- *While it is rare for the system to be globally congested, there is a continuous presence of highly congested regions (CRs) in the network, and they are severe enough to affect application performance*. Measurements show that (a) for more than 56% of system uptime, there exists at least one highly congested CR (i.e., a CR with a $P_{Ts} > 25\%$), and that these CRs have a median size of 32 links and a maximum size of 2,324 links (5.6% of total links); and (b) highly congested regions may persist for more than 23 hours, with a median duration time of 9 hours[3]. With respect to impact on applications, we observed 1000-node production runs of the NAMD [77] application [4] slowing down by as much as 1.89× in the presence of high congestion compared to median runtime of 282 minutes.
- *Once congestion occurs in the network, it is likely to persist rather than decrease, leading to long-lived congestion in the network*. Measurements show that once the network has entered a state of high congestion ($P_{Ts} > 25\%$), it will persist in high congestion state with a probability of 0.87 in the next measurement window.
- *Quick propagation of congestion can be caused by network component failures.* Network component failures (e.g., network router failures) that occur in the vicinity of a large-scale application can lead to high network congestion within minutes of the failure event. Measurements show that 88% of directional link failures [5] caused the formation of CRs with an average $P_{Ts} \geq 15\%$.
- *Default congestion mitigation mechanisms have limited efficacy.* Our measurements show that (a) 29.8% of the 261 triggers of vendor-provided congestion mitigation mechanisms failed to alleviate long-lasting congestion (i.e., congestion driven by continuous oversubscription, as opposed to isolated traffic bursts), as they did not address the root causes of congestion; and (b) vendor-provided mitigation mechanisms were triggered in 8% (261) of the 3,390 high-congestion events identified by our framework. Of these 3,390 events, 25% lasted for more than 30 minutes. This analysis suggests that augmentation of the vendor-supplied solution could be an effective way to improve overall congestion management.

In this paper, we highlight the utility of congestion regions in the following ways:

- We showcase the effectiveness of *CRs* in detecting long-lived congestion. Based on this characterization, we propose that *CR* detection could be used to trigger congestion mitigation responses that could augment the current vendor-provided mechanisms.
- We illustrate how *CRs*, in conjunction with network traffic assessment, enable congestion diagnosis. Our diagnosis tool attributes congestion cause to one of the following: (a) system issues (such as launch/exit of application), (b) failure issues (such as network link failures), and (c) intra-application issues (such as changes in communication patterns within an application). Such a diagnosis allows system managers to take cause-specific mitigating actions.

This paper's organization is illustrated in Figure 1. We present background information on the Gemini network, performance data, and congestion mitigation mechanisms in Section 2. In Section 3, we present our data collection methodology and tools. In Section 4, we present our methodology for characterizing congestion. We present our measurement-

[3] Note that *Blue Waters* allows applications to run for a maximum of 48 hours.
[4] NAMD is the top application running on Blue Waters consuming 18% of total node-hours [58].
[5] see Section 5.4 for the definition of directional link.

driven congestion characterization results in Section 5. In Section 6, we discuss the further utility of our methodology to inform targeted responses, and in Section 7, we discuss its use in diagnosing the root causes of congestion. We address related work in Section 8 and conclude in Section 9.

2 Cray Gemini Network and Blue Waters

A variety of network technologies and topologies have been utilized in HPC systems (e.g., [19, 21, 31, 36, 42, 59, 62, 75]). Depending on the technology, routing within these networks may be statically defined for the duration of a system boot cycle, or may dynamically change because of congestion and/or failure conditions. More details on HPC interconnects can be found in Appendix A. The focus of this paper is on NCSA's Cray XE/XK *Blue Waters* [1] system, which is composed of 27,648 nodes and has a large-scale (13,824 x 48 port switches) Gemini [21] 3D torus (dimension 24x24x24) interconnect. It is a good platform for development and validation of congestion analysis/ characterization methods as:

- It uses directional-order routing, which is predominantly static[6]. From a traffic and congestion characterization perspective, statically routed environments are easier to validate than dynamic and adaptive networks.
- *Blue Waters* is the best case torus to study since it uses topology-aware scheduling (TAS) [41, 82], discussed later in this section, which has eliminated many congestion issues compared to random scheduling.
- *Blue Waters* performs continuous system-wide collection and storage of network performance counters.

2.1 Gemini Network

In Cray XE/XK systems, four *nodes* are packaged on a *blade*. Each *blade* is equipped with a mezzanine card. This card contains a pair of *Gemini [21] ASICs*, which serve as network switches. The Gemini switch design is shown in Figure 2. Each Gemini ASIC consists of 48 *tiles*, each of which provide a duplex link. The switches are connected with one another in 6 directions, X+/-, Y+/- and Z+/-, via multiple links that form a 3D torus. The number of links in a direction, depends on the direction as shown in the figure; there are 8 each in X+/- and, Z+/- and 4 each in Y+/-. It is convenient to consider all links in a given direction as a *directionally aggregated link*, which we will henceforth call a *link*. The available bandwidth on a particular link is dependent on the link type, i.e., whether the link connects compute cabinets or *blades*, in addition to the number of tiles in the link [76]. X, Y links have aggregate bandwidths of 9.4 GB/s and 4.7 GB/s, respectively, whereas Z links are predominantly 15 GB/s, with 1/8 of them at 9.4 GB/s. Traffic routing in the Gemini network is largely static and changes only when failures occur that need to be routed around. Traffic is directionally routed in the X, Y, and Z dimensions, with the shortest path in terms of hops in + or - chosen for each direction. A deterministic rule handles tie-breaking.

To avoid data loss in the network [7], the Gemini HSN uses a credit-based flow control mechanism [61], and routing is done on a per-packet basis. In credit-based flow control networks, a source is allowed to send a quantum of data, e.g., a flit, to a next hop destination only if it has a sufficient number of credits. If the source does not have sufficient credits, it must stall (wait) until enough credits are available. Stalls can occur in two different places: within the switch (resulting in a *inq stall*) or between switches (resulting in an *credit stall*).

Definition 1 ***: A Credit stall*** *is the wait time associated with sending of a flit from an output buffer of one switch to an input buffer of another across a link.*

Definition 2 ***: An Inq stall*** *is the wait time associated with sending of a flit from the output buffer of one switch port to an input buffer of another between tiles within the same network switch ASIC.*

Congestion in a Gemini-based network can be characterized using both *credit* and *inq* stall metrics. Specifically, we consider the *Percent Time Stalled* as a metric for quantifying congestion, which we generically refer to as the *stall value*.

Definition 3 ***: Percent Time Stalled*** *(P_{Ts}) is the average time spent stalled (T_{is}) over all* tiles *of a directional network link or individual intra-Gemini switch link over the same time interval (T_i): $P_{Ts} = 100 * T_{is}/T_i$.*

Depending on the network topology and routing rules, (a) an application's traffic can pass through switches not directly associated with its allocated nodes, and multiple applications can be in competition for bandwidth on the same network links; (b) stalls on a link can lead to back pressure on prior switches in communication routes, causing congestion to spread; and (c) the initial manifestation location of congestion cannot be directly associated with the cause of congestion. Differences in available bandwidth along directions, combined with the directional-order routing, can also cause back pressure, leading to varying levels of congestion along the three directions.

2.2 Congestion Mitigation

Run-time evaluations that identify localized areas of congestion and assess congestion duration can be used to trigger *Congestion Effect Mitigating Responses (CEMRs)*, such as resource scheduling, placement decisions, and dynamic application reconfiguration. While we have defined a CEMR as a response that can be used to minimize the negative effects of network congestion, Cray provides a software mechanism [33] to directly alleviate the congestion itself. When a

[6] When network-link failures occur, network routes are recomputed; that changes the route while the system is up.

[7] The probability of loss of a quantum of data in credit-flow networks is negligible and mostly occurs due to network-related failures.

variety of network components (e.g., tiles, NICs) exceeds a high-watermark threshold with respect to the ratio of stalls to forwarded flits, the software instigates a *Congestion Protection Event* (CPE), which is a *throttling* of injection of traffic from all NICs. The CPE mechanism limits the aggregate traffic injection bandwidth over *all* compute nodes to less than what can be ejected to a *single* node. While this ensures that the congestion is at least temporarily alleviated, the network as a whole is drastically under-subscribed for the duration of the *throttling*. As a result, the performance of all applications running on the system can be significantly impacted. *Throttling* remains active until associated monitored values and ratios drop below their low-watermark thresholds. Applications with sustained high traffic injection rates may induce many CPEs, leading to significant time spent in globally throttling. Bursts of high traffic injection rates may thus trigger CPEs, due to localized congestion, that could have been alleviated without the global negative impact of *throttling*. There is an option to enable the software to terminate the application that it determines is the top congestion candidate, though this feature is not enabled on the *Blue Waters* system. The option to terminate application in a production environment is not acceptable to most developers and system managers as it will lead to loss of computational node-hours used by the application after the last checkpoint.

While some of this congestion may be alleviated by CEMRs such as feedback of congestion information to applications to trigger rebalancing [29] or to scheduling/resource managers to preferentially allocate nodes (e.g., via mechanisms such as slurm's [79] node weight), some may be unavoidable since all networks have finite bandwidth.

On *Blue Waters* a topology-aware scheduling (TAS) [41, 82] scheme is used to decrease the possibility of application communication interference by assigning, by default [12], node allocations that are constrained within small-convex prisms with respect to the HSN topology. Jobs that exceed half a torus will still route outside the allocation and possibly interfere with other jobs and vice versa; a non-default option can be used to avoid placement next to such jobs. The I/O routers represent fixed, and roughly evenly distributed, proportional portions of the storage subsystem. Since the storage subsystem components, including I/O routers, are allocated (for writes) in a round robin (by request order) manner independent of TAS allocations, storage I/O communications will generally use network links both within and outside the geometry of the application's allocation and can also be a cause of interference between applications.

3 Data Sources and Data Collection Tools

This section describes the datasets and tools used to collect data at scale to enable both runtime and long-term characterization of network congestion. We leverage vendor-provided and specialized tools to enable collection and real-time streaming of data to a remote compute node for analysis and characterization. Data provided or exposed on all Cray Gemini systems includes: OS and network performance counter data, network resilience-related logs, and workload placement and status logs. In this study, we used five months (Jan 01 to May 31, 2017) of production network performance-related data (15 TB), network resilience-related logs (100 GB), and application placement logs (7 GB). Note that the methodologies addressed in this work rely only on the availability of the data, independent of the specific tools used to collect the data.

Network Performance Counters: Network performance-related information on links is exposed via Cray's `gpcdr` [35] kernel module. Lustre file system and RDMA traffic information is exposed on the nodes via `/proc/fs` and `/proc/kgnilnd`. It is neither collected nor made available for analysis via vendor-provided collection mechanisms. On Blue Waters, these data are collected and transported off the system for storage and analysis via the Lightweight Distributed Metric Service (LDMS) monitoring framework [17]. In this work, we use the following information: directionally aggregated network traffic (bytes and packets) and length of stalls due to credit depletion; Lustre file system read and write bytes; and RDMA bytes transmitted and received. LDMS samplers collect those data at 60-second intervals and calculate derived metrics, such as the percent of time spent in stalls (P_{Ts}) and percent of total bandwidth used over the last interval. LDMS daemons synchronize their sampling to within a few *ms* (neglecting clock skew) in order to provide coherent *snapshots* of network state across the whole system.

Network Monitoring Logs: Network failures and congestion levels are monitored and mitigated by Cray's *xtnlrd* software. This software further logs certain network events in a well-known format in the *netwatch* log file. Significant example log lines are provided in Cray documents [33, 34]. Regular expression matching for these lines is implemented in LogDiver [66], a log-processing tool, which we use to extract the occurrences, times, and locations of link failures and CPEs.

Workload Data: Blue Waters utilizes the Moab scheduler, from which application queue time, start time, end time, exit status, and allocation of nodes can be obtained. The workload dataset contains information about 815,006 application runs that were executed during our study period. A detailed characterization of *Blue Waters* workloads can be found in Appendix B and *Blue Waters* workload study [58].

Note that we will only be releasing network data. Worload data and network monitoring logs will not be released due to privacy and other concerns.

4 *CR* Extraction and Characterization Tool

This section first describes our motivation for choosing congestion regions (CRs) as a driver for characterizing network congestion, and then describes our methodology (implemented as the *Monet* tool) for extracting *CRs* over each data

collection interval and the classification of those CRs based on severity.

4.1 Why Congestion Regions?

We seek to motivate our choice to characterize congestion regions (*CRs*) and the need for estimates for severity in terms of the stall values. We first show that the charcterization of hotspot links individually do not reveal the spatial and growth characteristics which is needed for diagnosis. Then, we show how characterizing *CRs* is meaningful.

Characterizing hotspot links individually do not reveal regions of congestion. Figure 3 characterizes the median, 99%ile and 99.9%ile duration of the hotspot links by generating the distribution of the duration for which a link persists to be in congestion at $P_{Ts} \geq P_{Ts}$Threshold value. For example, 99.9%ile duration for hotspot links with $P_{Ts} \geq 30$ is 400 minutes (6.67 hours). The measurements show that the median duration of hotspot link at different P_{Ts} thresholds is constantly at ~ 0, however, 99.9%ile duration of hotspot links linearly decreases with increasing P_{Ts} threshold value. Although such characterizations are useful to understand congestion at link-level, they hide the spatial characteristics of congestion such as the existence of multiple pockets of congestion and their spread and growth over time. The lack of such information makes it difficult to understand congestion characteristics and their root cause.

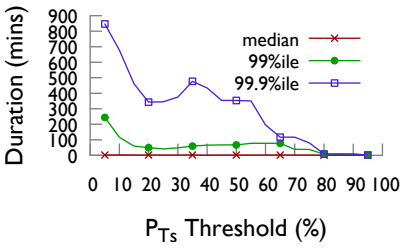
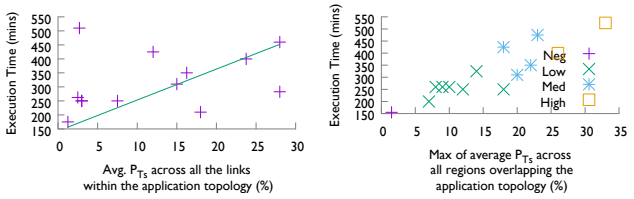


Figure 3: Duration of congestion on links at different P_{Ts} thresholds



(a) Not using **CRs** *(b) Using* **CRs**

Figure 4: Correlating congestion with NAMD application runtime

***CRs* captures relationship between congestion-level and application slowdown efficiently.** In order to determine possible severity values and show effectiveness of *CRs* in determining application slowdown, we extracted from the production Blue Waters dataset a set of NAMD [77][8] runs each of which ran on 1000 nodes with the same input parameters. We chose NAMD because it consumes approximately 18% of total node-hours available on Blue Waters[9]. Figure 4a shows the execution time of each individual run with respect to the *average* P_{Ts} over all links within the allocated application topology. (Here we leverage TAS to determine severity value estimates based on the values within the allocation; that is not a condition for the rest of this work.) Figure 4a shows that execution time is perhaps only loosely related to the average P_{Ts}; with correlation of 0.33 . In contrast, 4b shows the relationship of the application execution time with the *maximum* average P_{Ts} over all *CRs* (defined in 4.2) within the allocated topology; with correlation of 0.89. In this case, execution time increases with increasing maximum of average P_{Ts} over all regions. We found this relationship to hold for other scientific applications. This is a motivating factor for the extraction of such *congestion regions* (*CRs*) as indicators of 'hot-spots' in the network. We describe the methodology for *CR* extraction in the next section.

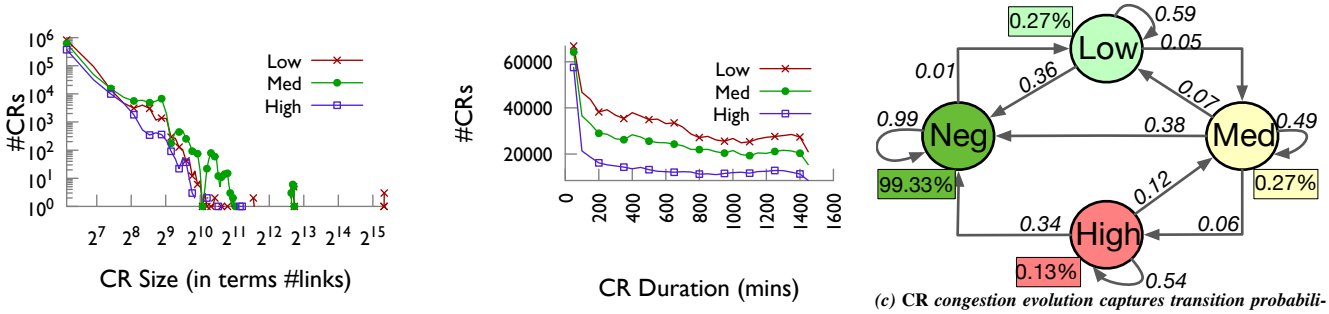
In addition, we selected approximate ranges of P_{Ts} values, corresponding to increasing run times, to use as estimates for the severity levels as these can be easily calculated, understood and compared. These levels are indicated as symbols in the figure. Explicitly, we assign 0-5% average P_{Ts} in a *CR* as *Negligible* or *'Neg'*, 5-15% as *'Low'*, 15-25% as *'Medium'*, and $> 25\%$ as *'High'*. These are meant to be qualitative assignments and not to be rigorously associated with a definitive performance variation for all applications in all cases, as the network communication patterns and traffic volumes vary among HPC applications. We will use these ranges in characterizations in the rest of this work. More accurate determinations of impact could be used in place of these in the future, without changing the validity of the *CR* extraction technique.

4.2 Extracting Congestion Regions

We have developed an unsupervised clustering approach for extracting and localizing regions of congestion in the network by segmenting the network into groups of links with similar congestion values. The clustering approach requires the following parameters: (a) network graph (G), (b) congestion measures (v_s for each vertex v in G), (c) neighborhood distance metric (d_δ), and (d) stall similarity metric (d_λ). The network is represented as a graph G. Each link in the network is represented as a vertex v in G, and two vertices are connected if the corresponding network links are both connected to the same switch (i.e., the switch is an edge in the graphs). For each vertex v, the congestion measures(s) are denoted by the vector v_s, which is composed of credit stalls and inq

[8]NAMD has two different implementations: (a) uGNI shared memory parallel (SMP)-based, and (b) MPI-based. In this work, unstated NAMD refers to uGNI SMP-based implementation. uGNI is user level Generic Network Interface [83].

[9]This was best effort extraction and the NAMD application runs may not be exactly executing the same binary or processing the same data, as user may have recompiled the code with a different library or used the same name for dataset while changing the data. There is limited information to extract suitable comparable runs from historical data that are also subject to allocation and performance variation.



(a) Distribution of CR sizes.

(b) Distribution of CRs.

(c) **CR** *congestion evolution captures transition probabilities from one severity* **state** *to another. Percentage in boxes indicates percentage of total link-hours spent in that* **state**.

Figure 5:* CR *size, duration, evolution characterization. # of* CRs *across* 'Low', 'Medium', *and* 'High' *are 9.4e05, 7.3e05, and 4.2e05 respectively.

stalls, which we use independently. Distance metrics d_δ and d_λ are also required, the former for calculating distances between two vertices and the latter for calculating differences among the stalls v_s. We assign each vertex the coordinate halfway between the logical coordinates of the two switches to which that vertex is immediately connected, and we set d_δ to be the L1 norm between the coordinates. Since the Blue Waters data consists of directionally aggregated information as opposed to counters on a per-tile-link (or buffer) basis, then, in our case, d_λ is simply the absolute difference between the two credit-stall or the two inq-stall values of the links, depending on what kinds of regions are being segmented. We consider credit and inq stalls separately to extract *CRs*, as the relationship between the two types of stalls is not immediately apparent from the measurements, and thus require two segmentation passes. Next, we outline the segmentation algorithm.

Segmentation Algorithm The segmentation algorithm has four stages which are executed in order, as follows.

- Nearby links with similar stall values are grouped together. Specifically, they are grouped into the equivalence classes of the reflexive and transitive closure of the relation $\sim_r$ defined by $x \sim_r y \Leftrightarrow d_\delta(x,y) \leq \delta \wedge d_\lambda(x_s - y_s) \leq \theta_p$, where x, y are vertices in G, and δ, θ_p are thresholds for distance between vertices and stall values, respectively.
- Nearby regions with similar average stall values, are grouped together through repetition of the previous step, but with regions in place of individual links. Instead of using the link values v_s, we use the average value of v_s over all links in the region, and instead of using θ_p, we use a separate threshold value θ_r.
- CRs that are below the size threshold σ are merged into the nearest region within the distance threshold δ.
- Remaining CRs with $< \sigma$ links are discarded, so that regions that are too small to be significant are eliminated.

The optimum values for the parameters used in segmentation algorithms, except for δ, were estimated empirically by knee-curve [63] method, based on the number of regions produced. Using that method, the obtained parameter values [10] are: (a) $\theta_p = 4$, (b) $\theta_r = 4$, and (c) $\sigma = 20$. In [63], the authors conclude that the optimum sliding window time is the knee of the curve drawn between the sliding window time and the number of clusters obtained using a clustering algorithm. This decreases truncation errors (in which a cluster is split into multiple clusters because of a small sliding window time) and collision errors (in which two events not related to each other merge into a single cluster because of a large sliding window time). We fixed δ to be 2 in order to consider only links that are two hops away, to capture the local nature of congestion [47]. It should be noted that the region clustering algorithm may discard small isolated regions (size $\leq \sigma$) of high congestion. If such *CRs* do cause high interference, they will grow over time and eventually be captured.

Our algorithm works under several assumptions: (a) congestion spreads locally, and (b) within a CR, the stall values of the links do not vary significantly. These assumptions are reasonable for k-dimensional toroids that use directional-order routing algorithm. The methodology used to derive *CRs* is not dependent on the resource allocation policy (such as TAS). The proposed extraction and its use for characterization is particularly suitable for analysis of network topologies that use directional- or dimensional-order routing. In principle, the algorithm can be applied to other topologies (such as mesh and high-order torus networks) with other metrics (such as stall-to-flit ratio). Furthermore, the region extraction algorithm does not force any shape constraints; thus *CRs* can be of any arbitrary shape requiring us to store each vertex associated with the *CR*. In this work, we have configured the tool to store and display bounding boxes over CRs, as doing so vastly reduces the storage requirements (from TBs of raw data to 4 MB in this case), provides a succinct summary of the network congestion state, and eases visualization.

We validate the methodology for determining the parameters of the region-based segmentation algorithm and its applicability for *CR* extraction by using synthetic datasets, as described in Appendix D.

4.3 Implementation and Performance

We have implemented the region-extraction algorithm as a modified version of the region growth segmentation algorithm [78] found in the open-source PointCloud Library (PCL) [9] [4]. The tool is capable of performing run-time extraction of *CRs* even for large-scale topologies. Using the

[10] stall thresholds are scaled by 2.55× to represent the color range (0-255) for visualization purposes

Blue Waters dataset, Monet mined *CRs* from each 60-second snapshot of data for 41,472 links in ∼7 seconds; Monet was running on a single thread of a 2.0 GHz Intel Xeon E5-2683 v3 CPU with 512 GB of RAM. Thus, on Blue Waters *Monet* can be run at run-time, as the collection interval is much greater than CR extraction time. Since Monet operates on the database, it works the same way whether the data are being streamed into the database or it is operating on historical data.

5 Characterization Results

In this section, we present results of the application of our analysis methodology to five months of data from a large-scale production HPC system (*Blue Waters*) to provide characterizations of *CRs*. Readers interested in understanding traffic characteristics at the link and datacenter-level may refer to a related work [16].

5.1 Congestion Region Characterization

Here we assess and characterize the congestion severity.

CR-level Size and Severity Characterizations: Figure 5a shows a histogram[11] of *CR* sizes in terms of the number of links for each congested *state* (i.e., not including *'Neg'*). Figure 5b show a histogram of the durations of *CRs* across *'Low'*, *'Medium'* and *'High'* congestion levels. These measurements show that unchecked congestion in credit-based interconnects leads to:

- High growth and spread of congestion leading to large *CRs*. The max size of *CRs* in terms of number of links was found to be 41,168 (99.99% of total links), 6,904 (16.6% of total links), and 2,324 (5.6% of total links) across *'Low'*, *'Medium'* and *'High'* congestion levels respectively, whereas the 99th percentile of the[12] *CR* size was found to be 299, 448, and 214 respectively.
- Localized congestion hotspots, i.e., pockets of congestion. *CRs* rarely spread to cover all of the network. The number of *CRs* decreases (see Figure 5a) with increasing size across all severity *states* except for *'Low'* for which we observe increase at the tail. For example, there are ∼16,000 CRs in the *'High'* which comprise 128 links but only ∼141 CRs of size ∼600.
- Long-lived congestion. The *CR* count decreases with increasing duration, however there are many long-lived *CRs*. The 50%ile, 99%ile and max duration of *CRs* across all states were found to be 579 minutes (9.7 hours), 1421 minutes (23.6 hours), and 1439 minutes (24 hours) respectively, whereas the 50%ile, 99%ile and max P_{Ts} of *CRs* was found to be 14%, 46%, and 92%, respectively. *CR* duration did not change significantly across *'Low'*, *'Medium'*, and *'High'*.

CR Evolution and State Probability: Figure 5c shows the transition probabilities of the *CR* states. The percentage in the box next to each *state* shows the percentage of total link-hours[13] spent in that *state*. It can be interpreted as the probability that a link will be congested at a severity *state* at a given time. For example, there is a probability of 0.10% that a link will be in the *'High'*. These measurements show that:

- The vast majority of link-hours (99.3% of total link-hours) on Blue Waters are spent in *'Neg'* congestion. Consideration of a grosser congestion metric, such as the average stall time across the entire network, will not reveal the presence of significant *CRs*.
- Once a *CR* of *'Low'*, *'Medium'* or *'High'* congestion is formed, it is likely to persist (with a probability of more than 0.5) rather than decrease or vanish from the network.

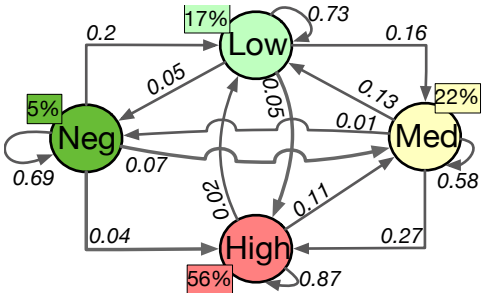


***Figure 6:* Network *congestion evolution captures transition probabilities from one severity* state *to another. Percentage numbers in boxes indicates percentage of total system wall clock time spent in that* state.**

5.2 Network-level Congestion Evolution and Transition Probabilities

In this section, we assess and characterize the overall network congestion severity state. The overall network congestion severity *state* is the *state* into which the highest CR falls. That assignment is independent of the overall distribution of links in each *state*. Figure 6 shows the probabilities that transitions between network *states* will occur between one measurement interval and the next. The rectangular boxes in the figure indicate the fraction of time that the network resides in each *state*. These measurements show the following:

- While each *individual* link of the entire network is most often in a *state* of *'Neg'* congestion, there exists at least one *'High'* CR for 56% of the time. However, *'High'* CRs are small; in Section 5.1, we found that 99th percentile size of *'High'* is 214 links. Thus, the *Blue Waters* network *state* is nearly always non-negligible (95%), with the "High" *state* occurring for the majority of the time.
- There is a significant chance that the current network *state* will persist or increase in severity in the next measurement period. For example, there is an 87% chance that it will stay in a *'High' state*.
- A network *state* is more likely to drop to the next lower *state* than to drop to *'Neg'*.
- Together these factors indicate that congestion builds and subsides slowly, suggesting that it is possible to fore-

[11] plotted as lines and every tenth point marked on the line using a shape for clarity.

[12] We will use %ile to denote percentile in the rest of the paper.

[13] Link-hours are calculated by $\sum$(#links in *Region*) × (measurement time-window) for each *state*.

cast (within bounds) congestion levels. Combined with proactive localized congestion mitigation techniques and CEMRs, such forecasts could significantly improve overall system performance and application throughput.

5.3 Application Impact of CR

The potential impact of congestion on applications can be significant, even when the percentage of link-hours spent in non-*'Neg'* congested regions is small. While we cannot quantify congestion's impact on all of the applications running on Blue Waters (as we lack ground truth information on particular application runtimes without congestion), we can quantify the impact of congestion on the following:

- Production runs of the NAMD application [77]. The worst-case NAMD execution runtime was 3.4× slower in the presence of high *CRs* relative to baseline runs (i.e., negligible congestion). The median runtime was found be 282 minutes, and hence worst-case runtime was 1.86× slower than the median runtime. This is discussed in more detail in Section 4.1.
- In [16], authors show that benchmark runs of PSDNS [74] and AMR [2] on 256 nodes slowed down by as much as 1.6× even at low-levels of congestion ($5\% < P_{Ts} \leq 15\%$).

To find a upper bound on the number of potentially impacted applications, we consider the applications whose allocations are directly associated with a router in a *CR*. Out of 815,006 total application runs on Blue Waters, over 16.6%, 12.3%, and 6.5% of the unique application runs were impacted by *'Low'*, *'Medium'*, and *'High'* CRs, respectively.

5.4 Congestion Scenarios

In this section, we show how *CRs* manifest under different congestion scenarios: (a) system issues (e.g. changes in system load), (b) network-component failures (e.g. link failures), and (c) intra-application contention. Since the *CRs* are described as bounding boxes with coordinates described in relation to the 3D torus, they can easily be visualized in conjunction with applications' placements at runtime on the torus. *CRs* of *'Neg'* congestion are not shown in the figures.

Congestion due to System Issues: Network congestion may result from contention between different applications for the same network resources. That can occur because of a change in system load (e.g. launches of new applications) or change in application traffic that increases contention on shared links between applications.

Figure 7(i) shows four snapshots, read clockwise, of extracted *CRs*, including size and severity state, for different time intervals during a changing workload. Figure 7(i)(a) shows that *'Low'* (blue) *CRs* when most of the workload consists of multiple instances of MPI-based *NAMD* [77]. The overall network state was thus *'Low'*. The *CRs* remained relatively unchanged for 40 minutes, after which two instances of NAMD completed and *Variant Calling* [37] was launched. Three minutes after the launch, new *CRs* of increased severity

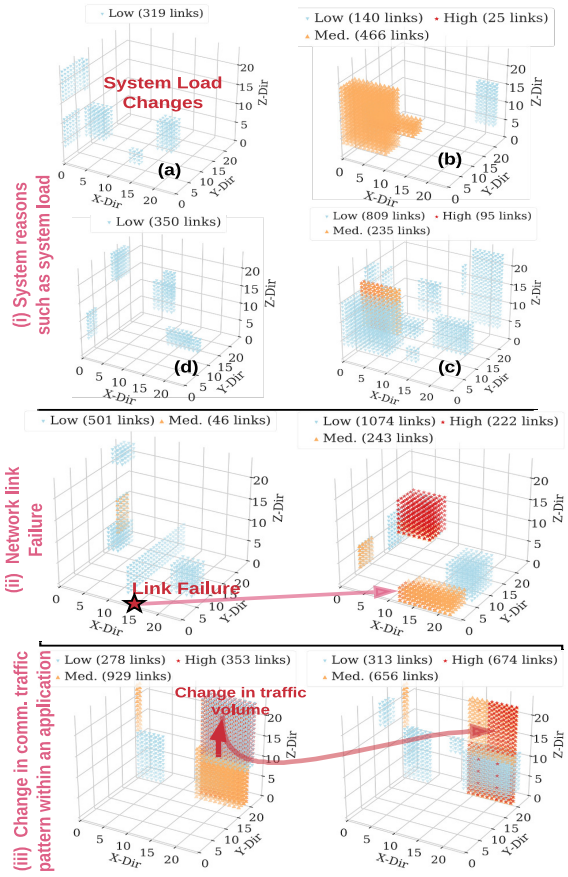


Figure 7: Case studies: network congestion is shown due to (i) system issues (such as introduction of new applications), (ii) failures (such as network link failure), and (iii) change in communication pattern within the application.

occurred (Figure 7(i)(b,c)). The *'High'* (red) [14] and *'Medium'* (orange) severity *CRs* overlapped with the applications.

The increase in the severity of congestion was due to high I/O bandwidth utilization by the *Variant Calling* application. The overall network state remained *'High'* for ∼143 minutes until the *Variant Calling* application completed. At that time, the congestion subsided, as shown in Figure 7(i)(d).

Congestion Due to Network-component Failures: Network-related failures are frequent [55, 68] and may lead to network congestion, depending on the traffic on the network and the type of failure. In [55], the mean time between failures (MTBF) for directional links in Blue Waters was found to be approximately 2.46*e*06 link-hours (or 280 link-years). Given the large number of links (41,472 links) on Blue Waters, the expected mean time between failure of a link across the system is about 59.2 hours; i.e., Blue Waters admins can expect one directional-link failure every 59.2 hours.

Failures of directional links or routers generally lead to

[14] not visible and hidden by other regions.

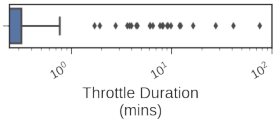
occurrences of *'High' CRs*, while isolated failures of a few switch links (which are much more frequent) generally do not lead to occurrences of significant *CRs*. In this work we found that 88% of directional link failures led to congestion; however, isolated failures of switch links did not lead to significant *CRs* (i.e., had *'Neg' CRs*).

Figure 7(ii) shows the impact of a network blade failure that caused the loss of two network routers and about 96 links (x,y,z location of failure at coordinates (12,3,4) and (12,3,3)). Figure 7(ii)(a) shows the congestion *CRs* before the failure incident and Figure 7(ii)(b) shows the *CRs* just after the completion of the network recovery. Immediately after failure, the stalls increased because of the unavailability of links, requiring the packets to be buffered on the network nodes. The congestion quickly spread into the geometry of nearby applications in the torus. Failure of a blade increased the overall size (in number of links) of *'Low' CRs* by a factor of 2, and of *'Medium' CRs* by a factor of 4.2, and created previously non existent *'High' CRs* with more than 200 links.

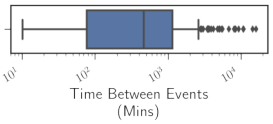
Congestion Due to Intra-Application Issues: Congestion within an application's geometry (intra-application contention) can occur even with TAS. Figure 7(iii) shows congestion *CRs* while the uGNI-based shared memory parallel (SMP) *NAMD* application on more than 2,000 nodes. The application is geometrically mapped on the torus starting at coordinates (15, 18, 0) and ending at coordinates (1, 21, 23) (wrapping around). The congestion *CRs* alternate between the two states shown (state 1 shown in Figure 7(iii)(a), and state 2, shown in Figure 7(iii)(b)) throughout the application run-time because of changes in communication patterns corresponding to the different segments of the NAMD code.

Intra-application contention is less likely to elevate to cause global network issue, unless the links are involved in global (e.g., I/O) routes, or if the resulting congestion is heavy enough to trigger the system-wide mitigation mechanism (see Section 2.2).

Importance of diagnosis: In this section, we have identified three high-level causes of congestion, which we categorize as (a) system issues, (b) network-component failures, and (c) intra-application contention. For each cause, system managers could trigger one of the following actions to reduce/manage congestion. In the case of intra-application congestion, an automated MPI rank remapping tool such as TopoMapping [46], could be used to change traffic flow bandwidth on links to reduce congestion on them. In the case of inter-application congestion (caused by system issues or network failures), a node-allocation policy (e.g., TAS) could use knowledge of congested regions to reduce the impact of congestion on applications. Finally, if execution of an application frequently causes inter-application congestion, then the application should be re-engineered to limit chances of congestion.



(a) Box plot of duration of throttling



(b) Box plot of time between triggers of congestion mitigation events

Figure 8: Characterizing Cray Gemini congestion mitigation events.

6 Using Characterizations: Congestion Response

In this section, we first discuss efficacy of Cray CPEs and then show how our *CR*-based characterizations can be used to inform effective responses to performance-degrading levels of congestion.

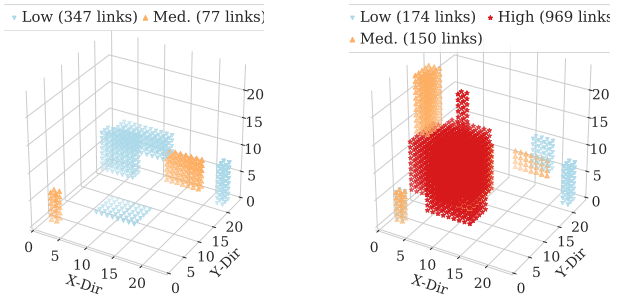
Characterizing Cray *CPEs*: Recall from Section 2 that the vendor-provided congestion mitigation mechanism throttles all NIC traffic injection into the network irrespective of the location and size of the triggering congestion region. This mitigation mechanism is triggered infrequently by design and hence may miss detections and opportunities to trigger more targeted congestion avoidance mechanisms. On Blue Waters, congestion mitigation events are generally active for small durations (typically less than a minute), however, in extreme cases, we have seen them active for as long as 100 minutes. Each throttling event is logged in *netwatch* log files.

We define a *congestion mitigation event (CME)* as a collection of one or more throttling events that were coalesced together based on a sliding window algorithm [63] with a sliding window of 210 seconds, and we use this to estimate the duration of the vendor-provided congestion mitigation mechanisms. Figure 8a and 8b shows a box plot of duration of and time between *CMEs* respectively. The analysis of *CMEs* shows that :

- CMEs were triggered 261 times; 29.8% of which did not alleviate congestion in the system. Figure 9 shows a case where the size and severity of *CRs* increases after a series of throttling events.
- The median time between triggers of CMEs was found to be 7 hours. The distribution of time between events is given in Figure 8b.
- CMEs are generally active for small durations (typically less than a minute), however, in extreme cases, we have seen them active for as long as 100 minutes.
- 8% of the application runs were impacted with over 700 of those utilizing $>$ 100 nodes.

These observations motivate the utility of augmenting the vendor supplied solution of global traffic suppression to manage exceptionally high congestion bursts with our more localized approach of taking action on *CRs* at a higher system-level of granularity to alleviate sources of network congestion.

***CR*-based congestion detection to increase mitigation effectiveness:** *CR* based characterizations can potentially im-



(a) Congestion before triggering of CPE

(b) Congestion after 30 minutes of CPE

Figure 9: A case in which a congestion protection event (CPE) failed to mitigate the congestion

prove congestion mitigation and CEMR effectiveness by more accurately determining which scenarios should be addressed by which mechanisms and by using the identified *CRs* to trigger localized responses more frequently than Cray *CMEs*. That approach is motivated by our discovery (see Section 5.2) that the network is in a *'High'* congestion *state* the majority of the time, primarily because of *CRs* of small size but significant congestion severity.

We define a *Regions Congestion Event (RCE)* as a time-window for which each time instance has at least one region of *'High'* congestion. We calculate it by combining the *CR* evaluations across 5-minute sliding windows. Figure 10 shows boxplots of (a) average credit P_{TS} across all extracted *CRs* during RCEs', (b) average inq P_{TS} across all RCEs', (c) times between RCE, and (d) durations of the RCEs'. These measurements show

- Relative to the vendor-provided congestion mitigation mechanisms, our characterization results in 13× more events (3390 RCEs) upon which we could potentially act.
- Vendor provided congestion mitigation mechanisms trigger on 8% (261 of 3390) of RCEs.
- The average P_{TS} of maximum inq- and credit-stall across all extracted regions present in *RCEs* is quite high, at 33.8% and 27.4%, respectively.
- 25% of 3390 RCEs lasted for more than 30 minutes, and the average duration was found to be approximately an hour.

CRs discovery could also be used for informing congestion aware scheduling decisions. Communication-intensive applications could be preferentially placed to not contend for bandwidth in significantly congested regions or be delayed from launching until congestion has subsided.

7 Using Characterizations: Diagnosing Causes of Congestion

Section 5.4 identifies the root causes of congestion and discusses the the importance of diagnosis. Here we explore that idea to create tools to enable diagnosis at runtime.

7.1 Diagnosis Methodology and Tool

We present a methodology that can provide results to help draw a system manager's attention to anomalous scenarios and potential offenders for further analysis. We can combine system information with the *CR*-characterizations to help diagnose causes of significant congestion. Factors include applications that inject more traffic than can be ejected into the targets or than the traversed links can transfer, either via communication patterns (e.g., all-to-all or many-to-one) or I/O traffic, and link failures. These can typically be identified by observation(s) of anomalies in the data.

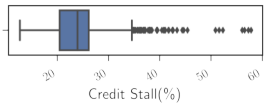
Mining Candidate Congestion-Causing Factors For each congestion Region, CR_i, identified at time T, we create two tables $\mathscr{A}_{CR_i}(T)$ and $\mathscr{F}_{CR_i}(T)$, as described below.

$\mathscr{A}_{CR_i}(T)$ *table*: Each row in $\mathscr{A}_{CR_i}(T)$ corresponds to an application that is within $N_{hops} \leq 3$ hops away from the bounding box of the congestion region CR_i. $\mathscr{A}_{CR_i}(T)$ contains information about the application and its traffic characteristics across seven *traffic features*: (a) application name, (b) maximum read bytes per minute, (c) maximum write bytes per minute, (d) maximum RDMA read bytes per minute, (e) maximum RDMA write bytes per minute, (f) maximum all-to-all communication traffic bytes per minute, and (g) maximum many-to-one communication traffic bytes per minute, where the maximums are taken over the past 30 minutes, i.e., the most recent 30 measurement windows. The list of applications that are within N_{hops} away from congestion region CR_i are extracted from the *workload data*. The measurements for features (a) to (e) are extracted by querying network performance counter data, whereas we estimate the features (f) and (g) are estimated from Network performance counter data by taking several bisection cuts over the application geometry and comparing node traffic ingestion and ejection bytes among the two partitions of the bisection cut.

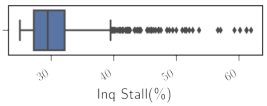
$\mathscr{F}_{CR_i}(T)$ *table*: Each row in $\mathscr{F}_{CR_i}(T)$ corresponds to an application that is within $N_{hops} \leq 3$ away from the congestion boundary of CR_i. $\mathscr{F}_{CR_i}(T)$ contains information about failure events across three *failure features*: (a) failure timestamp, (b) failure location (i.e., coordinates in the torus), and (c) failure type (i.e., switch link, network link, and router failures). Lists of failure events that are within N_{hops} away from congestion region CR_i are extracted from *network failure data*.

Identifying Anomalous or Extreme Factors: The next step is to identify extreme application traffic characteristics or network-related failures over the past 30 minutes that have led to the occurrence of CRs. For each traffic feature in $\mathscr{A}_{CR_i}(T)$, we use an outlier detection method to identify the top k applications that are exhibiting anomalous behavior. The method uses the numerical values of the features listed in table $\mathscr{A}_{CR_i}(T)$. Our analysis framework uses a median-based outlier detection algorithm proposed by Donoho [40] for each CR_i. According to [40], the median-based method is more robust than mean-based methods for skewed datasets. Because *CRs* due to network-related failure events [15] are rare relative to congestion caused by other factors, all failure events that

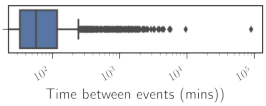
[15] In this paper, we do not consider the effect of lane failures on congestion.



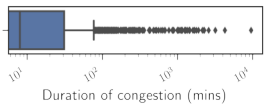
(a) ***Boxplot of average credit stall across extracted congestion events.***



(b) ***Boxplot of average inq-stall across extracted congestion events.***



(c) ***Boxplot of time between congestion events.***



(d) ***Boxplot of duration of congestion.***

Figure 10: Characterization of Regions Congestion Events (RCE).

occur within N_{hops} of CR_i in the most recent 30 measurement windows are marked as anomalous.

Generating Evidence: The last step is to generate evidence for determining whether anomalous factors identified in the previous step are truly responsible for the observed congestion in the *CR*. The evidence is provided in the form of a statistical correlation taken over the most recent 30 measurement time-windows between the moving average stall value of the links and the numerical traffic feature(s) obtained from the data (e.g., RDMA read bytes per minute of the application) associated with the anomalous factor(s). For failure-related anomalous factors, we calculate the correlation taken over the most recent 30 measurement time-windows between the moving average of observed traffic summed across the links that are within N_{hops} away from the failed link(s) and the stall values[16]. A high correlation produces the desired evidence. We order the anomalous factors using the calculated correlation value regardless of the congestion cause. Additionally, we show a plot of stall values and the feature associated with the anomalous factor(s) to help understand the impact of the anomalous factor(s) on congestion.

The steps in this section were only tested on a dataset consisting of the case studies discussed in Section 5.4 and 7 because of lack of ground truth labels on root causes. Creation of labels on congestion causes requires significant human effort and is prone to errors. However, we have been able to generate labels by using the proposed unsupervised methodology, which provides a good starting point for diagnosis.

7.2 Comprehensive Congestion Analysis

In this section, we describe an example use case in which our analysis methodologies were used to detect and diagnose the congestion in a scenario obtained from real data for which the ground truth of the cause was available. The overall steps involved in using our methodologies, included in our *Monet* implementation, for congestion detection and diagnosis are summarized in Figure 11 and described in Section 7. Not all of the steps discussed below are currently automated, but we are working on automating an end-to-end pipeline.

Step 1. Extraction of CR. Figure 11(a) shows that our analysis indicated wide spread high-level congestion across the system (see the left graph in Figure 11(a)). An in-depth analysis of the raw data resulted in identification/detection of congestion regions (see the top-right graph in Figure 11(a)).

Step 2. Congestion diagnosis. There are 3 steps associated with diagnosing the cause of the congestion.

Step 2.1. Mining candidate factors. To determine the cause of the congestion, we correlated the CR-data with application-related network traffic (for all applications that overlapped with or were near the congestion regions) and network information to generate candidate factors that may have led to congestion. In this example, there were no failures; hence, this analysis generated only application-related candidate factors $\mathscr{A}_{CR_i}$, as shown in Figure 11.

Step 2.2. Identifying anomalous factors. Next, we utilized the application traffic characteristics from candidate factors observed over the last 30 minutes (i.e., many-to-one or all-to-all traffic communication, and file system statistics such as read or write bytes) to identify anomalous factors by using a median-based outlier detection algorithm. In our example, as indicated in Figure 11(b), the offending application was "Enzo" which was running on 32 nodes allocated along the "Z" direction at location (X,Y,Z) = (0,16,16) (indicated by a black circle in Figure 11(a)). At the time of detection, "Enzo" was reading from the file system at an average rate of 4 GB/min (averaged over past 30 minutes and with a peak rate of 70 GB/min), which was 16x greater than the next-highest rate of read traffic by any other application in that time-window. The $\mathscr{A}_{CR_i}(T)$ for RDMA read bytes/min was 70 GB/min. The tool identified the RDMA read bytes/min of the "Enzo" application as the outlier feature. Hence, "Enzo" was marked as the anomalous factor that led to the congestion.

Step 2.3. Generating evidence. Once the potential cause had been established, further analysis produced additional evidence (e.g., distribution and correlation coefficient associated with link stalls in the congestion time window) to validate/verify the diagnosis results produced in Step 2.2. Figure 11(c), in the top graph, shows a plot of the sum of stall rates on all links for all the Gemini routers local to the compute nodes used by the offending application, (i.e., Enzo) (normalized to the total stall rate throughout the duration of the application run). The two peaks (marked) in this top plot correspond to the increase in read bytes (normalized to total read bytes during the application run) shown in the bottom plot. Note that abnormal activity (an excessive amount of traffic to the file system) occurred around 10:10 AM (as shown Figure 11(c)), which was about 20 minutes before the severe congestion developed in the system (seen in Figure 11(a)). A

[16] Increase in traffic near a failed link leads to congestion as shown in Section 5.4.

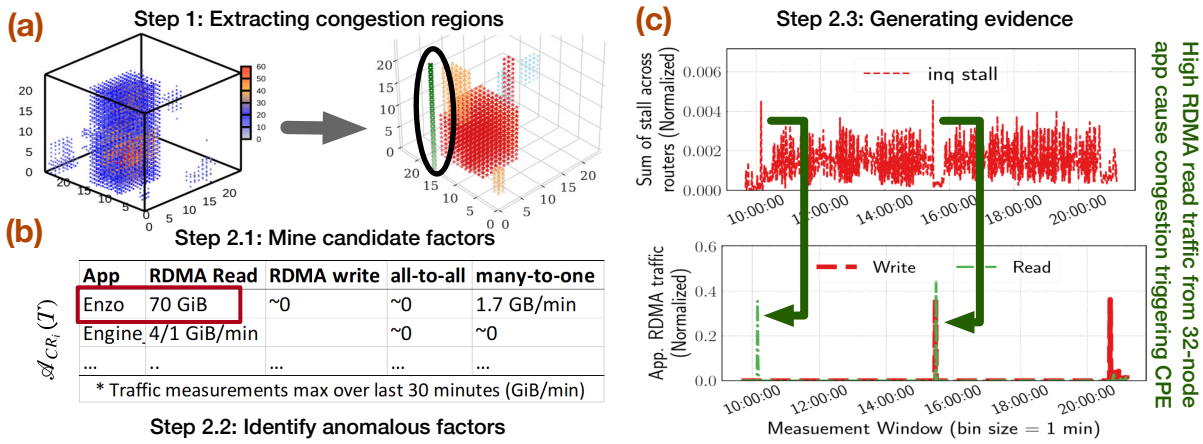


Figure 11: Detection and Diagnosis methodology applied to real-scenario

"Medium" level of congestion was detected in the system spanning a few links (i.e., the congestion region size was small) at the time of the increased read traffic. Thus the cause was diagnosed to be "Enzo". Although, in this example scenario, the Cray congestion mitigation mechanism was triggered, it was not successful in alleviating the network congestion. Instead, the CR size grew over time, impacting several applications. "Enzo" was responsible for another triggering of the congestion mitigation mechanism at 3:20 PM (see the top graph in Figure 11(c)). Monet detected and diagnosed it correctly.

8 Related Work

There is great interest in assessing performance anomalies in HPC systems with the goal of understanding and minimizing application performance variation [25, 86, 86, 88]. Monitoring frameworks such as Darshan [65], Beacon [87] and Kaleidoscope [54] focuses on I/O profiling and performance anomaly diagnosis. Whereas, our work focuses on assessing network congestion in credit-flow based interconnection networks. Typically congestion studies are based on measurements of performance variation of benchmark applications in production settings [25, 88] and/or modeling that assumes steady state utilization/congestion behavior [23, 52, 64, 73], and thus do not address full production workloads.

There are research efforts on identifying hotspots and mitigating the effects of congestion at the application or system-layer (e.g., schedulers). These approaches include (a) use of application's own indirect measures, such as messaging rates [25], or network counters from switch that are accessible only from within an allocation [38, 49, 50, 76], and therefore miss measurements of congestion along routes involving switches outside of the allocation; and (b) use of global network counter data [17, 26, 28, 30], however, these have presented only representative examples of congestion through time or executed a single application on the system [26].

In contrast, this work is the first long-term characterization of high-speed interconnect network congestion of a large-scale production system, where network resources are shared by nodes across disparate job allocations, using global network counters. The characterizations and diagnosis enabled by our work can be used to inform application-level [29] or system-level CEMRs (e.g., use of localized throttling instead of network-wide throttling). Perhaps, the closest work to ours is [22] which is an empirical study of cloud data center networks with a focus on network utilization and traffic patterns, and Beacon [87] which was used on TaihuLight [43] to monitor interconnection network inter-node traffic bandwidth. Like others, these works did not involve generation and characterization of congestion regions, diagnosis of congestion causes, nor a generalized implementation of a methodology for such, however, we did observe some complimentary results in our system (e.g., the existence of hot-spot links, the full bisection bandwidth was not always used, assessment of persistence of congestion in links).

Finally, for datacenter networks, efforts such as ExpressPass [32], DCQCN [89], TIMELY [69] focus on preventing and mitigating congestion at the network-layer whereas efforts such as PathDump [84], SwitchPointer [85], PathQuery [72], EverFlow [90], NetSight [51], LDMS [17] and TPP [53] focus on network monitoring. These approaches are tuned for TCP/IP networks and are orthogonal to the work presented here. Our approach is complementary to these efforts as it enables characterization of congestion regions (hotspots) and identification of congestion causing events.

9 Conclusions and Future Work

We present novel methodologies for detecting, characterizing, and diagnosing network congestion. We implemented these capabilities and demonstrated them using production data from NCSA's 27,648 node, Cray Gemini based, Blue Waters system. While we utilized the scale and data availability of the Blue Waters system to validate our approach, the methodologies presented are generally applicable to other credit-based k-dimensional meshes or toroidal networks. Our future work will involve extending the presented techniques to other network technologies and topologies (see Appendix C).

References

[1] Blue Waters. https://bluewaters.ncsa.illinois.edu.

[2] Charm++ MiniApps . http://charmplusplus.org/benchmarks/#amr.

[3] CLOUD TPU: Train and run machine learning models faster than ever before. https://cloud.google.com/tpu/.

[4] Color-based region growing segmentation. http://pointclouds.org/documentation/tutorials/region_growing_rgb_segmentation.php.

[5] Cray in Azure. https://azure.microsoft.com/en-us/solutions/high-performance-computing/cray/.

[6] Introducing the new HB and HC Azure VM sizes for HPC. https://azure.microsoft.com/en-us/blog/introducing-the-new-hb-and-hc-azure-vm-sizes-for-hpc/.

[7] Monet. https://github.com/CSLDepend/monet.

[8] NVIDIA DGX-1: The Fastest Deep Learning System. https://devblogs.nvidia.com/dgx-1-fastest-deep-learning-system.

[9] Point Cloud Library. http://pointclouds.org.

[10] Post-K Supercomputer Overview. http://www.fujitsu.com/global/Images/post-k-supercomputer-overview.pdf.

[11] Top 500 HPC systems. https://www.top500.org/.

[12] Topology Aware Scheduling. https://bluewaters.ncsa.illinois.edu/topology-aware-scheduling.

[13] https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/.

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

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

[16] Anonymized for double blind submission, 2019.

[17] A. Agelastos et al. Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications. In *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 154–165, 2014.

[18] Yuichiro Ajima, Tomohiro Inoue, Shinya Hiramoto, Shunji Uno, Shinji Sumimoto, Kenichi Miura, Naoyuki Shida, Takahiro Kawashima, Takayuki Okamoto, Osamu Moriyama, et al. Tofu interconnect 2: System-on-chip integration of high-performance interconnect. In *International Supercomputing Conference*, pages 498–507. Springer, 2014.

[19] Yuichiro Ajima, Shinji Sumimoto, and Toshiyuki Shimizu. Tofu: A 6d mesh/torus interconnect for exascale computers. *Computer*, 42(11), 2009.

[20] Yuuichirou Ajima, Tomohiro Inoue, Shinya Hiramoto, and Toshiyuki Shimizu. Tofu: Interconnect for the k computer. *Fujitsu Sci. Tech. J*, 48(3):280–285, 2012.

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

[22] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM, 2010.

[23] Abhinav Bhatele, Nikhil Jain, Yarden Livnat, Valerio Pascucci, and Peer-Timo Bremer. Analyzing network health and congestion in dragonfly-based supercomputers. In *Proc. IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS 2016)*, 2016.

[24] Abhinav Bhatelé and Laxmikant V Kalé. Quantifying network contention on large parallel machines. *Parallel Processing Letters*, 19(04):553–572, 2009.

[25] Abhinav Bhatele, Kathryn Mohror, Steven H Langer, and Katherine E Isaacs. There goes the neighborhood: performance degradation due to nearby jobs. In *2013 SC - International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 41:1–41:12, 2013.

[26] Abhinav Bhatele, Andrew R Titus, Jayaraman J Thiagarajan, Nikhil Jain, Todd Gamblin, Peer-Timo Bremer, Martin Schulz, and Laxmikant V Kale. Identifying the culprits behind network congestion. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 113–122. IEEE, 2015.

[27] Brett Bode, Michelle Butler, Thom Dunning, Torsten Hoefler, William Kramer, William Gropp, and Wen-mei Hwu. The blue waters super-system for super-science. In *Contemporary high performance computing*, pages 339–366. Chapman and Hall/CRC, 2013.

[28] J. Brandt, K. Devine, and A. Gentile. Infrastructure for In Situ System Monitoring and Application Data Analysis. In *Proc. Wrk. on In Situ Infrastructures for Enabling Extreme-scale Analysis and Viz.*, 2015.

[29] J Brandt, K Devine, A Gentile, and Kevin Pedretti. Demonstrating improved application performance using dynamic monitoring and task mapping. In *Cluster Computing, IEEE Int'l Conf. on*. IEEE, 2014.

[30] J. Brandt, E. Froese, A. Gentile, L. Kaplan, B. Allan, and E. Walsh. Network Performance Counter Monitoring and Analysis on the Cray XC Platform. In *Proc. Cray User's Group*, 2016.

[31] Dong Chen, Noel A Eisley, Philip Heidelberger, Robert M Senger, Yutaka Sugawara, Sameer Kumar, Valentina Salapura, David L Satterfield, Burkhard Steinmacher-Burow, and Jeffrey 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*, pages 1–10. IEEE, 2011.

[32] Inho Cho, Keon Jang, and Dongsu Han. Credit-scheduled delay-bounded congestion control for datacenters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 239–252. ACM, 2017.

[33] Cray Inc. Managing Network Congestion in Cray XE Systems. Cray Doc S-0034-3101a Cray Private, 2010.

[34] Cray Inc. Network Resiliency for Cray XE and Cray XK Systems. Cray Doc S-0032-B Cray Private, 2013.

[35] Cray Inc. Managing System Software for the Cray Linux Environment. Cray Doc S-2393-5202axx, 2014.

[36] William J. Dally and Charles L. Seitz. Torus routing chip, June 12 1990. US Patent 4,933,933.

[37] Van der Auwera et al. From fastq data to high-confidence variant calls: the genome analysis toolkit best practices pipeline. *Current protocols in bioinformatics*, pages 11–10, 2013.

[38] M. Deveci, S. Rajamanickam, V. Leung, K. Pedretti, S. Olivier, D. Bunde, U. V. Catalyurek, and K. Devine. Exploiting Geometric Partitioning in Task Mapping for Parallel Computers. In *Proc. 28th Int'l IEEE Parallel and Distributed Processing Symposium*, 2014.

[39] Catello Di Martino, William Kramer, Zbigniew Kalbarczyk, and Ravishankar Iyer. Measuring and understanding extreme-scale application resilience: A field study of 5,000,000 hpc application runs. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, pages 25–36. IEEE, 2015.

[40] David L Donoho. Breakdown properties of multivariate location estimators. Technical report, Technical report, Harvard University, Boston. URL http://www-stat. stanford. edu/˜ donoho/Reports/Oldies/BPMLE. pdf, 1982.

[41] J Enos et al. Topology-aware job scheduling strategies for torus networks. In *Proc. Cray User Group*, 2014.

[42] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, James Reinhard, et al. Cray Cascade: A Scalable HPC System Based on a Dragonfly Network. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 103:1–103:9, 2012.

[43] Haohuan Fu, Junfeng Liao, Jinzhe Yang, Lanning Wang, Zhenya Song, Xiaomeng Huang, Chao Yang, Wei Xue, Fangfang Liu, Fangli Qiao, et al. The sunway taihulight supercomputer: system and applications. *Science China Information Sciences*, 59(7):072001, 2016.

[44] Joshi Fullop et al. A diagnostic utility for analyzing periods of degraded job performance. In *Proc. Cray User Group*, 2014.

[45] Ana Gainaru, Guillaume Aupy, Anne Benoit, Franck Cappello, Yves Robert, and Marc Snir. Scheduling the i/o of hpc applications under congestion. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 1013–1022. IEEE, 2015.

[46] Juan J. Galvez, Nikhil Jain, and Laxmikant V. Kale. Automatic topology mapping of diverse large-scale parallel applications. In *Proceedings of the International Conference on Supercomputing*, ICS '17, pages 17:1–17:10, New York, NY, USA, 2017. ACM.

[47] Pedro Javier García, Francisco J Quiles, Jose Flich, Jose Duato, Ian Johnson, and Finbar Naven. Efficient, scalable congestion management for interconnection networks. *IEEE Micro*, 26(5):52–66, 2006.

[48] Paul Garrison. Why is Infiniband Support Important? https://www.nimbix.net/why-is-infiniband-support-important/.

[49] R. Grant, K. Pedretti, and A. Gentile. Overtime: A tool for analyzing performance variation due to network interference. In *Proc. of the 3rd Workshop on Exascale MPI*, 2015.

[50] T. Groves, Y. Gu, and N. Wright. Understanding Performance Variability on the Aries Dragonfly Network. In *IEEE Int'l Conf. on Cluster Computing*, 2017.

[51] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. I know

what your packet did last hop: Using packet histories to troubleshoot networks. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, pages 71–85, 2014.

[52] N. Jain, A. Bhatele, X. Ni, N. J. Wright, and L. V. Kalè. Maximizing Throughput on A Dragonfly Network. In *Proc. Int'l Conference on High Performance Computing, Networking, Storage and Analysis*, 2014.

[53] Vimalkumar Jeyakumar, Mohammad Alizadeh, Yilong Geng, Changhoon Kim, and David Mazières. Millions of little minions: Using packets for low latency network programming and visibility. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 3–14. ACM, 2014.

[54] Saurabh Jha, Shengkun Cui, Tianyin Xu, Jeremy Enos, Mike Showerman, Mark Dalton, Zbigniew T Kalbarczyk, William T Kramer, and Ravishankar K Iyer. Live forensics for distributed storage systems. *arXiv preprint arXiv:1907.10203*, 2019.

[55] Saurabh Jha, Valerio Formicola, Catello Di Martino, Mark Dalton, William T Kramer, Zbigniew Kalbarczyk, and Ravishankar K Iyer. Resiliency of HPC Interconnects: A Case Study of Interconnect Failures and Recovery in Blue Waters. *IEEE Transactions on Dependable and Secure Computing*, 2017.

[56] Saurabh Jha, Archit Patke, Jim M. Brandt, Ann C. Gentile, Mike Showerman, Eric Roman, Zbigniew T. Kalbarczyk, William T. Kramer, and Ravishankar K. Iyer. A study of network congestion in two supercomputing high-speed interconnects. *CoRR*, abs/1907.05312, 2019.

[57] Saurabh Jha, Archit Patke, Mike Showerman, Jeremy Enos, Greg Bauer, Zbigniew Kalbarczyk, Ravishankar Iyer, and William Kramer. Monet - Blue Waters Network Dataset. https://bluewaters.ncsa.illinois.edu/monet-bw-net-data/, 2019.

[58] Matthew D Jones, Joseph P White, Martins Innus, Robert L DeLeon, Nikolay Simakov, Jeffrey T Palmer, Steven M Gallo, Thomas R Furlani, Michael Showerman, Robert Brunner, et al. Workload Analysis of Blue Waters. *arXiv preprint arXiv:1703.00924*, 2017.

[59] John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable Dragonfly topology. In *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*, pages 77–88. IEEE, 2008.

[60] William Kramer, Michelle Butler, Gregory Bauer, Kalyana Chadalavada, and Celso Mendes. Blue waters parallel i/o storage sub-system. *High Performance Parallel I/O*, pages 17–32, 2015.

[61] HT Kung, Trevor Blackwell, and Alan Chapman. Credit-based flow control for atm networks: credit update protocol, adaptive credit allocation and statistical multiplexing. In *ACM SIGCOMM Computer Communication Review*, volume 24, pages 101–114. ACM, 1994.

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

[63] T.-T.Y. Lin and D.P. Siewiorek. Error log analysis: statistical modeling and heuristic trend analysis. *Reliability, IEEE Transactions on*, 39(4):419–432, 1990.

[64] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114. ACM, 2016.

[65] Huong Luu, Marianne Winslett, William Gropp, Robert Ross, Philip Carns, Kevin Harms, Mr Prabhat, Suren Byna, and Yushu Yao. A multiplatform study of i/o behavior on petascale supercomputers. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 33–44. ACM, 2015.

[66] Catello Di Martino, Saurabh Jha, William Kramer, Zbigniew Kalbarczyk, and Ravishankar K Iyer. Logdiver: a tool for measuring resilience of extreme-scale systems and applications. In *Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale*, pages 11–18. ACM, 2015.

[67] Celso L. Mendes, Brett Bode, Gregory H. Bauer, Jeremy Enos, Cristina Beldica, and William T. Kramer. Deploying a large petascale system: The blue waters experience. *Procedia Computer Science*, 29(0):198 – 209, 2014. 2014 International Conference on Computational Science.

[68] Justin Meza, Tianyin Xu, Kaushik Veeraraghavan, and Onur Mutlu. A large scale study of data center network reliability. In *Proceedings of the Internet Measurement Conference 2018*, pages 393–407. ACM, 2018.

[69] Radhika Mittal, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, David Zats, et al. Timely: Rtt-based congestion control for the datacenter. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 537–550. ACM, 2015.

[70] Hiroyuki Miyazaki, Yoshihiro Kusano, Naoki Shinjou, Fumiyoshi Shoji, Mitsuo Yokokawa, and Tadashi Watanabe. Overview of the k computer system. *Fujitsu Sci. Tech. J*, 48(3):302–309, 2012.

[71] Misbah Mubarak, Philip Carns, Jonathan Jenkins, Jianping Kelvin Li, Nikhil Jain, Shane Snyder, Robert Ross, Christopher D Carothers, Abhinav Bhatele, and Kwan-Liu Ma. Quantifying I/O and communication traffic interference on dragonfly networks equipped with burst buffers. In *Cluster Computing, 2017 IEEE Int'l Conf. on*, pages 204–215. IEEE, 2017.

[72] Srinivas Narayana, Mina Tahmasbi, Jennifer Rexford, and David Walker. Compiling path queries. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, pages 207–222, 2016.

[73] Vikram Nathan, Srinivas Narayana, Anirudh Sivaraman, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. Demonstration of the marple system for network performance monitoring. In *Proceedings of the SIGCOMM Posters and Demos*, pages 57–59. ACM, 2017.

[74] National Center for Supercomputing Applications. SPP-2017 Benchmark Codes and Inputs. https://bluewaters.ncsa.illinois.edu/spp-benchmarks.

[75] Zhengbin Pang, Min Xie, Jun Zhang, Yi Zheng, Guibin Wang, Dezun Dong, and Guang Suo. The TH express high performance interconnect networks. *Frontiers of Computer Science*, 8(3):357–366, 2014.

[76] K. Pedretti, C. Vaughan, R. Barrett, K. Devine, and S. Hemmert. Using the Cray Gemini Performance Counters. In *Proc. Cray User's Group*, 2013.

[77] James C Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with NAMD. *Journal of computational chemistry*, 26(16):1781–1802, 2005.

[78] T Rabbani, F.A. van den Heuvel, and George Vosselman. Segmentation of point clouds using smoothness constraint. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36, 01 2006.

[79] SchedMD. Slurm scontrol. https://slurm.schedmd.com/scontrol.html.

[80] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. Mesh-tensorflow: Deep learning for supercomputers. In *Advances in Neural Information Processing Systems*, pages 10435–10444, 2018.

[81] David Skinner and W. Kramer. Understanding the causes of performance variability in hpc workloads. In *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*, pages 137–149, 2005.

[82] Hari Subramoni, Sreeram Potluri, Krishna Kandalla, B Barth, Jérôme Vienne, Jeff Keasler, Karen Tomko, K Schulz, Adam Moody, and Dhabaleswar K Panda. Design of a scalable InfiniBand topology service to enable network-topology-aware placement of processes. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 70:1–70:12, 2012.

[83] Yanhua Sun, Gengbin Zheng, Laximant V Kale, Terry R Jones, and Ryan Olson. A ugni-based asynchronous message-driven runtime system for cray supercomputers with gemini interconnect. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pages 751–762. IEEE, 2012.

[84] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. Simplifying datacenter network debugging with pathdump. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 233–248, 2016.

[85] Praveen Tammana, Rachit Agarwal, and Myungjin Lee. Distributed network monitoring and debugging with switchpointer. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, pages 453–456, 2018.

[86] Ozan Tuncer, Emre Ates, Yijia Zhang, Ata Turk, Jim Brandt, Vitus J Leung, Manuel Egele, and Ayse K Coskun. Diagnosing performance variations in hpc applications using machine learning. In *International Supercomputing Conference*, pages 355–373. Springer, 2017.

[87] Bin Yang, Xu Ji, Xiaosong Ma, Xiyang Wang, Tianyu Zhang, Xiupeng Zhu, Nosayba El-Sayed, Haidong Lan, Yibo Yang, Jidong Zhai, et al. End-to-end i/o monitoring on a leading supercomputer. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 379–394, 2019.

[88] Xu Yang, John Jenkins, Misbah Mubarak, Robert B Ross, and Zhiling Lan. Watch out for the bully! job interference study on dragonfly network. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 750–760. IEEE, 2016.

[89] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming

Zhang. Congestion control for large-scale rdma deployments. *ACM SIGCOMM Computer Communication Review*, 45(4):523–536, 2015.

[90] Yibo Zhu, Nanxi Kang, Jiaxin Cao, Albert Greenberg, Guohan Lu, Ratul Mahajan, Dave Maltz, Lihua Yuan, Ming Zhang, Ben Y Zhao, et al. Packet-level telemetry in large datacenter networks. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 479–491. ACM, 2015.

A HPC Interconnect Background

Here we briefly give an overview of HPC interconnects and dive deeper into the details of torus networks.

A.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. Summit [13]), (2) DragonFly (e.g., Edison [15]), and (3) Torus (e.g., Blue Waters [67]).

A.2 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., 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 other network topolgoies such as Fat-Tree or DragonFly. On the other hand, extra hop counts provide path diversity, which is required for building fault-tolerant architecture.

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 directional-order routing algorithm. Directional-order routing does the following:

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

B Workload Information

On Blue Waters, all jobs execute in non-shared mode, without any co-location with another job on the same compute node. Users can submit batch or interactive jobs using Moab/Torque [14] and configure several parameters for job resource request such as: (i) number of nodes, (ii) the number of cores, and (iii) the system walltime (i.e., requested clock time for the job). Blue Waters puts a 48-hour walltime restriction. Blue Waters uses Integrated System Console (ISC) [44] to parse and store the job records and its associated metrics (performance and failure) in its database.

In [39], Di Martino *et al.* provided detailed characterization of more than 5 million HPC application runs completed during the first 518 production days of Blue Waters. However, for completeness, this section provides the workload characteristics of the jobs running on Blue Waters during our study period. Due to loss of data caused by a failure, we do not have workload information for Jan 2017 and hence the workload data shown here is from Feb 2017 - July 2017. During our study period, 2,219k jobs were executed by 467 unique users. We characterize the job characteristics in terms of i) job type, and ii) job size.

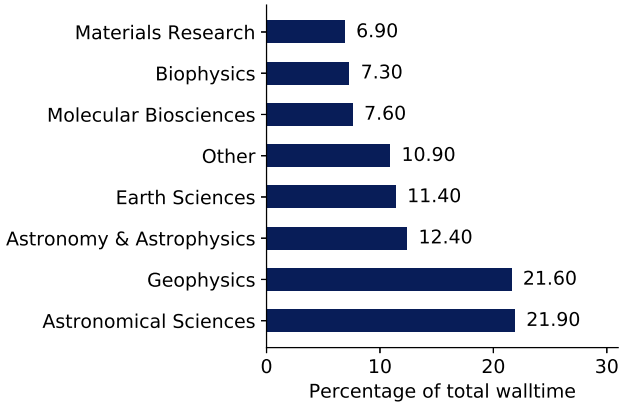
B.1 Job type

Blue Waters workload is predominantly composed of scientific applications. The most prolific scientific fields are summarized in Figure 12a in terms of node-hours. The top scientific discipline during our study period was 'Astronomical Sciences' (21.9%). However, the top scientific disciplines changes over time based on resource allocation awards given by US National Science Foundation and University of Illinois.

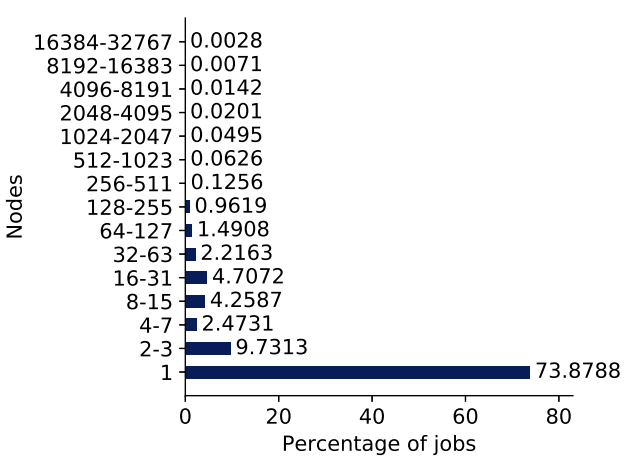
Definition 4 Node-seconds: *is the product of the number of nodes and the wallclock time (in seconds) used by a job. The metric captures the scale of the job's execution across space and time.*

B.2 Job Size

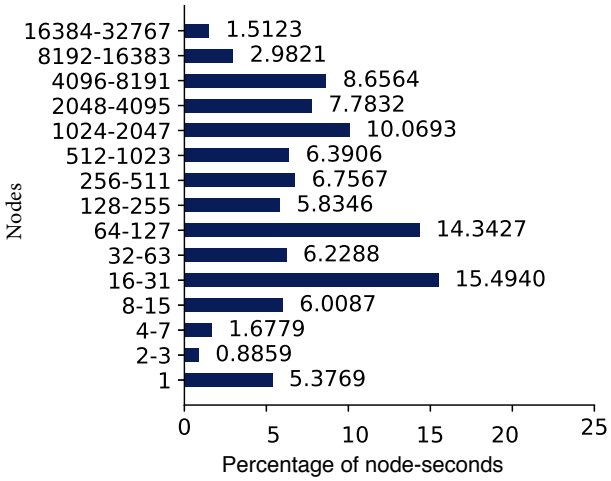
Figure 13b shows a bar plot summarizing relationship between percentage by node-seconds (see Def. 4) and percentage of jobs, whereas Fig. 13a shows a bar plot summarizing relationship between number of nodes and percentage of jobs. 74% of the jobs are single-node jobs. However, these jobs contribute *only* 5% by node-seconds. The large-scale jobs by number are small, they contribute to 94% of the total node-seconds.



(a) Breakdown of wallclock time of scientific application domains



(a) Percentage of jobs by node count



(b) Percentage of node-seconds

Figure 13: Characteristics of jobs running on Blue Waters.

C Existence of Congestion Hotspots and Regions in DragonFly Interconnect

Here we characterize hotspot links on Edison [15], a 2.57 petaflops production system, to showcase continued existence of network congestion problems on a current state of the art network interconnect. Edison uses Cray Aries interconnect which is based on DragonFly topology and uses adaptive routing [59]. We use one week of LDMS data that was collected from Edison at one second interval, and amounts to 7.7 TB. Our analyses shows 1) presence of long duration hotspot links, and 2) performance variability on a current state of the art network interconnect.

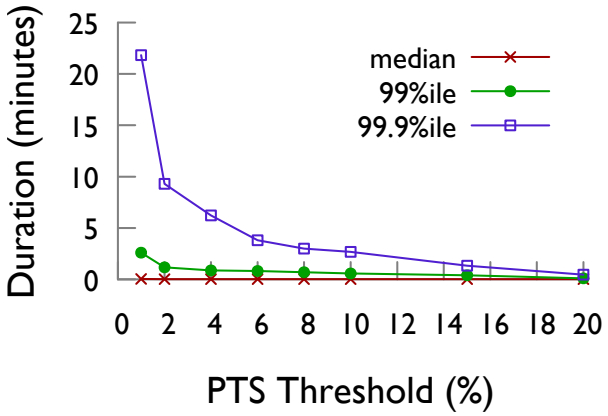


Figure 14: Distribution of hotspot link duration in Edison [replicated from [56]]

Figure 14 characterizes the median, 99%ile and 99.9%ile duration of the hotspot links by generating the distribution of the duration for which a link persists to be in congestion at $P_{Ts} \geq P_{Ts}$ Threshold. While the 99.9%ile hotspot duration is an order of magnitude lesser compared to the observed results in Gemini (see Figure. 3), which can be explained by the low diameter topology and use of congestion-aware routing policies in Aries. The duration of hotspot is longer than a minute for congestion thresholds less than 15% P_{Ts}. More details on characterization for Edison can be found in [56].

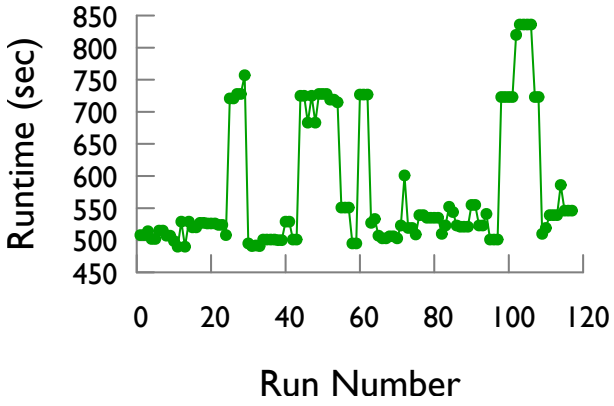


Figure 15: Variation of MILC runtime on Edison

Although the hotspot link duration has significantly decreased, the performance variation due to congestion continues to be a problem. For example, we observed significant performance variation of up to 1.67× compared to baseline for MILC application [25] on Edison (see Figure 15). MILC is a communication-heavy application susceptible to congestion on the interconnect. The reason for slowdown of the application can be attributed to presence of congestion in the links which rapidly evolves (i.e., a link may not be continuously congested but there is always groups of congested links). Preliminary analysis of network congestion counters

obtained from Edison [15] suggests existence of congestion regions that evolve rapidly. Our future work will focus on applying and extending our methodology to other interconnects technologies that use different topology other than torus (e.g., Fat-Tree or DragonFly) or use adaptive-routing (e.g., UGAL [59]). In emerging network technologies, vastly more network performance counters are available that can be used for detecting congestion and hence there is an increased need for algorithmic methods for discovering and assessing congestion at system and application-level.

D Validation of Congestion Regions Generated by Region Segmentation

Validation of the region segmentation algorithm was done by inspecting visualizations of both the unsegmented and segmented congestion data. We also generated synthetic congestion data and evaluated our algorithm's performance on it as a sanity check.

D.1 Results Analysis Discussion

As we do not have any ground truth for our clustering algorithm and the credit and inq stall on each link widely varies across the system with time (as discussed in previous subsection), we attempted some sanity checks in order to validate that the algorithm produced a sensible clustering of the data. To facilitate this, we implemented visualization tools for visualizing both the raw, unsegmented data, as well as the final, segmented data. We then ran our algorithm on the congestion data that was recorded at times when we knew there were congestion events (e.g. when Cray congestion protection events were triggered), as well as at multiple randomly sampled timepoints. We then visualized both sets of data and manually inspected the regions generated, checking visually to see if they lined up with the visualization of the raw congestion data. For samples that we inspected, the algorithm worked well for segmenting the data.

As a further test, we generated random congestion data following a simplified model and scored our algorithm's effectiveness at classifying those data. The data was created by randomly generating regions of congestion in a 24 x 24 x 24 cube representing the 3D torus of the Blue Waters Gemini interconnect. Each congestion region was created by randomly generating a) a cuboid in which each dimension was between 3 and 9 links inclusive, b) a random stall value s between 20% and 50%, and c) a random integer from $\{0, 1\}$. Depending on the value of the random integer, s was added to the credit-stall or inq-stall of all the links in the cuboid. Finally, after all regions were added random Gaussian noise with $(\mu, \sigma) = (0, 2.5)$ was added to both the credit- and inq-stalls of all the links in the cube to simulate small variances in the stall values of each link.

We then ran our algorithm on 100 samples, each with a random number of regions (from 1 to 8 inclusive), and assigned a score as well as calculating the precision and recall for that sample. We scored the match as follows: for a single sample, let $A_i, i = 1 \cdots n$ be the actual regions and $B_i, i = 1 \cdots m$ be the regions the algorithm produced. A single sample was then assigned the score $(\frac{1}{n}\sum_{i=1}^{n}\frac{|A_i \cap B_{j_i}|}{|A_i \cup B_{j_i}|})(\frac{n}{\max(n,m)})$, where $|A_i|$ represents the number of links in the enclosed region, and the B_{j_i} are the regions that "best" overlap their respective A_i. The B_{j_i} are chosen by going through the regions A_i in order from smallest to largest, and choosing j_i such that B_{j_i} has the largest possible overlap with A_i. Regions created by segmentation that have both inq- and credit-stall $< 5\%$ are considered insignificant and were excluded from this processing.

This scoring assigns a score of 1 to a perfect match, which degrades to 0 when a) the mismatch between the true and the matching generated region increases, or when b) the algorithm generates more regions than true regions.

Based on that scoring, the algorithm achieved an average score of 0.81 (maximum 1.0) with parameters $(\theta_p = 12, \theta_r = 8, \theta_d = 2, \theta_s = 20)$ over 100 samples, with an average precision of 0.87 and an average recall of 0.89.

D.2 Summary

While it is not possible to fully validate the efficacy of our segmentation algorithm, the synthetic datasets generated give us a degree of confidence that the algorithm does the right thing on simple models of congestion that satisfy our assumptions (i.e. that congestion tends to spread locally) and work well with noise. The comparison between the datasets before and after the segmentation suggests that algorithm does still work reasonably well in practice, on real datasets.

The region segmentation algorithm was applied to 5 months of production data collected as part of the operational environment of a system. The data was obtained from Blue Waters through various tools and counters(e.g., network performance counters, link-aggregated data, scheduler and log files) and hence is reliant on the correctness. The reliability of the system software for synchronized data collection at regular intervals (LDMS) is dependent on the system operation staff which supports it and performance details for this software are available in its documention.

Acknowledgment

We thank Jeremy Enos and Brett Bode (NCSA), Eric Roman (NERSC), anonymous reviewers of NSDI 2019 and our shepherd Lalith Suresh for insightful conversations and/or feedback. We thank K. Atchley and J. Applequist for their help in preparing the manuscript .

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award Number 2015-02674. This work is partially supported by NSF CNS 15-13051, and a Sandia National Lab contract number 1951381.

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 Applications (NCSA).

Sandia National Laboratories (SNL) is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.