Using Machine Learning Method for Probing Congestion in High-Performance Datacenter Network

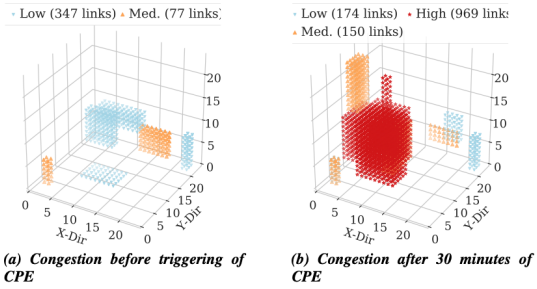
Sidharth Rupani, Fadel Adib
sidrup@alum.mit.edu

Abstract

Even if, as we all known, High Performance Computing (HPC) systems can greatly reduce application performance, there is little quantification of congestion on credit-based interconnected networks. This paper proposed a method for detecting, extracting and characterizing congested regions in the network. The authors have implemented this methodology in a deployable tool, Monet, which can provide such analysis and feedback at runtime. Using Monet, we can characterize and diagnose the congestion in Blue Waters, the world's largest 3D torus network. Blue Waters is a 13.3- petaflop supercomputer at the National Center for Supercomputing Applications.

Introduction

As we all know that network congestion in High Performance Computing (HPC) system can greatly degrade performance of applications. But there are few ways to quantify such congestion on credit-base interconnect networks. In this paper, we present a way to detect, extract and characterize congestion regions in networks. We use Monet, a tool that is able to help us to characterize and diagnose congestion in Blue Waters, a petaflop supercomputer at the National Center for Supercomputing Applications. There are two major contributions in this paper. First of all, this paper proposes a methodology for quantitative characterization of congestion in high-speed networks. Secondly, this paper introduced Monet that could employs such method. On Blue Waters, we will use gpcdr (**?**) and Lightweight Distributed Metric Service (LDMS) to monitor framework. These tools can help us to collect data on the network, file system traffic and appliations. We will use a network hotspot extraction and characterization tool that will extract Congestion Regions (CRs) at runtime. Also, a diagnosis tool which will determine the reason of the congestion will be used. Common causes include link failures or excessive file system traffic from applications. The utility of CRs could be shown by helping trigger congestion mitigation responses. This methodology is important because it is very common that HSN are vulnerable to congestion. As shown in Figure 1, CMEs(congestion mitigation events) were triggered 261 times and 29.8% of which did not reduce congestion in the system. Figure 1 shows a situation where the scale and severity of CRs increases after some throttling events.



(a) Congestion before triggering of CPE

(b) Congestion after 30 minutes of CPE

Figure 1: Examples to show the visualization of congestion regions.

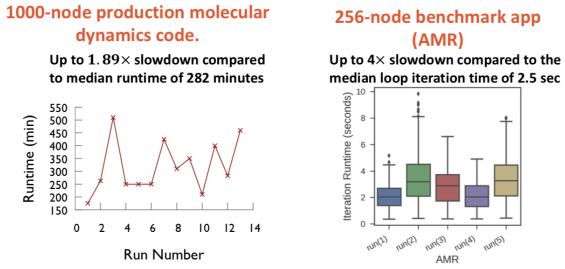


Figure 2: Examples to show high-speed networks (HSN) are susceptible to high congestion.

Our contribution is that we adopted unsupervised algorithm to run on AWS ubuntu system. Due to computing power limit, we only extract data in about 1 month frame. Then we experiments (CR size vs CR and CR duration vs CR) and have our own experiments results and analysis. Our results are following the same pattern but CR size and CR duration are much smaller.

Related Work

High-performance datacenters and clouds (HPC) (Shazeer et al. 2018) are widely used in high-speed interconnect networks, which used credit-based control algorithm. This advantages of this technique is to support low-latency communications, including low per-hop latency, low tail-latency variation and high bisection bandwidth. Significant performance variation has been observed in production systems running real-world workloads because of the network congestion. However, there has been only a little quantification

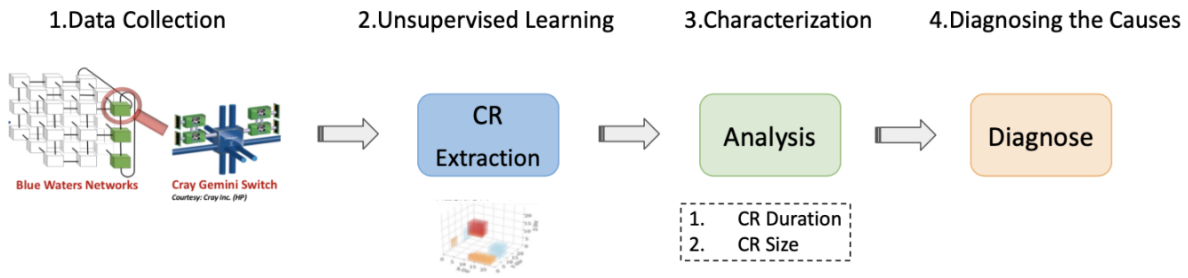


Figure 3: Characterization and diagnosis workflow for interconnection-networks.

Column Name	Column Description
#Time	*UNIX Epoch time* in seconds
***dir*_SAMPLE_GEMINI_LINK_CREDIT_STALL (% x1e6)**	*Credit Stall* which can be defined as the stall time associated with sending flits between different switches.
***dir*_SAMPLE_GEMINI_LINK_INQ_STALL (% x1e6)**	*Inq Stall* which can be defined as the stall time associated with sending flits within the same switch
***dir*_SAMPLE_GEMINI_LINK_USED_BW (% x1e6)**	*Bandwidth Utilization* which is the percentage of maximum link bandwidth used during the measurement interval
nettopo_mesh_coord_Z	*Z coordinate* in the torus topology
nettopo_mesh_coord_Y	*Y coordinate* in the torus topology
nettopo_mesh_coord_X	*X coordinate* in the torus topology

dir can be one of: Z+, Z-, Y+, Y-, X+, X-
Note: To get a percentage value (0-100) for stall/bandwidth divide the respective columns by 1000000.

Figure 4: Colume name and description of the dataset.

of congestion on such interconnect networks.

Previous works more focused on I/O profiling and performance anomaly diagnosis, including Darshan(**?**), Beacon(Yang et al. 2019) and Kaleidoscope(Jha et al. 2019). In addition, these works assumed steady state utilization/congestion behavior and thus do not address full production workloads. Different from these works, this paper focuses on assessing network congestion in credit-flow based interconnection networks. There are three main questions to solve in this paper. First, how often system/applications are experiencing congestion? Second, what are the culprits behind congestion? Third, how to avoid and mitigate effects of congestion. To address the congestion problem, some of the previous works (Deveci et al. 2014; Jha et al. 2019) employs indirect measures like messaging rates, network counters from switch. Other methods like (Agelastos et al. 2014; Bhatele et al. 2015) use the global network counter data.

Recently, some approaches are tuned for TCP/IP networks and this paper is complementary to these efforts. For example, ExpressPass(Cho, Jang, and Han 2017), DCQCN(Zhu et al. 2015a) and TIMELY(Mittal et al. 2015) focus on preventing and mitigating congestion. Other works like PathDump(Tammana, Agarwal, and Lee 2016; Liu et al. 2023; Liu, Tan, and Tensmeyer 2023), SwitchPointer(Tammana, Agarwal, and Lee 2018; Liu et al. 2020a,b) and EverFlow(Zhu et al. 2015b; Li et al. 2023) pay more attention to network monitoring.

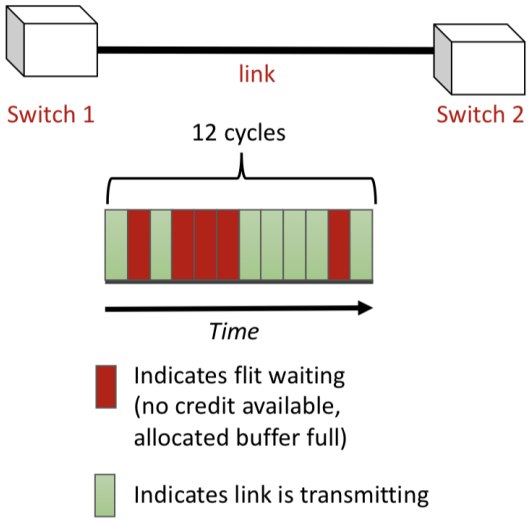


Figure 5: Mechanism behind the Percent Time Stalled.

Proposed Method

Comparing with Darshan (Yang et al. 2019; Liu, Yacoob, and Shrivastava 2023), Beacon (Yang et al. 2019) and Kaleidoscope (Jha et al. 2019), this paper used credit-based network to assess and reduce workload instead of I/O profiling and performance anomaly diagnosis. Then comparing to (Deveci et al. 2014; Liu et al. 2020a; Agelastos et al. 2014; Liu et al. 2023; Liu, Tan, and Tensmeyer 2023), this paper is the first one that is the 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. Another advantage is that this paper employs direct measurement to address the congestion instead of presenting only representative examples of congestion through time or executed a single application on the system. Comparing with (Yang et al. 2019; Fu et al. 2016), this paper involves the generation and characterization of congestion regions, diagnosis of congestion causes. In the following parts, we will first introduce the workflow of our algorithm. After that, we will describe the way we used to measure the congestion or severity level. Then we will talk a little bit reasons why we choose the congestion algorithm. Finally, we will discuss the

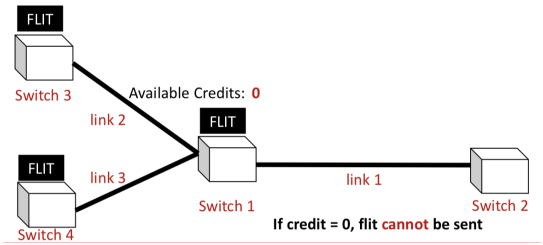


Figure 6: Credit-based flow control network.

$$P_{TS} = 100 * T^i_s / T^i$$

T^i: network cycles in i^{th} measurement interval.

T^i_s: total cycles the link was stalled in T^i.

Figure 7: How to calculate Percent Time Stalled (P_{TS}).

detailed segmentation algorithm.

Workflow

The workflow of our model is described in Figure 3. After the step: system description and data collection, we will go to the main part of the our implementation, which is the core task for us. The initial step for us is to use unsupervised algorithm to extract Congestion Region (CR). In details, we use the segmentation algorithm to segment the network into groups of links with similar congestion values. After that, we will implement the characterization operation to classify the congestion categories. Finally, we will use cases to check Congestion response and try to find mitigation methods. At the same time, Finding diagnosing causes of congestion is also our target. The first step of our workflow is the introduction of the dataset. The dataset here we used is the Blue Water Network, which is a 3-D topology instead of the 2-D topology graph. After getting the data, then unsupervised learning algorithm will be used to extract the congestion region. The third step is the analyze the results from the congestion regions. The congestion size and congestion duration are the main two aspects we will use to measure the congestion. Finally, we will also try to find the root causes behind the congestion. Our target mainly includes the following three questions: How often system/applications are experiencing congestion? What are the culprits behind congestion? How to avoid and mitigate effects of congestion?

Percent Time Stalled

In the previous sections, we talked a lot about the congestion definitions, effects and causes, but how to measure the congestion level or severity is still not mentioned. In the section, we will specifically talk about the measurement method: Percent Time Stalled (P_{TS}).

Figure 5 describes the behind mechanism of how congestion happens. In the figure, the red section means the flit waiting (no credit available allocated buffer full) since our data spreading mechanism depends on credit-based flow control network. If credit is 0, no flit will be sent, which is described in Figure 6. The insight of Figure 6 is congestion spreads locally (i.e., fans out from an origin point to other senders). The ways to calculate Percent Time Stalled (P_{TS}) is described in Figure 7. T^i means the network cycles in i^th measurement interval meanwhile the T^i_s means the total cycles the link was stalled in T^i. In Figure 5, the value of T^i_s is 5 and the value of T^i is 12. In this case, the Percent Time Stalled value is 0.42.

Apart from the Percent Time Stalled, we also assign 4 congestion or severity level in order to measure, understand and compare the extent of congestion. Specifically, we assign 0-5 percent average P_TS in a CR as Negligible or 'Neg', 5-15 percent as 'Low', 15-25 percent as 'Medium', and ¿25 percent as 'High'. We will then use these levels in characterizations in the following parts 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.

Why Congestion Regions?

In this section, we will describe the reasons why it's necessary to used congestion regions to analyze the severity level.

There are mainly two reasons. First, characterizing hotspot links individually do not reveal regions of congestion. This is simply because the hotspot links are useful to understand the severity level at the link level. However, they are possible to hide the spatial features of congestion including the existence of multiple pockets of congestion and their spread and growth overtime. If we lack this kind of information, it will make it difficult to understand congestion characteristics and their root cause in the future.

Second reason is that CRs captures relationship between congestion-level and application slowdown efficiently. As shown in Figure 8b, execution time increases with increasing maximum of average P_{TS} over all regions by using congestion regions. Without using congestion regions, the correlation value is only 0.33 in Figure 8a, which is much less than the correlation value 0.89 in Figure 8b.This is a motivating factor for the extraction of such congestion regions as indicators of 'hot-spots' in the network.

Segmentation Algorithm

Then the details of the cluster is described below. Before that, we have some parameters: network graph, congestion measures value, neighbor distance metric and the stall similarity. This algorithm of segmentation includes four phrases in sequence. First of all, we first group the close links which have the very similar stall values within the threshold. Second, the nearby regions with similar average stall values, are grouped together through repetition of the previous step. Finally the congestion regions that are below the size threshold are merged into the nearest region within the distance threshold. We also have the following assumptions: the congestion runs locally and the link stall values are stable in the same congestion regions.

The detailed steps are includes 4 steps. First, the close links will be grouped or classified into the same categories if they meet the following requirements. For example, there are two links x and y. If the distance between the two vertices are below the threshold δ meanwhile the stall values

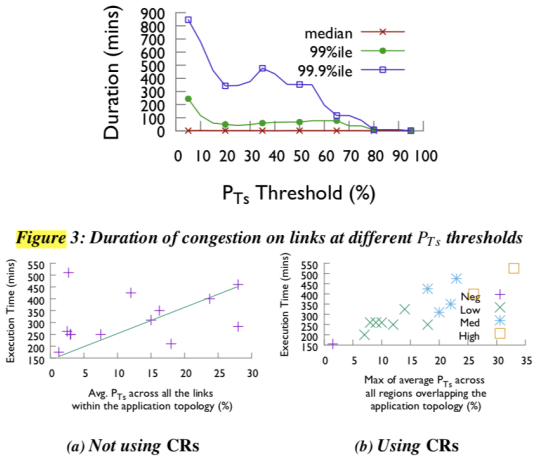


Figure 8: Why choose congestion regions?

between the two are below the threshold θ_p, they can be put into the same class. Then as for the second step, if the close regions have the similar stalled values and they are close to each other. The stalled values here we used are the average values between the stalled value of the links in this region. Then if the congestion regions that are below the size threshold and the distance is also within the threshold, they will be merged into one group. Then finally, if the left small congestion regions will be discarded since they are too small to analyze.

Dataset

We downloaded and analyzed the dataset, the Blue Waters Network Dataset, which contains collected and aggregated network information from NCSA's Blue Water system, which is comprised of 27648 nodes connected via Cray Germini 3D torus (dimension 24x24x24) interconnect, from Jan/01/2017 to May/31/2017. All LDMS/OVIS data is stored in Comma Separated Values (CSV) format. Each file name follows the format YYYYMMDD where YYYY, MM and DD is the year, month and date when the data was collected. The column name and description shows in the figure 4. Since the size of the source dataset is too large 140G, we only include the sample data in the final submission. The location of the original dataset is this **link**.

Experiment

Evaluation

As described in the paper (Fu et al. 2016), there are three steps for diagnosis. First of all, Mining candidate congestion-causing factors. For each CR, identified at time T, we create two tables A and F. Each row in table A corresponds to an application that is within Nhops ¡ 3 hops away from the bounding box of the congestion region CR. Table A has characteristics across 7 traffic features such as application name, maximum read bytes per minute and so on. Each row in table F corresponds to an application that is within Nhops ¡= 3 away from the congestion boundary of CR. For table F, it contains information about failures events across 3 failure features such as failure timestamp, failure location and failure type. Secondly, identifying anomalous or extreme factors. Next, we will identify extreme application traffic characteristics or network-related failures over the past 30-mins. The last step is to generate evidence in order to decide whether anomalous factors are really responsible for the congestion observed in the network.

Implementation

We are implementing a hotspot extraction and characterization tool which will enable us to extract and visualize Congestion Regions at run time. The tool will be realizing the unsupervised region-growth clustering algorithm as mentioned in the paper (Jha et al. 2019). This clustering approach requires us to have specification of congestion metrics such as percent time stalled and stall-to-flit-ratios. Also, a network topology graph is needed for us to extract regions of congestion which can be used for network characterization if time permits. Moreover, we will find a congestion vizualization tool to display the congestion regions. So far, we decided to build our tool based on the region growth segmentation algorithm in open-source PointCloud Library(PCL). However, there were some challenges. For example, we tried to ran PCL on several operation systems but they all encounter different kinds of errors such as the index-pack error shown in Figure 11 when running on Mac OS and the c++ compiler error shown in Figure 12 when running on Centos system. Finally, we found we successfully installed PCL docker and implement our tool on an AWS EC2 instance with Ubuntu 18.04 operation system.

Results

Due to computing power and time limit, we are not able to simulate the whole Blue Water network. Instead, we extracted part of the data collected within only one month. We re-experimented two experiments in the paper(Jha et al. 2020). The first one is recording the relationship between size of congestion region and the number of congested links. The second one is recording the relationship between congestion duration and the number of congestion regions. According to the paper, we use run time as estimate for different kinds of severity levels because they can be easily calculated and are handy for us to compare. In our experiments, we assign 0-5 percent of 400 min as 'Low', 15-18 percent as 'Medium' and 'High' when the run time exceed 25 percent of 400 min. As shown in figure 13, our results are showing that our average duration for Low is about 300 mins. The average for Med is about 400 mins and the average for high is about 150 mins. Also, as shown in figure 15, compared to the results from authors, our results are showing the same pattern which is that the longer CR duration the more congested regions can be detected and this pattern is the same for all severity. We also noticed that low severity happened the most frequent as more than 50,000 linked can be detected in just 100 mins. However, the low severity line also drop very rapidly as time go on. Compared to low severity, medium severity is more stable as time go on as the range of the largest number of congested regions detected and the least of congested regions detected is about

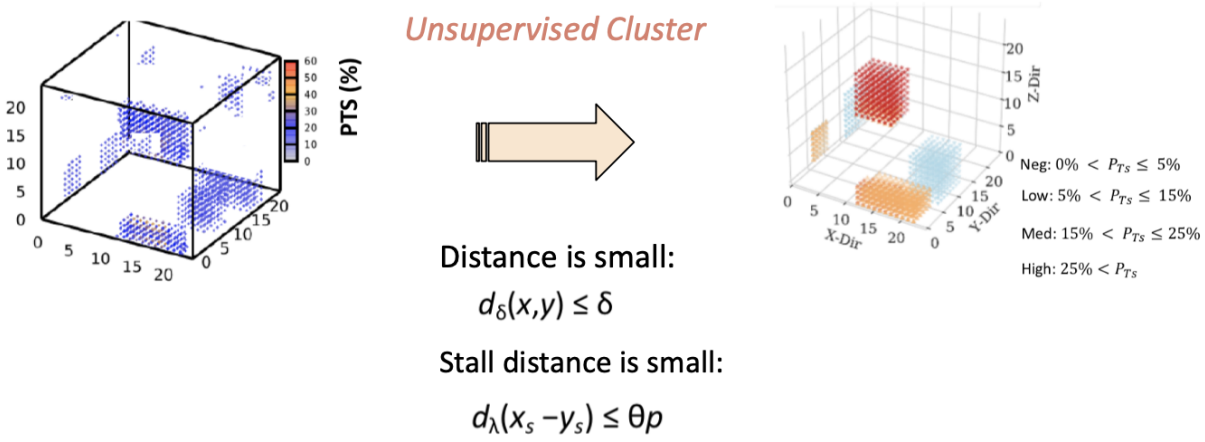


Figure 9: Segmentation algorithm behind congestion regions, where x,y are vertices in the graph, δ, θ_p are the thresholds of the distance and stall values

Figure 10: screenshot when compiling on Mac OS

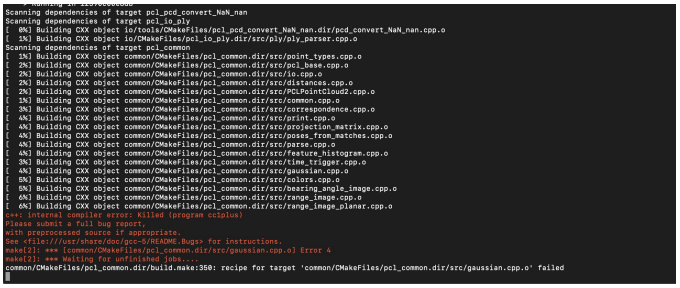


Figure 11: screenshot when compiling on Centos OS

only 17,000. However, we also found something in our results very different from those of authors. As shown in figure 15, all Low, Med and High severity are very fast within first 100 mins. But ours does not show such drastic decline. We believe this situation is caused by small size of data and shorter of running time. For another experiment about CR size, we also have similar pattern with the results from the authors as shown in figure 16. Still, most of the congested links are spreading in low severity CRs and high severity has the least. We can also see the figure 17 that all severity lines are decline as congestion size increase, which means there are not many large size CRs. Low severity CR is very stable as its number of congested regions do not decrease much as CR size increases. However, both medium and low severity are decreasing dramatically along with increasing CR size. we also noticed that low and medium severity lines are basically overlapping under the size of 350 and the reason could be high and medium severity congestion both happen with

CR Duration	100 mins	200 mins	400 mins	800 mins	1000 mins
Low	50391	39427	37963	29627	27038
Med	37642	30048	27691	24041	20074
High	21350	12401	10047	6387	7421

Figure 12: CR sizes in terms of the number of links for each congested state from our results

the same frequency under this size. Another point worthy mentioning is that after separating at size of 350, the two lines end up with similar congestion region numbers, which proves that high and medium severity congestion region are very similar to each other in a certain size. Another thing worth to be noted is that compared to the results from the authors, we found our results do not show 0 congested region situation. But we can see that the number of CRs drop to 0 for all severity when CR size reaches 2048 links. We believe the reason is due to our limited CR size in our network.

Diagnosis

The authors showed three congestion scenarios: (a) system issues (e.g. changes in system load), (b) network-component failures (e.g. link failures), and (c) intra-application contention.

(a) 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. (b) Network-related failures are frequent and may lead to network congestion, depending on the traffic on the network and the type of failure. Failures of directional links or

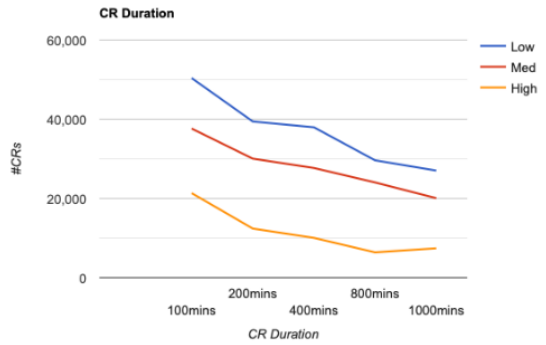
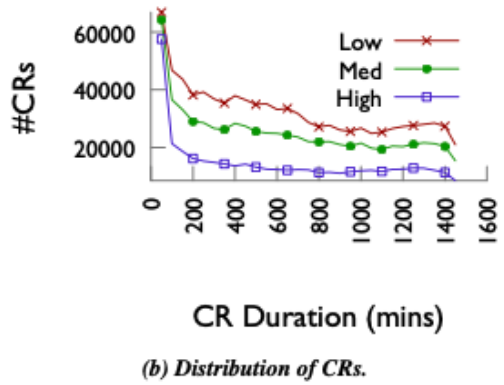


Figure 13: CR sizes in terms of the number of links for each congested state from our results



(b) Distribution of CRs.

Figure 14: CR sizes in terms of the number of links for each congested state from authors' results

routers generally lead to 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. (c) Congestion within an application's geometry (intra-application contention) can occur even with TAS. 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. 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.

How do we divide these congestion causes in order to help draw a system manager's attention to anomalous scenarios and potential offenders for further analysis? The authors presented a methodology which combine system information

CR Size	300	350	400	450	500
Low	925	863	778	722	673
Med	856	525	415	322	235
High	837	523	469	365	254

Figure 15: CR sizes in terms of the number of links for each congested state from our results

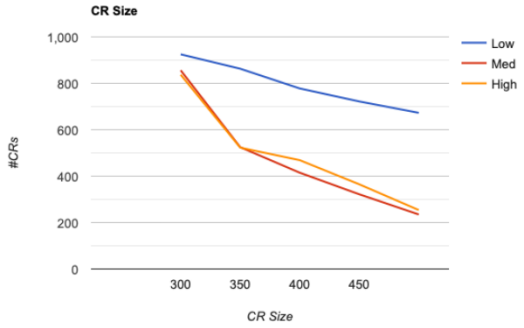


Figure 16: CR sizes in terms of the number of links for each congested state from our results

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 of anomalies in the data.The first step is mining candidate congestion-causing factors. The second step is identifying extreme application traffic characteristics or network-related failures over the past 30 minutes that have led to the occurrence of CRs.The last step is generating 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).

Conclusion and Future Work

In this paper, the authors presented methodologies for detecting, characterizing, and diagnosing network congestion. It implemented these capabilities and demonstrated them using production data from NCSA's 27,648 node, Cray Gemini based, Blue Waters system. While they utilized the scale and data availability of the Blue Waters system to validate the approach, the methodologies presented are generally applicable to other credit-based k-dimensional meshes or toroidal

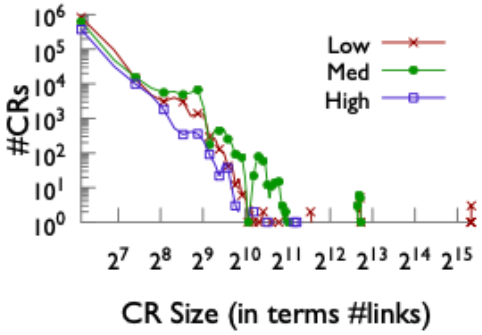


Figure 17: CR sizes in terms of the number of links for each congested state from authors' results

networks.

The future work will involve extending the presented techniques to other network technologies and topology. According to our process now, we have the following future plans. First, we will solve the required package problem by working with the original authors together. Second, we will experiment the model on the dataset with different time periods since the total amount of the data is too large. Third, we will write the code to achieve the visualization of the characterizations of the congestion regions. Finally, we plan to automate the congestion region characterization, diagnosis and visualization since the current version of Monet is not totally automated.

References

Agelastos, A.; Allan, B.; Brandt, J.; Cassella, P.; Enos, J.; Fullop, J.; Gentile, A.; Monk, S.; Naksinehaboon, N.; Ogden, J.; et al. 2014. The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 154–165. IEEE.

Bhatele, A.; Titus, A. R.; Thiagarajan, J. J.; Jain, N.; Gamblin, T.; Bremer, P.-T.; Schulz, M.; and Kale, L. V. 2015. Identifying the culprits behind network congestion. In *2015 IEEE International Parallel and Distributed Processing Symposium*, 113–122. IEEE.

Cho, I.; Jang, K.; and Han, D. 2017. Credit-scheduled delay-bounded congestion control for datacenters. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 239–252.

Deveci, M.; Rajamanickam, S.; Leung, V. J.; Pedretti, K.; Olivier, S. L.; Bunde, D. P.; Catalyürek, U. V.; and Devine, K. 2014. Exploiting geometric partitioning in task mapping for parallel computers. In *2014 IEEE 28th international parallel and distributed processing symposium*, 27–36. IEEE.

Fu, H.; Liao, J.; Yang, J.; Wang, L.; Song, Z.; Huang, X.; Yang, C.; Xue, W.; Liu, F.; Qiao, F.; et al. 2016. The Sunway TaihuLight supercomputer: system and applications. *Science China Information Sciences* 59(7): 1–16.

Jha, S.; Cui, S.; Xu, T.; Enos, J.; Showerman, M.; Dalton, M.; Kalbarczyk, Z. T.; Kramer, W. T.; and Iyer, R. K. 2019. Live Forensics for Distributed Storage Systems. *arXiv preprint arXiv:1907.10203* .

Jha, S.; Patke, A.; Brandt, J.; Gentile, A.; Lim, B.; Showerman, M.; Bauer, G.; Kaplan, L.; Kalbarczyk, Z.; Kramer, W.; and Iyer, R. 2020. Measuring Congestion in High-Performance Datacenter Interconnects. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 37–57. Santa Clara, CA: USENIX Association. ISBN 978-1-939133-13-7. URL https://www.usenix.org/conference/nsdi20/presentation/jha.

Li, Z.; Xu, P.; Liu, F.; and Song, H. 2023. Towards Understanding In-Context Learning with Contrastive Demonstrations and Saliency Maps. *arXiv preprint arXiv:2307.05052* .

Liu, F.; Lin, K.; Li, L.; Wang, J.; Yacoob, Y.; and Wang, L. 2023. Aligning Large Multi-Modal Model with Robust Instruction Tuning. *arXiv preprint arXiv:2306.14565* .

Liu, F.; Tan, H.; and Tensmeyer, C. 2023. DocumentCLIP: Linking Figures and Main Body Text in Reflowed Documents. *arXiv preprint arXiv:2306.06306* .

Liu, F.; Wang, Y.; Wang, T.; and Ordonez, V. 2020a. Visual news: Benchmark and challenges in news image captioning. *arXiv preprint arXiv:2010.03743* .

Liu, F.; Wang, Y.; Wang, T.; and Ordonez, V. 2020b. Visualnews: Benchmark and challenges in entity-aware image captioning. *arXiv preprint arXiv:2010.03743* .

Liu, F.; Yacoob, Y.; and Shrivastava, A. 2023. COVID-VTS: Fact Extraction and Verification on Short Video Platforms. *arXiv preprint arXiv:2302.07919* .

Mittal, R.; Lam, V. T.; Dukkipati, N.; Blem, E.; Wassel, H.; Ghobadi, M.; Vahdat, A.; Wang, Y.; Wetherall, D.; and Zats, D. 2015. TIMELY: RTT-based congestion control for the datacenter. *ACM SIGCOMM Computer Communication Review* 45(4): 537–550.

Shazeer, N.; Cheng, Y.; Parmar, N.; Tran, D.; Vaswani, A.; Koanantakool, P.; Hawkins, P.; Lee, H.; Hong, M.; Young, C.; et al. 2018. Mesh-tensorflow: Deep learning for supercomputers. *arXiv preprint arXiv:1811.02084* .

Tammana, P.; Agarwal, R.; and Lee, M. 2016. Simplifying datacenter network debugging with pathdump. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 233–248.

Tammana, P.; Agarwal, R.; and Lee, M. 2018. Distributed network monitoring and debugging with switchpointer. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 453–456.

Yang, B.; Ji, X.; Ma, X.; Wang, X.; Zhang, T.; Zhu, X.; El-Sayed, N.; Lan, H.; Yang, Y.; Zhai, J.; et al. 2019. End-to-end I/O monitoring on a leading supercomputer. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, 379–394.

Zhu, Y.; Eran, H.; Firestone, D.; Guo, C.; Lipshteyn, M.; Liron, Y.; Padhye, J.; Raindel, S.; Yahia, M. H.; and Zhang,

M. 2015a. Congestion control for large-scale RDMA deployments. *ACM SIGCOMM Computer Communication Review* 45(4): 523–536.

Zhu, Y.; Kang, N.; Cao, J.; Greenberg, A.; Lu, G.; Mahajan, R.; Maltz, D.; Yuan, L.; Zhang, M.; Zhao, B. Y.; et al. 2015b. Packet-level telemetry in large datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 479–491.