Quantifying the impact of network congestion on application performance and network metrics

Yijia Zhang*, Taylor Groves†, Brandon Cook†, Nicholas J. Wright† and Ayse K. Coskun*
* Boston University, Boston, MA, USA; E-mail: {zhangyj, acoskun}@bu.edu
† National Energy Research Scientific Computing Center, Berkeley, CA, USA; E-mail: {tgroves, bgcook, njwright}@lbl.gov

***Abstract*—In modern high-performance computing (HPC) systems, network congestion is an important factor that contributes to performance degradation. However, how network congestion impacts application performance is not fully understood. As Aries network, a recent HPC network architecture featuring a dragonfly topology, is equipped with network counters measuring packet transmission statistics on each router, these network metrics can potentially be utilized to understand network performance. In this work, by experiments on a large HPC system, we quantify the impact of network congestion on various applications' performance in terms of execution time, and we correlate application performance with network metrics. Our results demonstrate diverse impacts of network congestion: while applications with intensive MPI operations (such as HACC and MILC) suffer from more than 40% extension in their execution times under network congestion, applications with less intensive MPI operations (such as Graph500 and HPCG) are mostly not affected. We also demonstrate that a stall-to-flit ratio metric derived from Aries network counters is positively correlated with performance degradation and, thus, this metric can serve as an indicator of network congestion in HPC systems.**

***Index Terms*—HPC, network congestion, network counters**

I. Introduction

High-performance computing (HPC) systems play an important role in accelerating scientific research in various realms. However, applications running on HPC systems frequently suffer from performance degradation [1]. Network congestion is a major cause of performance degradation in HPC systems [2]–[4], leading to extention on job execution time 6X longer than the optimal [5]. Although performance degradation caused by congestion has been commonly observed, it is not well understood how that impact differs from application to application. Which network metrics could indicate network congestion and performance degradation is also unclear. Understanding the behavior of network metrics and application performance under network congestion on large HPC systems will be helpful in developing strategies to reduce congestion and improve the performance of HPC systems.

In this paper, we conduct experiments on a large HPC system called Cori, which is a 12k-node Cray XC40 system. We run a diverse set of applications while running network congestors simultaneously on nearby nodes. We collect application performance as well as Aries network counter metrics. Our results demonstrate substantial difference in the impact of network congestion on application performance. We also demonstrate that certain Aries network metrics are good indicators of network congestion.

The contributions of this work are listed as follow:

- In a dragonfly-network system, we quantify the impact of network congestion on the performance of various applications. We find that while applications with intensive MPI operations suffer from more than 40% extension in their execution times under network congestion, the applications with less intensive MPI operations are negligibly affected.
- We find that applications are more impacted by congestor on nearby nodes with shared routers, and are less impacted by congestor on nodes without shared routers. This suggests that a compact job allocation strategy is better than a non-compact strategy because sharing routers among different jobs is more common in a non-compact allocation strategy.
- We show that a *stall-to-flit ratio* metric derived from Aries network tiles counters is positively correlated with performance degradation and indicative of network congestion.

II. Aries network counters and metrics

In this section, we first provide background on the Aries network router. Then, we introduce our network metrics derived from Aries counters. The value of these metrics in revealing network congestion is evaluated in Section IV.

A. Aries network router

Aries is one of the latest HPC network architectures [6]. Aries network features a dragonfly topology [7], where multiple routers are connected by row/column links to form a virtual high-radix router (called a "group"), and different groups are connected by optical links in an all-to-all manner, giving the network a low-diameter property, where the shortest path between any two nodes is only a few hops away.

Figure 1 shows the 48 tiles of an Aries router in a Cray XC40 system. The blue tiles include the optical links connecting different groups; the green and grey tiles include the electrical links connecting routers within a group; and the yellow tiles include links to the four nodes connected to this router. In the following, we call the 8 yellow tiles as processor tiles (**ptiles**); and we call the other 40 as network tiles (**ntiles**).

B. Network metrics

In each router, Aries hardware counters collect various types of network transmission statistics [8], including the number of

978-1-7281-6677-3/20/$31.00 ©2020 IEEE

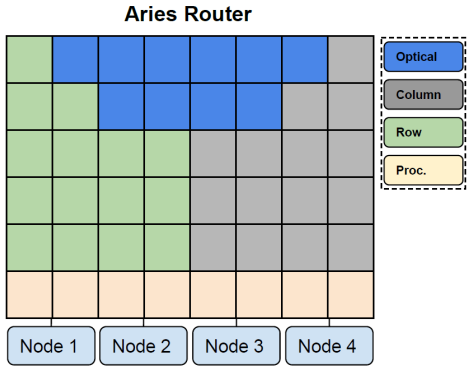


Fig. 1: Aries router architecture in a dragonfly network.

flits/packets travelling on links and the number of stalls that represent wasted cycles due to network congestion.

In this work, we use a *stall-to-flit ratio* metric derived from ntile counters. As the number of network stalls represents the number of wasted cycles in transmitting flits from one router to the buffer of another router, we expect the stall/flit ratio to be an indicator of network congestion. For each router, we define the ntile stall/flit ratio as:

$$\text{Ntile Stall/Flit Ratio} = \text{Avg}_{r\in 0..4, c\in 0..7} \frac{\text{N\_STALL\_}r\text{\_}c}{\sum_{v\in 0..7} \text{N\_FLIT\_}r\text{\_}c\text{\_}v}$$

Here, N_FLIT_r_c_v is the number of incoming flits per second to the v-th virtual channel of the r-th row, c-th column network tile. N_STALL_r_c is the number of stalls per second in all virtual channels on that ntile. As the stalls and flits collected from a specific ntile cannot be attributed to a certain node, we take an average over all the 40 ntiles (represented as "Avg") and use it as the ntile stall/flit ratio of the router. Because the 40 ntiles are the first 5 rows and all 8 columns in Fig. 1, the metric takes the average for $r \in 0..4$, and $c \in 0..7$.

In comparison to ntile counters, we also analyze ptile flits per second collected by P_REQ_FLIT_n and P_RSP_FLIT_n, which are request and response flits received by a node, respectively. In this paper, we always take the sum of these two metrics when we refer to ptile flit-per-second. Similarly, we refer to the sum of P_REQ_STALL_n and P_RSP_STALL_n as the ptile stalls per second. In these metrics, $n \in 0..3$ corresponds to the four nodes connected with this router. Thus, ptile counters specify the contribution from a certain node. The full names of the counters we mentioned are listed in Table I. The original counters record values cumulatively, so we take a rolling difference to estimate instantaneous values.

In addition, when we calculate stall/flit ratio, we ignore the samples where stall-per-second is smaller than a threshold. This is because when both the stall and the flit number in a second are too small, the stall/flit ratio could occasionally surge while it does not reflect influential congestion. We set the threshold as the median stall-per-second value of data taken over a three-month period from the entire system. For electrical link ntiles and optical link ntiles, the thresholds are 5410794 and 933, respectively.

TABLE I: Aries network counters used in this work [8].

Abbreviation	Full counter name
N_STALL_r_c	AR_RTR_r_c_INQ_PRF_ROWBUS_STALL_CNT
N_FLIT_r_c_v	AR_RTR_r_c_INQ_PRF_INCOMING_FLIT_VCv
P_REQ_STALL_n	AR_NL_PRF_REQ_PTILES_TO_NIC_n_STALLED
P_REQ_FLIT_n	AR_NL_PRF_REQ_PTILES_TO_NIC_n_FLITS
P_RSP_STALL_n	AR_NL_PRF_RSP_PTILES_TO_NIC_n_STALLED
P_RSP_FLIT_n	AR_NL_PRF_RSP_PTILES_TO_NIC_n_FLITS

III. Experimental Methodology

We conduct experiments on Cori, which is a 12k-node Cray XC40 system located at the Lawrence Berkeley National Laboratory, USA. On Cori, network counter data are collected and managed by the Lightweight Distributed Metric Service (LDMS) tool [9]. LDMS has been continuously running on Cori and collecting counter data for years for every node. The data collection rate is once per second.

To characterize job execution performance, we experiment with the following real-world and benchmark applications:

- **Graph500**. We run breadth-first search (BFS) and single-source shortest path (SSSP) from Graph500, which are representative graph computation kernels [10].
- **HACC**. The Hardware Accelerated Cosmology Code framework uses gravitational N-body techniques to simulate the formation of structure in an expanding universe [11].
- **HPCG**. The High Performance Conjugate Gradient benchmark models the computational and data access patterns of real-world applications that contain operations such as sparse matrix-vector multiplication [12].
- **LAMMPS**. The Large-scale Atomic/Molecular Massively Parallel Simulator is a classical molecular dynamics simulator for modeling solid-state materials and soft matter [13].
- **MILC**. The MIMD Lattice Computation performs quantum chromodynamics simulations. Our experiments use the `su3_rmd` application from MILC [14].
- **miniAMR**. This mini-application applies adaptive mesh refinement on an Eulerian mesh [15].
- **miniMD**. This molecular dynamics mini-application is developed for testing new designs on HPC systems [15].
- **QMCPACK**. This is a many-body ab initio quantum Monte Carlo code for computing the electronic structure of atoms, molecules, and solids [16].

To create network congestion on HPC systems in a controlled way, we use the Global Performance and Congestion Network Tests (GPCNeT) [17], which is a new tool to inject network congestion and benchmark communication performance. When launched on a group of nodes, GPCNeT runs congestor kernels on 80% of nodes, and the other 20% runs a canary test in a *random-ring* or *allreduce* communication pattern [17] to evaluate the impact of the congestor kernel. Our experiments run the RMA Broadcast congestor kernel. By comparing running the canary test in isolation with running the canary test together with congestor kernels, GPCNeT reports the intensity of congestion by the following impact factor metrics: bandwidth ratio, latency ratio, and allreduce bandwidth ratio. For example, bandwidth ratio represents the

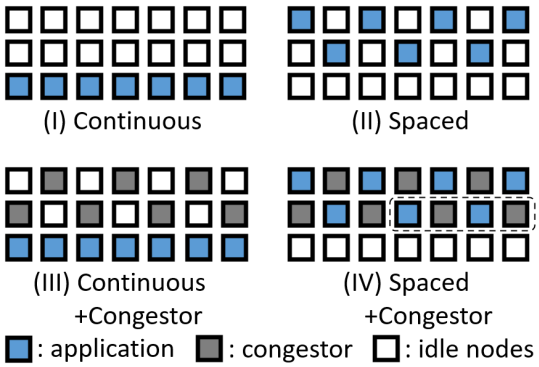


Fig. 2: The four experimental settings. Each square is a node. Blue squares run a parallel application. Grey squares run the GPCNeT congestor. White ones are idle.

canary test's effective bandwidth when running with congestor, divided by the bandwidth when running in isolation.

We quantify the impact of network congestion on applications by comparing the execution time of the applications when running them with or without congestors. We also differentiate between *endpoint congestion* and *intermediate congestion*. Endpoint congestion refers to the congestion generated by traffic from other nodes that share the same routers as our application. Intermediate congestion refers to the congestion caused by traffic not from nodes sharing the same routers but from intermediate links. We design experiments as follows.

Assume we are going to run an application on N nodes (we use $N = 7$ in Fig. 2 as an example, and we actually experiment with $N = 64$), then we reserve $3N$ nodes from the system. Most of these $3N$ nodes are groups of consecutive nodes as the Slurm scheduler on Cori is configured to prioritize reserving consecutive nodes. We have four experimental settings shown in Fig. 2 and described below:

- In Setting I ("Continuous"), we run the application on N nodes continuously selected from the list (shown in blue), and the other $2N$ nodes are left idle.
- In Setting II ("Spaced"), we run the application on N nodes selected by choosing every other node from the list.
- In Setting III ("Continuous+Congestor"), besides the application in a "continuous" way, we simultaneously run GPCNeT on another N nodes selected in a "spaced" manner (shown in grey). In this case, the application is mostly affected by intermediate congestion because the majority of blue nodes do not share routers with grey nodes.
- In Setting IV ("Spaced+Congestor"), the nodes for the application and the nodes for the congestor are interleaved. In this case, the application is also affected by endpoint congestion because sharing router among application nodes and congestor nodes is common. As an example, assume the dashed line shows the four nodes connected to the same router, then, the two grey nodes create endpoint congestion on the other two blue nodes. Although every four nodes are not always connected to the same router, because Cori's scheduler prioritizes allocating contiguous nodes for us, nodes sharing the same router are common.

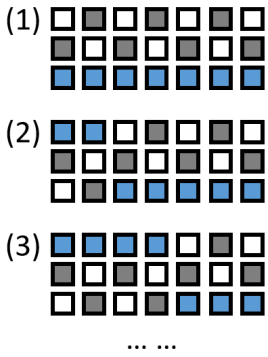


Fig. 3: To mitigate variations from background traffic, we repeat experiments with the placement of application/congestor rotationally shifted (first three shifts for Setting III are drawn).

In our experiments, we always run an application on 64 nodes, and the congestor also occupies 64 nodes. We did not experiment with larger workloads to avoid too much impact on other users in a production system. All experiments run on Haswell nodes and use all 32 cores of each node. The same inputs are used during different runs of an application.

Since the experiments are done in a production system, network traffic generated by jobs from other users may go through the routers we use. To reduce the impact of this background traffic, we repeat each setting 10 times by rotationally shifting the application's and congestor's placement, as illustrated in Fig 3. Each shift rotates $1/10$ of the node list length.

In addition to the experiments discussed above, we also experiment with GPCNeT alone without our applications, and the details are discussed in Section IV-C.

IV. Results and Discussion

In this section, we first analyze the impact of network congestion on applications in Section IV-A. Next, we show the correlation between network metrics and application execution time in Section IV-B. Then, we show that ntile stall/flit ratio is correlated with network congestion intensity in Section IV-C.

A. Impact of network congestion on applications

Figure 4 summarizes the impact of network congestion on applications. The execution times are normalized separately for each application with regard to the median value in Setting I. Because LAMMPS's values exceed the range of Fig. 4(a), we also draw them separately in Fig. 4(b).

These results demonstrate that network congestion has diverse impact on applications. Some applications such as Graph500 and HPCG are negligibly affected by congestion, where the median execution time difference between with and without congestor is less than 10%. On the other hand, applications including HACC, LAMMPS, MILC, miniAMR, and miniMD are significantly affected by congestion, where the median performance with endpoint congestion (Setting IV) is 0.4X to 7X higher than the performance without congestion (Setting II). QMCPACK shows a medium-level impact from congestion, where endpoint congestion extends the median execution time by 11%. The slightly shorter

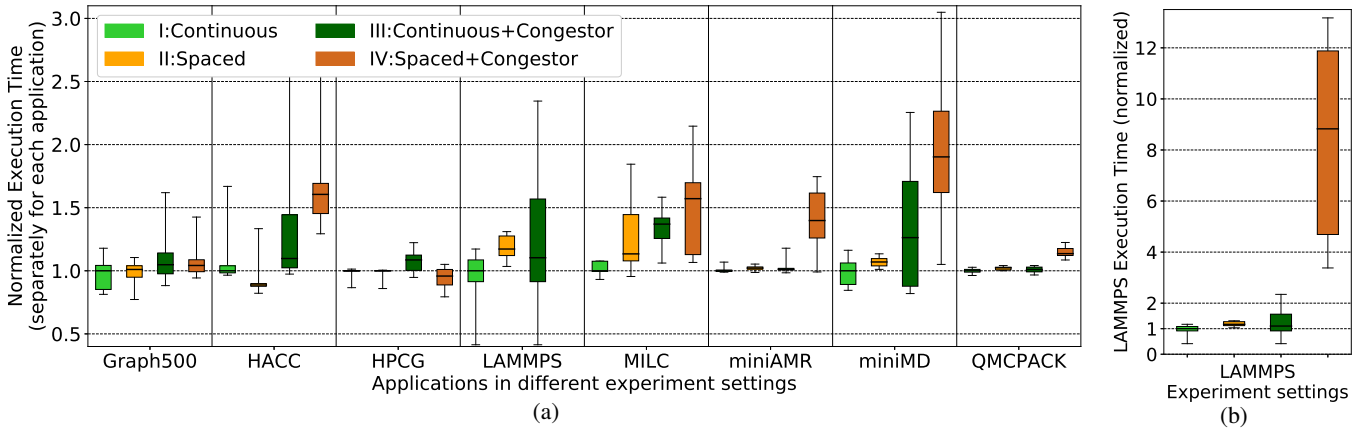


Fig. 4: Normalized application execution time under four experimental settings. Normalization is done separately for each application. Each bar summarizes the 10 runs for an application. Errorbars are min/max; edges of box are the first and third quartiles; middle line is the median. Setting IV of LAMMPS exceeds the range and it is separately drawn in (b).

execution time in Setting II of HACC and Setting IV of HPCG should be due to variations caused by background traffic.

To understand why applications are impacted differently, we use CrayPat [18] to profile the MPI characteristics of the applications (in separate runs without congestor). Table II shows the percentage of time spent on certain MPI operations, and Table III shows the aggregate bytes transferred per second, average message size, and MPI call counts.

From Table II, we see that the applications impacted more by congestion, including HACC, LAMMPS, MILC, miniAMR, and miniMD, share a common feature of more time spent on MPI operations. On the contrary, HPCG and QMCPACK have only 3.7% and 6.0% time on MPI operations, respectively. In addition, more MPI collective operations (such as MPI_Allreduce, MPI_Bcast) implies more intensive communication, making the application sensitive to congestion. Therefore, as LAMMPS spends 25.4% time on MPI_Allreduce, larger than any other applications, its execution time is extended by more than 7X in Setting IV. Similarly, QMCPACK has 5.4% time on MPI_Allreduce, higher than the 0.2% from HPCG, which explains why QMCPACK is impacted more by congestion than HPCG. On the other hand, Graph500 has only 2.3% time on MPI_Allreduce, and less than 8% time on all other MPI calls except for MPI_Test, which explains why it is only slightly affected by congestion.

These findings suggest several key criteria for predicting congestion's impact on an application. The first is the amount of time an application spends performing MPI operations. Intuitively, an application not spending much time on communication will not be sensitive to congestion. Secondly, the type of communication matters. In our experiments, when collectives such as MPI_Allreduce, MPI_Bcast, and MPI_Barrier occupy more than 5% of time, we regard the application as having intensive MPI operation and expect it to be sensitive to congestion. Lastly, MPI_Wait(all) is important as they often indicate synchronization points where the slowest communication dominates performance (as is the case with MILC). Conversely, though Graph500 performs reasonable amounts of communication, the communications are uncoupled from each other as MPI_Test(any) calls indicate communication events that are completely independent of many other communications. Applying this understanding to Table II, we consider HACC, LAMMPS, MILC, miniAMR, miniMD, and QMCPACK as having intensive MPI operations.

From Table III, we see the relationship between average message size and sensitivity to congestion is not clear. HACC, LAMMPS and MILC use very different message sizes but each seems sensitive to congestion. Other studies have found that small-size, latency-sensitive communications are more sensitive to congestion than bandwidth benchmarks typically with large message size [17]. However, this relationship is not as clear cut for real applications.

Based on our results, aggregate data transfer rate is not indicative of congestion sensitivity either. For example, although Graph500 transfers data at 50 MB/s, it is less impacted by congestion than LAMMPS and QMCPACK which transfer data at merely 5 MB/s and 600 KB/s, respectively.

From Fig. 4, we also notice that the applications are more impacted by endpoint congestion than by intermediate congestion. Comparing Setting II with IV, we see HACC, LAMMPS, MILC, miniAMR, miniMD, and QMCPACK are all significantly impacted by endpoint congestion. Comparing Setting I with III, we see only MILC and miniMD are significantly impacted by intermediate congestion. This observation suggests that a compact job allocation strategy is better than a non-compact one because a non-compact allocation increases a job's probability to share routers with other jobs and are more likely to suffer from endpoint congestion.

B. *Correlating network metrics with application performance*

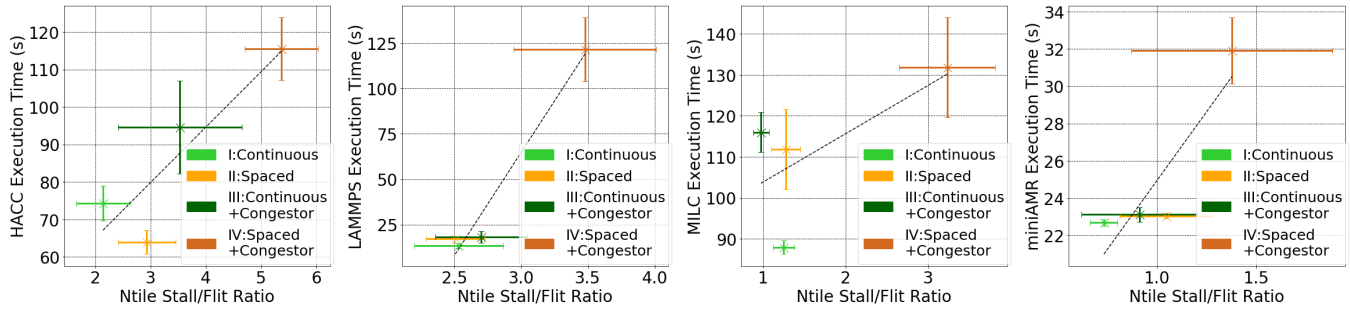
From the same experiments in Section IV-A, we correlate execution time with ntile stall/flit ratio in Fig. 5. Each cross represents the average value of the ten runs in each setting, and errorbars show their standard error. The ntile stall/flit ratio is calculated using the formula in Section II-B, and averaged only

TABLE II: Application MPI profiles collected by CrayPat. "MPI Operation" shows the percentage of execution time spent on MPI operations, and the MPI call breakdown is shown in other columns. "MPI_(other)" is the sum of other MPI calls not specified here. Applications with more time spent on MPI operations, especially MPI collective operations (MPI_Allreduce, MPI_Bcast, MPI_Barrier, etc.), are impacted more by network congestion than applications with less intensive MPI operations.

Application	MPI Operation	MPI_Allreduce	MPI_Sendrecv (or Send, Isend)	MPI_Bcast	MPI_Test (or Testany)	MPI_Wait (or Waitall)	MPI_Barrier	MPI_(other)
Graph500	31.4%	2.3%	2.6%	0.2%	21.3%	<0.1%	4.4%	0.6%
HACC	67.1%	<0.1%	0.2%	0	0	66.2%	0	0.7%
HPCG	3.7%	0.2%	2.1%	0	0	1.2%	0	0.2%
LAMMPS	47.3%	25.4%	8.6%	<0.1%	0	12.2%	<0.1%	1.1%
MILC	61.9%	1.9%	0.6%	<0.1%	0	58.5%	<0.1%	0.9%
miniAMR	26.8%	9.2%	0.5%	<0.1%	0	14.2%	0	2.9%
miniMD	83.4%	0.5%	82.5%	0	0	0	0.2%	0.2%
QMCPACK	6.0%	5.4%	<0.1%	<0.1%	<0.1%	<0.1%	0.5%	0.1%

TABLE III: Execution time, aggregate data transfer rate, average message size, and MPI call counts collected by CrayPat. Columns starting with "MPI_" are breakdown of "MPI Call". Non-dominant MPI call types are not listed. "Exec Time" is the median of the 10 runs in Setting I. "Agg Data Trans Rate" shows the aggregate bytes of data transferred per second.

Application	Exec Time	Agg Data Trans Rate	Avg Msg Sz.	MPI Call	MPI_Allreduce	MPI_Sendrecv (or Send, Isend)	MPI_Test (or Testany)	MPI_Wait (or Waitall)
Graph500	122 s	50 MB/s	9 KB	45,724,042	7,251	607,409	43,852,063	2
HACC	69 s	100 MB/s	6 MB	7,252	41	1,504	0	2,748
HPCG	68 s	3 MB/s	4 KB	128,940	555	40,353	0	40,353
LAMMPS	15 s	5 MB/s	70 B	3,012,409	125,170	893,845	0	812,673
MILC	87 s	90 MB/s	16 KB	10,842,875	13,224	528,076	0	1,056,152
miniAMR	23 s	40 MB/s	22 KB	789,958	10,326	18,366	0	35,540
miniMD	65 s	90 MB/s	3 KB	2,736,074	4,810	2,064,036	0	0
QMCPACK	67 s	600 KB/s	13 KB	8,160	2,004	390	2,697	195



(a) HACC (b) LAMMPS (c) MILC (d) miniAMR

Fig. 5: There are positive correlations between ntile stall/flit ratio and application execution time. A cross represents the average of 10 runs for each setting. Errorbars are standard errors. The dashed line is a linear fit. These positive correlations suggest that ntile stall/flit ratio metric is indicative of performance degradation caused by network congestion.

over routers that contain nodes running our application. The metric is also averaged over the duration of the application.

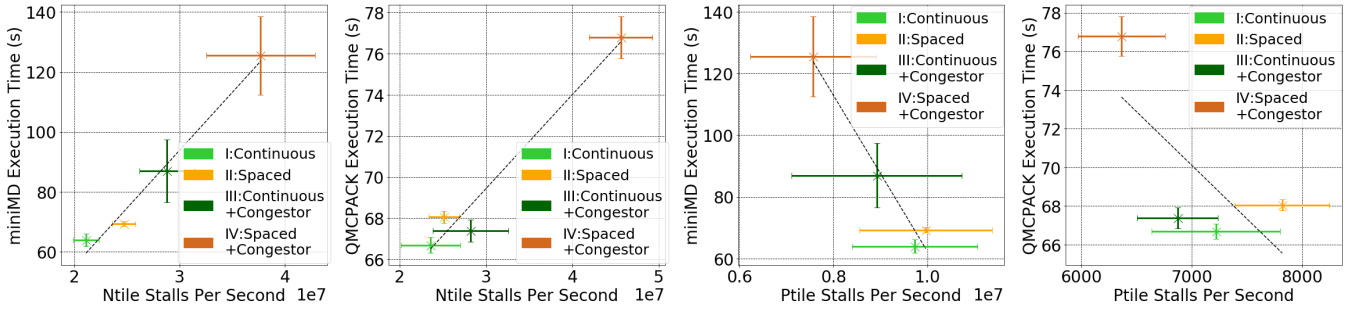
In each case, we notice a positive correlation between job execution time and ntile stall/flit ratio, which demonstrates that this metric is indicative of application performance. Because the ntile counters collect not only local communications directly related to the host router but also communications that travel through the host router as an intermedium, our metric is only statistically correlated with job performance and suffers from variations caused by background traffic.

We also show the stall per second values on either ntiles or ptiles in Fig. 6. The stall count is averaged over routers and durations. While ntile stall per second shows a similar trend as ntile stall/flit ratio, the ptile stall per second shows a negative correlation with execution time. Although this negative correlation seems counter-intuitive at first thought, it in fact implies that ntile links, instead of the ptile-to-node links, are the communication bottleneck in these experiments.

When ntiles are the bottleneck, performance degradation causes an application to run slower and receive less messages per second. As a result, there are less flits per second in the ptile-to-node links. Less flits per second leads to less stalls per second on ptiles since these ptile-to-node links are not the bottleneck. Another way to explain the phenomenon is that the convergence of traffic occurs before the final hop within a switch. Once traffic makes it past the bottleneck, the rest of the path is relatively clear. This explains the negative correlation we see in Fig. 6(c,d). Therefore, we conclude that ntile metrics are better indicators for congestion than ptile metrics since ntiles links, rather than ptiles, are mostly the bottleneck.

C. Correlating network metrics with network congestors

We also conduct experiments that run GPCNeT on either 16, 32, 64, 86, 106, or 128 nodes without our applications. We use



(a) miniMD - Ntile Stall (b) QMCPACK - Ntile Stall (c) miniMD - Ptile Stall (d) QMCPACK - Ptile Stall

Fig. 6: There are positive (negative) correlations between ntile (ptile) stalls per second and application execution time, respectively. The negative correlations in (c) and (d) imply that ptile-to-node links are not the bottleneck of the network.

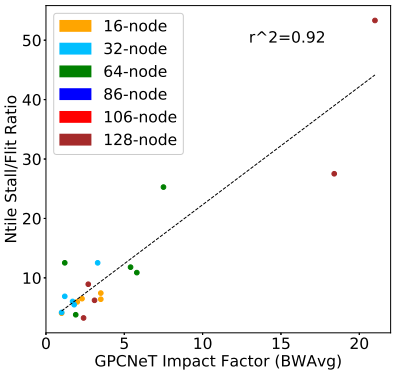


Fig. 7: Correlating ntile stall/flit ratio with GPCNeT congestor impact factor. Different colors represent experiments where we run GPCNeT on different number of nodes.

the impact factor metrics reported by GPCNeT to quantify the intensity of congestion created by GPCNeT. Figure 7 shows the correlation between impact factor (bandwidth ratio) and ntile stall/flit ratio. Each point represents an experiment run. The ntile stall/flit ratio is averaged similarly as before. We see a rough correlation between GPCNeT congestion intensity (quantified by impact factor) and ntile stall/flit ratio, which demonstrates that ntile stall/flit ratio is indicative of network congestion created by GPCNeT.

V. Related work

Network contention/congestion is an important topic in HPC research. It has been reported that, on HPC systems, inter-job network contention causes performance variability as high as 2X [2], 2.2X [3], 3X [4], or 7X [5]. Analysis has shown that network contention, rather than other factors such as OS noise, is the dominant reason for performance variability [2]–[4].

Some prior works have analyzed the statistics of flit or stall counts on HPC systems. Jha et al. analyzed packet and stall count on a 3d-torus network and provided a visualization method to identify network hot spots [19]. Brandt et al. measured network stall/flit ratio and showed its variation across links and over time [20]. These works have not analyzed the relation between network counters and job performance.

A few works have explored the relation between network metrics and job performance using machine learning. Jain et al. trained tree-based classifiers to predict job execution time on a 5d-torus system [21], [22]. They found that buffers and injection FIFOs are important metrics and that the hop-count metric is not helpful in predicting performance. On Cori, Groves et al. demonstrated strong correlations between communication latency and Aries network counters [23]. They built machine learning models to forecast sub-optimal performance and identified network counters related to performance variability [4]. Machine learning methods in this domain often focus on predicting performance or other outcomes; in contrast, our work's focus is on providing an analysis on selected network counters' role in understanding job performance.

On dragonfly systems, prior works have also studied various system setup or management factors that affect job performance. These factors include job allocation [24]–[27], task mapping [28], [29], routing algorithm [30]–[33], link bandwidth [34], global link arrangement [35], [36], etc.

VI. Conclusion

In this work, we show that applications demonstrate substantial difference under network congestion. Applications with intensive MPI operations suffer from 0.4X to 7X extension in execution times under network congestion, while applications with less intensive MPI operations are negligibly affected. By analyzing Aries network counters, we observe a positive correlation between network-tile stall/flit ratio and application execution time, which demonstrates that this metric is indicative of network congestion and job performance. We also show that processor-tile metrics are not good indicators because processor-tile-to-node links are mostly not the bottleneck in our experiments. This work enhances our understanding of application performance under congestion and forms the necessary basis for rational design of congestion-aware HPC system scheduling based on network metrics.

Acknowledgment

This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] D. Skinner and W. Kramer, "Understanding the causes of performance variability in hpc workloads," in *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.*, 2005, pp. 137–149.

[2] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs, "There goes the neighborhood: Performance degradation due to nearby jobs," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 41:1–41:12.

[3] S. Smith, D. Lowenthal, A. Bhatele, J. Thiagarajan, P. Bremer, and Y. Livnat, "Analyzing inter-job contention in dragonfly networks," 2016. [Online]. Available: https://www2.cs.arizona.edu/∼smiths949/dragonfly.pdf

[4] A. Bhatele, J. J. Thiagarajan, T. Groves, R. Anirudh, S. A. Smith, B. Cook, and D. K. Lowenthal, "The case of performance variability on dragonfly-based systems," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020.

[5] S. Chunduri, K. Harms, S. Parker, V. Morozov, S. Oshin, N. Cherukuri, and K. Kumaran, "Run-to-run variability on xeon phi based cray xc systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: ACM, 2017, pp. 52:1–52:13.

[6] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray xc series network," https://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf, 2012.

[7] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," *International Symposium on Computer Architecture (ISCA)*, pp. 77–88, 2008.

[8] Cray Inc., "Aries hardware counters (4.0)," http://pubs.cray.com/content/S-0045/4.0/aries-hardware-counters, 2018.

[9] A. Agelastos *et al.*, "The lightweight distributed metric service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2014, pp. 154–165.

[10] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the graph 500." in *Cray Users Group (CUG)*, vol. 19, 2010, pp. 45–74.

[11] K. Heitmann, H. Finkel, A. Pope, V. Morozov *et al.*, "The outer rim simulation: A path to many-core supercomputers," *The Astrophysical Journal Supplement Series*, vol. 245, no. 1, p. 16, nov 2019.

[12] J. Dongarra, M. A. Heroux, and P. Luszczek, "Hpcg benchmark: a new metric for ranking high performance computing systems," Tech. Rep. ut-eecs-15-736, 2015-01 2015.

[13] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," 5 1993.

[14] G. Bauer, S. Gottlieb, and T. Hoefler, "Performance modeling and comparative analysis of the milc lattice qcd application su3_rmd," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, 2012, pp. 652–659.

[15] M. A. Heroux *et al.*, "Improving performance via mini-applications," Sandia National Laboratories, Tech. Rep. SAND2009-5574, 2009.

[16] J. Kim, A. D. Baczewski, T. D. Beaudet, A. Benali *et al.*, "QMCPACK: an open sourceab initioquantum monte carlo package for the electronic structure of atoms, molecules and solids," *Journal of Physics: Condensed Matter*, vol. 30, no. 19, p. 195901, apr 2018.

[17] S. Chunduri, T. Groves, P. Mendygral, B. Austin, J. Balma, K. Kandalla, K. Kumaran, G. Lockwood, S. Parker, S. Warren, N. Wichmann, and N. Wright, "GPCNeT: Designing a benchmark suite for inducing and measuring contention in hpc networks," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019.

[18] Cray Inc., "Cray performance measurement and analysis tools user guide (7.0.0)," https://pubs.cray.com/content/S-2376/7.0.0/cray-performance-measurement-and-analysis-tools-user-guide/craypat, 2020.

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

[20] J. M. Brandt, E. Froese, A. C. Gentile, L. Kaplan, B. A. Allan, and E. J. Walsh, "Network performance counter monitoring and analysis on the cray xc platform," 2016.

[21] N. Jain, A. Bhatele, M. P. Robson, T. Gamblin, and L. V. Kale, "Predicting application performance using supervised learning on communication features," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 95:1–95:12.

[22] A. Bhatele, A. R. Titus, J. J. Thiagarajan, N. Jain, T. Gamblin, P. Bremer, M. Schulz, and L. V. Kale, "Identifying the culprits behind network congestion," in *2015 IEEE International Parallel and Distributed Processing Symposium*, May 2015, pp. 113–122.

[23] T. Groves, Y. Gu, and N. J. Wright, "Understanding performance variability on the aries dragonfly network," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, Sep. 2017, pp. 809–813.

[24] N. Jain, A. Bhatele, X. Ni, N. J. Wright, and L. V. Kale, "Maximizing throughput on a dragonfly network," *SC*, pp. 336–347, 2014.

[25] R. Budiardja, L. Crosby, and H. You, "Effect of rank placement on Cray XC30 communication cost," *The Cray User Group Meeting*, 2013.

[26] X. Yang, J. Jenkins, M. Mubarak, R. B. Ross, and Z. Lan, "Watch out for the bully! Job interference study on dragonfly network," *SC*, pp. 750–760, 2016.

[27] Y. Zhang, O. Tuncer, F. Kaplan, K. Olcoz, V. J. Leung, and A. K. Coskun, "Level-spread: A new job allocation policy for dragonfly networks," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018, pp. 1123–1132.

[28] B. Prisacari, G. Rodriguez, P. Heidelberger, D. Chen, C. Minkenberg, and T. Hoefler, "Efficient task placement and routing in dragonfly networks," *ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pp. 129–140, 2014.

[29] O. Tuncer, Y. Zhang, V. J. Leung, and A. K. Coskun, "Task Mapping on a Dragonfly Supercomputer," *IEEE High Performance Extreme Computing Conference (HPEC)*, 2017.

[30] D. De Sensi, S. Di Girolamo, and T. Hoefler, "Mitigating network noise on dragonfly networks through application-aware routing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019.

[31] M. Garcia, E. Vallejo, R. Beivide, M. Odriozola, and M. Valero, "Efficient routing mechanisms for dragonfly networks," *Proceedings of the International Conf. on Parallel Processing*, pp. 582–592, 2013.

[32] P. Faizian, S. Rahman, A. Mollah, X. Yuan, S. Pakin, and M. Lang, "Traffic pattern-based adaptive routing for intra-group communication in dragonfly networks," *IEEE Annual Symposium on High-Performance Interconnects (HOTI)*, 2016.

[33] N. Jiang, J. Kim, and W. J. Dally, "Indirect adaptive routing on large scale interconnection networks," *ACM SIGARCH Computer Architecture News*, vol. 37, p. 220, 2009.

[34] T. Groves, R. E. Grant, S. Hemmer, S. Hammond, M. Levenhagen, and D. C. Arnold, "(SAI) Stalled, Active and Idle: characterizing power and performance of large-scale dragonfly networks," *CLUSTER*, pp. 50–59, 2016.

[35] E. Hastings, D. Rincon-Cruz, M. Spehlmann, S. Meyers, A. Xu, D. P. Bunde, and V. J. Leung, "Comparing global link arrangements for dragonfly networks," *CLUSTER*, pp. 361–370, 2015.

[36] M. Belka, M. Doubet, S. Meyers, R. Momoh, D. Rincon-Cruz, and D. P. Bunde, "New link arrangements for dragonfly networks," *IEEE International Workshop on High-Performance Interconnection Networks in the Exascale and Big-Data Era (HiPINEB)*, pp. 17–24, 2017.