Modeling Communication Latency in High-speed Interconnection Networks

Archit Patke*, Saurabh Jha*, Jim Brandt†, Ann Gentile†, Zbigniew T. Kalbarczyk*, and Ravishankar K. Iyer*
*University of Illinois at Urbana-Champaign
†Sandia National Laboratories

***Abstract*—Estimating performance degradation due to network congestion in data centers is important for identifying and mitigating sources/effects of contention. We propose a model-driven methodology that estimates performance degradation experienced by applications in the presence of congestion. Using Gaussian Linear models to approximate latency from network performance counters we were able to correctly estimate 89% of completion times within a 35% error bound.**

I. Introduction

Modern data centers (e.g., massively parallel supercomputers and clouds) may be executing hundreds to thousands of interactive and batch-processing jobs at any given time. These jobs contend with one another for network resources (e.g., bandwidth) leading to significant application runtime variation [1]. The amount of contention (i.e., congestion) experienced by application communication primitives (e.g., send and recv) determines the extent of application performance degradation. Contention for network resources can translate into an increase in latency for application communication primitives. Modeling and estimating communication latency can be used to pinpoint sources of contention and validate its impact on applications meeting their service level objectives. However, accurate latency estimation remains an elusive goal despite the availability of network performance counters and monitoring tools because: 1) performance measurements are inherently noisy, 2) congestion varies significantly both in space and time [2], and 3) there is a lack of fine-grained labeled datasets on performance impact corresponding to varied levels of congestion

In this paper, we propose a model-driven methodology, that continuously determines the extent of performance degradation experienced by application-specific communication primitives in the presence of congestion. We estimate the completion time (latency) of a communication primitive in following steps:

S1: **Targeted experiments and data collection**. We record completion-time of communication primitives (henceforth, referred as latency tests) as well as network-wide performance counters (collected via LDMS [3]) while executing congestion-causing benchmarks (i.e., GPCNet [4]) and applications invoking various communication primitives (e.g., MPI_SEND).

S2: **Modeling and training.** We model the dependency between latency test completion times and network performance counters using a Gaussian Linear Model (GLM). In this model, the completion time of the communication primitive is a function of path taken by the requests (generated by the communication primitive) and the congestion on each link along the path.

S3: **Inference.** We use the trained GLM together with performance counters that are collected online to estimate the completion-time of a communication primitive.

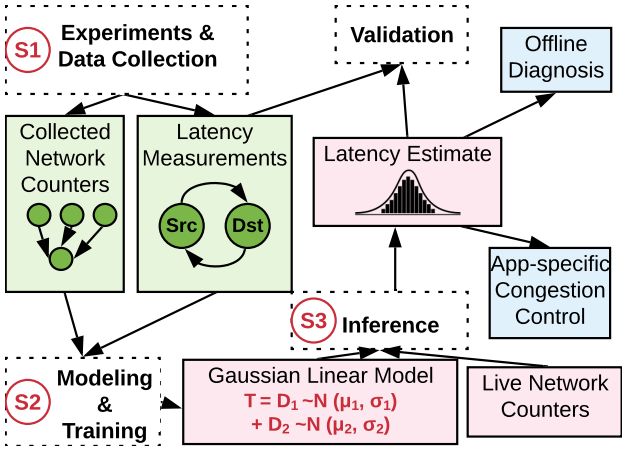


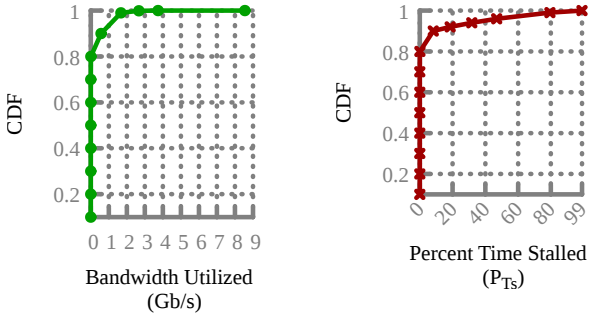
Figure 1: Modeling and Monitoring Workflow

Figure 1 shows the sequence of steps used to generate latency estimates. We execute S1 and S2 only once for collecting and training the data, whereas S3 is executed at each-measurement epoch. Potential use-cases of this approach include: 1) characterization and diagnosis of application performance degradation post-completion of the application, and 2) active mitigation of congestion impact on applications which fail to meet their completion time guarantees by adaptively re-distributing network resources (e.g., via throttling congestor applications).

We implemented and tested our methods on a Cray high-performance computing system consisting of 64 nodes connected via the Cray Aries interconnect [5] which uses a DragonFly-based topology and credit-based control flow algorithms.

II. Model

In our approach, we use Gaussian Linear models to approximate latency using network performance counters, specifically the percent time stalled metric (P_{Ts}) (see Equation 2). We calculate delay for sending messages across network links (D_l) and sum up these delays along a path to obtain a communication latency estimate between two nodes (T_{n_1,n_2}). As the experimental setup system has a limited size of 64 nodes, there exists exactly one minimal path between any pair of nodes. In this model, we choose to ignore effects of non-minimal paths and dynamic routing. Due to the congestion sensitivity of our latency tests, and dedicated use of nodes by applications, OS level delay variation does not significantly contribute to total delay variation. The standard deviation of average latency for tests run in isolation was 0.5μs. This can primarily be attributed to OS noise. In presence of congestors,



(a) Traffic **(b)** Congestion

Figure 2: Traffic and congestion distribution during latency tests with congestors running

average latency rises by $\sim$5.67μs which significantly eclipses the OS noise.

A. Metrics

For our model, we need to define the following metrics: a) effective bandwidth (B_{eff}, the traffic metric) and percent time stalled (P_{Ts}, the congestion metric). Percent time stalled is a measured metric from which effective bandwidth can be derived (See Equation 1).

Effective bandwidth (B_{eff}) is the highest number of flits which can be transmitted on the link per measurement interval. Effective bandwidth and percent time stalled are related by:

$$\frac{B_{eff}}{B_{max}} = \left(1 - \frac{P_{Ts}}{100}\right) \tag{1}$$

where B_{max} is the maximum possible link bandwidth.

Percent Time Stalled (P_{Ts}) is the percentage of time spent stalled (T_{is}) in a network link between two switch ports over the measurement interval (T_i).

$$P_{Ts} = 100\frac{T_{is}}{T_i} \tag{2}$$

B. Estimating Path Latency

The path (P) between 2 nodes n_1 and n_2 is comprised of links $l_1, l_2 ... l_k$.

$$P = (l_1, l_2, l_3, ..., l_{k-1}, l_k) \tag{3}$$

End-to-end latency (T_{n_1,n_2}) can be inferred by summing up individual delays in the links along the path.

$$T_{n_1,n_2} = \sum_{i=1}^{|P|} D_{P_i} \tag{4}$$

For each link l, delay (D_l) can be modelled as a Gaussian process whose mean is proportional to the effective bandwidth of the link.

$$D_l \sim N(\mu = c_1 + m_1\frac{1}{B_{eff}}, \sigma = c_2 + m_2\frac{1}{B_{eff}}) \tag{5}$$

We chose to model this relation using a Gaussian process to account for measurement noise and inaccuracy arising due to averaging percent time stalled across the measurement interval. The parameters $m1$, $m2$, $c1$ and $c2$ were inferred during the training phase.

C. Model Validation

Latency tests were run on 2 randomly chosen nodes from 64 nodes in the system for 100 iterations while varying congestion via congestion-causing benchmarks. Each latency test involved 10000 message exchanges and average message latency for each run was recorded. A large number of messages were sent to compensate for the massive difference in latency test time (2-20 μs) and granularity of measurement (1 second). Figure 2 shows the distribution of traffic and congestion created during these latency tests. Percent time stalled measured ranges from 0 to 99% which indicates that the data set is comprehensive and strong congestion was successfully generated.

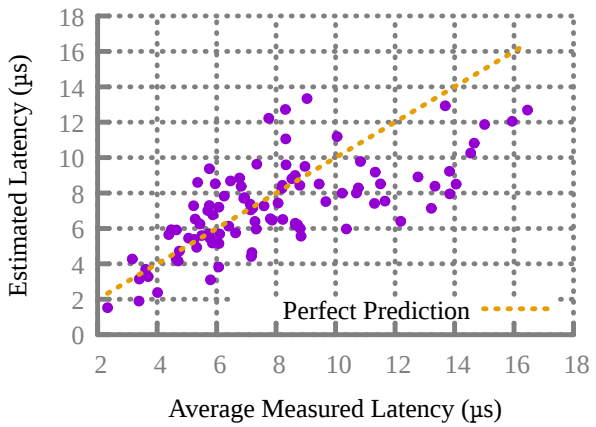


Figure 3: Validating the probabilistic model

Figure 3 shows the expected latency calculated by sampling from the model against the actual measured latency. The Pearson correlation coefficient between the measured and expected latency is 0.68. While using a simple linear regression model would give us a similar correlation, the probabilistic model also enables us to calculate error bounds for the estimated latency. In our experiments, 89% of latency measurements fell within a 35% error bound around the estimated latency.

III. Conclusion

Using our methodology we were able to estimate latency test completion times in the presence of congestion-causing benchmarks with reasonable confidence, i.e., 89% of latency measurements fell within a 35% error bound around the estimated latency. Our future work will involve a) reducing the error bounds, b) validating the model for more realistic scientific applications (e.g., molecular dynamics, weather forecasting, and chemical engineering), and c) scaling the model for larger HPC systems with adaptive routing. Additionally, we will investigate congestion mitigation algorithms that factor in application-specific requirements (e.g., latency guarantees).

References

[1] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs, "There goes the neighborhood: performance degradation due to nearby jobs," in *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 41:1–41:12.

[2] S. Jha, A. Patke, B. Lim, J. Brandt, A. Gentile, G. Bauer, M. Showerman, L. Kaplan, Z. Kalbarczyk, W. T. Kramer, and R. Iyer, "Measuring congestion in high-performance datacenter interconnects," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, Feb 2020.

[3] 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*, 2014, pp. 154–165.

[4] S. Chunduri, S. Warren, N. Wichmann, N. Wright, T. Groves, P. Mendygral, B. Austin, J. Balma, K. Kandalla, K. Kumaran *et al.*, "Gpcnet," 2019.

[5] G. Faanes, A. Bataineh, D. Roweth, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, J. Reinhard *et al.*, "Cray Cascade: A Scalable HPC System Based on a Dragonfly Network," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, 2012, pp. 103:1–103:9.