Diagnosing Network Bottlenecks: One-sided Message Contention

Nathan R. Tallent, Abhinav Vishnu, Hubertus Van Dam, Jeff Daily, Darren Kerbyson, Adolfy Hoisie
Pacific Northwest National Laboratory
{tallent,vishnu,HubertusJJ.vanDam,Jeff.Daily,Darren.Kerbyson,Adolfy.Hoisie}@pnnl.gov

I. Introduction

Two trends suggest that one-sided message network contention is poised to become a cause of concern for scientific application developers. First, there is an increased interest in one-sided messages motivated by Global Address Space (GAS) programming models such as Unified Parallel C (UPC) [1], Co-Array Fortran (CAF) [2], [3], Global Arrays [4], and Chapel [5]. The GAS programming model provides a global shared address space abstraction along with 'memory operations' — such as load, store, and atomic increment — for distributed data structures. GAS programming models use one-sided messages for performance: such messages are readily implemented atop the Remote Direct Memory Access (RDMA) hardware of modern interconnects [6], [7]. However, RDMA operations must be used carefully because network interconnect latency is 1–2 orders of magnitude larger than a typical local memory access. Even worse, network contention can increase that latency by integer factors. Second, there is a growing ratio of hardware threads to network injection bandwidth [8]. When more than one hardware thread initiates a message at the same time, the messages may contend for network resources.

Unfortunately, it is difficult to reason about network contention and one-sided messages. One of the attractions of one-sided programming models is that they enable more asynchronous applications, a trait that is already influencing scientific applications [9], [10]. The asynchrony can makes it difficult to reason about when and where contention occurs. Because asynchronous tasks avoid bursts of communication during synchronization, they have the potential to lower network contention by avoiding demand spikes for network resources. However, asynchronous tasks can also lead to hot spots 'reading' or 'storing' remote data where network contention becomes severe.

This poster presents a portable tool for diagnosing the causes and severity of one-sided message contention.

II. Contributions

Our work makes the following contributions.

First, we characterize contention in NWChem [11], a production computational chemistry application with a highly asynchronous computation structure. We show results for the Carbon 240 benchmark from the MSC Benchmark 2.0 [12] with some modified parameters. This benchmark reflects an important computational work load from the Environmental Molecular Science Laboratory. The benchmark is designed to use strong scaling to obtain a solution as fast as possible.

The graphs in the top-right quadrant characterize the benchmarks's (a) runtime, (b) blocked synchronization time (t_{sync}), (c) blocked one-sided message time (t_{blocked}), and (d) extra message time from contention ($t_{\text{contention}}$) for three machines: PIC, a cluster using QDR InfiniBand (oversubscribed fat tree, static routing); Eos, a Cray XC30 (Cray Aries, dragonfly topology, adaptive routing), and Mira, an IBM Blue Gene/Q (5D torus, zone-adaptive routing). Since the task size in the Carbon 240 benchmark can affect contention, we also show results (when interesting) for two different task sizes. The 'Runtime' graphs show that for each machine, the benchmark hits the point at which strong scaling ends. Our tool's measurement overhead ranged from 1–5%.

Depending on the platform, the source of blocking time changes. Consider the t_{sync} and t_{blocked} graphs, which show the time the benchmark spends blocked for collective synchronization and one-sided messaging, respectively, as a percentage of runtime. On PIC, t_{sync} remains low but t_{blocked} scales poorly. On Eos, t_{sync} grows surprisingly large; and the behavior of t_{blocked} depends radically on the task size. As on PIC, on Eos t_{blocked} grows for the small task size (42). Unlike PIC, on Eos t_{blocked} is a relatively minor problem for the large task size (54). On Mira, in contrast to Eos and PIC, t_{blocked} remains small for even the small task size (42).

Contention changes substantially with task size, interconnect, and data layout. Frequently, $t_{\text{contention}}$ closely corresponds to t_{blocked}. The $t_{\text{contention}}$ graphs show the time one-sided messages spend contending for network resources as a percentage of runtime. When the benchmark blocks waiting for contended messages, the extra time due to contention accounts for significant portions of runtime — as much as 65%. However, this contention does not always lead to slowdowns. The Eos and Mira graphs show that contention suddenly spikes at 256 and 1024 nodes, respectively. The graphs also show contention suddenly changes from being at messages' targets to their initiators. It turns out that this initiator contention is not a problem because it derives from an algorithm that uses double buffering and is therefore mostly hidden.

The reason t_{sync} grows on Eos and Mira is that two non-scalable algorithms (with load imbalance) become important in the context of otherwise good scaling behavior. Explaining the dip on Eos at 256 nodes requires out-of-scope details.

Second, our techniques pinpoint and quantify network contention. The bottom-right quadrant of the poster summarizes the challenges of detecting and quantifying network contention, which is not an easy problem.

Our work is based on two observations. First, unlike two-sided messages, it is possible to reason about network contention without considering synchronization. Second, we distinguish between contention within a network and at its boundaries and focus on the latter. We define *network contention* as contention for a NIC's in or out links and *network congestion* to be contention for forward-links within a network. Several considerations (beyond the scope of this poster) suggest that *network contention* is a good proxy for identifying the effects of contention in general. Furthermore, we can show that when message delivery time deviates from an ideal model, there are nearly always other messages contending for the same network links.

To detect that a message is being affected by network contention, we maintain statistics representing instantaneous (non-local) network resource demand. For low overhead and portability, we maintain network demand by sampling messages and exploiting common RDMA hardware features. To estimate the severity of contention on message latency, we use lightweight in situ modeling and analysis. We identify the portion of a message's latency that is due to contention and whether that contention is at the message initiator or target. We pinpoint the sources of contention by attributing these metrics to program statements in their full static and dynamic context. The results pinpoint the sources of contention, classify its type (initiator/target) and estimate its severity. This information shows when and where contention is a problem and by how much. Our contention metric is a good indicator of how much better an application would perform if contention were eliminated.

Finally, we diagnose and correct network contention in the Carbon 240 benchmark. Consider the screenshot in the lower left quadrant of the poster. This screenshot shows that at 64 nodes on PIC, 100% of contention — 65% of the runtime — is due to a get operation reading data The get's derive from three different call sites and three sets of distributed (global) arrays. Since the contention is due to a dynamically moving hot-spot reading data, we hypothesized that using a block-cyclic distribution for the relevant distributed arrays would reduce get contention. The PIC graphs labeled 'blk-cyc' demonstrate that a block-cyclic distribution can reduce total runtime by up to 20% on PIC. The reason is that the block-cyclic distribution reduces contention by about 50%. We believe it is possible to obtain further gains, but limitations in our Global Arrays implementation prevent us from using the optimal block size for block-cyclic distributions.

III. Discussion

We know of no application tools that diagnose network contention. There has been substantial work understanding contention during two-sided collectives, but that is inapplicable to point-to-point messages. There has been substantial work understanding the latency of two-sided messages, but that work rarely focuses on contention and is inapplicable to one-sided messages. Most prior work on one-sided contention falls into three categories. Queuing-based analytical models such as the LoGPC [13] require a user to supply key parameters such as message injection rates. With a tool, one could measure such parameters and refine the models. Simulations offer insight, but require substantial computing resources [14], [15]. There are a few tools that use back pressure to report contention, but they are system-oriented and hardware-specific [14], [16], [17]. We are aware of no tools that offer application-oriented and measurement-based insight into the problem of one-sided message contention.

References

[1] UPC Consortium, "The UPC language specification, v. 1.2," Lawrence Berkeley National Lab Tech Report LBNL-59208, May 2005.

[2] R. W. Numrich and J. Reid, "Co-array Fortran for parallel programming," *SIGPLAN Fortran Forum*, vol. 17, no. 2, pp. 1–31, Aug. 1998.

[3] J. Mellor-Crummey, L. Adhianto, G. Jin, and W. N. Scherer III, "A new vision for Coarray Fortran," in *Proc. of the Third Conf. on Partitioned Global Address Space Programming Models*, 2009.

[4] J. Nieplocha, B. Palmer, V. Tipparaju, M. Krishnan, H. Trease, and E. Aprà, "Advances, applications and performance of the Global Arrays shared memory programming toolkit," *Int. J. High Perform. Comput. Appl.*, vol. 20, no. 2, pp. 203–231, 2006.

[5] B. Chamberlain, D. Callahan, and H. Zima, "Parallel programmability and the Chapel language," *International Journal of High Performance Computing Applications*, vol. 21, no. 3, pp. 291–312, 2007.

[6] A. Vishnu, D. Kerbyson, K. Barker, and H. van Dam, "Building scalable PGAS communication subsystem on Blue Gene/Q," in *2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops*, May 2013, pp. 825–833.

[7] A. Vishnu, J. Daily, and B. Palmer, "Designing scalable PGAS communication subsystems on Cray Gemini interconnect," in *19th International Conference on High Performance Computing*, Dec 2012, pp. 1–10.

[8] A. Geist and R. Lucas, "Major computer science challenges at exascale," *Int. J. High Perform. Comput. Appl.*, 2009.

[9] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov, "Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects," *Journal of Physics: Conference Series*, vol. 180, no. 1, 2009.

[10] A. Haidar, H. Ltaief, and J. Dongarra, "Parallel reduction to condensed forms for symmetric eigenvalue problems using aggregated fine-grained and memory-aware kernels," in *Proc. of the 2011 ACM/IEEE Conf. on Supercomputing*. New York, NY, USA: ACM, 2011.

[11] M. Valiev, E. Bylaska, N. Govind, K. Kowalski, T. Straatsma, H. V. Dam, D. Wang, J. Nieplocha, E. Apra, T. Windus, and W. de Jong, "NWChem: A comprehensive and scalable open-source solution for large scale molecular simulations," *Computer Physics Communications*, vol. 181, no. 9, pp. 1477–1489, 2010.

[12] Environmental Molecular Science Laboratory, "MSC Benchmark, v. 2.0," http://www.emsl.pnl.gov/capabilities/computing/msc/msc_benchmark/, February 2012.

[13] C. Moritz and M. Frank, "LoGPC: Modeling network contention in message-passing programs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 4, pp. 404–415, 2001.

[14] G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy, and J. Duato, "Solving hot spot contention using InfiniBand architecture congestion control," in *Workshop on High Performance Interconnects for Distributed Computing*, 2005.

[15] J. Peter D. Barnes, C. D. Carothers, D. R. Jefferson, and J. M. LaPre, "Warp speed: Executing time warp on 1,966,080 cores," in *Proc. of the 2013 ACM SIGSIM Conf. on Principles of Advanced Discrete Simulation*, ser. SIGSIM-PADS '13. New York, NY, USA: ACM, 2013, pp. 327–336.

[16] M. Gusat, D. Craddock, W. Denzel, T. Engbersen, N. Ni, G. Pfister, W. Rooney, and J. Duato, "Congestion control in InfiniBand networks," in *Proc. of the 13th Symp. on High Performance Interconnects*, 2005, pp. 158–159.

[17] A. Vishnu, A. Mamidala, H.-W. Jin, and D. Panda, "Performance modeling of subnet management on fat tree InfiniBand networks using OpenSM," in *Proc. of the First Intl. Workshop on System Management Techniques, Processes, and Services (held in conjunction with IPDPS'05)*, April 2005.