Application Communication Emulation for Performance Management of NOW Clusters

Jeffrey J. Evans
Department of Electrical and Computer Engineering Technology
Purdue University
401 N. Grant Street
West Lafayette, IN, USA
jje@purdue.edu

Cynthia S. Hood
Department of Computer Science
Illinois Institute of Technology
10 W. 31st Street
Chicago, IL, USA
hood@iit.edu

Abstract

Performance management of clusters and Grids poses many challenges. The sharing of large distributed sets of resources can provide efficiencies, but it also introduces complexity in terms of providing and maintaining adequate performance. Current application requirements focus on the amount of resources needed without explicitly characterizing the performance required from those resources. Inconsistent, or highly variable run-time of applications is an indication of systemic inconsistency and is an issue with implications for those running the application as well as those managing the resources. We are focusing on the contribution of the interconnection network to application run-time variability. This work presents experimental results characterizing parallel application run-time sensitivity to communication performance variability using an *Application Communication Emulator* (ACE).

Keywords

cluster, application run-time variability, network performance management

1. Introduction and Background

In this paper we focus on cluster communication performance and demonstrate a high variability. For some applications, this variability could have a significant negative impact. Generally the application designer will specify the number of processors, the length of the job, and the required interconnection network on systems equipped with multiple interconnection systems. When the specified requirements are the same, it is desirable that large systems of resources operate in a consistent manner. To best manage network performance we first must understand and quantify communication performance variability. Previous work [1, 2, 3] provides motivation for analysis into the application's perception of sensitivity to communication cost variability.

Applications on clusters and Grids tend to be parallel programs which are distributed among many processes and processors. We focus on systems where a one-to-one relationship of processes to processors exists. Processes tend to operate in *cycles* of computation and data exchange (communication). Communication between processes reduces to communication between processors, requiring the services of a communications "stack" and interconnection network.

The communication time of a parallel program T_{comm} is "the time it takes to send and receive messages" [4]. From the application's perspective, message communication can be divided into components relating to CPU and memory system interactions (including the NIC) and network interactions. The modeling of communication cost in parallel programs has been developed into a well-known linear 2-parameter model, which can be expressed as

$$T_{comm} = \sum_{i=0}^{m} T_{msg}^{i} = \sum_{i=0}^{m} \alpha + \beta n, \tag{1}$$

where α is the per message startup cost (independent of message size) and β is the cost per unit of data. When more than one application is executing simultaneously (a common practice) modeling communication performance in this manner for estimating run-time performance may be less effective, since systemic communication patterns and task locations for the set of applications is unknown and potentially fragmented [3].

A study on the Intel Paragon [7] is the closest work to our own in this regard. Two approaches were used to degrade network bandwidth. Messages were exchanged multiple times in an effort to consume hardware bandwidth while preserving contention effects of the 2D mesh interconnect. A second approach was to apply synthetic perturbations in the communications routines by inserting "spin wait loops".

2. Experiments and Results

Our method of isolating network performance without instrumenting the application called "application communication emulation", uses a two-phase approach. The first measures communication time then uses the 2-parameter linear cost model to determine the communication time of messages used in the second phase. The second phase performs compute / communicate cycles, emulating a parallel application *run*. Actual compute and communication times are also measured during the run. ACE uses the Message Passing Interface [5] for communication and acquisition of timing information. One or more independent *emulated applications* (EAs) can be instantiated using MPI *communicators*.

The compute time algorithm (T_{comp}) is designed to run consistently close to its calculated value. Any variation in run-time should therefore be attributable to variations in communication time[*]. The compute time is derived from the notion that the compute/communicate cycle time is

$$T_{cycle} = T_{comp} + T_{comm}. \tag{2}$$

Using measured T_{comm} and the user supplied percentage of time an EA spends communicating (*load*) and number if iterations, the compute, cycle, and run-time is calculated (predicted). Undesired memory exchanges are avoided by pre-allocating communication buffers. Each process records its run-time, communication, compute, and synchroniza-

[*]The exception is in the event of a system administrator or inadvertent user gaining access to a compute node and performing operations in addition to those of the emulator

tion time statistics in simple (small) data structures. Run-time error is then the difference between the predicted (computed) and actual run-time.

Evaluations of the ACE framework were performed on two different clusters, Chiba City [6] at Argonne National Laboratory and the PC Linux cluster at Purdue University. Both clusters operate such that users "own" each of their allocated nodes. Tests were performed over a wide range of node allocations, message sizes, communication loads, message patterns, and number of EAs. Here we present results from Chiba City using its Myrinet interconnection network for large (1MB) messages, communication load from 10 to 80%, and up to 2 EAs using point-to-point and collective message exchange operations on up to 32 nodes. It is noteworthy that sequential runs across the communication load range were performed on the same node set and a new benchmark (phase one) was performed prior to each run. This is critical as it provides a direct measure of the most up-to-date network performance.

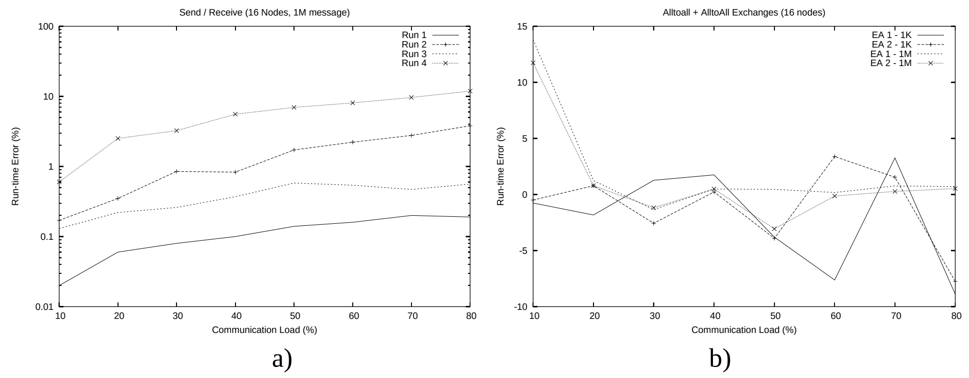

Figure 1: Run-time Variability a) Blocking Send b) Collective Exchange (2 EAs)

Figure 1a shows how ACE exposes network performance variability. 1,000 iterations (T_{cycle}) were performed for each test, thus run-times ranged from a few seconds to several minutes. Run-time prediction errors of nearly two orders of magnitude led to investigating the ACE logs for anomolies in compute or communication times.

Statistics on these runs revealed nearly identical mean compute times ($< 1\%$ of calculated) and small standard deviations, on the order of 0.005%. Comparing runs where up to two orders of magnitude difference in run-time prediction error was observed, we found anomolies in dispersion statistics. Mean values of measurerd T_{comm} are within $\pm 3\%$, however the standard deviations were different by a factor of nearly 70. We attribute these anomalies to network performance since the machines are of identical capability (CPU, memory, OS, libraries, etc.).

Collective operations (scatter, gather, Alltoall) are used frequently in parallel programs such as those solving computational fluid dynamics and N-body particle problems. Figure 1b captures significant run-time variability behavior of each of two EAs, using 16 total nodes, both executing Alltoall message exchanges.

3. Summary and Future Work

This work attempts to gain insight into the relationship of run-time variability and communication performance, as motivated by our previous work [1, 2, 3]. Testing and validation activities of ACE on one experimental and one production NOW cluster has produced insightful results.

The capturing of dispersion statistics has been shown to be useful in order to gain insight into run-time variability when benchmarked communication coefficients and average communication times are nearly identical, implying consistent run-time. This work also illustrates that measuring communication performance only moments prior to program execution provides no guarantee that performance will not degrade (or improve) during the actual running of the application. For an actual parallel application this imprecision manifests itself as a variability in the ratio of computation to communication.

We anticipate using the ACE framework as a means to characterize other parallel applications in terms of their sensitivity to network performance. The ability to characterize acceptable network performance will facilitate system adaptation in the face of unacceptable performance. To remedy poor network performance, the network may adapt it's routes or the application may be migrated to another part of the cluster with more consistent network performance.

4. Acknowledgements

This work was supported in part by the U.S. Department of Energy, under Contract W-31-109-Eng-38 and NSF 9984811. The authors also thank the research computing department of Information Technology at Purdue (ITaP) for insights and use of their Linux clusters.

References

[1] Jeffrey J. Evans, Seongbok Baik, Cynthia S. Hood, and William Gropp. Toward understanding soft faults in high performance cluster networks. In *Proceedings of the 8th IFIP/IEEE International Symposium on Integrated Network Management*, pages 117–120, March 2003.

[2] Jeffrey J. Evans, Seongbok Baik, Joseph Kroculick, and Cynthia S. Hood. Network Adapability in Clusters and Grids. In *Proceedings from the Conference on Advances in Internet Technologies and Applications (CAITA)*. IPSI, July 2004.

[3] Jeffrey J. Evans, Cynthia S. Hood, and William D. Gropp. Exploring the relationship between parallel application run-time variability and network performance in clusters. In *Workshop on High Speed Local Networks (HSLN) from the Proceedings of the 28th IEEE Conference on Local Computer Networks (LCN)*, pages 538–547, October 2003.

[4] Ian Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley Publishing Company, 1995.

[5] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA, 2nd edition, 1999.

[6] Argonne National Laboratory. Chiba City, the Argonne scalable cluster. Online Document, 1999. http://www-unix.mcs.anl.gov/chiba/.

[7] P. H. Worley, A. C. Robinson, D. R. Mackay, and E. J. Barragy. A study of application sensitivity to variation in message passing latency and bandwitdh. In *Concurrency: Practice and Experience*, volume 10, pages 387–406. John Wiley & Sons, April 1998.