Network Performance in Distributed HPC Clusters

Ben Huang, Michael Bauer, Michael Katchabaw
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada N6A 5B7

Abstract - Linux-based clusters have become prevalent as a foundation for High Performance Computing (HPC) systems. As these clusters become more affordable and available, and with the emergence of high speed networks, it is becoming more feasible to create HPC grids consisting of multiple clusters. One of the attractions of such grids is the potential to scale applications across the various clusters. This creates a number of interesting opportunities as well as introduces a number of challenges. A key question is the impact of the inter-cluster network on the overall performance of the computing environment. In this paper, we report on recent experiments to evaluate such performance.

Keywords: High performance computing, grids, network performance analysis, benchmarking tools.

1. INTRODUCTION

Being able to solve larger, more complex problems in a shorter period of time is a key motivator in building High Performance Computing (HPC) systems. Today's computers are becoming more and more powerful and high-speed, low-latency networks are becoming increasingly available and affordable. It is now possible to build powerful HPC platforms from off-the-shelf computers and networks. In particular, the Linux-based commodity cluster constructed by general purpose computers is an increasingly popular model and seems to be a trend for future HPC [1].

Commodity cluster computing can be characterized as cost effective, extensible, and easy to maintain. With off-the-shelf components and operating systems it becomes feasible to interconnect such systems over local and wide area networks, particularly, as the speed of commodity networks increase. Such *distributed HPC clusters*, or *HPC grids*, offer the potential of large computational spaces when needed. However, since these commodity clusters are built with a variety of computers and network devices, they do not necessarily guarantee high performance. The performance of a standalone computer depends on its operating system, CPU, memory speed, and a variety of other factors. For HPC clusters, network communication is another key factor in performance - communication bottleneck in an HPC cluster may lead to a significant loss of overall performance in the cluster. Inter-cluster performance is also critically dependent on network performance. Detailed network performance analyses that identify these bottlenecks are capable of yielding insight that developers can use to build better applications and administrators can use to better configure, manage, and administer their clusters and grids.

In this paper, we report on measurements of network performance within and between HPC clusters. In Section 2 we provide a brief overview of previous work on network measurements and in Section 3 we provide an introduction to a tool for HPC environments. In Section 4, we describe our HPC grid and the clusters used for experimentation and report on experiments measuring UDP and TCP communication under a variety of parameters. We conclude with some directions for future work.

2. RELATED WORK

To measure network performance, such as throughput and latency, two types of measurements are commonly used: active and passive. Active measurement introduces a workload into the network, usually through the execution of a special application. Typically, this is done in a client/server model, in which the client and server exchange probe packets across the network, with both the client and server collecting timing measurements[2]. These models assume that characteristics of each packet traveling a link are related to the bandwidth and network delay. This type of measurement is considered non-intrusive, since it does not significantly increase the network traffic during the testing. Examples of tools adopting these techniques include Pathchar [3] and Pathrate [4].

Passive measurement, in contrast, probes existing network traffic to compute various network metrics. For example, in TCP the time interval between a SYN packet and the corresponding SYN/ACK packet can be used as a measure of the round trip time (RTT) between the two hosts involved. Passive measurement does not create any additional network traffic, and consequently does not require the execution of traffic generating application processes on hosts in the network. A simple example of a passive measurement application is tcpdump[5]. People often use this tool to capture a selection of packets and analyze the appropriate header fields to help understand network usage and behaviour. Other passive measurement tools include ntop[6] and nettimer[7]. Generally, however, passive measurement is not used for network benchmarking as it is not flexible, is difficult to control, and is not repeatable.

To study networks in HPC environments, it is preferable to use an active measurement model as it allows direct measurements of network performance and behavior. While this can cause disruptions during experimentation, this inconvenience is well worth the better quality results that can be obtained. There are many existing tools available that involve active measurements, including Udpmon[8], Netperf[9], Iperf[10] and NetPIPE[11]. All of these tools were designed as general purpose network tools, and have limitations and restrictions that make them unsuitable for HPC environments. For example, none of the above tools test non-blocking communication and none specialize in high performance interconnects, nor are they capable of testing all three of the most common communication protocols in commodity clusters: UDP, TCP, and MPI. In theory, it is possible to modify these tools, but, their implementations are quite complex and difficult to extend. For example, the Netperf utility is comprised of more than 40,000 lines of C code.

3. DESIGN of HPCBENCH

With this in mind, we chose to implement our own benchmarking tool, Hpcbench, focusing specifically on HPC environments. Hpcbench was designed to measure the high-speed, low-latency communication networks in Linux-based HPC systems. The objectives of Hpcbench include: 1) High accuracy and efficiency; 2) Support for UDP, TCP and MPI communications; 3) Tunable communication parameters of interest in an HPC environment; 4) Detailed recording of test results, communication settings, and system information. Hpcbench was written in C and uses BSD socket and MPI APIs. It has three independent sets of benchmarks measuring UDP, TCP, and MPI. For UDP and TCP communication tests, we employ a client/server model that uses two channels during testing: a control channel and a test channel. The first is a reliable TCP connection for critical data communication for controlling the test run, while the second is used for carrying test data packets. This two-channel design makes it easier to control the tests and gather results. The control channel is used by Hpcbench solely for control of the tests and only involves data transfer before each test starts and after each test ends; thus, it does not introduce additional traffic or overhead during actual testing..

Another reason for two communication channels in Hpcbench is for test configuration purposes. Hpcbench supports many test modes for various protocols, with numerous tunable parameters for each protocol; all of this must be configured for each test. For example, some socket options, such as a socket's buffer size, should be set before establishing a connection for test data packets. With only one communication channel between the client and server, the server process must be initialized with a long and cumbersome argument set according to the client's test setting. For further details, the reader is urged to refer to [12].

4. NETWORK PERFORMANCE

In this section, we present results of analyzing UDP and TCP throughput for inter-cluster and intra-cluster communication within SHARCNET [13], the Shared Hierarchical Academic Research Computing Network. SHARCNET is a multi-institutional HPC network distributed across 9 universities in Ontario. We look at network performance involving three clusters in SHARCNET: *greatwhite* (Western), *deeppurple* (Western), and *hammerhead* (Guelph). These are Linux clusters with 39, 12 and 28 Alpha ES40 4-cpu, 833MHz processors, respectively, with 4Gb RAM per ES40 and Gigabit Ethernet and Quadrics QSW Elan3 [14] interconnects; more details can be found in [12].

We focus on blocking, unidirectional stream experiments, because they more directly show the network behavior with less benchmark overhead. Our experiments include both intra-cluster and inter-cluster communications in our test-bed. For inter-cluster tests, we examined the communication between *greatwhite* and *deeppurple*, which were connected via an optical fiber (1KM distance) and communication between *greatwhite* and *hammerhead* with a long distance fiber optic link (150KM). All three clusters use the same network devices (Alteon AceNIC, HP Passport 8600 switch) and software (Linux 2.4.21).

4.1 UDP Communication

Table 1 summarizes the unidirectional tests using UDP. We make three observations based on the results. First, the throughput increased when datagram size was increased from 1KB to 1460-bytes, but it dropped when the datagram size was increased from 1460-bytes to 4KB. When datagram size increased to 40KB and the socket buffer size was relatively large (1MB and 10MB), network throughput decreased drastically. Second, throughput increased when socket buffer size increased, but remained reasonably steady for larger socket buffer sizes. Finally, throughput varied slightly for different links, but not significantly.

To explain the first observation, we look at the packet size for the different datagrams. The MTU size in our test-bed was 1500-bytes, implying a maximum 1472-byte UDP payload for each packet. When datagram size was increased from 1KB to 1460-bytes, fewer packets needed to be transmitted to transfer the same amount of data, and system overhead was reduced, resulting in a higher throughput. Throughput decreased, however, when datagram size was increased from 1460-bytes to 4KB. This likely occurred because each 4KB datagram had to be fragmented into 3 packets to fit the MTU size before transmission in the network, resulting in some smaller packets requiring transmission, and more overhead.

Table 1: Intra/Inter-cluster UDP Throughput (Mbps)

UDP throughput (Mbps) (mean of ten replications)					
Datagram (Bytes)	***Link***	***Socket buffer***			
		10KB	**100KB**	**1MB**	**10MB**
1024	gw→gw	165.11	485.22	575.30	574.79
	gw→dp	162.27	471.38	556.33	559.54
	gw→hh	162.31	459.07	557.15	568.03
1460	gw→gw	177.75	557.43	649.83	647.75
	gw→dp	177.68	541.88	628.19	630.91
	gw→hh	177.59	539.45	636.77	638.83
4KB	gw→gw	147.64	549.46	586.02	583.32
	gw→dp	146.55	540.11	539.13	541.22
	gw→hh	147.30	536.34	538.54	537.49
40KB	gw→gw	---	567.09	1.15	1.13
	gw→dp	---	564.46	1.16	1.13
	gw→hh	---	565.78	1.15	1.14

Further, an entire 4KB datagram would be discarded if there was a single packet lost.. When datagram size was increased to 40KB, this made the situation worse, since one datagram was segmented into at least 28 packets, and the entire 40KB of data would be considered lost if any of these 28 packets were lost. When the socket buffer size was of a medium size, 100KB in our experiments, the socket buffer was itself a bottleneck (the application was blocked from sending because the socket buffer was frequently full). Consequently, there was relatively little data loss during kernel processing, and network throughput was expectedly high. However, when the socket buffer was made large enough, 1MB and 10MB in our case, the socket buffer limitation was eliminated, and UDP data was periodically dropped when the data transfer from the application exceeded the kernel's (or network interface's) capabilities. When data loss caused by the sender itself was considerable, few complete datagrams could be reassembled at the server. System log files collected by Hpcbench during these tests verified this.

Without considering the effects of data loss, the kernel was able to process more packets in one transmission with larger socket buffers, so the overhead of the sending process was reduced, and the throughput could increase. When the maximum sustainable throughput was reached, larger socket buffers produced no further gains: throughput increased dramatically when socket buffer size was increased from 10KB to 100KB, but there was relatively little change when socket buffer size was increased from 100KB to 1MB, and from 1MB to 10MB.

Table 1 shows that the maximum UDP throughput was approximately 630~650 Mbps for both intra-cluster and inter-cluster communication, which was achieved with a relatively large socket buffer size (1MB and 10MB). Similar throughputs observed during both intra-cluster and inter-cluster UDP communications demonstrate that the fiber optic network was able to provide sufficient bandwidth for Gigabit Ethernet communications over a long distance.

Table 2: Intra/Inter-cluster TCPThroughput (Mbps)

TCP throughput tests (Mbps) (means of ten replications)					
Message Size	***Link***	***Socket buffer***			
		10KB	**100KB**	**1MB**	**10MB**
10K	gw→gw	108.34	513.86	568.13	587.71
	gw→dp	88.79	495.32	565.79	572.33
	gw→hh	12.30	152.04	527.51	535.40
100K	gw→gw	119.85	515.47	574.22	589.43
	gw→dp	98.41	504.30	570.44	573.27
	gw→hh	13.82	157.65	541.89	549.20
10MB	gw→gw	117.27	510.82	573.15	590.54
	gw→dp	98.22	504.41	567.08	567.18
	gw→hh	13.85	155.33	534.67	550.14

4.2 TCP Communication

For TCP throughput tests, we chose 10KB, 100KB, 1MB, and 10MB socket buffer sizes with three different message sizes: 100KB, 1MB, and 10MB. Tests were conducted in the same way as the UDP tests. Table 2 shows the unidirectional test results.

In contrast to UDP, TCP throughput decreased significantly for cross-cluster communication when the socket buffer size was small, particularly in the long distance communication between *greatwhite* and *hammerhead* with a 10KB socket buffer. UDP could achieve more than 140 Mbps throughput with any datagram size, while TCP throughput was less than 14 Mbps for all message sizes. This is likely due to TCP's transmission rate being controlled by the TCP sliding window. To guarantee reliable delivery of data, the TCP active window shrinks during transmission, and no more data will be sent when the TCP window closes. The theoretical maximum throughput is window-size/RTT-time if there is sufficient bandwidth for the data transfer. In our example, the RTT time between *greatwhite* and *hammerhead* was measured to be about 2.9ms. As a result, the maximum throughput for a 10KB socket buffer (with an actual usable buffer space of only 5KB in Linux) is about 5*1024*8/0.0029 ≈ 14.12 Mbps and for a 100KB socket buffer, this value is about 5*1024*1024*8/0.0029 ≈ 144.63, both very close to our test results.

We also observe that message size had little impact on measured throughput since TCP is a byte-stream protocol. The achievable maximum TCP throughput in our tests was about 590 Mbps on the cluster *greatwhite*, 570 Mbps between *greatwhite* and *deeppurple*, and 550 Mbps between *greatwhite* and *hammerhead*.

5. CONCLUSION

Given the emergence of high performance commodity clusters and the ability to connect them via high bandwidth networks, such as in SHARCNET's case, the potential of large scale applications operating over HPC grids becomes a real possibility. For consortia of institutions, this becomes a very viable approach to large scale HPC. This approach is, however, not without challenges. In particular, as with any cluster, network performance and tuning is crucial. In this paper, we looked at UDP and TCP communications within and between clusters. We introduced and illustrated the use of a Linux-based network measurement toolset designed for high performance networks, to do these measurements. Some conclusions from this work include:

- With appropriate settings and configuration, we can achieve about as high a throughput whether inside or between clusters for UDP and TCP communications.
- The maximum UDP throughput was approximately 630~650 Mbps (1 Gb connection) for both intra-cluster and inter-cluster communication. This occurred with 1460-byte datagrams and a relatively large socket buffer size (1MB and 10MB). Larger datagrams, especially at 40KB, were extremely ineffective.
- Hpcbench proved to be a useful tool in evaluating UDP, TCP, and MPI communication throughput and latency with a variety of configurable parameters.

There are many interesting directions for our work to take. While Hpcbench already supports a large number of variables and protocol options, there is always more that can be added, for example support for other MPI methods of communication besides point-to-point.

Currently, tracing MAC layer statistics and other low-level network information is only possible with Gigabit Ethernet and other TCP/IP-based networks, but for proprietary technologies, such as Myrinet and QsNet, it is possible to trace this information with vendor-dependent APIs. Extensions to the tool to include this support would also be useful.

A topic for future experimental study is the relationship between network performance and computational performance, e.g. different network interface cards can lead to different network throughput and latency [12]. How does the computational capacity of a HPC cluster change with different network behavior introduced by different configurations or underlying technologies? Such experimentation is important for developing guidelines for building commodity clusters and grids in the future.

REFERENCES

[1] The TOP500 Supercomputers list, http://www.top500.org.

[2] Kevin Lai, Mary Baker. *Measuring Link Bandwidths Using a Deterministic Model of Packet Delay*. **Proceedings of ACM SIGCOMM 2000**, August 2000.

[3] A. B. Downey. *Using pathchar to Estimate Link Characteristics*. In **Proc. of ACM SIGCOMM**, 1999.

[4] C. Dovrolis, P. Ramanathan, and D. Moore, *What Do Packet Dispersion Techniques Measure?* **IEEE INFOCOM**, April 2001.

[5] Tcpdump utility and packet capture library, http://www.tcpdump.org.

[6] L. Deri and S.Suin, *Effective Traffic Measurement using ntop*, **IEEE Communications Magazine**, May 2000.

[7] K. Lai and M. Baker. *Nettimer: A tool for measuring bottleneck link bandwidth*. In **Proceedings of the USENIX Symposium on Internet Technologies and Systems**, San Francisco, California, March 2001.

[8] Udpmon network measurement tool, R. Hughes-Jones, http://www.hep.man.ac.uk/u/rich/net/.

[9] Netperf Homepage, http://www.netperf.org.

[10] Iperf Homepage, http://www.iperf.org.

[11] Dave Turner, Adam Oline, Xuehua Chen, Troy Benjegerdes, *Integrating New Capabilities into NetPIPE*. PVM/MPI 2003: 37-44.

[12] Ben Huang, Master's thesis, **Network Performance Studies in HPC environments**. Department of Computer Science, The University of Western Ontario, Canada. (http://hpcbench.sourceforge.net)

[13] SHARCNET Homepage, http://www.sharcnet.ca.

[14] Fabrizio Petrini et al. *The Quadrics Network (QsNet): High-Performance Clustering Technology*. **IEEE Micro**, January-February 2002, pp. 46-57.