Evaluation of Point-to-Point Network Performance of HPC Clusters at the Level of UDP, TCP, and MPI

Eric Gamess[1,2], Humberto Ortiz-Zuazaga[2]
eric.gamess@ciens.ucv.ve, humberto@hpcf.upr.edu

[1] School of Computing, Central University of Venezuela, Caracas, Venezuela
[2] Department of Computer Science, University of Puerto Rico, San Juan, Puerto Rico

Abstract: Linux-based clusters and commodity hardware are becoming more prevalent in the field of HPC (High Performance Computing), replacing the expensive proprietary solutions, and making HPC more affordable. Hence, the community has developed all-in-one software solutions to rapidly deploy HPC clusters, based on the integration of different open source projects into a Linux distribution, allowing provisioning, configuration, management, and monitoring. Rocks Cluster is a famous cluster manager solution that has been widely accepted in academic, governmental, and commercial environments, since it is open source, it has been actively developed, and its main objective is to make easier the life cycle of a HPC cluster. Network performance evaluation at different protocol level is crucial for the management and administration of such clusters, since it allows the validation and troubleshooting of the networking system. To assist in this process, several benchmarking tools have been proposed, with their own limitations and strengths, reporting evaluation results at different levels of the OSI model. In this paper, we describe some popular tools for network measurement, at the most important levels for HPC developers: UDP, TCP, and MPI (Message Passing Interface). We do some experiments in a testbed environment. Our results show that choosing one or the other consolidate compilers and MPI implementations, will not change the network performance results significantly, allowing users to select the tools that are more adequate to their needs. Also, our experiments of throughput and RTT (Round Trip Time) for the three protocols are valuable for HPC researchers since they give insight on how much an application can be affected at the networking level by choosing a convenient high level paradigm such as MPI, in contrast of working at a more primitive transport layer such as UDP and TCP.

Keywords: Performance Evaluation; Rocks Clusters; High Performance Computing; MPI; Benchmarks; IMB; Hpcbench.

1. Introduction

To help administrators and researchers that work in the field of HPC (High Performance Computing), a wide variety of tools are available for provisioning, configuration, management, and monitoring HPC clusters. The tools differ in complexity, costs, type of license, supported hardware, targeted operating systems, provisioning methods, scalability, supported job schedulers, additional tools such as compilers and libraries, etc. Well-known cluster managers include BCM[1] [1][2] (Bright Cluster Manager), Rocks Cluster[2] [3][4], and Stacki[3]. BCM is a commercial product from Bright Computing, with the largest part of the market for commercial cluster managers. Bright Computing provides support, training, certification, and consulting services. BCM lets customers deploy complete clusters over bare metal and manage them effectively, with cmsh (a command line manager) o cmgui (a GUI oriented manager). BCM has good support for hardware such as GPU (including the latest version of the NVIDIA CUDA toolkit), Myrinet [5] (a legacy high performance network), and InfiniBand [5][6]. Rocks Cluster [3][4] is an open source HPC cluster manager, used by academic, government, and commercial organizations. According to its register page, more than 2,000 Rocks Clusters have been installed around the world. Stacki is a fork of Rocks Cluster, maintained by StackIQ. Three versions of Stacki are available: (1) Stacki Community, (2) Stacki Professional, and (3) Stacki Enterprise. Stacki Community is the open source version of Stacki, and can be downloaded and installed on an HPC cluster for free. It has no support from StackIQ. Stacki Professional and Stacki Enterprise are the commercial versions of Stacki. There are enhanced versions of Stacki Community that include cluster validation, remote logfile streaming, system inventory, Puppet integration, a powerful GUI, etc.

Network is an essential part of a HPC cluster. Hence, performance evaluation must be done to validate the correct setup of a cluster. In this paper, we present several benchmarks to evaluate point-to-point network performance at different levels: UDP, TCP, and MPI [7][8] (Message Passing Interface). We use some of these measurement tools to evaluate the network of a Rocks Cluster in a controlled environment. One of the goals of our work is to study the impact of the selection of the development tools (compilers and MPI implementations) over the performance of point-to-point communications in UDP, TCP, and MPI. Also, we present the results of the evaluation of the throughput and the RTT (Round Trip Time) in our testbed, to quantify the overhead of using MPI (a well-know and accepted message passing library) in comparison to UDP or TCP.

[1] http://www.brightcomputing.com
[2] http://www.rocksclusters.org
[3] http://www.stacki.com

The rest of this paper is organized as follows: Related work is discussed in Section 2. Rocks Cluster is introduced in Section 3. A survey of actual benchmarking tools for point-to-point network evaluation is presented in Section 4. In Section 5, we briefly introduce our testbed. Our network performance evaluation is presented and discussed in Section 6. Finally in Section 7, we provide concluding remarks and directions for future work in this area.

2. Related Work

Some work on evaluating the network performance on national o international HPC clusters has been done. Quan [9] evaluated the network of ARCHER (Advanced Research Computing High End Resource), the latest UK national supercomputing service. This service was started in November 2013 and is expected to run for 5 years. Quan focused on the performance variations through different communication paths, in particular for bandwidth and latency, by using the HPC Challenge benchmarks [10] and the IMB [11] (Intel MPI Benchmarks) benchmarks. Other work is focused on HPC services offered by cloud providers. Hassan, Mohamed, and Sheta [12] evaluated the scalability and performance of High Performance Cloud Computing on Microsoft Azure Cloud infrastructure by using the IMB benchmarks [11] and the NAS Parallel Benchmarks[4] (NPB). Similar work is done in [13].

Saini et al [14] used the HPC Challenge [10] and the IMB [11] benchmarks to evaluate the combined performance of processor, memory subsystem, and interconnect fabric of five supercomputers (SGI Altix BX2, Cray X1, Cray Opteron Cluster, Dell Xeon Cluster, and NEC SX-8). The interesting point of this paper is that the five systems use five different networks (SGI NUMAlink4, Cray network, Myrinet, InfiniBand, and NEC IXS). In [15], Ismail, Hamid, Othman, Latip, and Sanwani presented the measurements of MPI point-to-point communications using benchmarking tools such as SKaMPI [16], IMB [11], and MPBench over HPC clusters for Gigabit Ethernet and InfiniBand. In [17], the same authors of [15] studied the performance of MPI collective communications (broadcasting, scattering, gathering, etc) over Gigabit Ethernet and InfiniBand in HPC clusters using the SKaMPI [16] benchmarks.

As seen in the previous mentioned work, most of the research in this area is focused on the evaluation of networking performance at the level of MPI. In our paper, the goal is to make an evaluation comparison of three communication environments (aka, UDP, TCP, and MPI) for HPC clusters, based on commodity hardware and an open source cluster manager (Rocks Cluster).

3. Rocks Clusters

Rocks Cluster [3][4] is a complete open-source HPC solution for x86 64-bit architecture, developed by UCSD (University of California at San Diego). It combines popular cluster management tools on top of CentOS, a widely used Linux distribution. Rocks Cluster allows users with little experience from a wide range of scientific fields (biology, chemistry, physics, mathematics, etc) to easily build computational clusters, which is one of the main goals of the project. Hence, thousands of researchers from around the world have used Rocks Cluster to deploy their own HPC cluster.

As shown in Figure 1, a Rocks Cluster is made of a front-end node, compute nodes (back-end nodes), and an Ethernet switch. The front-end node generally has two Ethernet NICs (e.g., eth0 and eth1). eth1 is used for public communications with the outside world, while eth0 is limited to private communications with compute nodes. It runs many services (NFS, NIS, DHCP, NAT, MySQL, HTTP, etc), and it is where users login in, develop and compile code, submit jobs, etc. This node also acts as a router for compute nodes by using NAT. In some implementation of Rocks Clusters, the front-end node is also the file server for the cluster, and has several hard disks organized in RAID (Redundant Array of Independent Disks). The compute nodes are the workhorse nodes, and generally have one Ethernet NIC (e.g., eth0), for communications with the front-end node and other compute nodes. That is, these nodes are not seen on the public Internet. They should have a small hard disk for storing the OS (CentOS), libraries, and administration tools. The basic idea of the Ethernet switch is to connect the front-end and compute nodes on the private network, for PXE [18] (Preboot Execution Environment) installation of compute nodes, administration, monitoring, and file sharing.

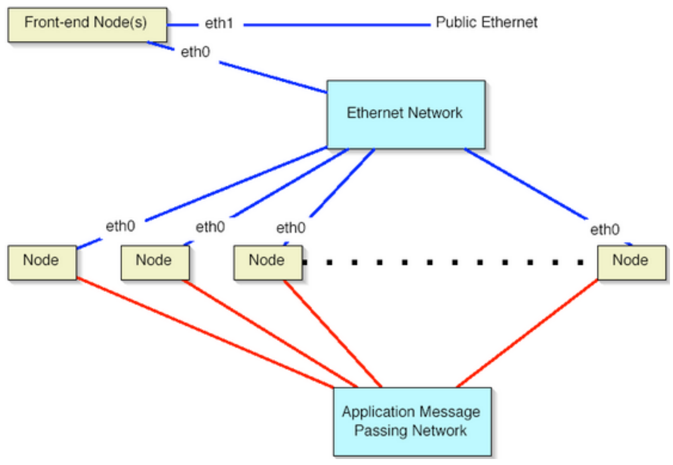

Figure 1: Basic Architecture of a Rocks Cluster

For high speed network between the compute nodes, many Rocks Clusters also offer a second private network, based on Myrinet [5] or InfiniBand [5][6], as shown in Figure 1.

The software stack of Rocks Clusters usually include: (1) Kickstart as the provisioning system that facilitates the installation and configuration of compute nodes, (2) Ganglia as a tool for monitoring and managing the cluster's status, configuration and resources, (3) MPICH and Open MPI to allow users to develop parallel applications based on the message passing paradigm, (4) SGE as a jobs scheduler, (5) Environment Modules for managing different libraries and compilers, and (6) NFS for sharing the file system. For a better performance, NFS is commonly replaced by Lustre[5], an open-source parallel distributed file system. Lustre achieves high-performance by striping across disks attached to multiple I/O

[4] http://www.nas.nasa.gov/publications/npb.html

[5] http://www.lustre.org

nodes. This improves performance since retrieving a file consists in getting chunks of it from a number of distributed disks where the file is striped across, and by sharing the network load across multiple connections on multiple nodes. Kickstart makes provisioning of Rock Clusters very simple. It requires that the compute nodes have support for PXE [18] for installation over the network. If it is not the case, it is necessary to burn the kernel/boot roll into a CD and boot these compute nodes using the CD for their installation.

The distribution of Rocks Cluster is done by rolls. A roll is a collection of software intended for a specific task. Open source rolls include: kernel/boot, base, os, Ganglia (cluster monitoring system from UCB), HPC (MPICH, Open MPI, PVM, etc), area51 (Tripwire for analyzing the integrity of files and chkrootkit for locally checking for signs of rootkits), web-server (Wordpress and MediaWiki), SGE (SUN Grid Engine), etc. Other rolls have been contributed, such as TORQUE (Terascale Open-source Resource and QUEue Manager), SLURM (Simple Linux Utility for Resource Management), CUDA (Compute Unified Device Architecture), and Lustre. TORQUE and SLURM are alternatives to the SGE job scheduler.

4. Benchmarks for Point-to-Point Network Evaluation in HPC Clusters

Many benchmarking tools have been proposed for network performance evaluation at the level of UDP and TCP. Some of the popular open source projects include Netperf, Iperf [19], and NetPIPE [20]. Netperf[6] is a benchmarking tool that can be used to measure various aspects of networking performance. Its primary focus is on bulk data transfer (TCP_STREAM, UDP_STREAM, etc) and request/response performance (TCP_RR and UDP_RR) using either TCP or UDP. It is designed around the basic client/server model. In the TCP_STREAM test, a constant bitrate of data is transferred from the client (netperf) to the server (netserver), and the actual throughput is reported as the result. It worth mentioning that the reported throughput is equal to the maximum throughput, since Netperf saturates the communication link. The UDP_STREAM test is similar to the TCP_STREAM test, except that UDP is used as the transport protocol rather than TCP. In the TCP_RR test, a fixed quantity of data is exchanged between the client (netperf) and the server (netserver) a number of times, and the benchmark reports the transaction rate which is the number of complete round-trip transactions per second. The UDP_RR is very much the same as the TCP_RR test, except that UDP is used rather than TCP. Iperf[7] [19] has a client/server functionality, based on bulk data transfer, either unidirectionally (from the client to the server) or bidirectionally. In the case of UDP, users can specify the read/write data size and the required bandwidth. Iperf will report the actual throughput, the jitter, and the packet lost. In the case of TCP, users can specify the read/write data size. Iperf will saturate the TCP connection and report the maximum throughput for this data length. NetPIPE[8] [20] (Network Protocol Independent Performance Evaluator) is based on the client/server model, and focused in request/ response performance. It is independent of the lower protocols, that is, NetPIPE can be used at the level of TCP, at the message passing level (MPI and PVM), or at the native communications level (Myrinet and InfiniBand). For each experiment, users must specify a lower and upper bound for the size of the messages that will be exchanged by the client and the server. NetPIPE tests network performance by bouncing a number of messages at each block size, starting from the lower bound of the message sizes. The message size is incremented until the upper bound on the message size is reached. Message sizes are incremented at regular intervals. For each message size, NetPIPE reports the actual throughput and the OWD (One Way Delay).

At the message passing layer (aka, MPI), many benchmarking tools have been proposed by the community to test point-to-point communications (e.g., SKaMPI[9] [16], MPBench[10], Mpptest[11], and IMB). IMB [11] (Intel MPI Benchmarks) is a set of open source benchmarks (IMB-MPI1, IMB-EXT, IMB-IO, IMB-NBC, and IMB-RMA) from Intel to fully characterizes the performance of a cluster system, including node performance, network latency and throughput, and efficiency of the MPI implementation. IMB-MPI1 is limited to MPI-1 functions. It is divided in 3 categories: (1) simple transfer benchmarks, (2) parallel transfer benchmarks, and (3) collective benchmarks. The PingPong benchmark is one of the simple transfer benchmarks, and it uses the request/response scheme, that is, a message is bounced several times between 2 MPI processes (rank 0 and rank 1). The benchmark reports the OWD and the actual throughput. The PingPing version is similar to PingPong, except that PingPing is a bidirectional version of PingPong. The parallel transfer benchmarks are for parallel communications, involving several processes (more than two processes). The collective benchmarks evaluate collective functionality, such as broadcasting, reductions, scattering, and gathering. IMB-EXT and IMB-IO evaluate MPI-2 functionality. IMB-EXT is for one-sided communication benchmarks, while IMB-IO is focused on I/O (Input/Output). IMB-NBC and IMB-RMA cover MPI-3 functions. IMB-NBC is a set of non-blocking collective benchmarks that provides measurements of the computation/communication overlap and of the pure communication time. IMB-RMA are Remote Memory Access (RMA) benchmarks that use passive target communication to measure one-sided communication.

Hpcbench[12] [21][22] is a network benchmarking tool focused on HPC environments. It is comprised of three independent sets of benchmarks measuring UDP, TCP and MPI communications. Similar to many other tools, the UDP and TCP tests are based on the client/server paradigm. Basically, Hpcbench allows users to evaluate the maximum throughput and the RTT, in any of the three communication standards, for a message with a length that users must specified. For throughput measurements, Hpcbend floods the network with messages of the accorded size, and reports the observed

[6] http://www.netperf.org
[7] http://iperf2.sourceforge.net
[8] http://bitspjoule.org/netpipe

[9] http://liinwww.ira.uka.de/~skampi
[10] http://icl.cs.utk.edu/llcbench/mpbench.html
[11] http://www.mcs.anl.gov/research/projects/mpi/mpptest
[12] http://hpcbench.sourceforge.net

throughput, which also correspond to the maximum throughput for this message size. For the RTT measurements, the two processes of Hpcbend bounce a message of the specified size, and report the average RTT by dividing the total time for the experiment, by the number of transactions. For UDP and TCP, users can set the client and server buffer size, the read/write size, and the size of the message. For throughput tests, unidirectional and bidirectional flows are allowed, however the RTT evaluation is limited to unidirectional flows.

5. Testbed for our Experiments

For our experiments, the testbed was based on Rocks Cluster v6.1. As shown in Figure 2, the cluster was made of a head node (front-end node), four compute nodes, a file server, and two Ethernet switches (SW1 and SW2). The Ethernet switches were Dell PowerConnect 6224 with 24 10/100/1000BASE-T Gigabit Ethernet ports. An extension module was added to the switches to get 2 additional 10 Gigabit Ethernet ports.

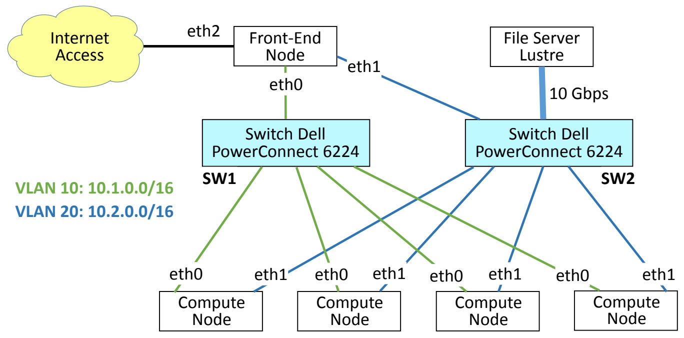

Figure 2: Testbed for our Experiments

Two VLANs were defined: VLAN 10 for the communication between the front-end node and the compute nodes. VLAN 20 for the communication between the nodes and the file server. The file server was running Lustre[13], an open-source parallel distributed file system, and connected to SW2 through a 10 Gigabit Ethernet link. Through SW1, we connected the compute nodes to the front-end for administration, monitoring, and submission of jobs, using Gigabit Ethernet links. Through SW2, we connected the nodes to the Lustre file server, using Gigabit Ethernet links. The compute nodes were identical (Dell PowerEdge R410 rack mounted - 1U), with the following characteristics:

- Processors: 2 Quad-core Intel Xeon E5520 2.27 GHz
- RAM: 32 GB (DDR3) – 8 x 4 GB DIMM
- NIC: Embedded dual-port Broadcom NetXtreme II BCM5716 Gigabit Ethernet
- Hard Disk: 1 internal hard disk (250 GB) for a local installation of the operating system (Rocks Cluster v6.1)
- Remote Management: IPMI2.0.

6. Results and Analytical Comparision

In this section, we present and report the results of our experiments. We start comparing the variation of the point-to-point network performance when using different compilation tools and MPI implementations. Then, we study the throughput and RTT of the three chosen communication systems (UDP, TCP, and MPI). It is worth mentioning that each experiment was run several times and we report the average results.

6.1 Influence of Compilers and MPI Implementations

Since they are different C/C++ compilers (GNU gcc/g++, PGI pgcc/pgc++, etc), different implementations of MPI (MPICH, Open MPI, etc), the goal of this first experiment is to study how the selection of the tools can affect the performance of point-to-point communications. For this experiment, we chose the PingPong and PingPing benchmarks of IMB [11].

We had some hard time running the benchmarks with Open MPI. By default, Open MPI automatically chooses the fastest available network. Even if we did not have an InfiniBand network in our testbed, Rocks Cluster will automatically load the InfiniBand drivers in the nodes, resulting as the first choice network of Open MPI. We could solve this issue by forcing Open MPI to use the Gigabit Ethernet NICs, as shown in Figure 3.

```
mpirun  --mca btl tcp,self  --mca btl_tcp_if_include eth0    \
     -np 2  -host compute-0-2,compute-0-3                     \
     ./IMB-MPI1 PingPong
```

Figure 3: Command to Run PingPong with Open MPI

We used two C/C++ compilers: (1) GNU gcc/g++ v4.9.2 and (2) PGI pgcc/pgc++ v14.9. PGI (Portland Group Inc.) is a company that makes well-known commercial Fortran, C, and C++ compilers for HPC systems. PGI was acquired by NVIDIA Corporation in July 2013. We chose two MPI implementations: (1) MPICH v3.2 and Open MPI v1.10.2. The results of running the PingPong benchmark between two compute nodes in our testbed are shown in Table I and Table II. We varied the length of data to be exchanged from 512 to 4,194,304 bytes. Table I shows the RTT in microseconds, while Table II reports the throughput in Mbps. Note that the PingPong benchmark gives results in Mbyte/s, and we multiplied them by 8 to get the results in Mbps.

Table I: RTT in Microseconds for the PingPong Benchmark

Size (bytes)	gcc 4.9.2	PGI 14.9	gcc 4.9.2	PGI 14.9
	MPICH 3.2		Open MPI 1.10.2	
512	43.49	44.26	48.26	48.02
2,048	77.89	82.66	75.10	75.10
8,192	151.16	152.03	152.07	151.83
32,768	389.45	382.47	394.52	390.63
65,536	674.92	665.89	774.84	763.21
131,072	1,290.16	1,280.65	1,325.49	1,319.62
262,144	2,430.13	2,412.70	2,434.68	2,426.36
524,288	4,674.08	4,652.48	4,678.53	4,672.13
1,048,576	9,154.45	9,119.86	9,137.03	9,135.45
2,097,152	18,099.22	18,028.25	18,054.32	18,054.35
4,194,304	35,980.06	35,840.50	35,859.50	35,862.71

As we can see from these experiments, the results are very similar for the four possibilities. Hence, our recommendation for programmers is to use the tools that better feet their needs. Also, it is worth to clarify that we obtain greater throughput for large data sizes. For small data sizes, the overhead of MPI is significant compared to the user data sent, resulting in low throughput at the level of MPI. We also want to point-out that

[13] http://www.lustre.org

Table II does not reflect the maximum experimental throughput for the specified data size, since the PingPong benchmark does not saturate the communication link.

Table II: Throughput in Mbps for the PingPong Benchmark

Size (byte)	gcc 4.9.2	PGI 14.9	gcc 4.9.2	PGI 14.9
	MPICH 3.2		Open MPI 1.10.2	
512	89.84	88.24	80.96	81.36
2,048	200.56	189.04	208.08	208.08
8,192	413.44	411.12	410.96	411.60
32,768	641.92	653.68	633.68	640.00
65,536	740.80	750.88	645.28	655.12
131,072	775.12	780.88	754.40	757.76
262,144	823.04	828.96	821.44	824.24
524,288	855.76	859.76	854.96	856.16
1,048,576	873.92	877.20	875.52	875.68
2,097,152	884.00	887.52	886.24	886.24
4,194,304	889.36	892.88	892.40	892.32

The results of running the PingPing benchmark (the bidirectional version of the PingPong benchmark) between two compute nodes in our testbed are shown in Table III and Table IV. We varied the length of data to be exchanged from 512 to 4,194,304 bytes. Table III shows the RTT in microseconds, while Table IV reports the throughput in Mbps. These experiments confirm that we did not get significant differences when varying the tools (compilers and MPI implementations). However, although the network is full-duplex, we can see that using bidirectional communications at the same time, resulted in a small degradation of the RTT and the throughput.

Table III: RTT in Microseconds for the PingPing Benchmark

Size (byte)	gcc 4.9.2	PGI 14.9	gcc 4.9.2	PGI 14.9
	MPICH 3.2		Open MPI 1.10.2	
512	43.93	43.45	48.78	51.14
2,048	95.23	94.53	105.38	109.55
8,192	226.66	228.32	244.57	248.07
32,768	390.09	387.14	393.89	396.32
65,536	678.17	680.90	762.89	769.96
131,072	1,305.77	1,306.50	1,335.84	1,340.16
262,144	2,504.70	2,540.38	2,553.84	2,585.97
524,288	4,955.01	4,993.35	4,989.02	5,002.70
1,048,576	9,951.65	10,045.67	9,966.45	10,063.92
2,097,152	20,030.94	20,097.24	19,978.20	20,202.76
4,194,304	39,444.80	39,758.40	39,894.39	40,054.39

Table IV: Throughput in Mbps for the PingPing Benchmark

Size (byte)	gcc 4.9.2	PGI 14.9	gcc 4.9.2	PGI 14.9
	MPICH 3.2		Open MPI 1.10.2	
512	88.96	89.92	80.08	76.40
2,048	164.08	165.28	148.24	142.64
8,192	275.76	273.76	255.52	251.92
32,768	640.88	645.76	634.72	630.80
65,536	737.28	734.32	655.44	649.36
131,072	765.84	765.44	748.56	746.16
262,144	798.48	787.28	783.12	773.44
524,288	798.80	796.16	801.76	785.44
1,048,576	811.28	796.40	802.72	794.88
2,097,152	811.92	801.04	800.88	792.00
4,194,304	815.28	804.88	802.08	798.88

6.2 *Throughput Experiments with a Unidirectional Flow*

The objective of these experiments is to get the maximum throughput that can be obtained between two processes, running in different compute nodes. We used Hpcbench [21][22] (UDP, TCP, and MPI), compiled with GNU gcc v4.9.2 and MPICH v3.2.

We run the UDP experiments as specified in Figure 4. Line 01 is the command for the server. Line 02 is the command for the client, where <size> represents the desired data size for UDP. Fragmentation of IP will be required for data sizes greater than 1,472 bytes.

```
01: ./udpserver  -v
02: ./udptest   -h <remoteIPAddress>   -l <size>
```

Figure 4: Commands for UDP in Throughput Experiments

We run the TCP experiments as specified in Figure 5. Line 01 is the command for the server. Line 02 is the command for the client, where <size> represents the desired data size for TCP. Option –N is to disable Nagle algorithm [23], and –n is for the usage of non-blocking communication functions.

```
01: ./tcpserver  -v
02: ./tcptest   -h <remoteIPAddress>   -N  -n  -m <size>
```

Figure 5: Commands for TCP in Throughput Experiments

We run the MPI experiments as specified in Figure 6. In file <hosteFile>, we put the IP address of the two nodes, and <size> represents the desired size of data to be sent with MPI. Option –n is for the usage of non-blocking communication functions.

```
mpirun  -np 2  --hostfile <hostFile>  mpitest  -n  -m <size>
```

Figure 6: Commands for MPI in Throughput Experiments

Figure 7 shows the obtained results. We can observe that UDP has the best throughput, and MPI has the lowest one. For the three protocols, the throughput increases with the size of data, and reaches an upper bound (close to 1 Gbps) for large values. UDP and TCP reach their upper bound around 1,024 bytes, while MPI reaches it around 4,096 bytes.

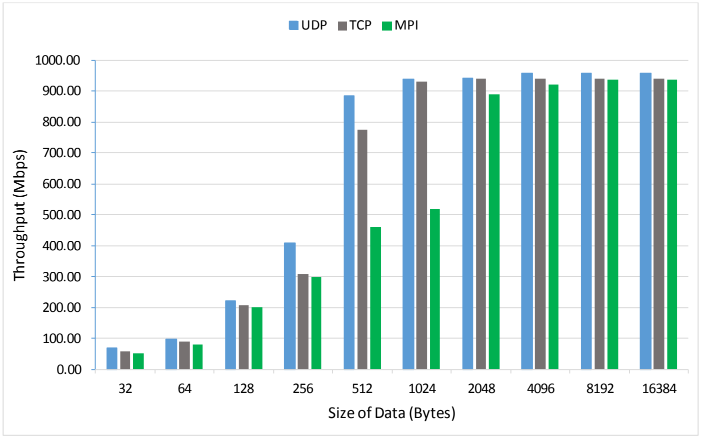

Figure 7: Throughput Experiments – Unidirectional Flow

6.3 *Throughput Experiments with a Bidirectional Flow*

The experiments realized in this section are similar to the ones of Section 6.2, except that they are two flows between the two processes (bidirectional experiments). As we can infer from Figure 8, the throughput increases according to the size of the data and reaches an upper bound for large values (close to 1 Gbps). It is noticeable that the rise of the bidirectional experiments (see Figure 8) is slower than the rise of the unidirectional experiments (see Figure 7). UDP reaches its upper bound for the throughput around 2,048 bytes, TCP does it around 4,096 bytes, while MPI does it around 262,144 bytes (not shown in Figure 8).

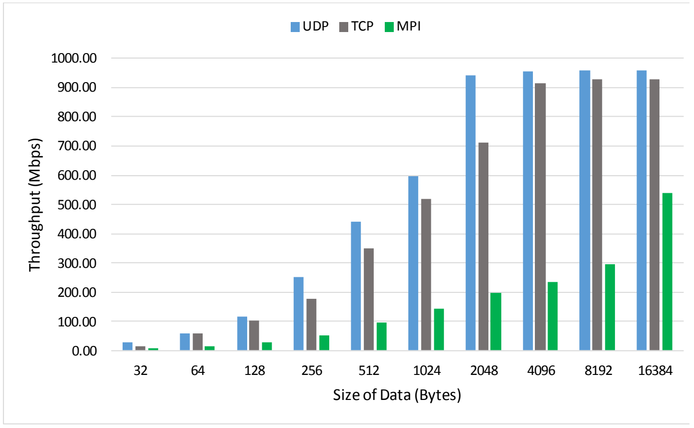

Figure 8: Throughput Experiments – Bidirectional Flow

6.4 Round-Trip-Time Experiments

The objective of these experiments is to get the RTT that can be obtained between two processes, running in different compute nodes. We used Hpcbench (UDP, TCP, and MPI), compiled with GNU gcc v4.9.2 and MPICH v3.2.

We run the UDP experiments as specified in Figure 9. Line 01 is the command for the server. Line 02 is the command for the client, where <size> represents the desired data size for UDP.

```
01: ./udpserver  -v
02: ./udptest   -h <remoteIPAddress>   -A <size>
```

Figure 9: Commands for UDP in RTT Experiments

We run the TCP experiments as specified in Figure 10. Line 01 is the command for the server. Line 02 is the command for the client, where <size> represents the desired data size for TCP. Option –N is to disable Nagle algorithm [23], and –n is for the usage of non-blocking communication functions.

```
01: ./tcpserver  -v
02: ./tcptest   -h <remoteIPAddress>   -N  -n  -A <size>
```

Figure 10: Commands for TCP in RTT Experiments

We run the MPI experiments as specified in Figure 11. In file <hosteFile>, we put the IP address of the two nodes, and <size> represents the desired size of data to be sent with MPI. Option –n is for the usage of non-blocking communication functions.

```
mpirun  -np 2  --hostfile <hostFile>  mpitest  -n  -A <size>
```

Figure 11: Commands for MPI in RTT Experiments

In Figure 12, we plotted the results obtained with the RTT experiments, for message sizes varying from 32 to 16,384 bytes. The lowest RTT is achieved by UDP, and the highest by MPI. However, the differences are small and seem to indicate that UDP will be the best choice for application that need low latency, for small data sizes. However, the differences between the three communication methods (UDP, TCP, and MPI) over the overall latency performance will not be much significant when using large data sizes.

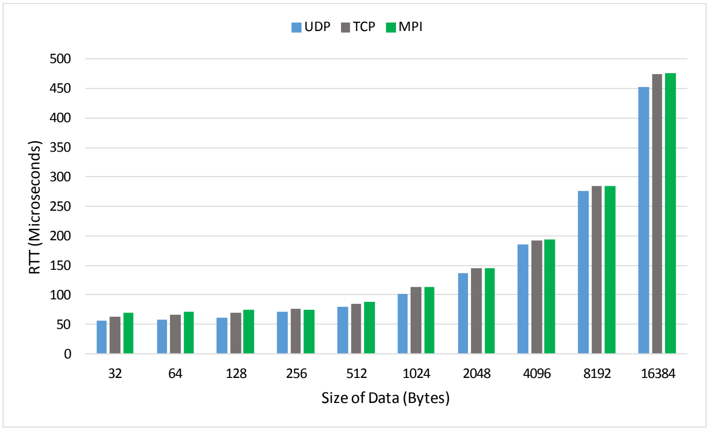

Figure 12: Round-Trip-Time Experiments

7. Conclusion and Future Work

In this paper, we compiled information about point-to-point network benchmarking tools and presented their limitations and strengths. We also specified how do they differ from each other, and how can they be used to evaluate point-to-point communications in HPC clusters. We run some of these benchmarks in a controller testbed based on commodity hardware and Rocks Cluster, when varying the compilers and MPI implementations. Our experiments showed that the usage of different compilation tools (aka, GNU gcc/g++ and PGI pgcc/pgc++) gave results with similar point-to-point network performance. Furthermore, doing experiments with MPICH and Open MPI also took us to comparable results.

We also did other tests to evaluate the throughput and the RTT of UDP, TCP, and MPI. For small message sizes, the throughput at the user application is small and far from the bandwidth of the network (1 Gbps), since the three communication methods add a significant overhead compared to the user data. However, as the size of the data increases, the throughput also increases and reaches an upper bound close to the bandwidth (1 Gbps), since the user data become more predominant than the overhead introduced by the communication environment. Moreover, these experiments also give insight of the relative different overheads produced by the three communication methods. The RTT experiments showed small differences between UDP, TCP, and MPI. If these differences will not affect the latency in applications that use data of large sizes, it can be more noticeable in applications with small data sizes, since these differences cannot be neglected in comparison to the nominal values.

InfiniBand is a recent computer-networking communication standard used in high-performance computing, and promoted by the IBTA (InfiniBand Trade Association). It features very high throughput and very low latency. Currently, InfiniBand offers 5 data rates (SDR=2.5 Gbps, DDR=5.0 Gbps, QDR=10.0 Gbps, FDR=14.0625 Gbps, and EDR=25.78125 Gbps), and for higher speeds, links can be bonded together (1x, 4x, 8x, and 12x). As future work, we plan to make a comparative performance evaluation between InfiniBand, Gigabit Ethernet, and 10 Gigabit Ethernet. Also, we want to further investigate BCM, and realize an analytical comparison with Rocks Cluster.

ACKNOWLEDGMENT

We want to thank the NSF (National Science Foundation) which partially supported this research under grant number 1010094 through the EPSCoR Track 2 program.

REFERENCES

[1] Bright Computing, *insideHPC Five Essential Strategies for Successful HPC Clusters*, April 2014.

[2] Bright Computing, *Breaking the Rules: Bright Cluster Manager with Cisco UCS, a Complete HPC Solution*, November 2014.

[3] J. Sloan, *High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI*, 1st edition, O'Reilly, November 2004.

[4] M. Trangoni and M. Cabral, *A Comparison of Provisioning Systems for Beowulf Clusters*, XVIII Congreso Argentino de Ciencias de la Computación (CACIC 2012), Bahia Blanca, Argentina, October 2012.

[5] J. Liu, B. Chandrasekaran, J. Wu, W. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff, and D. K. Panda, *Performance Comparison of MPI Implementations over InfiniBand, Myrinet and Quadrics*, in proceeddings of the 2003 ACM/IEEE Supercomputing Conference, Phoenix, Arizona, USA, November 2003.

[6] B. Madani and R. Al-Shaikh, *Performance Benchmark and MPI Evaluation Using Westmere-based InfiniBand HPC Cluster*, International Journal of Simulation -- Systems, Science & Technology, Vol. 12, Nro. 1, pp. 20-26, February 2011.

[7] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard, Version 3.1*, High Performance Computing Center Stuttgart, June 2015.

[8] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, 3d edition, MIT Press, November 2014.

[9] C. Quan, *Network Interconnect Evaluation on ARCHER*, Master Thesis, The University of Edinburgh, UK, August 2014.

[10] P. Luszczek, D. Bailey, J. Dongarra, J. Kepner, R. Lucas, R. Rabenseifner, and D. Takahashi, *The HPC Challenge (HPCC) Benchmark Suite*, Tutorial of the 2006 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'06), Tampa, Florida, USA, November 2006.

[11] Intel, *Intel MPI Benchmarks - Users Guide and Methodology Description*, 2015.

[12] H. Hassan, S. Mohamed, and W. Sheta, *Scalability and Communication Performance of HPC on Azure Cloud*, Egyptian Informatics Journal, In press.

[13] R. Expósito, G. Taboada, S. Ramos, J. Touriño, and R. Doallo, *Evaluation of Messaging Middleware for High Performance Cloud Computing*, Personal and Ubiquitous Computing, Vol. 17, Nro. 8, pp. 1709-1719, December 2013.

[14] S. Saini, R. Ciotti, B. Gunney, T. Spelce, A. Koniges, D. Dossa, P. Adamidis, R. Rabenseifner, S. Tiyyagura, and M. Mueller, *Performance Evaluation of Supercomputers using HPCC and IMB Benchmarks*, Journal of Computer and System Sciences, Vol. 74, Nro. 6, pp. 965-982, September 2008.

[15] R. Ismail, N. Hamid, M. Othman, R. Latip, and M. Sanwani, *Point-to-Point Communication on Gigabit Ethernet and InfiniBand Networks*, in proceedings of the International Conference on Informatics Engineering and Information Science, Kuala Lumpur, November 2011.

[16] R. Reussner, P. Sanders, and J. Larsson Träff, *SKaMPI: a Comprehensive Benchmark for Public Benchmarking of MPI*, Scientific Programming, Vol. 10, Nro. 1, pp. 55-65, 2002.

[17] R. Ismail, N. Hamid, M. Othman, and R. Latip, *Performance Analysis of Message Passige Interface Collective Communication on Intel Xeon Quad-Core Gigabit Ethernet and InfiniBand Clusters*, Journal of Computer Science, Vol. 9, Nro.4, pp. 455-462, 2013.

[18] Intel Corporation and SystemSoft, *Preboot Execution Environment (PXE) Specification, Version 2.1*, September 1999.

[19] A. Tirumala, L. Cottrell, and T. Dunigan, *Measuring End-to-End Bandwidth with Iperf using Web100*, in proceeding of the 2003 Passive and Active Monitoring Workshop (PAM 2003), San Diego, California, USA, April 2003.

[20] Q. Snell, A. Mikler, and J. Gustafson, *NetPIPE: A Network Protocol Independent Performance Evaluator*, in proceedings of the IASTED International Conference on Intelligent Information Management and Systems, Washington, DC, USA, June 1996.

[21] B. Huang, M. Bauer, and M. Katchabaw, *Network Performance in High Performance Linux Clusters*, in proceedings of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2005), Las Vegas, Nevada, USA, June 2005.

[22] B. Huang, M. Bauer, and M. Katchabaw, *Hpcbench – A Linux-Based Network Benchmark for High Performance Networks*, in proceedings of the 19th International Symposium on High Performance Computing Systems and Applications (HPCS 2005), Guelph, Ontario, Canada, May 2005.

[23] J. Nagle, *Congestion Control in IP/TCP Internetworks*, RFC 896, January 1984.