Understanding the Significance of Network Performance in End Applications: A Case Study with EtherFabric and InfiniBand

K. VAIDYANATHAN, S. BHAGVAT, P. BALAJI AND D. K. PANDA

Technical Report
Ohio State University (OSU-CISRC-2/06-TR19)

Understanding the Significance of Network Performance in End Applications: A Case Study with EtherFabric and InfiniBand*†

K. Vaidyanathan S. Bhagvat P. Balaji D. K. Panda

Dept. of Computer Science and Engineering,
Ohio State University,
{vaidyana, bhagvat, balaji, panda}@cse.ohio-state.edu

Abstract

Due to the low speeds of earlier generation networks such as Fast Ethernet, network communication was considered to be one of the primary bottlenecks in cluster computing. Accordingly, researchers used a number of techniques to hide the communication overheads in networks. In order to alleviate this problem, several researchers and industries proposed and implemented faster networks such as Gigabit Ethernet (GigE). This trend soon led to the development of even higher speed networks such as 10-Gigabit Ethernet (10GigE) and InfiniBand (IBA). Today, industries are taking the next step in high-speed networking with multi ten-gigabit networks such as the Mellanox 20-Gigabit IBA DDR adapters, IBM 30-Gigabit IBA adapters, etc. Many of these innovations emerged due to the underlying assumption that the network is still one of the primary bottlenecks in cluster computing. However, whether this assumption is in fact a reality for current environments is yet to be understood. In this paper, we take a step back to understand if such multi-gigabit network speeds justify the cost of these newer networks with respect to the performance gains they provide for end applications. Specifically, we perform comparative analysis with conventional 1-GigE adapters, 2-GigE adapters (using two 1-GigE ports in adapters such as EtherFabric) and 8-Gigabit network adapters such as IBA[1] for end applications in two broad domains: (i) message passing interface (MPI) based scientific applications such as GROMACS and the NAS parallel benchmarks and (ii) data-intensive applications relying on parallel file-systems (e.g., PVFS), such as MPI-Blast. Our experimental results show that for many of the end-applications that we studied, EtherFabric was able to achieve 70-90% of the performance gains achievable by IBA.

*This research is funded in part by DOE Grant #DE-FC02-01ER25506, NSF grants #CCR-0204429 and #CCR-0311542 and technical and equipment support from Level-5 Networks, Mellanox Corporation and Intel

†We would like to thank Heshan Lin, Jeremy Archuleta and Dr. Wu-chun Feng for providing us with the several significant details about the MPI-Blast application and helping us tremendously in our evaluations.

[1]Though several people believe that the IBA (SDR) is a 10-gigabit network, it only provides a signaling rate of 10-gigabit per second and uses a 8b/10b encoding; thus the actual theoretical peak data rate it can achieve is only 8Gbps.

Further, due to their low-cost and compatibility with the existing TCP/IP/Ethernet infrastructure (include copper port based traditional GigE switches), such adapters seem to hold a lot of promise for many of the end-applications.

Keywords: High-speed Networks, InfiniBand, EtherFabric

1 Introduction

Clusters consisting of commodity off-the-shelf (COTS) components have become the predominant architecture for solving today's grand challenge problems in the fields of nuclear research, life sciences and engineering. Cluster systems are now present in all aspects of high-end computing, due to the increasing performance of commodity processors, memory and networking technologies. A typical cluster system consists of several hardware and software components integrated with a common network.

Due to the low speeds of earlier generation networks such as Fast Ethernet [27], network communication was considered to be one of the primary bottlenecks in cluster computing. Accordingly, researchers used a number of approaches, including various caching techniques [14, 18], prefetching algorithms [35], etc., to hide communication overheads. In order to alleviate this problem, several researchers and industries proposed and implemented faster networks such as Gigabit Ethernet (GigE) [17]. This trend soon led to the development of even higher speed networks such as 10-Gigabit Ethernet (10GigE) [21, 16, 15] and InfiniBand (IBA) [2]. Today, several industries are taking the next step in high-speed networking with multi ten-gigabit networks such as the Mellanox 20-Gigabit IBA DDR adapters [4], IBM 30-Gigabit IBA adapters [1], etc.

Many of these innovations emerged due to the underlying assumption that the network is one of the primary bottlenecks for end applications in cluster computing. However, in reality, is this assumption still true? Or in our quest to alleviate the network bottleneck, have we overshot the requirements of end applications and created faster-than-needed network interconnects?

Figure 1 shows the impact of increasing network speed on an application's communication time. If we consider x as the current speed of the network and C as the total data to be transferred,

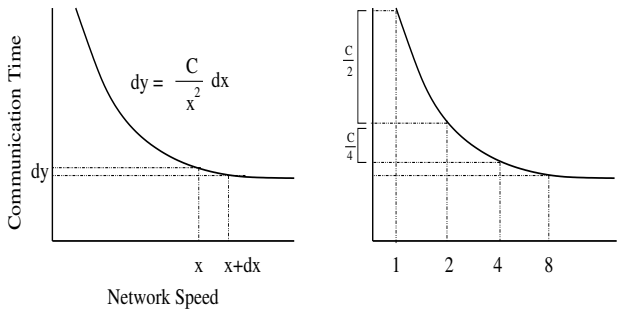

Figure 1: Communication Time with increasing network speed

a first-level approximation of the total time spent in communication (y) is given by Equation 1.

$$y = \frac{C}{x} \tag{1}$$

If we increase this network speed by dx, the total time spent in communication will be $\frac{C}{x+dx}$. Hence, the improvement in communication time (dy) is given by $\frac{C}{x} - \frac{C}{x+dx}$, which simplifies to,

$$dy = \frac{C}{x^2}dx \tag{2}$$

As indicated in equation 2, the decrease in communication time reduces *exponentially* as we increase the network speed, i.e., a unit increase in network speed at 1Gbps provides a four times larger benefit as compared to a unit increase in network speed at 2Gbps!

However, the first-level approximation given in Equation 2 does not completely capture the characteristics of all networks and applications. For example, (i) high-speed networks not only provide raw performance but also offer several hardware features such as one-sided operations which can potentially increase the application performance; thus the gain high-speed networks provide could be greater than what equation 2 indicates and (ii) an increase in network performance only impacts the communication portion of end applications; thus for applications which perform both computation and communication, the actual amount of gain in the overall execution time of the application could be even lesser than what equation 2 indicates. Based on these two conflicting examples, the actual performance benefits achievable for end applications by increasing network

speeds is not completely clear.

In this paper, we use an experimental approach to address this issue. Specifically, we perform comparative analysis with conventional 1-GigE adapters, 2-GigE adapters (using two 1-GigE ports in adapters such as EtherFabric) and 8-Gigabit network adapters such as IBA for end applications in two broad domains: (i) message passing interface (MPI) based scientific applications such as GROMACS [19] and the NAS parallel benchmarks [7] and (ii) data-intensive applications relying on parallel file-systems (e.g., PVFS), such as MPI-Blast [5].

Our experimental results demonstrate that though at a micro-benchmark level, IBA is close to four times faster than the 2-Gigabit EtherFabric (EF) network, this difference is not reflected in the applications that we studied. We notice that for applications which rely on small or medium message transfers, EF can provide 70-90% of performance gain achievable by IBA. For applications which rely on large message transfers, EF can provide up to 55% of the performance gain achievable by IBA. Due to the low-cost of EtherFabric adapters and its compatibility with the existing TCP/IP/Ethernet infrastructure (include copper port based traditional GigE switches), such adapters seem to hold a lot of promise for many of the end-applications.

The rest of the paper is organized as follows: we present an overview of IBA and EF in the Section 2. Section 3 describes the message passing interface (MPI) programming model and the parallel virtual file-system (PVFS) middleware layer used in this paper. Detailed evaluation using MPI and PVFS over the IBA and EF networks is discussed in Sections 6 and 7. Section 8 discusses related work and concluding remarks are presented in Section 9.

2 Overview of the Networking Stacks

In this section, we provide an overview of the two networks used in this paper, namely EtherFabric (EF) and InfiniBand (IBA).

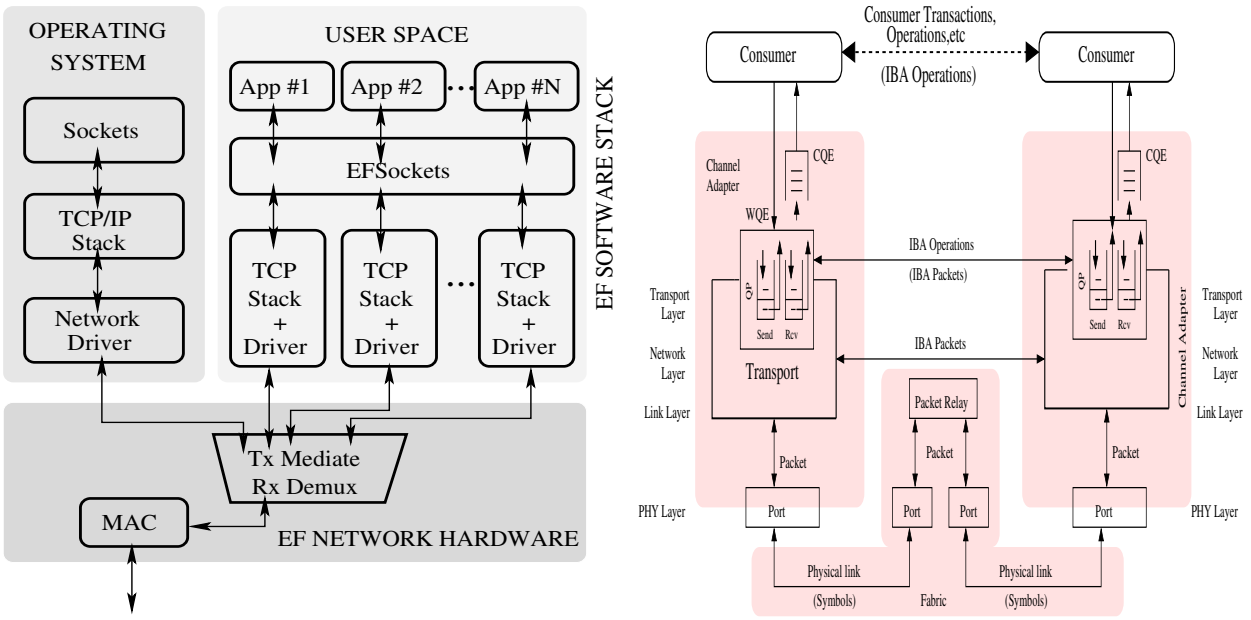

Figure 2: Network Architecture (a) EtherFabric (b) InfiniBand

2.1 EtherFabric Network and Software Stack

EtherFabric (EF) is a recently introduced network interconnect by Level-5 Networks [3]. EF is a dual port 1-GigE network adapter and supports striping of data across both the ports, achieving an aggregate throughput of 2Gbps in each direction. As illustrated in Figure 2a, the EF network consists of two components: (i) the EF network hardware and (ii) the EF software stack.

EF Network Hardware: The EF network adapter hardware supports basic offloading of certain aspects of protocol processing, including connection-specific demultiplexing of incoming data, etc. Similar to other high-speed network adapters, EF is designed to allow multiple applications to access it directly (without the operating system intervention) and simultaneously (using multiple doorbells implemented on hardware). The hardware design relies on a technique called *virtualized interface* to establish a direct data path between each application and the network hardware.

EF Software Stack: The EF software stack runs at the user-level and is not affected by host overheads such as kernel context switches. The stack utilizes the hardware connection-specific demultiplexing engine to allow applications to avoid host memory copies. The main novelty of the

EF software stack lies in its usage of user-level TCP/IP drivers to allow clusters connected with EF maintain wire-protocol compatibility with the existing TCP/IP/Ethernet infrastructure. Further, the EF software stack also provides a user-level sockets implementation, termed as *EFSockets*, to take advantage of the user-level TCP/IP drivers and other features of the network. This allows traditional sockets-based applications to be directly and transparently deployed on clusters connected with EF.

2.2 InfiniBand Architecture

InfiniBand Architecture (IBA) (Figure 2b) is an industry standard that defines a System Area Network (SAN) to design clusters offering low latency and high bandwidth. A typical IBA cluster consists of switched serial links for interconnecting both the processing nodes and the I/O nodes. The IBA specification defines a communication and management infrastructure for both inter-processor communication as well as inter and intra node I/O. IBA also defines several network hardware supported features including RDMA, multicast, QoS, etc.

IBA interacts with the applications using a thin layer known as the verbs interface (e.g., VAPI). This verbs interface is considered the native communication mechanism for IBA and extracts the best performance of the network. In order to maintain compatibility with existing applications, several researchers have implemented various standard programming models (e.g., MPI, Sockets) and communication middleware (e.g., PVFS) over IBA using this verbs interface. In Section 3, we describe the approaches used to implement these programming models over the verbs interface.

3 Programming Models and Middleware

Traditional programming models and middleware layers used TCP/IP sockets for communication. As mentioned in Section 2, both EF and IBA provide efficient support for allowing upper layers (e.g., programming models and middleware layers) to take advantage of the high performance offered by the networks. Specifically, the EF network exposes the EFSockets interface and IBA

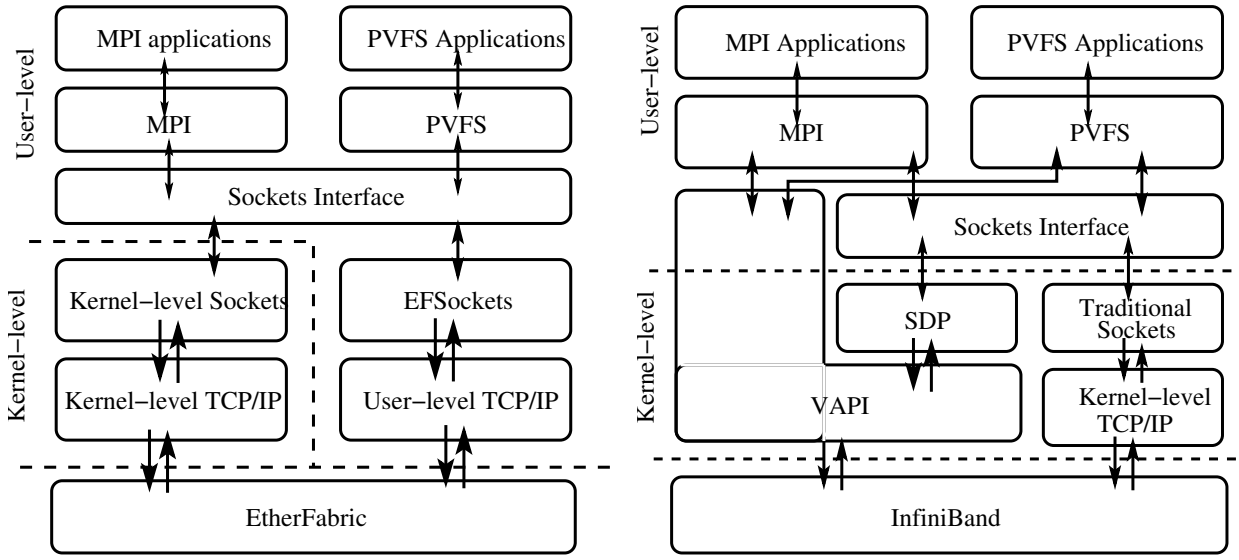

Figure 3: MPI and PVFS on: (a) EtherFabric (b) InfiniBand

exposes the verbs interface to upper layers (these are considered to be the native protocol layers exposed by these networks); several researchers have proposed a number of approaches to design and implement other upper layers on top of these native protocol layers with high efficiency.

In this section, we describe some of the common programming models and middleware supported on EF and IBA. Specifically, we describe the Message Passing Interface (MPI) programming model and a file-system middleware known as the Parallel Virtual File System (PVFS).

3.1 Message-Passing Programming Model

The message-passing programming model is based on the distributed memory model with explicit communication between processes. MPI is a message passing interface that, together with the protocol, is used for communicating between processes. It is designed to be a portable, efficient and flexible standard for communication among nodes and for running parallel applications.

For IBA(Figure 3b), MPI can either be implemented directly over the verbs interface (e.g., MVAPICH2 [29, 20]), or over the sockets interface (e.g., LAM-MPI [12]). Further, to support the sockets interface, we can either use the traditional sockets implementation present in the kernel or a high-performance implementation of sockets such as the Sockets Direct Protocol (SDP) [8, 10]. Amongst these implementations, MPI implemented over the native verbs interface provides the

best performance since it can directly take advantage of the features provided by the underlying network instead of being abstracted by another interface (e.g., SDP). Thus, for all MPI-level evaluation we use the MVAPICH2 implementation of MPI.

In the case of EF (Figure 3a), the native layer itself exposes the sockets interface. Thus, only a sockets-based implementation of MPI (e.g., LAM-MPI) can be used. However, for supporting the sockets interface, we can either use the EFSockets implementation or the traditional sockets implementation in the kernel. The traditional sockets implementation uses only one port of EF and thus uses the network as a conventional GigE network (throughput of 1Gbps). The EFSockets implementation, on the other hand, stripes data across both ports on the network to achieve a peak theoretical throughput of 2Gbps.

3.2 Parallel File-system Middleware

Parallel file-system middleware provide high-speed data access for applications to meet the increasing I/O demands of parallel applications. The Parallel Virtual File-System (PVFS) [13] is one example of such a middleware. In *PVFS*, a number of nodes in the cluster system can be configured as I/O servers and one of them (either on a I/O server node or on a different node) as a metadata manager. *PVFS* achieves high performance by striping a file across a set of I/O server nodes allowing parallel access to the file. It uses the native file system on the I/O servers to store individual file stripes. An I/O daemon runs on each I/O node and services requests from the client nodes. A manager daemon running on the metadata manager node handles metadata operations like file permissions, file stripe characteristics, etc., but does not participate in the read/write operations. The metadata manager provides a cluster-wide consistent name space to applications.

Figure 3 shows the various approaches in which PVFS can be implemented over EF and IBA. Since EF only exposes the sockets interface, we use the sockets implementation of PVFS-2. For the EF network, as indicated in Section 3.1, the sockets interface can be supported using either the traditional kernel-level implementation of sockets (using EF as a conventional GigE adapter) or

using the high-performance EFSockets implementation (with a peak throughput of 2Gbps).

Three alternatives for implementing PVFS exist with IBA, i.e., using the native verbs interface, using traditional kernel-level sockets or using SDP. The PVFS implementation on top of the native verbs interface is currently implemented as a research prototype and is unstable; hence we use the sockets-based implementation of PVFS over SDP (due to its better performance as compared to the traditional kernel-level sockets implementation).

4 Testbed and Evaluation Metrics

Our experimental testbed consists of an 8 node cluster with dual Intel Xeon 3.4 GHz CPU-based EM64T systems. Each node is equipped with 1 GB of DDR400 memory. The nodes were connected with two networks: (i) MT25128 Mellanox HCAs (SDK v1.8.0) connected through a InfiniScale MT43132 24-port completely non-blocking switch and (ii) EtherFabric EF1-21022T network adapters connected through a Netgear GS524T GigE store-and-forward switch (software stack v1.1.5403-0).

Evaluation Metric: To understand the benefits of EFSockets and IBA in terms of communication time as compared to TCP/IP/EF, we define a metric, Relative Performance Improvement (RPI) in the following way:

RPI refers to the percentage of the communication time gain achievable by IBA that is already covered by EFSockets, i.e., if X, Y and Z are the time spent in communication with TCP/IP/EF, EFSockets and IBA respectively and C is the total computation time spent by the application, RPI is provided by equation in Figure 4. Also, we define the total execution time of TCP/IP/EF, EFSockets and IBA as $E_{TCP/IP/EF} = X+C$, $E_{EFSockets} = Y+C$ and $E_{IBA} = Z+C$ respectively.

In Sections 5, 6 and 7, we evaluate the EF and IBA networks with several micro-benchmarks and applications and provide detailed analysis for the same.

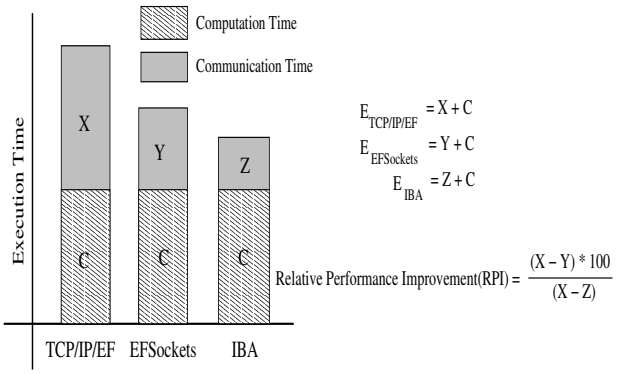

Figure 4: Evaluation Metrics: Relative Performance Improvement (RPI)

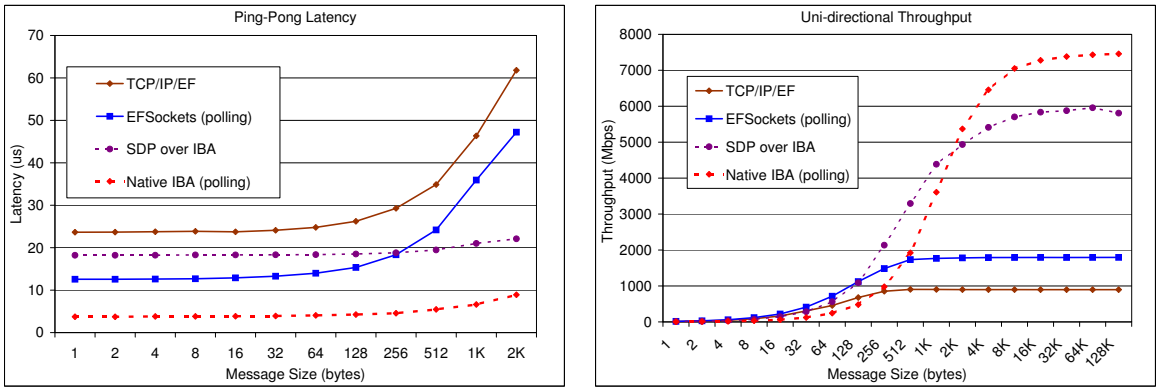

Figure 5: Native Protocol Evaluation: (a) Latency (b) Throughput

5 Native Protocol Performance

In this section, we present latency and throughput micro-benchmark evaluations of TCP/IP/EF and native protocols such as EFSockets and VAPI/IBA.

Figure 5a shows the latency of TCP/IP/EF, the native EFSockets, SDP/IBA and the native IBA verbs (VAPI). We observe that TCP/IP/EF is able to achieve a latency of close to 23μs as compared to the 12μs, 18μs and 3μs achieved by EFSockets, SDP/IBA and VAPI respectively, i.e., VAPI provides close to four times improvement as compared to EFSockets. Also, as the message size increases, we observe that the latency of TCP/IP/EF and EFSockets increases at a faster rate as compared to VAPI.

Figure 5b shows the throughput achieved by the different stacks. Again, we see that VAPI achieves a throughput of close to 7457Mbps as compared to the 897Mbps, 1794Mbps and 5808Mbps achieved by TCP/IP/EF, EFSockets and SDP/IBA respectively; an improvement of close to four

times for IBA as compared to EF. However, for small message sizes, we see that the throughput of VAPI is not as good due to a technique known as reverse packetization [9] used by the other schemes.

6 MPI Evaluation and Analysis

In this section, we evaluate MPI with EF and IBA with detailed micro-benchmarks and several applications such as the NAS benchmarks and GROMACS.

6.1 MPI Micro-Benchmarks

In this section, we present latency and throughput micro-benchmark evaluations of MPI over three networking stacks, namely TCP/IP/EF, EFSockets and VAPI/IBA. TCP/IP/EF uses EF as a conventional 1-GigE adapter (with a 1Gbps peak theoretical throughput). EFSockets uses both the ports of EF and thus achieves a 2Gbps theoretical throughput. Thus, this comparison addresses three different network speeds, viz., 1Gbps, 2Gbps and 8Gbps. For VAPI/IBA, we use MVAPICH2 [29] and for TCP/IP/EF and EFSockets, we use LAM-MPI for the comparison.

The latency test is conducted in a ping-pong fashion and the latency is derived from round-trip time. Figure 6a shows the latencies achieved by MPI on top of the different stacks. We observe that the 1-byte latency achieved by MPI/IBA is 4μs as compared to the 13μs and 28μs delivered by MPI/EFSockets and MPI/TCP/IP/EF, respectively; an improvement of more than 3 times for IBA as compared to EF. In addition, we observe that, as the message size increases, the latency of both MPI/EFSockets and MPI/TCP/IP/EF increases at a faster rate as compared to MPI/IBA.

The bandwidth test is used to determine the maximum sustained data rate that can be achieved at the network level. The test uses non-blocking MPI calls. In this test, the sender keepds sending back-to-back messages to the receiver until it has reached a pre-defined window size Q. Then it waits for these messages to finish and sends out another Q messages. Figure 6b shows the throughput for MPI with the different stacks. The peak throughput achieved by MPI/IBA is 7321Mbps

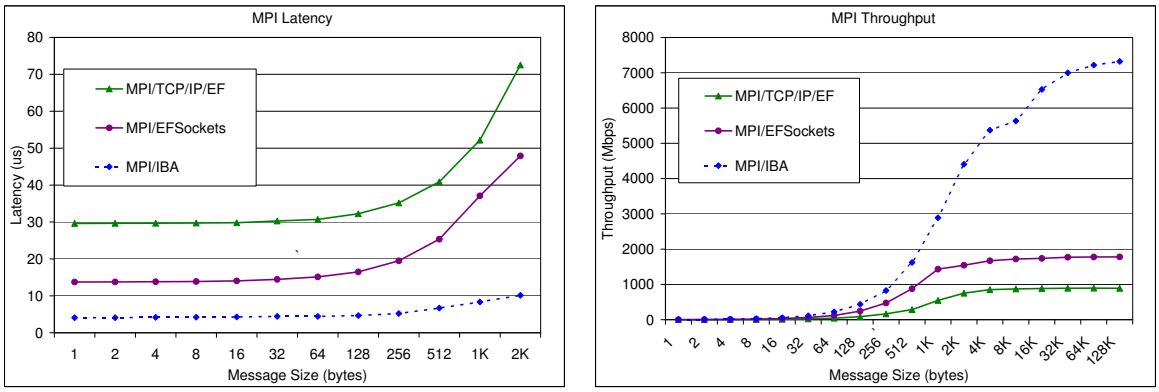

Figure 6: Micro-benchmarks with MPI/TCP/IP/EF, MPI/EFSockets, MPI/IBA (a) Latency (b) Throughput

as compared to the 1780Mbps and 890Mbps delivered by MPI/EFSockets and MPI/TCP/IP/EF, respectively, i.e., IBA achieves a 4 times higher throughput as compared to EF.

6.2 MPI Application Evaluation and Analysis

In this section, we evaluate various MPI-based applications with the different networking stacks. Specifically, we choose two well known application suites for the evaluation - NAS Parallel Benchmarks and GROMACS.

Evaluation of NAS benchmarks: NAS benchmarks, dervied from computational fluid dynamics application, are a set of benchmarks (IS, MG, EP, CG, LU, SP and BT) which are designed to evaluate the performance of parallel supercomputers. These benchmarks provide several problem sizes (e.g., Class W, S, A, B, C) for evaluating different supercomputers. We use Class B problem size, which tests the performance of small clusters in all our experiments.

Figures 7a and Figure 7b show the evaluation of NAS benchmarks using the three networking stacks. These figures show multiple evaluation metrics, viz., execution times, computation time, communication time and RPI. The measured execution time is the overall time taken by the application as reported by the benchmarks themselves. RPI is relative performance improvement achieved by IBA as compared to EFSockets as defined in Section 4; in short RPI refers to the percentage of communication benefit that we retain by replacing IBA with EFSockets.

We see that for most of the NAS benchmarks (e.g., MG, EP, LU, SP and BT), MPI/EFSockets

Table 1: NAS Benchmarks: Communication time and Message Size breakup

Application	Comm. Time TCP/IP/EF	<2K	2K -16K	16K - 1M	>1M
IS	84.18%	14	11	0	11
CG	65.48%	16113	0	11856	0
MG	64.08%	1607	630	3702	0
LU	54.68%	100021	0	1008	0
SP	51.32%	9	0	9636	0
BT	39.57%	9	0	4836	0

achieves a RPI value of 90-100%, i.e., replacing IBA with EF will affect the communication performance of such applications by less than 10%. For the other NAS benchmarks (IS and CG), we see that RPI values are close to 55%. It is to be noted that though these values are lower than those of the other NAS benchmarks, they are still impressive considering the fact that EF is a four times slower network as compared to IBA.

In order to further understand the reason for the low RPI values of EF for some of the NAS benchmarks, we profile these applications using a light-weight MPI profiling library, *mpiP* [6]. Also, in order to understand the message size breakdown at the MPI level, we use the numbers reported in [25] (note that the message size breakup does not change with the experimental testbed).

Table 1 shows the communication time spent using TCP/IP/EF and message size distribution of all the benchmarks used. We note two main insights from the table. First, IS spends close to 85% of its time in communication and 15% of its time in computation, i.e., its total execution time and communication time are close to each other. Benchmarks such as CG have 65% of its time spent in communication and the remaining 35% in computation. The second point to note in the table is that IS and CG use the largest overall message sizes (IS uses several messages over 1Mbyte in size and CG uses close to 12,000 messages that are between 16Kbyte and 1Mbyte in size). As shown in Figure 6, IBA achieves close to four times the performance of EF for large message sizes. Accordingly, communication-intensive applications which use large message sizes benefit significantly resulting in lesser RPI values.

Evaluation with GROMACS: Groningen Machine for Chemical Simulations (GROMACS) is a molecular dynamics simulation package developed at the University of Groningen. GROMACS

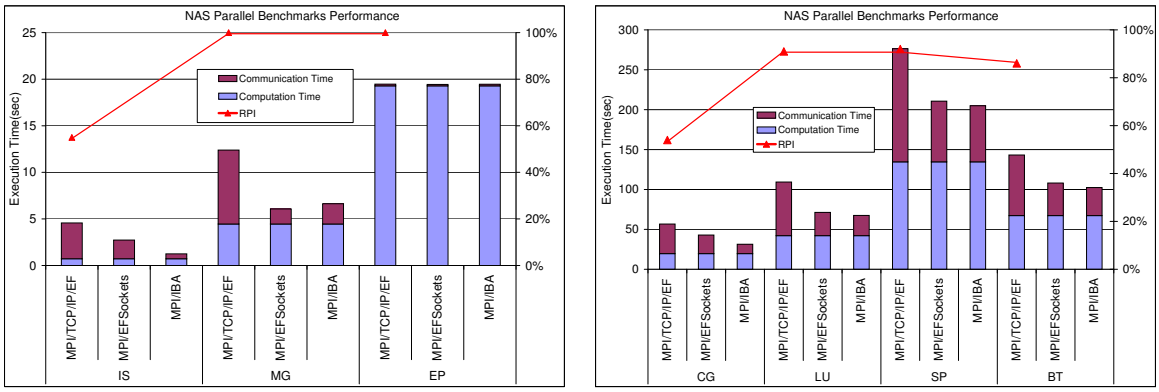

Figure 7: NAS Benchmarks Evaluation with MPI/TCP/IP/EF, MPI/EFSockets, MPI/IBA: (a) IS, MG and EP(b) CG, LU, SP and BT

is a distributed molecular dynamics implementation, for systems with hundreds to millions of particles. It is a communication-intensive benchmark. When partitioned on to multiple nodes, a major portion of the data can reside in L2 cache, reducing the main memory accesses and resulting in super-linear speedup. We evaluate the performance of three different GROMACS applications, viz., DPPC, VILLIN and POLY-CH2 on 16 processes (8 nodes).

Figure 8 shows the performance of all three applications for the three stacks. As shown in the figure, MPI/EFSockets performs significantly better than MPI/TCP/IP/EF for DPPC. MPI/IBA, however, performs only slightly better than MPI/EFSockets. This is also reflected in the RPI value which is 92%. This shows that replacing IBA with EF only matters minimally for this application in the communication time as well as the overall execution time.

In the case of VILLIN and POLY-CH2, the performance improvement of MPI/IBA is better compared to MPI/EFSockets. The RPI values for these applications are 56% and 55% respectively. The reason being both these application are heavily communication-intensive (with 99% of the total execution involved in communication with large messages of sizes close to 500Kbytes as shown in Table 2 and Figure 8). We believe that the huge communication time spent in such slow networks in a 16 processor run is mainly due to the process skew waiting for other processes to synchronize. As mentioned earlier, IBA achieves close to four times the performance of EF for large message sizes. Accordingly, applications such as VILLIN and POLY-CH2 benefit significantly.

Table 2: GROMACS: Communication Time and Message Size Distribution

Application	Comm. Time TCP/IP/EF	Message Size
DPPC	82.77%	91KB
VILLIN	99.06%	375KB
POLY-CH2	99.3%	480KB

In summary, we notice that in the MPI applications that we studied, for applications which use small or medium message sizes, EF is able to provide 80-95% of the performance gains achievable by IBA. For applications which use large message sizes, EF is able to provide 55-60% of the performance gains achievable by IBA.

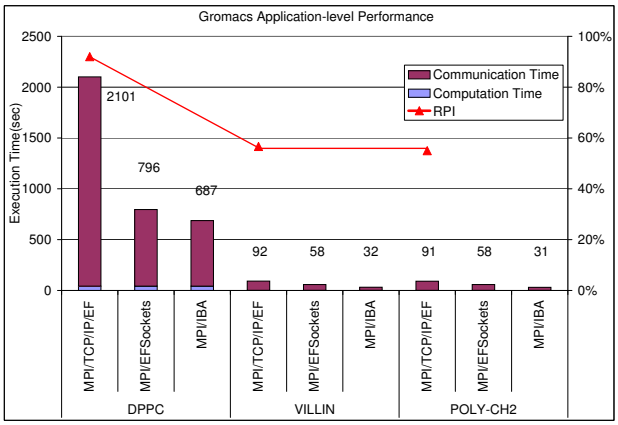

Figure 8: Gromacs Evaluation with TCP/IP/EF, EFSockets and IBA: DPPC, VILLIN, POLY-CH2

7 Parallel File-system Evaluation

In this section, we evaluate the three network stacks with several PVFS-based micro-benchmarks and the MPI-Blast application.

7.1 File-system Micro-Benchmarks

Figure 9 shows a typical PVFS setup. As shown in the figure, a number of nodes in the cluster system can be configured as I/O servers and one of them (either an I/O server or a different node) as a metadata manager. Compute nodes contact the metadata manager for file information and get the actual data from different I/O servers over the network in parallel. I/O servers use the local file-system to store the individual file stripes. It is to be noted that, due to the wide difference between

the network and the disk speeds, the effective throughput achieved by PVFS could be limited by that of the disk. However, data used by applications is frequently in server memory, e.g., due to file caching and read-ahead mechanisms. Thus, applications can still benefit from fast networks in such cases. Hence, we designed our experiments based on a memory-resident file system, *ramfs*. These tests are designed to stress the network data transfer independent of any disk activity.

As mentioned in Section 3.2, PVFS can be implemented either using the sockets interface or over the native protocol stack exposed by each network. For EF, the native protocol itself exposes the sockets interface through the EFSockets protocol stack; hence for this network, we use the sockets-based implementation of PVFS-2. For IBA, there exist research prototypes of PVFS and PVFS-2 which have been implemented directly on the native IBA interface [36] and give the best performance. However, these implementations are currently unstable, due to which we stick to the sockets implementation of PVFS in this paper. Further, we use SDP as the sockets protocol stack due to its better performance as compared to the traditional kernel-level sockets implementation.

Performance of Contiguous File I/O: In this test, we evaluate the performance of PVFS concurrent read/write operations. For this purpose, we use a program to parallelize file read/write accesses of contiguous data buffers from each compute node. Each compute node simultaneously reads or writes a single contiguous region of size $64N$ Kbytes, where N is the number of I/O nodes in use. For example, if the number of I/O nodes is 4, the request size is 256 Kbytes (striped as four 64Kbyte chunks).

Figure 10 shows the PVFS read/write performance with TCP/IP/EF, EFSockets and SDP. In this experiment, we use four servers and vary the clients from one to four to measure the aggregate throughput. As shown in the figure, PVFS/SDP considerably outperforms the PVFS/TCP/IP/EF and PVFS/EFSockets, achieving close to twice the throughput as PVFS/EFSockets and thrice that of PVFS/TCP/IP/EF. This follows the same trend as shown by the basic throughput micro-benchmark results in Figures 5.

Performance of Non-Contiguous File I/O: In this test, we evaluate the performance of PVFS

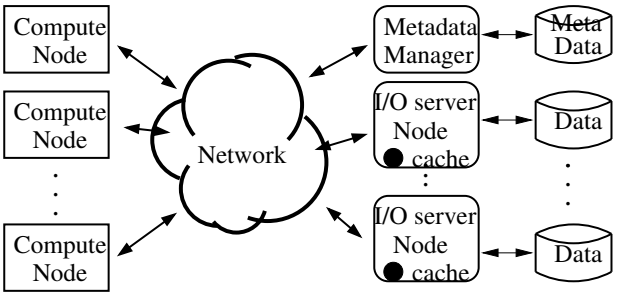

Figure 9: PVFS Setup

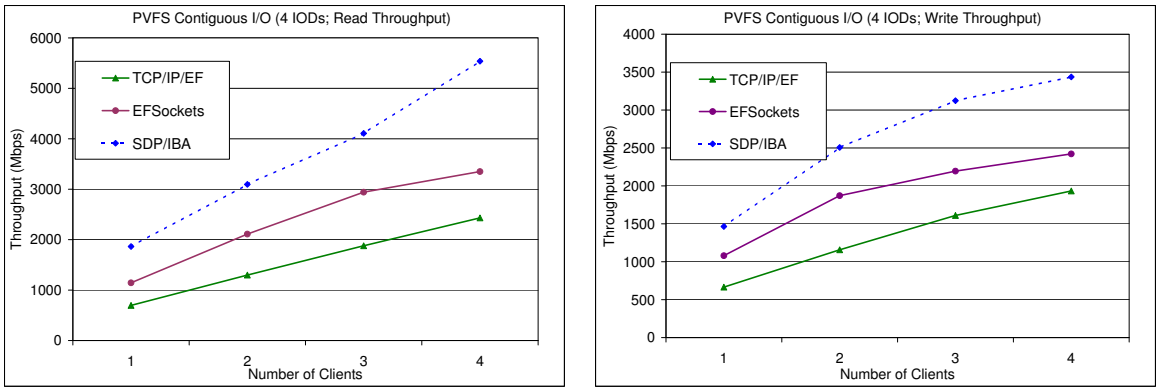

Figure 10: PVFS over TCP/IP/EF, EFSockets and SDP (a) Read Performance (b) Write performance

with noncontiguous read and write accesses. MPI-Tile-IO [32] is a tile-reading MPI-IO benchmark that performs non-contiguous read or write accesses. It tests the performance of tiled accesses to a two-dimensional dense dataset, simulating the type of workload that exists in some visualization applications and numerical applications. In our experiments, four nodes are used as server nodes and the other four as client nodes running the MPI-tile-IO processes. Each process renders a 1×2 array of displays, each with 1024×768 pixels. The size of each element is 32 bytes, leading to a file size of 48 MB.

We evaluate the read and write performance of mpi-tile-io over PVFS. As shown in Figure 11, the peak read throughput achieved by PVFS/SDP is 1119Mbps as compared to 764Mbps and 756Mbps achieved by PVFS/EFSockets and PVFS/TCP/IP/EF, respectively. We see a similar trend in PVFS write performance as well. One interesting point to be noted is that the performance of all the networks is considerably worse in this test as compared to the concurrent file I/O test; this is due to the non-contiguous data access pattern of the MPI-tile-IO benchmark which adds significant

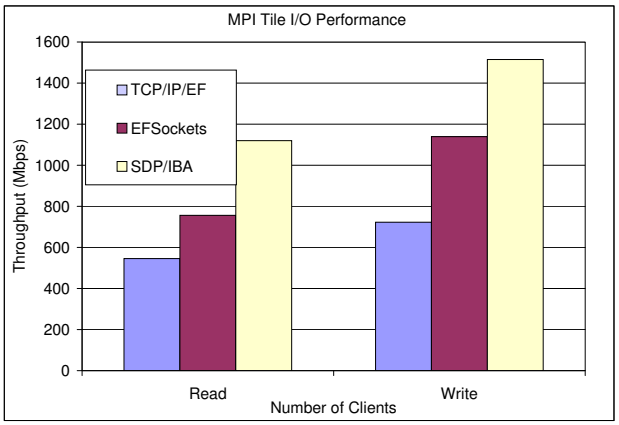

Figure 11: Non-contiguous Read and Write Performance of PVFS over TCP/IP/EF, EFSockets and SDP

amount of overhead.

7.2 File-system Application Evaluation

In this section, we evaluate the three networking stacks using MPI-Blast, a parallel sequence matching application.

Overview of MPI-BLAST: BLAST is a Basic Local Alignment Search Tool that is designed for comparing a sequence of a particular gene or protein with other sequences from a variety of organisms. It uses an algorithm developed by NCBI; by searching for a portion of alignment, BLAST identifies regions of similarity within two protein sequences.

MPI-BLAST is a parallel version of NCBI BLAST developed by the Los Alamos National Laboratory. MPI-BLAST segments the BLAST database and distributes it across cluster nodes, permitting BLAST queries to be processed on many nodes simultaneously. Further, since each node's segment is smaller in size, it can reside in the memory cache, yielding a significant speedup. For our experiments, we use four PVFS I/O servers and 4 nodes as both PVFS clients and MPI processes to access the shared storage. We vary the database sizes from 1GB to 4GB.

Figure 12a shows the total execution time of MPI-Blast while running a single query with

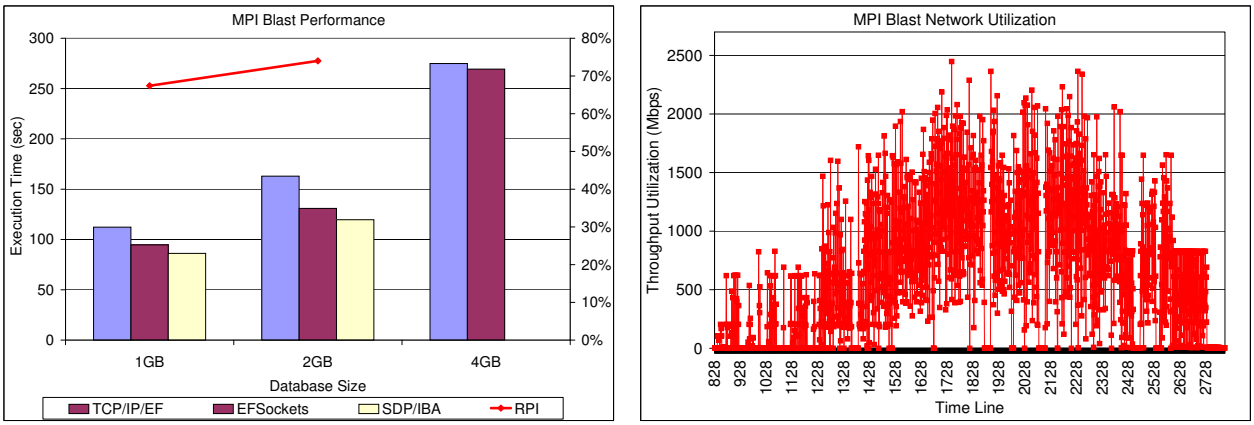

Figure 12: MPI-Blast Application (a) Execution Time with PVFS/TCP/IP/EF, PVFS/EFSockets and PVFS/SDP (b) Network Utilization

varying database sizes[2]. From the figure, we see that for a 1GB database, MPI-Blast using PVFS/TCP/IP/EF, EFSockets and SDP have execution times of 112 secs, 94 secs and 86 secs, respectively; the RPI value for this experiment is 67%.

Since, the 1GB database used in this experiment completely fits into the host memory of the PVFS servers, the file-system communication is primarily bound by the network performance. This reflects in the RPI value where EF is only able to cover 67% of the performance capability of IBA (which is, though impressive, lesser than what we observed for some of the NAS benchmarks in Section 6.2). Further, since MPI-Blast also performs a significant amount of computation once the data is fetched, i.e., the percentage of communication in the overall application execution can be expected to be less. This leads to lesser improvement for applications replacing EF with IBA in terms of total execution time. Similarly, for the 2GB database, we see that 74% of the communication performance gain (RPI) achievable by PVFS/SDP/IBA is already provided by PVFS/EFSockets.

Despite the fact that there is close to 3 times improvement in terms of bandwidth between EFSockets and SDP/IBA as shown in Figure 5, we see that the performance of MPI-Blast with PVFS/SDP is closely comparable to that of PVFS/EFSockets. In order to understand this further, we measure the network utilization of MPI-Blast using PVFS/SDP.

[2]For the 4GB database, we had stability issues with SDP and hence could not include this result. We plan to include this in the final version of the paper.

Figure 12b shows the network utilization of MPI-Blast with PVFS/SDP for a 1GB database. As shown in the figure, the IBA network is severely under utilized during the execution of the application. The peak network bandwidth achievable using SDP is close to 5.8Gbps whereas only 2Gbps of network bandwidth is utilized during most of the execution; a requirement which slower 2-Gbps networks such as EF can easily meet. Since the network utilization is more than the peak bandwidth of TCP/IP/EF, we see that the performance of MPI-Blast using PVFS/TCP/IP/EF is significantly worse compared to the other two networks. However, as it is within the reach of EFSockets, there is not much difference between the performance of EFSockets and SDP.

In summary, we see that for applications such as MPI-Blast, 69-75% of the communication performance gain achievable by SDP/IBA is already provided by up to four-times slower networks such as EF.

8 Discussion and Related Work

In the past decade, there has been significant research on various programming models such as Sockets [33, 24, 11, 10] and MPI [26, 30, 31, 22], as well as parallel file-systems over high-speed networks [36]. In most of these implementations, researchers have used the advanced features of high-speed networks (e.g., RDMA, non-blocking communication, etc.) to design these programming models in an efficient manner. In our previous work [34, 28, 23], we have also shown the benefits of the advanced hardware features of 10-Gigabit as well as 1-Gigabit networks to efficiently support applications outside these standard programming models. In this paper, we try to understand and point out that current generation networks offer advantages in two directions: (i) raw network performance and (ii) advanced communication features (e.g., RDMA, multicast). Of these, the advanced communication features are the most helpful in achieving high-performance for applications, scalability for large clusters, etc. However, the raw network performance does not seem to play a significant role in these performance benefits.

9 Concluding Remarks

Recent trends in high-speed networking technology emerged due to the underlying assumption that the network is one of the primary bottlenecks in cluster computing. However, whether this assumption is in fact a reality for current environments is yet to be understood. In this paper, we try to understand if such multi-gigabit network speeds justify the cost of these newer networks with respect to the performance gains they provide for end applications. Specifically, we performed comparative analysis with conventional 1-GigE adapters, 2-GigE adapters (using two 1-GigE ports in adapters such as EtherFabric) and 8-Gigabit network adapters such as IBA for end applications in two broad domains: (i) message passing interface (MPI) based scientific applications such as GROMACS and the NAS parallel benchmarks and (ii) data-intensive applications relying on parallel file-systems (e.g., PVFS), such as MPI-Blast. Our experimental results show that for many of the end-applications that we studied, EtherFabric was able to achieve 70-90% of the performance gains achievable by IBA. Due to the low-cost of EtherFabric and its compatibility with the existing TCP/IP/Ethernet infrastructure (include copper port based traditional GigE switches), such adapters seem to hold a lot of promise for many of the end-applications. We plan to study this further with a wider range of applications and also understand the impact of varying network speeds.

References

[1] IBM Corporation. http://www.ibm.com/.

[2] InfiniBand Trade Association. http://www.infinibandta.com.

[3] Level5 Networks. http://www.level5networks.com/.

[4] Mellanox Technologies. http://www.mellanox.com.

[5] mpiblast: Open-source parallel blast. http://mpiblast.lanl.gov/.

[6] mpiP: Lightweight, Scalable MPI Profiling. http://www.llnl.gov/CASC/mpip/.

[7] NAS Parallel Benchmarks. http://www.nas.nasa.gov.

[8] Sockets Direct Protocol. http://www.infinibandta.org.

[9] P. Balaji, S. Bhagvat, H.-W. Jin, and D. K. Panda. Asynchronous Zero-Copy Communication for Synchronous Sockets Direct Protocol (SDP) over InfiniBand. In *CAC*, 2006.

[10] P. Balaji, S. Narravula, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Sockets Direct Protocol over InfiniBand in Clusters: Is it Beneficial? In *ISPASS '04*.

[11] P. Balaji, P. Shivam, P. Wyckoff, and D. K. Panda. High Performance User Level Sockets over Gigabit Ethernet. In *Cluster Computing '02*.

[12] G. Burns. LAM: Local Area Multicomputer. Technical report, Ohio Supercomputer Center, 1995.

[13] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A Parallel File System for Linux Clusters. In *4th Annual Linux Showcase and Conference*. USENIX Association, 2000.

[14] B. D. Davison. *The Design And Evaluation Of Web Prefetching and Caching Techniques*. PhD thesis, Department of Computer Science, Rutgers University, October 2002.

[15] W. Feng, P. Balaji, C. Baron, L. N. Bhuyan, and D. K. Panda. Performance Characterization of a 10-Gigabit Ethernet TOE. In *HotI*, 2005.

[16] W. Feng, J. Hurwitz, H. Newman, S. Ravot, L. Cottrell, O. Martin, F. Coccetti, C. Jin, D. Wei, and S. Low. Optimizing 10-Gigabit Ethernet for Networks of Workstations, Clusters and Grids: A Case Study. In *SC '03*.

[17] H. Frazier and H. Johnson. Gigabit Ethernet: From 100 to 1000Mbps.

[18] S. Glassman. A Caching Relay for the World Wide Web. 1994.

[19] D. van der Spoel H. J. C. Berendsen and R. van Drunen. GROMACS: A Message-passing parallel molecular dynamics implementation. In *Comp. Phys. Comm. 91, 43-56*, 1995.

[20] W. Huang, G. Santhanaraman, H.-W Jin, and D. K. Panda. Scheduling of MPI-2 One Sided Operations over InfiniBand. In *CAC*, 2005.

[21] J. Hurwitz and W. Feng. End-to-End Performance of 10-Gigabit Ethernet on Commodity Systems. *IEEE Micro '04*.

[22] J. Liu and A. Mamidala and D. K. Panda. High Performance MPI-Level Broadcast on InfiniBand with Hardware Multicast Support. In *IPDPS*, 2004.

[23] H. W. Jin, S. Narravula, G. Brown, K. Vaidyanathan, P. Balaji, and D. K. Panda. Performance Evaluation of RDMA over IP: A Case Study with the Ammasso Gigabit Ethernet NIC. In *HPI-DC*, 2005.

[24] J. S. Kim, K. Kim, and S. I. Jung. SOVIA: A User-level Sockets Layer over Virtual Interface Architecture. In *Proceedings of Cluster Computing*, 2001.

[25] 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 *SC*, 2003.

[26] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. In *ICS*, 2003.

[27] L. Melatti. Fast Ethernet: 100 Mbit/s Made Easy. *Data Communications on the Web*, Nov. http://www.data.com/tutorials/100mbits-made-easy.html.

[28] S. Narravula, P. Balaji, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Supporting strong coherency for active caches in multi-tier data-centers over infiniband. In *SAN*, 2004.

[29] Network-Based Computing Laboratory. MVAPICH: MPI for InfiniBand on VAPI Layer. http://nowlab.cse.ohio-state.edu/projects/mpi-iba/index.html.

[30] OpenMPI: . Open Source High Performance Computing. http://www.open-mpi.org/.

[31] A. Pant and H. Jafri. Communicating Efficiently on Cluster based Grids with MPICH-VMI. In *2004 IEEE International Conference on Cluster Computing*, September 2004.

[32] Rob B. Ross. Parallel i/o benchmarking consortium. http://www-unix.mcs.anl.gov/rross/pio-benchmark/html/.

[33] H. V. Shah, C. Pu, and R. S. Madukkarumukumana. High Performance Sockets and RPC over Virtual Interface (VI) Architecture. In *Proceedings of CANPC workshop*, 1999.

[34] K. Vaidyanathan, H.-W. Jin, S. Narravula, and D. K. Panda. Accurate Load Monitoring for Cluster-based Web Data-Centers over RDMA-enabled Networks. Technical report, The Ohio State University, 2005.

[35] S. VanderWiel and D. J. Lilja. A Survey of Data Prefetching Techniques.

[36] J. Wu, P. Wyckoff, and D. K. Panda. PVFS over InfiniBand: Design and Performance Evaluation. In *ICPP*, Oct. 2003.