Contents lists available at SciVerse ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

RXIO: Design and implementation of high performance RDMA-capable GridFTP ☆

Yuan Tian [a,*], Weikuan Yu [a,b], Jeffrey S. Vetter [b]

[a] Dept. of Computer Science, Auburn University, AL 36849, United States
[b] Computer Science and Mathematics, Oak Ridge National Laboratory, Bldg 5100, MS-6173, 1 Bethel Valley Rd., Oak Ridge, TN 37831, United States

ARTICLE INFO

Article history:
Available online 21 December 2011

ABSTRACT

For its low-latency, high bandwidth, and low CPU utilization, Remote Direct Memory Access (RDMA) has established itself as an effective data movement technology in many networking environments. However, the transport protocols of grid run-time systems, such as GridFTP in Globus, are not yet capable of utilizing RDMA. In this study, we examine the architecture of GridFTP for the feasibility of enabling RDMA. An RDMA-capable XIO (RXIO) framework is designed and implemented to extend its XIO system and match the characteristics of RDMA. Our experimental results demonstrate that RDMA can significantly improve the performance of GridFTP, reducing the latency by 32% and increasing the bandwidth by more than three times. In achieving such performance improvements, RDMA dramatically cuts down CPU utilization of GridFTP clients and servers. These results demonstrate that RXIO can effectively exploit the benefits of RDMA for GridFTP. It offers a good prototype to further leverage GridFTP on wide-area RDMA networks.

1. Introduction

State-of-the-art networking technologies, such as InfiniBand [1] and 10GigE [2,3], provide high bandwidth of more than 10 Gbps and sub-microsecond latency. This trend of network acceleration has led to a new class of communication protocols called Remote Direct Memory Access (RDMA) [4]. RDMA has been extensively exploited in local area networks for various data transfer protocols. It is used to leverage the benefits from latest networking technologies such as InfiniBand [1] and iWARP [5]. Numerous RDMA-based communication and transport protocols for programming models [6,7], file systems [8–11], and storage paradigms [12–14] have been proposed and implemented.

Recently, motivated by the potential to extend InfiniBand performance to the wide-area, there were hardware implementations of InfiniBand over long-distance network devices, in particular Longbow XR from Obsidian Research Corporation [15] and NX5010ae from Network Equipment Technologies [16]. These products extend the reach of InfiniBand connections to several thousand miles, and open up the possibility of connecting supercomputers, clusters and storage systems located far apart. With these capabilities, RDMA through InfiniBand has become a promising contender for high-performance data transport in the wide-area. Initial results indicated that InfiniBand-based RDMA technologies can sustain multiple Gbps network transfer rates over long distance, for example, 7 Gbps over 8600 mile connections [17–19].

Many geographically distributed high-performance computing systems and data centers are producing and/or supplying large volumes of data sets. Such data must be transported to storage, visualization and analysis systems that are often remotely located. The traditional approach of utilizing TCP/IP tools such as GridFTP [20] and bbcp [21] for data movement

☆ Reviews processed and proposed for publication to Editor-in-Chief by Guest Editor Prof. Fangyang Shen.
* Corresponding author.
E-mail addresses: tianyua@auburn.edu (Y. Tian), wkyu@auburn.edu (W. Yu), vetter@ornl.gov (J.S. Vetter).

0045-7906/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compeleceng.2011.11.008

on the wide-area network (WAN) requires sophisticated per-connection optimizations. To support such applications, dedicated 10 Gbps connections can be provisioned over several networks, including Internet2 [22], ESnet's Data Sciences Network [23], UltraScience Net [24], CHEETAH [25], and others. However, the research community is yet to see RDMA-based data movement protocols for grid run-time systems such as the Globus Toolkit [26]. As a popular grid-based run-time environment, Globus uses GridFTP that extends the FTP protocol for data movement on high-bandwidth wide area networks. With more communication protocols being integrated, an extensible Input/Output (XIO) system has evolved as the I/O abstraction layer of GridFTP. However, XIO still relies on the legacy TCP or UDP protocols for data movement, thus it has not yet taken advantage of the benefits of RDMA such as low-latency, high-bandwidth and low CPU utilization.

In this paper, we examine the software architecture of GridFTP for the feasibility of enabling RDMA. We propose to design a new I/O system RXIO that retains the original XIO on top of TCP/IP-based protocols, but extends it with additional features to match RDMA's semantics in connection establishment and communication progress. Accordingly, an RDMA driver for RXIO is designed to enable GridFTP on RDMA. As a case study, we have implemented RXIO on InfiniBand using its native RDMA operations for data movement. Zero-copy RDMA operations are utilized to minimize memory copies and reduce CPU involvement. A credit-based buffer management scheme is implemented to allow a GridFTP sender to send data as quick as possible. An RDMA write-based zero-copy noncontiguous data transfer is also implemented in the prototype. We have validated our design and implementation in local area InfiniBand networks. Our experimental results show that the GridFTP InfiniBand implementation is able to exploit the benefits of RDMA including low-latency, high-bandwidth and low CPU utilization. Compared to IPoIB and 10GigE, GridFTP on RDMA achieves significant performance improvements, reducing the latency by 32% and increasing the bandwidth by more than three times. In addition, when performing long streams of bandwidth tests, GridFTP on RDMA is able to maximize the performance of data movement using only a fraction of CPU cycles. At this point we do not have access to long-distance RDMA networks, however, we are keenly looking forward to an extensive evaluation and tuning of this RXIO-based GridFTP on long-distance RDMA networks as they become available. Nonetheless, our experimental results adequately demonstrate the effectiveness of our design and implementation of RXIO in exploiting the benefits of RDMA for GridFTP. It also provides a good prototype to further examine and prepare GridFTP on wide-area RDMA networks.

The rest of the paper is organized as follows. We first introduce the background knowledge and an overview of related work in Section 2. Then we describe the design of RXIO and the RDMA driver in Section 3. Section 4 provides a discussion on the implementation of RXIO on InfiniBand. Experimental results are provided in Section 5. Finally, we conclude the paper in Section 6.

2. Background and related work

2.1. Overview of GridFTP and Globus XIO

In Grid computing environment, applications often times need to access data that is located at different sites. Thus, transmitting large files across multiple autonomies are frequently required. FTP protocol has been widely used as a standard for data movement over a TCP-based network. However, despite the performance limitation inherent within the protocol itself, it is common that different data site has different self-defined interface for data access. Such difference restrains the efficiency of data movement and increases the complexity of data sharing. As part of the data management component within Globus Toolkit [26], GridFTP is designed to solve the aforementioned incompatibility problem of accessing data among different resources, and improve data transfer performance. Many features were designed and implemented to provide a secure, reliable and high-performance framework for bulk data movement in grid environment. For example, GridFTP supports parallel TCP streams to achieve high-performance data transfer. TCP buffer/widows size can be negotiated automatically or manually to achieve optimal performance. Data transfer is restartable and can be coordinated among different sites. GridFTP even allows user to specify the start point of transportation in the file. Security is ensured through providing encryption options such as PKI and SSH-based encryption. In general, GridFTP provides an uniformed and efficient mechanism to access data in the Grid. Such compatibility is achieved by utilizing Globus Extensible Input/Output (XIO) [27] framework. XIO can be seen as a thin mapping layer between application and underlying XIO *drivers*. To applications and users, XIO presents a single standard OCRW (open/close/read/write) interface. Below XIO is a group of data drivers that can be divided into two categories: transform driver and transport driver. Transform drivers are used to manipulate data in the memory buffer, e.g. GSI authentication driver. While transport drivers are in charge of sending data over the wire. One or multiple XIO drivers can be loaded as a stack to operate on the same data. XIO identifies each I/O request from application and maps it to corresponding driver's interface from top to the bottom of the stack. Currently XIO only supports transport drivers that are legacy IP-based, such as TCP [27] and UDT [28,29], which internally employ kernel-level memory copies. Thus the performance for data movement is constrained. Our goal essentially is to design and implement an RDMA-based transport driver conforming XIO OCRW interface. Therefore upper layer applications, such as GridFTP, is able to take advantage of the cutting-edge data movement technology to achieve high-performance data transfer.

2.2. Related work on GridFTP and RDMA

Efficient data movement in the grid environment attracts research interests from many organizations. Allcock et al. [30] designed a striped GridFTP protocol that achieved efficient data-transfer using parallel TCP streams in GridFTP. In view of continuously evolving communication protocols for network I/O, researchers from the same group proposed an abstraction layer, eXtensible Input/Output (XIO), to hide the differences of protocol variants and facilitate the porting of GridFTP on new protocols. Bresnahan et al. [29] then investigated UDT [28] (a high performance UDP-based reliable communication protocol) as an alternative transport method for GridFTP using the XIO abstraction. Kissel et al. [31] recently designed a new XIO driver on top of the phoebus infrastructure to ensure better end-to-end performance for GridFTP. Ito et al. [32] explored an automatic tuning mechanism called GridFTP with Automatic Parallelism Tuning (GridFTP-APT) that dynamically determines the number of parallel TCP connections for best performance based on network status information from the Grid middleware. All these efforts focused on TCP/IP-based communication protocols or their variants.

Leveraging RDMA from high speed networks for high-performance data movement has been very popular in various programming models and storage paradigms. Liu et al. [6] designed RDMA-based MPI over InfiniBand. Zhou et al. [33] studied the benefits of VIA networks in database storage. DeBergalis et al. [8] implemented a file system DAFS on top of VIA. Implementations of PVFS [34] on top of RDMA networks such as InfiniBand and Quadrics were described in [10] and [11], respectively. Callaghan et al. [9] provided an initial implementation NFS over RDMA (NFSoRDMA) on Solaris. An implementation of NFSoRDMA was made available for Linux [35] systems. RDMA has also been exploited for data movement in SCSI-based storage protocols including efforts from academia and industry [12–14]. Our work complements these efforts to enable RDMA for grid-oriented applications.

Enabling high-performance and grid-computing applications using RDMA recently attracted a lot of interests, both in terms of infrastructure deployment and research investigation. A group of researchers from Oak Ridge National Laboratory extensively studied the performance of InfiniBand-based communication protocols, programming models as well as storage protocols across long-distance OC192 connections on UltraScience Net [36,18,19]. Their work revealed the strength of InfiniBand across long distance. Narravula et al. [37] studied the performance of different HPC middleware across simulated long-distance InfiniBand connections. Together, these efforts exhibited the performance impact of different network parameters and reliability configurations for InfiniBand on WAN. Lai et al. [38] designed an efficient FTP protocol for high performance data-transfer by exploiting a communication library Advanced Data Transfer Service (ADTS) over InfiniBand. Built on top of the earlier work [38], Subramoni et al. [39] have extended the RDMA-based FTP implementation for GridFTP. They have achieved as much as 100% performance improvement using the ADTS library. Our work accomplished the similar goal yet taking a completely different approach. We examine the need of XIO protocol extensions to enable GridFTP on top of any RDMA networks, without having to create a separated library for communication establishment and progression. An RDMA-capable XIO (RXIO) system is designed to closely match communication semantics and connection management of RDMA. We demonstrate an implementation of the RXIO RDMA driver on InfiniBand, and also show the benefits of InfiniBand to GridFTP on different transfer patterns such as noncontiguous I/O.

3. Design RXIO to enable GridFTP on RDMA networks

To integrate RDMA for high-performance zero-copy I/O in GridFTP, the design of RXIO needs to handle the following issues: (a) how to extend XIO to match the communication characteristics of RDMA networks; (b) how to establish RDMA connections inside RXIO; and (c) how to manage the progress of RDMA communication protocols.

3.1. RDMA-Capable XIO (RXIO)

XIO uses system file descriptors to represent and manage network connections, such as TCP/IP sockets. Built on top of that, RXIO provides extensions to support RDMA-based connections. It removes the existing dependence of XIO on system file descriptors, and introduces new handles to identify network connections based on the underlying networks. In addition, RXIO is expanded with functionalities to invoke different callback routines for these networks. For example, for events from an RDMA network, a callback routine from the RDMA driver will be called by RXIO to process RDMA events without RXIO knowing the internal details of an RDMA network. Fig. 1 shows the software architecture of RXIO with these extensions. RXIO is designed to work with all RDMA networks including InfiniBand [1], iWARP [5] and RDMAoE (RDMA over Ethernet) [40]. The RDMA driver groups functionalities of RDMA networks into three categories: connection management, communication progress, and data transfer. Data transfer functions are network-specific. We describe them with an example implementation of RXIO on InfiniBand in Section 4. Connection management and communication progress are more generically designed, meant to work with RDMA-compliant [4] networks.

3.2. Connection management and communication progress

The original connection management and communication progress functionalities of XIO are centered around a poll/select mechanism that detects the events from connection request/establishment and network communication. This works fine for legacy IP-based protocols because their connections are represented as system files and are compatible with the poll/select interface. In addition, the operating system (OS) handles all internal events for data communication and connection establishment. XIO only needs to initiate a non-blocking I/O call and then poll on the completion as the OS finishes handling internal communication or connection establishment tasks.

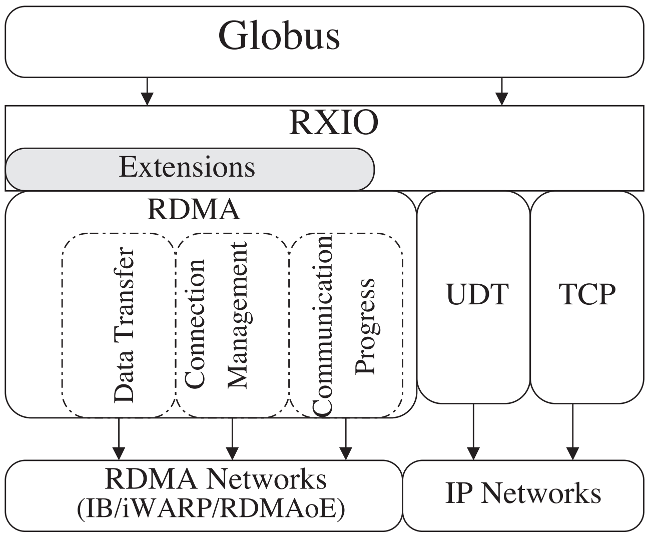


Fig. 1. Software architecture of RXIO.

In contrast, RDMA provides an OS-bypass communication. The intermediate network events from connection establishment and data communication no longer go through the operating system, instead are directly available at the user level. For this reason, programming libraries (or tools) built on top of RDMA must handle all network events associated with a single message. To avoid a significant change on the core infrastructure of XIO, RXIO is designed to conform to the poll/select mechanism. Fig. 2 shows the diagram of RDMA-based connection management in RXIO. An additional thread managing network event is created for each GridFTP endpoint, a client or a server. Within an endpoint, a one-way pipe is created for communication between the network thread and the main GridFTP thread. In the case of a client, it allocates a new connection (a.k.a Queue Pair in the context of InfiniBand) and sends a connection request via *rdma_connect()* to the server. The network thread listening for incoming requests on the server receives this connection request and handles a series of events that are detected on the associated RDMA event channel. This server then allocates a new RDMA connection. Via *rdma_accept()*, it accepts and confirms the connection request to the client. The successful completion of the accept() call will be detected via an *established* event by network threads on both server and client sides. Each thread writes a notification to the pipe. The main thread, polling on the other end of the pipe, detects the establishment of a new connection. This completes the establishment of a new RDMA connection.

On RDMA networks, each connection is no longer represented by an individual file descriptor. For efficiency reasons, communication progress of all connections is reported through a shared Complete Queue (CQ) and its associated event channel. Compared to the TCP driver, the RXIO RDMA driver cannot complete message transmission with only one I/O event. This is because it has to detect and process completion events of send and receive messages, as well as intermediate events of control messages and intermediate messages. Section 4 can be referred to for more details on the number of events and the associated RDMA operations for RXIO. We design RXIO to report communication progress in a way that is similar to the establishment of connections. The network thread as described above is used to detect communication events. A notification to the main thread via a pipe is generated only after all the events pertaining to a message are complete. The main thread in GridFTP then marks the completion of this message for the application.

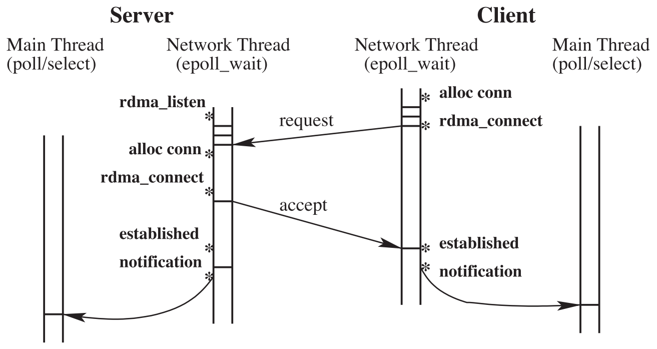


Fig. 2. Connection management in RXIO.

4. A case study: implementing RXIO on InfiniBand

With the aforementioned design of RXIO and its RDMA driver, we have implemented GridFTP on InfiniBand as a case study. It is developed on top of OpenFabrics Enterprise Distribution (OFED) version-1.4. We describe a couple of implementation issues including data transfer for short and long messages, noncontiguous message transfer, and credit-based buffer management.

4.1. Data transfer of short and long messages

A number of fixed-size, registered memory buffers are allocated for data communication. InfiniBand requires data communication to fall in registered memory regions. To save the registration cost, we enable a registration cache for memory regions. The left diagram of Fig. 3 shows the transfer of a short message. From the source memory of the sender, a short message is first copied into a buffer and then sent across the network. At the receiver side, it is received into a pre-posted receive buffer. The incoming message remains in the receive buffer, until the receiver has posted a matching receive operation. Then the message is copied into the destination memory at the receiver. This is also called the eager protocol because the sender does not wait for the receiver to become ready on the destination memory. However, a sender can never overrun the receiver. This is guaranteed through a credit scheme that is described later in this section.

A *rendezvous* protocol is designed to transfer long messages in a zero-copy manner using RDMA write operations. As shown by the right diagram of Fig. 3, for a long message (a message that is longer than a single memory buffer), a *rendezvous* request (*req*) is sent to the receiver. The receiver returns an acknowledgment (*ack*) to the sender when it matches a receive operation with this request. The sender then sends the long message with a RDMA write operation. At the completion of RDMA write, a *done* message is sent to the receiver as transfer completion notification. Except for three control messages (req, ack, and done), the actual data for a long message is sent without any intermediate buffering through zero-copy RDMA write operations.

4.2. Noncontiguous data transfer

As noncontiguous I/O is a common access pattern for scientific applications, it is crucial for GridFTP to provide an efficient solution as a general-purpose data transportation mechanism. On the other hand, InfiniBand supports scatter/gather in its NIC hardware, which matches the characteristic of noncontiguous I/O. Scatter/gather is provided in both InfiniBand channel semantics and memory semantics. In the channel semantics, at the sender side, InfiniBand NIC hardware packs a list of source data segments on the fly and inject them into the network; at the receiver side, NIC drains the data from the network, and correspondingly unpacks them into a list of destination memory segments. In contrast, in the memory semantics, data can be directly moved between source and destination memory. It allows much simpler implementation in which only one side is needed to initiate the data movement, either store data to or load data from, remote memory. Two different operations are available in memory semantics, RDMA write and RDMA read. They work only with non-contiguous data segments in the local memory. In our prototype, we use RDMA write for noncontiguous data transfer. It gathers data from multiple segments and writes them into a remote contiguous memory segment.

RXIO is designed with two different methods for noncontiguous I/O. In the first method, noncontiguous I/O was implemented through data packing and unpacking, in which data segments are packed into a contiguous memory region on the sender side. The packed data are then sent over to another contiguous memory region at the receiver side, wherein they are scattered to the list of targeted memory segments. In the second method, RXIO takes advantage of the NIC-level support and implements noncontiguous I/O for GridFTP with zero-copy RDMA write. Fig. 4 shows the diagram of GridFTP noncontiguous I/O using zero-copy RDMA write on InfiniBand. Since RDMA write only gathers local segments and delivers data into a single segment in the remote memory, we invoke an RDMA write operation for each destination segment. Data segments in the source memory are sequentially distributed into these destination segments. Such RDMA write operations are posted together and followed by an RDMA send operation to notify the remote side about the completion of data transfer.

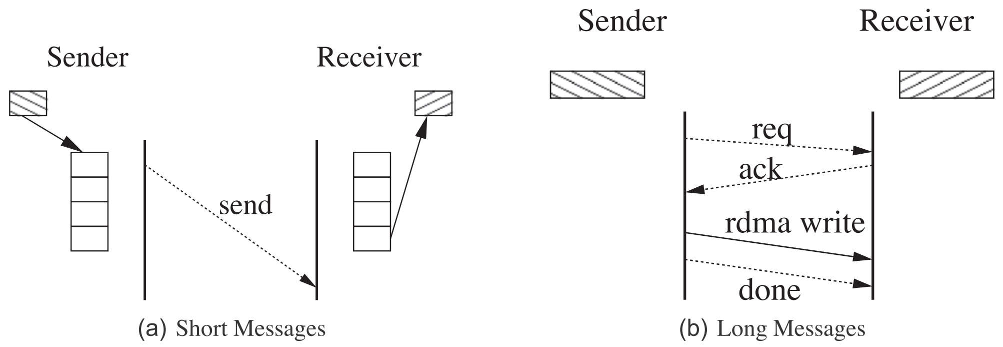


Fig. 3. RXIO data transfer on InfiniBand.

4.3. Credit-based buffer management

Short messages in the eager protocol and control messages in the rendezvous protocol require the availability of memory buffers, one per message, at the receiver. Otherwise, the InfiniBand network will retry for a few times and eventually cause a retry error if a buffer is not posted in time. Ideally, the number of memory buffers should be only limited by the available memory. In practice the RDMA driver shall not exhaust the memory and a sender shall not be allowed to swamp the receiver. We manage the memory buffers using a credit scheme. A pair of client and server start with an adjustable number of fixed-size buffers. One buffer is reserved for communicating the status of credits under urgent scenarios. So the number of credits is always set to be one less than the available buffers. A sender reduces its credits for every short or control message. A receiver accumulates new credits when buffers are recycled. The number of new credits is normally piggybacked and returned to the sender when both sides are actively exchanging messages. But the communication may be unidirectional at times and one GridFTP endpoint can be predominantly receiving. In this case, the receiver then informs the sender about the available new credits when half of the memory buffers are recycled. The reserved buffer is used for this purpose. With this credit scheme, a sender cannot overwhelm the receiver in a pathological situation. At the same time, a receiver can keep replenishing the sender with credits for an uninterrupted sequence of transmissions, so long as the receiver can recycle its buffers sufficiently fast. In addition, the reserved memory buffer offers an exit channel for credit synchronization, and breaks possible deadlocks that may be caused by the lack of credits.

5. Performance evaluation

Our experiments are conducted on systems with a variety of different configurations. Table 1 shows the configuration details. We primarily utilize a pair of Type-III hosts with 2.1 GHz 64-bit quad-core Intel Harpertown processors. Each node is equipped with 8x PCI-Express Gen 2.0 bus. These hosts run Linux 2.6.18 kernels. They are also equipped with 1-Gigabit Ethernet (1GigE) and Myricom 10-Gigabit Ethernet (10GigE) cards. For the sake of comparison with other configurations, we utilize two pairs of nodes with Types-I and II configurations, respectively. Type-I nodes have a single-core single-socket 32-bit Intel Xeon processor and a PCI-X 2.0 bus, running Linux-2.6.9. Type-II nodes have quad-core 64-bit Intel Harpertown processors and a 8x PCI-Express Gen 2.0 bus, running Linux-2.6.18. Types-I and II nodes are connected to a Mellanox 144-port InfiniBand DDR switch, through single data rate (SDR) and dual data rate (DDR) Host Channel Adaptors (HCA), with peak bandwidths of 8 Gbps and 16 Gbps, respectively. Type-III nodes are equipped with InfiniBand QDR (quad data rate) cards that are connected to a Mellanox 36-port QDR switch. QDR cards have a peak bandwidth of 32 Gbps.

5.1. Latency and bandwidth

We have developed a set of latency and bandwidth benchmarks to measure the performances of GridFTP I/O operations under different protocols. RXIO is enabled through calling the standard OCRW interface inherited from XIO, the same way as application invoking XIO drivers. Both benchmarks require a pair of server and client on separated nodes for data transportation purpose. The latency test is a program that measures half of the round-trip message latency between an RXIO server and a client. The bandwidth test is a program that measures data-transfer bandwidth when a client (sender) sends 1000 messages to the server (receiver). In all tests, 50 initial iterations are executed before measurements are taken. We use

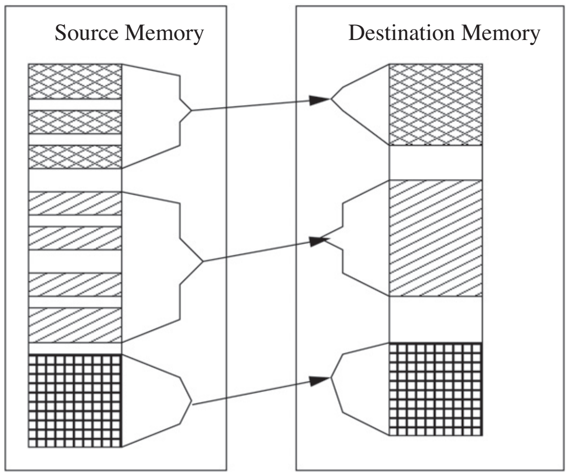


Fig. 4. Noncontiguous data transfer using zero-copy RDMA write.

Table 1
System configurations.

	Hosts	Processors	Ethernet NIC	PCI bus	HCA
I	single-core single-socket	2.6 Xeon 32-bit	Gigabit	PCI-X 2.0	SDR
II	quad-core dual-socket	2.4 GHz Xeon 64-bit	Gigabit	PCI-Express 2.0	DDR
III	quad-core dual-socket	2.1 GHz Xeon 64-bit	Gigabit & Myricom 10GigE	PCI-Express 2.0	QDR Connect X

the emulated IP implementation (IPoIB) for TCP tests. By using the same underlying InfiniBand network, the comparison of GridFTP performance is then focused on protocol differences.

Fig. 5 shows the latency performance of GridFTP using TCP and RDMA on InfiniBand. Compared to the TCP driver (with IPoIB), RXIO with RDMA reduces the message latency for small messages by more than 10 μs, a 32% improvement. This demonstrates that the RXIO/RDMA driver can significantly speed up GridFTP I/O. Note that GridFTP detects message completion via the event-based mechanism. Its basic latency on InfiniBand (20 μs) includes the interrupt overhead, and therefore is higher than the best microsecond-level latency that can be achieved via polling based completion detection [41].

Fig. 6 shows the bandwidth performance of RXIO on the InfiniBand network with different GridFTP I/O drivers. Compared to the TCP driver (with IPoIB), RDMA improves the peak bandwidth by more than three times. For mid-range messages, RDMA performs worse. This is because the choice of transition point at 8 KB from the eager protocol to the *rendezvous* protocol, in which more control messages have to be communicated. As described later in Section 5.4, tuning on the message buffer size changes the comparison and remedies this performance behavior.

5.2. Comparison to Gigabit and 10-GigaBit Ethernet

Using the same latency and bandwidth programs, we evaluate the performance of GridFTP on different networks. Type-III hosts are used in this set of experiments with the 1-Gigabit Ethernet (1GigE) network cards and 10-Gigabit Ethernet (10GigE) cards.

Fig. 7 shows the latency performance of RXIO on different networks. Similar to the earlier latency comparison, RDMA reduces the message latency for small messages by more than 10 μs, compared to 1GigE and 10GigE. For large messages, RDMA cuts down the latency by 4 times compared to 10GigE, and much more compared to 1GigE. Note that the Myricom 10GigE network cards by default are enabled with interrupt coalescing, which adds 75 μs before an interrupt can be triggered. This leads to in an increased latency. Interrupt coalescing is disabled for measuring the latency of 10GigE.

Fig. 8 shows the bandwidth performance of RXIO on different networks. For these tests, 10GigE cards are enabled with interrupt coalescing for best bandwidth results. 1GigE performs well for small messages but soon reaches its peak at 118 MB/s. 10GigE reaches a peak bandwidth of 1184 MB/s, while RDMA improves the peak bandwidth up to 3018 MB/s. For mid-range messages, RDMA performs worse. This again is because the choice of transition point at 8 KB between the eager and the *rendezvous* protocols. As described later in Section 5.4, tuning on the message buffer size changes the comparison and corrects this performance behavior.

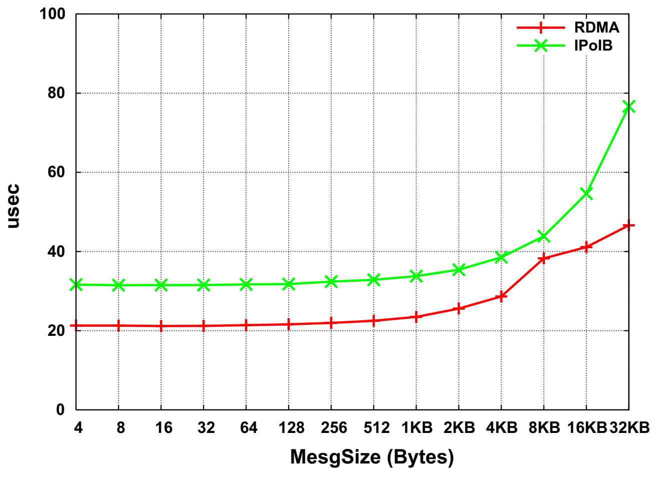


Fig. 5. The latency of RXIO.

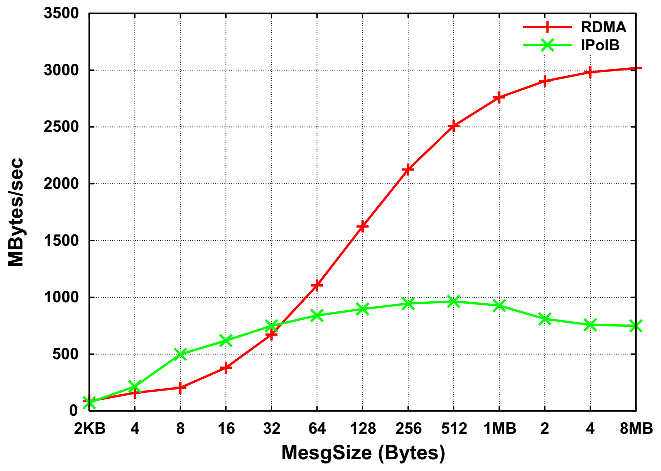


Fig. 6. The bandwidth of RXIO.

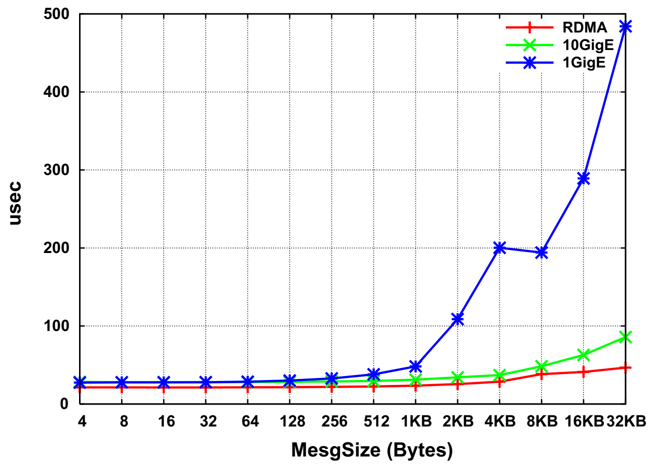


Fig. 7. Latency comparison of different networks.

5.3. Performance with different InfiniBand host channel adaptors

To examine the performance of RXIO on different generations of InfiniBand Host Channel Adaptors (HCAs), we use all three types of hosts as detailed in Table 1. Three generations of InfiniBand HCAs are tested, including single data rate (SDR), double data rate (DDR), and quad data rate (QDR) Connect-X HCAs. Figs. 9 and 10 show the comparisons of latency and bandwidth, respectively. Type-I hosts have the rather old single-core Intel Xeon processor. Their latency is particularly high compared to DDR and QDR, due to less capable SDR cards and the threads that compete for the only CPU. Not surprisingly, QDR performs the best in terms of both latency and bandwidth, but SDR and DDR still reach the peak bandwidth on their respective systems.

5.4. Tuning RXIO on InfiniBand

As described in Section 4, the RDMA driver for RXIO is implemented with different protocols for short and large messages. The threshold between the two is determined by the size of message that can fit into a receiver buffer. To examine the impact of this threshold, we measure the bandwidth of RXIO using different buffer sizes. Fig. 11 shows the bandwidth comparison for medium messages around the threshold. We choose only medium messages because only their bandwidths are affected by the threshold value. As shown in the figure, the best performance for medium messages can be obtained when the buffer size is tuned at 32 KB. This effectively changes the earlier bandwidth comparisons for medium messages shown in Figs. 6 and 8.

5.5. Performance of noncontiguous data transfer

We conduct an experiment to measure the performance of GridFTP noncontiguous I/O operations. The data-transfer bandwidth is measured when a client (sender) sends 2000 messages to the server (receiver). Each message is divided

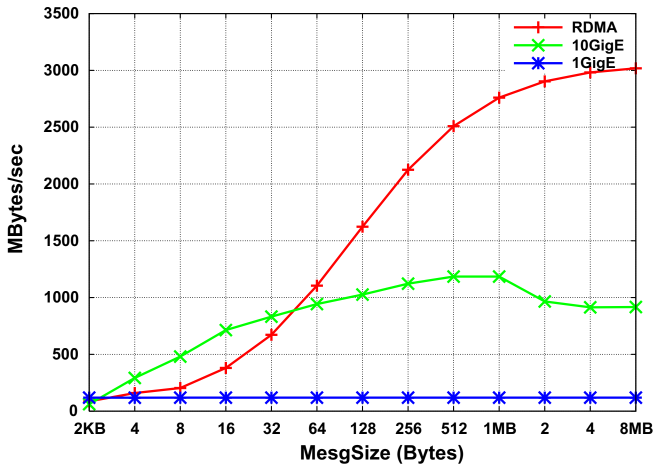


Fig. 8. Bandwidth comparison of different networks.

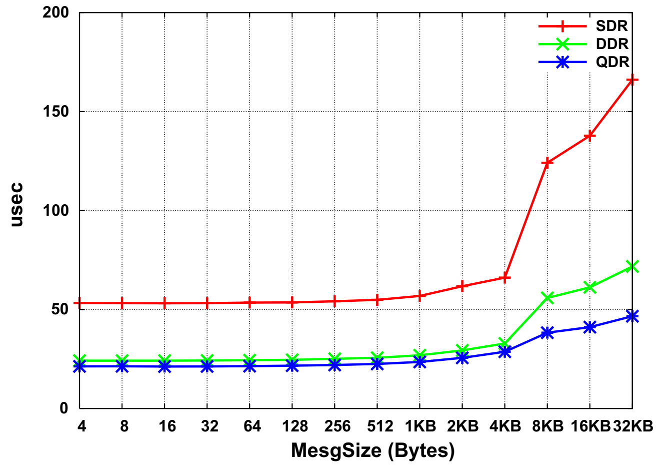


Fig. 9. Latency comparison of different HCAs.

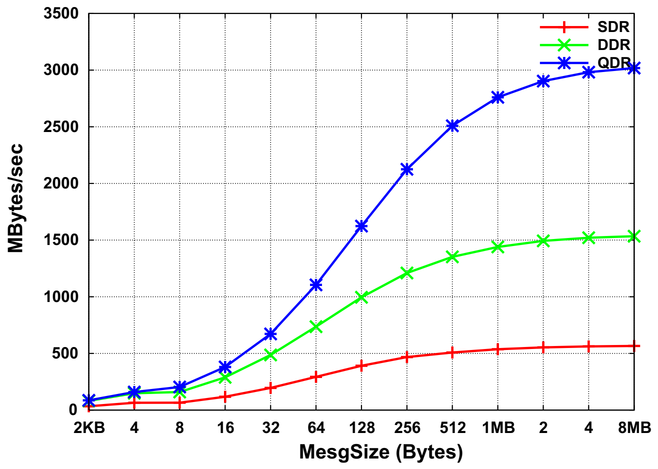


Fig. 10. Bandwidth comparison of different HCAs.

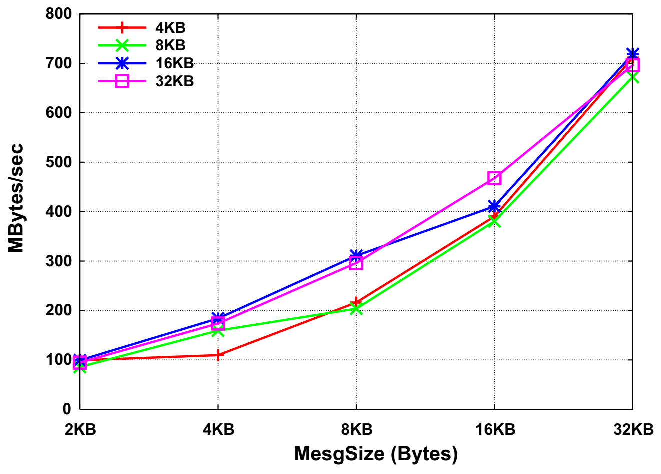


Fig. 11. Bandwidth of RXIO with different buffer size.

into 16 segments, separated by 1 MB apart. In all tests, 50 initial iterations are executed before measurements were taken.

Fig. 12 shows the bandwidth of RDMA write-based noncontiguous data transfer in RXIO on the InfiniBand network. We compare this to a simple packing/unpacking based implementation for noncontiguous data. As shown in the figure, zero-copy RDMA improves the bandwidth for noncontiguous I/O by up to three times. Due to the high demand on memory bandwidth, the packing/unpacking method is peaked at 1500 MB/s, and then tailed off for messages bigger than 2 MB. To show the effectiveness of noncontiguous I/O, we also measure the bandwidth of transferring contiguous messages of the same size. As shown in the figure, our zero-copy noncontiguous I/O is able to achieve the performance close to that of contiguous I/O. Among all the message sizes, the efficiency stays above 70%, and more than 95% for large messages.

5.6. CPU utilization

Another significant strength of RDMA is their benefit in reducing the CPU involvement in network communication. We extend the bandwidth test with the capability to report CPU utilization. To this purpose, we increase the number of messages from 1000 to 1,000,000. This prolonged bandwidth test does not improve the bandwidth numbers much, but it allows us to measure the CPU utilization for every message size. We record the time stamp counter (TSC) before launching and after completion of the measurement for each message size. Then we record the increment of CPU utilization during the same period. The ratio of these two is taken as the percentage of CPU utilization. This measurement is conducted on both the sender side and the receiver side. Type-III nodes are used for these experiments. As we use our local network for the experiment, interference is minimized to ensure the accuracy.

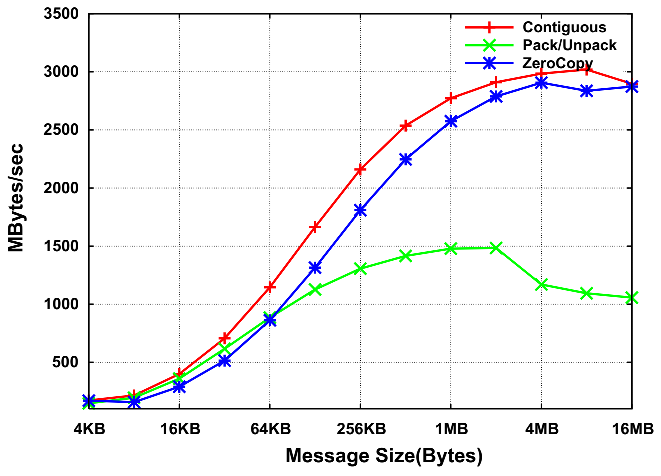
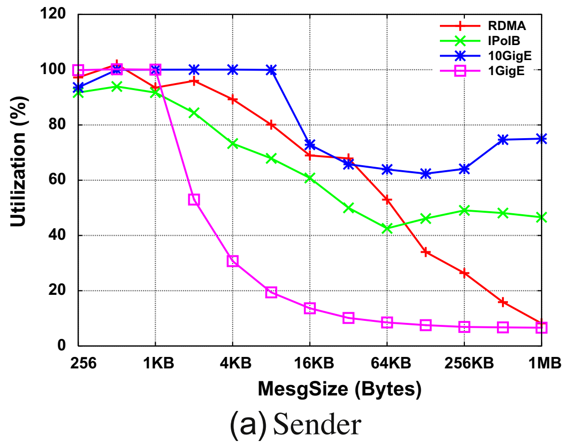


Fig. 12. Benefits of zero-copy RDMA write for noncontiguous data transfer.



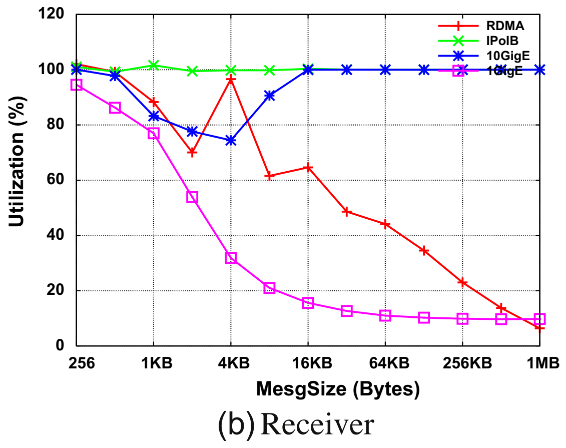


Fig. 13. Comparison of CPU utilization.

Fig. 13 shows the comparison of GridFTP CPU utilization on top of 1GigE, 10GigE, IPoIB and RDMA. Figs. 6 and 8 can be referred to for information on the corresponding GridFTP bandwidths. For small messages, both the sender's and receiver's CPUs are fully utilized because the network is fast enough to keep up with the processing speed of CPU. The sender uses slightly less than 100% CPU to keep the receiver busy. The reason of higher CPU utilization at the receiver side is because of the heavier processing overhead that is caused by message matching and data copying (for small messages in the case of InfiniBand). As the message size increases, the sender and the receiver on 1GigE gradually become more idle. This is because the network is slow. It takes longer time for big size messages to go through the network. The number of messages communicated per second becomes smaller, therefore CPU utilization is lower. In the cases of IPoIB and 10GigE, the network is fast enough to keep the CPU busy. The receiver side CPU is particularly saturated because of the heavier processing overhead, while the sender side only needs 40–50% and 60–75% CPU, respectively for IPoIB and 10GigE, to keep the receiver busy. In contrast to all these three, GridFTP with RDMA uses much less CPU on both the sender and the receiver sides, while at the same time it is able to transmit messages to exploit the best network bandwidth. A spike is observed at the server side when message size is 4 KB, where communication and processing overhead reaches the peak for small messages. By switching to *rendezvous* mode at 8 KB, CPU utilization immediately decreases as *rendezvous* involves much less communication overhead.

These results adequately demonstrate the effectiveness of RXIO in making use of the latest network technologies such as RDMA. They also showcase the benefits of RDMA in increasing I/O bandwidth for different I/O patterns.

6. Conclusions

RDMA networks provide low-latency, high bandwidth communication with low CPU utilization. In this paper, we have described the design of RXIO and the RDMA driver that enables GridFTP on RDMA networks such as InfiniBand [1], iWARP [5], and RDMAoE (RDMA over Ethernet) [40]. RXIO extends XIO with additional features that match the characteristics of RDMA networks. In particular, RXIO is designed to match RDMA connection management and communication progress semantics and yet conforms to the original XIO framework. An example implementation on InfiniBand is accordingly provided. The I/O operations in GridFTP are efficiently mapped to RDMA data transfer operations. An RDMA write-based zero-copy noncontiguous data transfer is also implemented in this prototype. A credit-based buffer management is developed to prevent a sender from inundating a slow receiver. Our experimental results demonstrate that, compared to IPoIB and 10GigE, RDMA can significantly improve the performance of GridFTP, reducing the latency by 32% and increasing the bandwidth by more than three times. While achieving such performance improvements, out RDMA driver is able to dramatically cut down CPU utilization for both GridFTP clients and servers.

We plan to study further optimizations of GridFTP on different RDMA networks. Particularly, when long-distance RDMA networks become available, we will carry out an extensive evaluation and tuning of this RXIO-based GridFTP. We also plan to study adaptive mechanisms in selecting the best networking drivers for optimal GridFTP performance in different network environments, or different combinations of them.

Acknowledgments

This work was funded in part by a UT-Battelle grant (UT-B-4000087150) to Auburn University, and in part by a National Science Foundation award CNS-1059376. This research was also sponsored in part by the Office of Advanced Scientific Computing Research; US Department of Energy. This research used resources of the Experimental Computing Laboratory at Oak Ridge National Laboratory, which is supported by the Office of Science of the US Department of Energy under Contract No.

DE-AC05-00OR22725. We are very thankful for an InfiniBand equipment donation from HPC Advisor Council to Auburn University. Portions of this research were also conducted with high performance computational resources provided by the Louisiana Optical Network Initiative (http://www.loni.org). We are very grateful for the technical support from the LONI team.

References

[1] Infiniband Trade Association. http://www.infinibandta.org.
[2] Hurwitz J, Feng W. End-to-end performance of 10-Gigabit Ethernet on commodity systems. IEEE Micro '04.
[3] Feng W, Hurwitz J, Newman H, Ravot S, Cottrell L, Martin O, et al. Optimizing 10-Gigabit Ethernet for networks of workstations, clusters and grids: a case study. In: SC '03.
[4] Recio R, Culley P, Garcia D, Hilland J. An RDMA protocol specification (version 1.0), October 2002.
[5] Romanow A, Bailey S. An overview of RDMA over IP. In: Proceedings of international workshop on protocols for long-distance networks (PFLDnet2003), 2003.
[6] Liu J, Wu J, Panda DK. High performance RDMA-based MPI implementation over InfiniBand. Int J Parallel Programm 2004;32.
[7] Farreras M, Almasi G, Cascaval C, Cortes T. Scalable RDMA performance in PGAS languages. Parallel Distrib Process Sympos Int 2009;0:1–12.
[8] DeBergalis M, Corbett P, Kleiman S, Lent A, Noveck D, Talpey T, et al. The direct access file system. In: Proceedings of second USENIX conference on file and storage technologies (FAST '03), March 2003.
[9] Callaghan B, Lingutla-Raj T, Chiu A, Staubach P, Asad O. NFS over RDMA. In: Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence. ACM Press; 2003. p. 196–208.
[10] Wu J, Wychoff P, Panda DK. PVFS over InfiniBand: design and performance evaluation. In: Proceedings of the international conference on parallel processing '03. Kaohsiung, Taiwan, 2003.
[11] Yu W, Liang S, Panda DK. High performance support of parallel virtual file system (PVFS2) over quadrics. In: Proceedings of international conference on supercomputing (ICS'05), 2005.
[12] Ko M, Chadalapaka M, Elzur U, Shah H, Thaler P, Hufferd J. iSCSI extensions for RDMA specification. http://www.ietf.org/internet-drafts/draft-ietf-ips-iser-05.txt.
[13] Chadalapaka M, Shah H, Elzur U, Thaler P, Ko M. A study of ISCSI extensions for RDMA (ISER). Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence (NICELI) 2003:209–19.
[14] Dalessandro D, Devulapalli A, Wyckoff P. iSER storage target for object-based storage devices. Proc SNAPI Workshop 2007;0:107–13.
[15] http://www.obsidianresearch.com/. Obsidian Research Corporation.
[16] Network Equipment Technologies, http://www.net.com.
[17] Rao NSV, Hicks WRWSE, Poole SW, Denap FA, Carter SM, Wu Q. Ultrascience net: high-performance network research test-bed. In: International symposium on computer and sensor network systems, 2008.
[18] Yu W, Rao N, Vetter J. Experimental analysis of infiniband transport services on WAN. In: International conference on networking, architecture and storage, 2008.
[19] Rao NSV, Yu W, Wing WR, Poole SW, Vetter J. Wide-area performance profiling of 10gige and infiniband technologies, November 2008.
[20] GT 4.0 GridFTP. http://www.globus.org.
[21] bbcp. http://www.slac.stanford.edu/abh/bbcp/.
[22] Internet2. http://www.internet2.edu.
[23] Energy Sciences Network. http://www.es.net.
[24] Rao N, Wing W, Carter S, Wu Q. Ultrascience net: network testbed for large-scale science applications. Commun Mag IEEE 2005;43(11):S12–7.
[25] End-To-End Provisioned Optical Network Testbed for Large-Scale eScience Application. http://www.ece.virginia.edu/mv/html-les/ein-home.html.
[26] Foster I, Kesselman C. Globus: a metacomputing infrastructure toolkit. Int J High Perform Comput Appl 1997;2:115–28.
[27] Allcock WE, Bresnahan J, Kettimuthu R, Link JM. The globus extensible input/output system (xio): a protocol independent IO system for the grid. In: IPDPS, 2005.
[28] Gu Y. UDT: a high performance data transport protocol. Ph.D. thesis. Chicago, IL, USA: Chairperson-Grossman, Robert L.; 2005.
[29] Bresnahan J, Link M, Kettimuthu R, Foster I. Udt as an alternative transport protocol for gridftp. In: Proceedings of the second international workshop on protocols for fast long-distance networks, 2009.
[30] Allcock W, Bresnahan J, Kettimuthu R, Link M. The globus striped gridftp framework and server. SC'05 2005;0:54.
[31] Kissel E, Swany M, Brown A. Improving gridftp performance using the phoebus session layer. In: SC09: Proceedings of the conference on high performance computing networking, storage and analysis. New York, NY, USA: Springer; 2009. p. 1–10.
[32] Ito T, Ohsaki H, Imase M. Gridftp-apt: automatic parallelism tuning mechanism for data transfer protocol gridftp. IEEE Int Sympos Cluster Comput Grid 2006;0:454–61.
[33] Zhou Y, Bilas A, Jagannathan S, Dubnicki C, Philbin JF, Li K. Experiences with VI communication for database storage. In: Proceedings of the 29th annual international symposium on computer architecture. IEEE Computer Society, 2002. pp. 257–68.
[34] Carns PH, Ligon III WB, Ross RB, Thakur R. PVFS: a parallel file system for linux clusters. In: Proceedings of the 4th annual linux showcase and conference. Atlanta, GA 2000. p. 317–27.
[35] Talpey T, et. al. NFS/RDMA ONC transport. http://sourceforge.net/projects/nfs-rdma.
[36] Cater S, Minich M, Rao N. Experimental evaluation of infiniband transport over local and wide-area networks. In: High performance computing symposium (HPC'07), September 2008.
[37] Narravula S, Subramoni H, Lai P, Noronha R, Panda DK. Performance of HPC middleware over infiniband wan. In: Proceedings of international conference on parallel processing (ICPP' 08), September 2008.
[38] Lai P, Subramoni H, Narravula S, Mamidala A, Panda D. Designing efficient FTP mechanisms for high performance data-transfer over infiniband. In: Proceedings of international conference on parallel processing (ICPP' 09), September 2009.
[39] Subramoni H, Lai P, Kettimuthu R, Panda DK. High performance data transfer in grid environment using gridftp over infiniband. IEEE Int Sympos Cluster Comput Grid 2010;0:557–64.
[40] Technologies M. RDMA over Ethernet (RDMAoDCB) A Technology Brief. http://www.ethernetalliance.org/.
[41] Liu J, Chandrasekaran B, Yu W, Wu J, Buntinas D, Kini SP, et al. Micro-benchmark performance comparison of high-speed cluster interconnects. IEEE Micro 2004;24:42–51.

Yuan Tian is a Ph.D. student in the Department of Computer Science and Software Engineering at Auburn University. She earned her bachelors degree from Chengdu University of Technology, Chengdu, China. Prior to enter Auburn, she worked as a Software Engineer in both China and Japan. She holds a masters degree in Computer Science from Auburn University. Her research interests mainly focus on High Performance Computing, High Speed Networking, Network and Grid Computing, File Systems, and Parallel IO. She is a Research Assistant advised and supported by Dr. Weikuan Yu. Tian is a member of ACM, IEEE.

Weikuan Yu is currently an Assistant Professor in the Department of Computer Science and Software Engineering at Auburn University. He leads the Parallel Architecture and System Laboratory for research on high-end computing, parallel and distributing networking, storage and file systems, as well as interdisciplinary topics on computational biology. Prior to joining Auburn, he served as a Research Scientist for two and a half years at Oak Ridge National Laboratory. He is also a Joint Faculty at ORNL. He earned his Ph.D. in Computer Science from the Ohio State University in 2006. He is a member of AAAS, ACM, and IEEE.

Jeff S. Vetter is a computer scientist in the Computer Science and Mathematics Division (CSM) of Oak Ridge National Laboratory (ORNL), where he leads the Future Technologies Group and directs the Experimental Computing Laboratory. He is also a Joint Professor in the College of Computing at the Georgia Institute of Technology, where he earlier earned his Ph.D. He joined ORNL in 2003, after four years at Lawrence Livermore National Laboratory. His interests span several areas of high-end computing – encompassing architectures, system software, and tools for performance and correctness analysis of applications.