2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum

Middleware Support for RDMA-based Data Transfer in Cloud Computing

Yufei Ren, Tan Li, Dantong Yu, Shudong Jin, Thomas Robertazzi
Department of Electrical and Computer Engineering
Stony Brook University, Stony Brook, NY, 11794
Email: yufren@ic.sunysb.edu, tanli@ic.sunysb.edu, dtyu@bnl.gov, shujin@notes.cc.sunysb.edu, tom@ece.sunysb.edu

***Abstract*—Providing high-speed data transfer is vital to various data-intensive applications in cloud computing systems. We design a middleware layer of high-speed communication based on Remote Direct Memory Access (RDMA) that serves as the common substrate to accelerate various data transfer applications in cloud computing, such as FTP, HTTP, file copy, sync and remote file I/O. This middleware offers higher end-to-end bandwidth than the traditional TCP-based alternatives, while it hides the heterogeneity of the underlying high-speed architecture. In this paper, we describe the design of this middleware, including resource abstraction, and task synchronization and scheduling. We provide a reference implementation of the file-transfer protocol over this RDMA-based middleware. Our experimental results show that it outperforms several TCP-based FTP tools, such as GridFTP, while maintaining very low CPU consumption on a variety of platforms. Furthermore, these results confirm that our middleware achieves near line-speed performance in both LAN and MAN, and scales consistently from 10Gbps Ethernet to 40Gbps Ethernet and InfiniBand environments.**

***Keywords*-Data Center Networks, Middleware, Remote Direct Memory Access, Protocol offloading**

I. Introduction

Data-intensive applications in grid and cloud computing environment are expected to generate extremely high-volume data in the future. The data is usually transferred, visualized, and analyzed by geographically distributed teams of users. High-performance network capabilities must be available to these users to support these applications.

Protocol offload and hardware acceleration are among the most desired techniques to achieve high data transfer rate with marginal host resource consumption. The TCP/IP Offload Engine (TOE) is one of the early examples of protocol offload to meet above requirements. The idea of TOE is to use a dedicated hardware module on a network adaptor card to process the TCP/IP internal operations, such as segmenting, framing, reassembling the payload, timing, and flow control. Research and implementation works [1], [2] have shown that TOE is a cost-effective technique to free the host processors from onerous protocol processing, and therefore enhance the concurrency between communication and computation. Thereafter, Remote Direct Memory Access (RDMA) is proposed as a hardware-based Protocol Offload Engine (POE) realization to move bulk data from the source memory directly to the remote host memory with kernel-bypass and zero-copy operations. Its use has recently become popular when converged Ethernet and data center bridge technologies were implemented.

In addition to achieving near line-speed data transfer, another challenge is to manage a heterogeneity of underlying RDMA architectures for applications. Various RDMA implementations offer opportunities to enhance the performance of data transfer service, and overcome the limitations of kernel-based TCP/IP realization [3], [4]. However, despite of the emergence of industry standards such as OpenFabrics Enterprise Distribution (OFED) [5], it could be a distraction if the applications have to manage the heterogeneous devices.

In this paper, we describe the design of a middleware to support RDMA-based data transfer for grid and cloud computing applications. This middleware integrates network access, memory management, and multitasking. We address various issues related to its efficient implementation, for instance, buffer management and memory registration, and parallelization of RDMA operations, that are vital to delivering the benefit of RDMA to the applications. Built on top of this middleware, an implementation of a RDMA-based FTP software, RFTP, is described. This application has been implemented by our team to exploit the full capabilities of advanced RDMA mechanisms for ultra-high speed bulk data transfer applications on U.S. Department of Energy, Energy Sciences Network (ESnet) [6]. Our contributions include,

- We design the core of our middleware that offers data transfer and access primitives. This core is designed to have a multi-threaded architecture and facilitates multi-stream data transfer to exploit parallelism of RDMA operations. We implement task synchronization mechanisms to allow maximum buffer reuse, and to minimize synchronization and RDMA connection setup cost. Thus, it serves as a reliable base for the development of data transfer applications in grid and cloud computing systems.
- We implement and test a RDMA extension for the File Transfer Protocol defined in RFC 959. The experiments are done on actual Linux cluster over a variety of testbed platforms and networks, and the results validate our middleware design.

The rest of this paper is organized as follows. Background information of our work is presented in Section II. Sec-

978-0-7695-4676-6/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPSW.2012.133

tion III describes the design of our middleware layer, and Section IV briefly describe RFTP design. In Section V, we describe the hardware and software setup of our evaluation platforms and present the experiment results. A conclusion of this paper is given in Section VI.

II. Background

The performance benefit of RDMA technology for high performance computing has attracted substantial interests in both academia and industry. The original RDMA architecture, InfiniBand [7], supports top-down RDMA message service with its own layer-4 to layer-2 implementation of the OSI protocol stack. Unlike the best-effort frame delivery service in Ethernet, the link layer of InfiniBand provides reliability and in-order delivery through its credit-based flow control and virtual lane mechanism. Two other implementations, Internet Wide Area RDMA Protocol (iWARP) and RDMA over Converged Ethernet (RoCE), were proposed to extend the advantages of InfiniBand to ubiquitous IP/Ethernet-based networks and integrate the IP networks with RDMA mechanisms. iWARP offloads the whole TCP/IP stack, and transfers data in user-space buffer directly to remote application memory. RoCE techniques allow the IB transport protocol to run over Ethernet.

One objective of our middleware is to support applications across all aforementioned RDMA architectures. We build our middleware with the common Verb Application Programming Interface (API) in OpenFabrics Enterprise Distribution (OFED) [5], a unified, cross-platform, transport-independent software stack for RDMA. The OFED offers a uniform application programming interface, native IB verbs, to access various RDMA architectures. The OFED software also offers several middleware packages to allow socket based applications running over RDMA devices without rewriting the program. The User Direct Access Programming Library (uDAPL) [8] also provides RDMA capabilities for applications, and has been used in other studies [9], [10]. These extensions introduce additional overhead and performance penalties compared to the native RDMA IB verbs interface [11].

RDMA provides two message transfer semantics: channel semantic and memory semantic. The channel semantic, SEND/RECEIVE, is also referred as two-sided operation in RDMA since the kernels at both the data source and the sink are involved in the data transfer after a connection is established [12]. The communication channel between the source and the sink is modeled as a queue pair (QP). Each QP consists of one send queue and one receive queue, and each queue represents one end of the channel. Before a data transfer, the receiver posts a work request to the receive queue, and the sender can post a work request to the send queue. Both the send and receive sides will get a completion event after the data transfer is finished. On the other hand, the memory semantic, or RDMA READ/WRITE, is regarded as one-sided operations. The receiver advertises its available registered memory to the sender, including the key of memory region and the address, so that the sender can directly RDMA WRITE data to the specified memory space at the receiver.

Our middleware uses the one-sided WRITE operation for its better performance and lower synchronization cost. The use of RDMA two-sided operations can be found in [11]. Their FTP implementation is based on two-sided zero-copy operations in IB networks. In addition, studies [13], [14] have shown that even though there are some benefits of using RDMA over LAN with short latency, there are challenges to achieve good performance in WAN with a long latency due to the low performance issue with RDMA READ operation and the lack of RDMA capability in handling non-contiguous data.

III. Middleware Design

We design a middleware layer between applications and the RDMA interface, and target at making this layer a general architecture convenient for developing various applications that can take advantage of RDMA techniques. In this section, we describe the function and design of this middleware.

A. Overview of the Architecture

The middleware layer, as shown in Figure 1, implements a set of function modules, and provides an abstraction of the computational resources including main memory and network cards. The middleware layer contains two primary sections: one data structure section, which is used to keep track of data structure necessary for data communication and memory access, and one thread pool, which implements all function modules related to data communication, synchronization, and task scheduling. These data structures and threads are all pre-allocated and reused to avoid the overhead of resource allocation and release during program execution.

The middleware interacts with the host computer adaptor (HCA) via an array of queue pairs, which are supported by the OpenFabrics standard. A separate queue, the completion queue (CQ), is also maintained by the same standard. The threads in the middleware layer gain access to the queue pairs and the completion queue via standard programming interface.

B. Resources Abstraction and Management

The middleware accesses the local and remote memory by maintaining several data block/message lists. This data structure is maintained and updated by the threads, according to certain communication semantics to be described later. This data structure contains:

- A data block is used to contain user payload data, and it is temporarily kept in the data block list before the

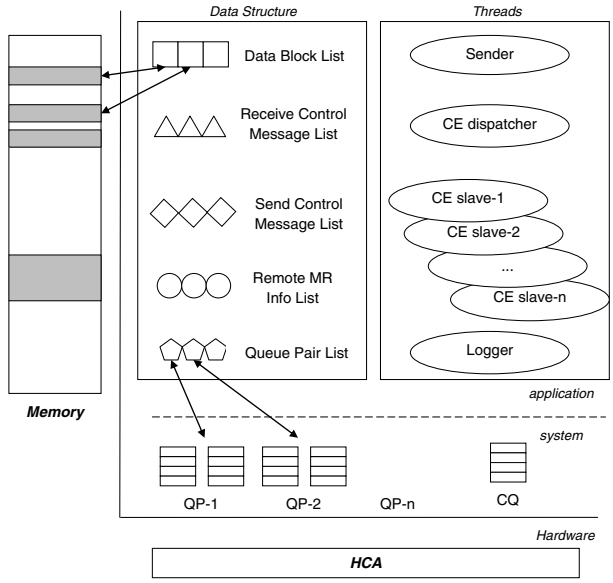

Figure 1. Multi-threaded architecture and data structure of middleware

actual delivery. A user can copy data into the block or extract data from it.

- The send and receive control message lists are used to keep outgoing and incoming control messages, respectively.
- The remote memory region information is used to store the memory region information such as key, logical buffer address, and maximum length of acceptable payload at the remote side of the communication.
- A queue pair list is maintained to keep track of the status of the actual queue pairs implemented in the underlying OpenFabrics standard software.

These resource abstractions are managed and maintained by a pool of threads that implement the communication semantics. There is one sender thread, which posts control message tasks or payload tasks into the send queue of the queue pairs. Once RDMA WRITE operation is completed, a completion event (CE) is generated. A master-slave thread pool handles various types of completion events as follows. The master, i.e., the CE dispatcher thread, on the detection of a completion event, encapsulates this event into a task structure and dispatch it to a CE slave. This CE slave then parses the task to determine the appropriate action, for example, updating the status of a data block. Finally, we have a logger thread which checks the status of ongoing data transfers, for example, whether the transfer is finished or aborted for any reason. The logger also monitors and reports the performance of data transfer.

C. Communication Semantics

Our middleware layer uses the channel semantic to exchange control messages and the memory semantic to deliver user data payload for low latency and high performance. A dedicated queue pair is used to exchange control messages between two communication parties, and a channel semantic is adopted for this communication. For data payload transfer, we allocate, possibly multiple, additional queue pairs, the number of which is user-configurable. Data packets are exchanged using these queue pairs.

There are four categories of control messages to support multi-task, parallel reliable data transfer through RDMA channel. We describe each of them as follows.

- *Session ID negotiation:* Before the data transfer, the middleware at the client side and server side will choose a unique session identifier for each transfer task.
- *Number of data connection negotiation:* The middleware is designed to support multiple parallel streams for a single data transfer. The client and server will exchange message to agree and establish the number of parallel connections.
- *Memory region block request and response:* The memory semantic requires one-sided operations, with which the active side acquires the memory region information of the passive remote side prior to actual data transportation. Before the client issues a RDMA WRITE, for example, it has to compose the sending task information such as the key of the remote memory region, the address of remote memory block, and the maximum allowed length of the remote block. When the server receives this memory region request, it searches for available memory regions for the client and sends back its information.

  Pre-request and batch-mode are used to reduce this control message overhead. “Pre-request” means the data source requests free memory regions before an actual data transfer task, and thus the data source could send data without the time-consuming memory region information exchange. “Batch-mode” means the data sink side feedbacks a list of free memory regions available at the time in a single response.
- *RDMA completion notification:* Since the middleware transfers user payload using one-sided RDMA WRITE, the data sink is not aware of the completion of the data transfer. Therefore, the data source should issue a notification to the data sink.

D. Parallel and Pipelined Data Transfer

The middleware implements an end-to-end, connection-oriented data transfer. It attempts to maximize the data transfer performance by exploiting parallelism of RDMA operations. We achieve this goal via two ways. First, we allow multiple active data streams. Second, each single data stream uses a pipelined execution, i.e., multiple data blocks can be posted before any of them is acknowledged.

In particular, the implementation of multiple active streams facilitates parallel data transfer between a single

pair of data source and sink. For instance, a single file can be partitioned (or striped), and each partition can be delivered using a single data channel, i.e., a stream. This implementation will potentially increase the data transfer rate for data-intensive applications. With this multi-stream transfer, many data blocks are transferred through queue pairs simultaneously. There is one minor problem related to this multi-stream data transfer, though, because it is possible data arrives at the sink out-of-order. For example, magnetic disk write performance would deteriorate during random access. To solve this problem, our middleware contains a plug-in module which gathers out-of-order blocks temporarily, until they can be delivered to the applications (for example, the disk writer here) in a sequential order.

IV. An Example Data Transfer Application

Our middleware facilitates the development of various data-intensive applications that rely on RDMA techniques for high-performance data transfer. In this section, we show one example, the design and implementation of our RDMA-based FTP service, RFTP.

A. RFTP Modules

The RFTP has a layered architecture as shown in Figure 2. This RFTP application is modified from the traditional FTP protocol. We support two modes: one is solely based on the conventional TCP/IP stack with optimized socket operations, and the other is based on our RDMA middleware layer as our extension to the standard FTP protocol. The middleware layer itself consists of several modules, such as buffer management and connection management.

This RDMA middleware is in turn supported by low layer protocols. It uses IB verbs through the InfiniBand, RoCE, and iWARP provided by InfiniBand Verbs library (libib-verbs) and RDMA communication manager (librdmacm). Thus, the middleware layer hides all the specifics related to the hardware to provide the desirable transparency, while the applications do not have to be aware of those specifics.

During the development of RFTP, we also notice that disk access is critical to the performance of the applications. Thus in our RFTP implementation, we design an additional disk I/O module based on direct I/O operations. This module includes an I/O scheduler, which is responsible for supporting disk readers/writers. The direct I/O access is particularly efficient for solid-state disks (SSD).

B. RDMA extension to standard FTP protocol

The standard FTP protocol implements a set of commands that are exchanged between the client and server entities. A control channel is used for exchanging the commands, while file content is transmitted over another data channel.

To support data transfer over our RDMA middleware, we extend the set of commands in FTP protocol. Table I lists the RDMA-extension FTP commands, RADR, RSTR, and RRTR, to support RDMA-enabled data transfer. They correspond to the PORT, STOR, and RETR commands in RFC 959 of FTP. These extensions explicitly request the remote side to adopt RDMA-based data transfer and therefore negotiate particular RDMA capabilities and ensure the compatibility between between sender and receiver. When these extended commands are exchanged, a RDMA control message channel and one or more RDMA data channels are initiated.

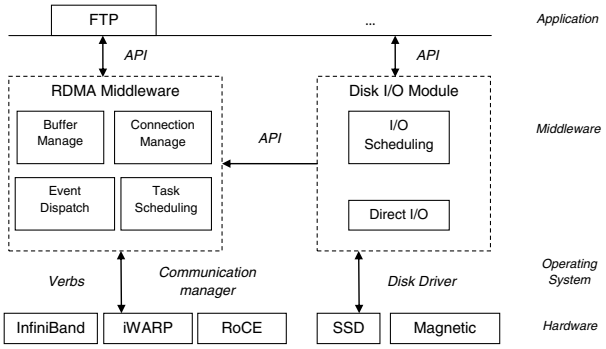

Figure 2. Modules in RFTP

Table I
RDMA-extension to standard FTP protocol

Command	Function and Procedure
RADR	RDMA address information exchange
RSTR	1) Establish RDMA data transfer channel; 2) Send data from client to server using RDMA operations, and store file at server side; 3) Tear down RDMA data transfer channel.
RRTR	1) Establish RDMA data transfer channel; 2) Get data from server to client using RDMA operations, and store the file at client side; 3) Tear down the RDMA data transfer channel

We also provide user commands, *rget* and *rput*, which are extensions to the original FTP commands. We provide a command line parameter to allow users to choose between two modes, i.e., the traditional FTP service that is built on top of the socket interface and the RFTP service that is built on top of our middleware. The advantages of extension instead of rewriting the entire FTP service include, 1)users can easily choose to use the one that is fit with their data transfer environment, and, 2) it is relatively easy to make FTP service available on these RDMA architectures.

V. Experimental Results

To validate our middleware, we conduct a comprehensive experimental study on actual test platforms. We first describe the test setup with difference RDMA architectures and network configurations. Then we compare the performance of RFTP with Netkit FTP and GridFTP.

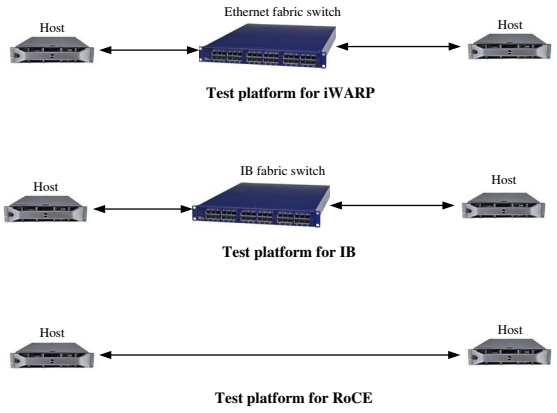

Figure 3. Test platform with different RDMA implementations in LAN

A. Testbed Setup

We consider both memory-to-memory and memory-to-disk data transfer between local and remote hosts. For memory-to-memory data transfer, we generate and transfer data between client memory and server memory. In this configuration, our focus is to evaluate the network bandwidth performance and protocol offload efficacy, but not the file system performance. For data-intensive applications it is a reasonable simplification to avoid the disk I/O bottleneck, and the applications usually use solid state disk arrays or distributed file systems to achieve superior disk I/O performance. In our experiments, the bandwidth performance only counts the pure payload without all the control messages. For the memory-to-disk data transfer experiments, we set up a disk system at the receiving server side using Fusion-io's solid state disk arrays.

1) High-bandwidth low-latency LANs: In order to provide an application-level performance test over different RDMA architectures, we setup two local-area test platforms. The first test platform is described in Figure 3. We have the client and server hosts set in the Brookhaven National Laboratory, and the propagation delay between them is less than 0.1ms. The detailed configurations for the nodes and switches are listed in Table II. Each host is equipped with a 40Gbps IB HCA and a 10Gbps iWARP HCA. The IB Mellanox RNIC we use can support both IB and RoCE.

The second test platform contains a 10Gbps link, between two hosts located in University of Michigan, Ann Arbor. We use this platform to evaluate and compare the performance of our RFTP and GridFTP. In order to maximize the performance of GridFTP over 10GB links, we tried several TCP parameter tunings, including setting the MTU at 9000, i.e., jumbo frames, according to [15]. The default congestion control algorithm, BIC, is used in the version of Linux we used. The extended block mode (MODE E) of GridFTP is enabled throughout the test for high performance. We did not enable the security feature of GridFTP for a fair comparison with our RFTP, which does not provide security function either.

Table II
LAN CLUSTER CONFIGURATION

Hardware	24 Intel(R) Xeon(R) CPU X5660 @ 2.80GHz cores 64GB memory 12288KB cache iWARP HCA: NetEffect NE020 10Gb Adapter IB HCA: Mellanox MT26428 ConnectX VPI PCIe IB QDR
Software	RHEL 5 with kernel-2.6.18 OFED Version 1.5.2 Netkit FTP version 0.17
Network	Mellanox MTS 3600 InfiniBand switch Juniper EX2500 Ethernet fabric switch

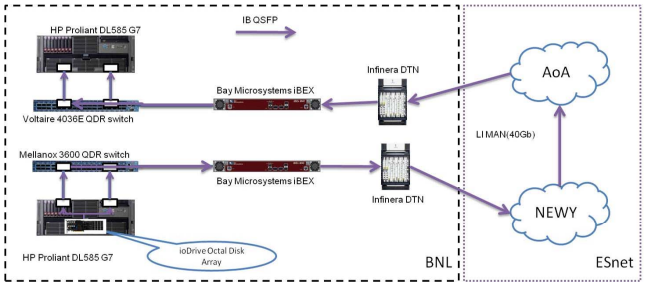

Figure 4. Host connectivity with long haul-link for MAN experiments

2) High-bandwidth medium-latency MAN: We set up a communication path in the New York metropolitan area. We have two hosts located inside the Brookhaven National Laboratory. These two hosts are connected by a long distance SONET link that goes through New York city. This infrastructure is also part of the Energy Science Network (ESnet) [6]. The capacity of this link is 40Gbps in each direction, with a minimum round trip time (RTT) of 3.6ms.

B. Experimental Results over LAN

In this set of experiments, we use memory-to-memory data transfer as the baseline results. We compare the performance between RFTP and Netkit FTP, and between RFTP and GridFTP.

1) Bandwidth and CPU usage of RFTP and Netkit FTP: We consider the aggregate bandwidth and CPU utilization as the main performance metric, and the performance numbers are obtained as follows. For both Netkit FTP and RFTP, multiple clients are initiated and connected to the server concurrently, and then each client process transfers 100GB of data to the server memory using the *put/rput* command. High-volume data movement is important in grid computing, and less so in cloud computing environment. The server always listens to potential incoming client connection requests, and on each connection request, forks a child process to handle data transfer. The aggregate bandwidth is obtained by monitoring the entire transfer period of all connections, and then we take the average bandwidth during the time

Table III
BANDWIDTH AND CPU UTILIZATION RATIO WITH SINGLE CLIENT

	iWARP	InfiniBand	RoCE
Bandwidth ratio	1.492	2.468	3.074
Server CPU ratio	1.168%	1.435%	1.774%
Client CPU ratio	8.906%	15.802%	19.251%

period. We use "nmon" tool to obtain the CPU utilization of the application. Note, we have 24 cores in our hosts, and therefore the total CPU utilization can be up to $24 \times 100\%$.

Figures 5–7 show the aggregate bandwidth and CPU utilization performance of RFTP and Netkit FTP over iWARP, InfiniBand, and RoCE in LAN, respectively, with different numbers of concurrent clients. We have two important observations:

- First, RFTP utilizes the bare metal bandwidth better with few concurrent clients. RFTP saturates the bare metal bandwidth with only two clients, compared to 4-8 clients needed by Netkit FTP. To further explain this, we calculate the ratios of aggregate bandwidth and CPU utilization between RFTP and Netkit FTP when there is only one client. The results are shown in Table III. We find the bandwidth increases by 49.2% on iWARP, 146.8% on InfiniBand, and 207.4% on RoCE, respectively.
- Second, RFTP improves the bandwidth performance at significantly lower CPU consumption. The figures show the CPU consumption of Netkit FTP increases faster than that of RFTP with the increase of the number of clients. TCP/IP stack is known as high overhead protocol for high-speed data transfer applications. Unnecessary intermediate buffering, associated context switching, and overhead from flow and congestion control, all contribute to higher CPU processing cost and slower data transfer.

We note that there are other interesting observations from our experiments. First, we implement one-sided RDMA WRITE in the middleware, and thus much of the processing cost is left to the clients who use the *rput* operation in the experiments. Second, the bandwidth and CPU load performance do not strictly increase with the number of clients. The reason is that, during our experiments, there were other active processes in the hosts, competing against our test processes, and thus introduced some noises.

2) Bandwidth and CPU usage of RFTP and GridFTP: GridFTP is a popular tool for data transfer in high-performance computing environment. As RFTP, GridFTP allows the use of multiple parallel connections to improve bandwidth utilization. In this experiment, we compare the performance of GridFTP and RFTP, in terms of both aggregate bandwidth and CPU consumption.

Figure 8 shows the performance comparison between GridFTP and RFTP in the LAN environment with 10Gbps

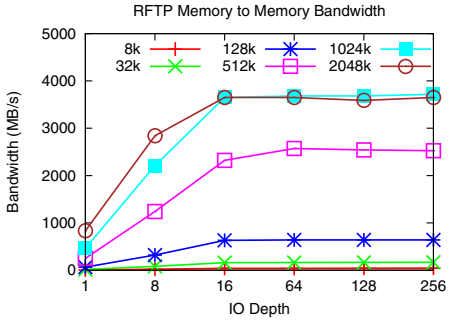

Figure 9. RFTP memory-to-memory bandwidth in MAN

iWARP connection. We run RFTP with one stream and eight streams. We also test GridFTP with a single TCP connection and eight parallel connections. In terms of bandwidth performance, RFTP consistently outperforms GridFTP in this setting. Notice that with RFTP, the bandwidth is almost fully utilized when block size is large enough, for example, 128K bytes. For CPU utilization, Figure 8 shows, with a small block size, both RFTP client and server run at higher CPU utilization, due to the higher overhead for handling more control message and completion events. Because our middleware has a multi-thread design, RFTP takes advantage of multi-core CPU resources efficiently. When the block size is large enough, we observe that the CPU consumption of RFTP drops drastically, as much of the protocol processing and data copy has been offloaded. On the other hand, GridFTP, in particular its server, requires almost a constant CPU consumption even when block size is large.

We note that performance evaluation of GridFTP alone with InfiniBand was done in another study [16]. Our work is different in that we consider another RDMA architecture, and we compare the performance of GridFTP to our application.

C. Experiment Results over MAN

In this setting, our first experiment evaluates the network performance of our middleware layer and RFTP without bandwidth limitation from the disks. At the data source, the disk I/O module loads data from the special file /dev/zero, and simply copies data into /dev/null at the data sink. The application's throughput performance is calculated by the logging thread in the middleware layer.

We configure with only one data stream in this experiment to check the impact of I/O depth and data block size of each trunk. Figure 9 shows the bandwidth of the RFTP payload. The throughput of the middleware increases as the size of each data block increases. However, the throughput hits the hardware limit offered by the network at certain block size, and further increasing the size does not improve the bandwidth performance. On our testbed, we have found that 2048 kilobytes block size results in the same throughput as

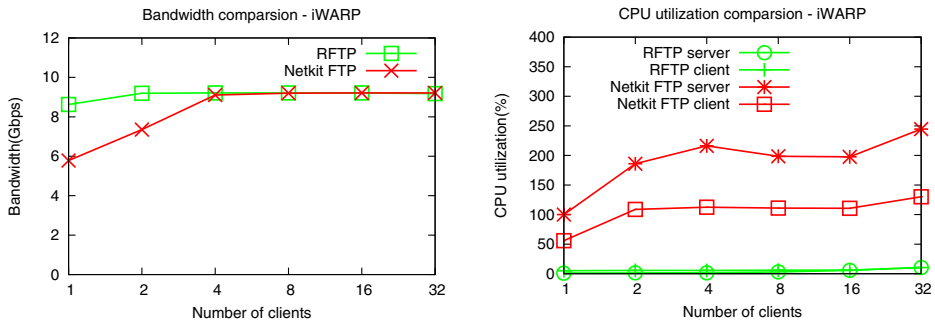

Figure 5. Bandwidth and CPU utilization comparison between RFTP and Netkit FTP over iWARP in LAN

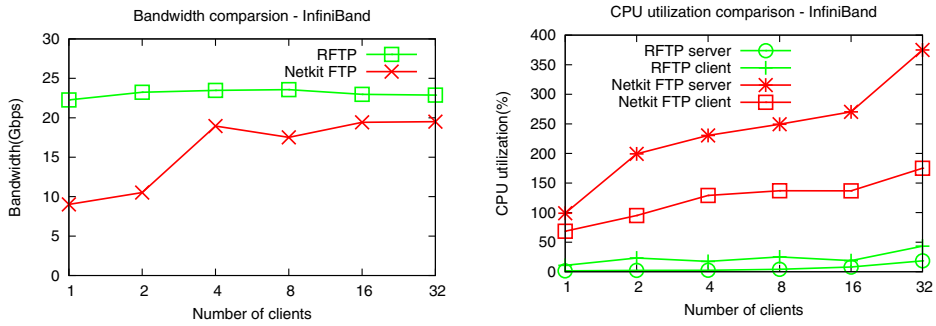

Figure 6. Bandwidth and CPU utilization comparison between RFTP and Netkit FTP over InfiniBand in LAN

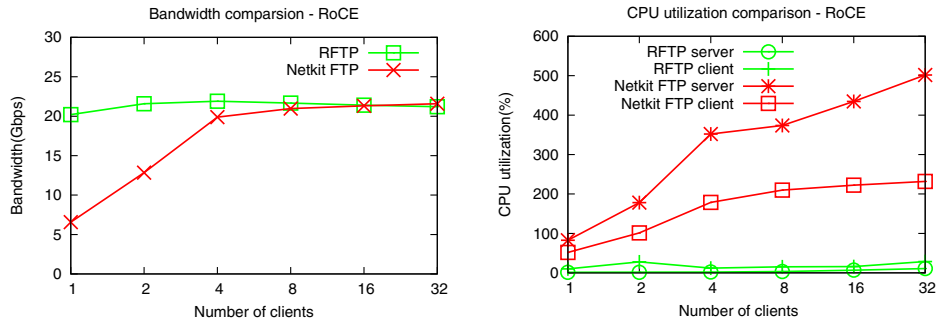

Figure 7. Bandwidth and CPU utilization comparison between RFTP and Netkit FTP over RoCE in LAN

1024 kilobytes' when the RDMA I/O queue depth is greater than 16.

Our middleware design supports multiple packets in flight in each transfer pipeline. It means the RFTP application can post many RDMA tasks into the send queue simultaneously. With the same block size, the larger I/O depth usually means a better performance.

As shown in Figure 10, when the I/O depth is low and the block size is small, .e.g., less than 1024 kilobytes, the CPU utilization remains mostly a constant. This is because the network bandwidth is under-utilized, as shown in the bandwidth figure. However, when the I/O depth is larger than 16 and the block size is greater than 1024 kilobytes, the bandwidth reaches its maximum. At this point, the use of large block size leads to lower CPU consumption. For example, the CPU usage at 2048-kilobytes block size is half that at 1024-kilobytes block size.

In our second experiment, data will be sent from /dev/zero at the data source to a Fusion-io disk set, which contains eight physical solid state disks at the data sink. The experiment launches two concurrent processes, each responsible for delivering four files into four independent disks. The performance of RFTP in this case is similar to the performance in the previous memory-to-memory experiment when the block size is smaller than 128 kilobytes. When the block size is small, the disks are not the bottleneck. When the block size is larger, the disk bandwidth becomes the bottleneck. We find the RFTP's performance is close to the peak disk rate that is measured with local disk performance tools on the installed Fusion-io disks.

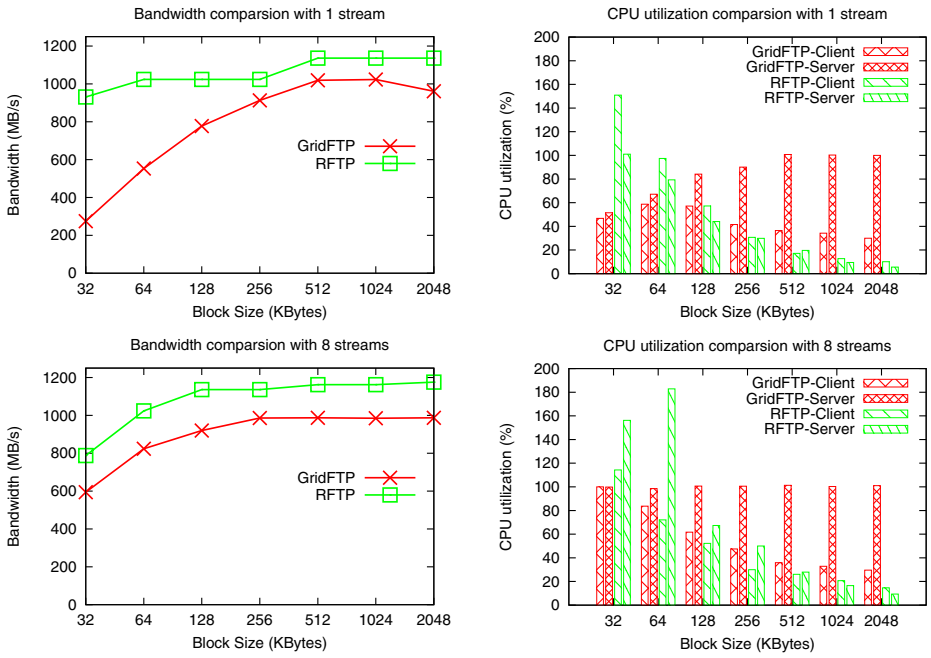

Figure 8. Bandwidth and CPU comparison between GridFTP and RFTP over iWARP in LAN

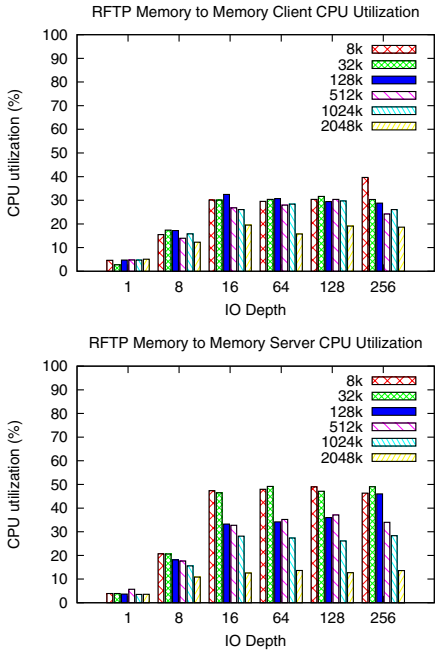

Figure 10. CPU utilization of RFTP with different block size and I/O depth

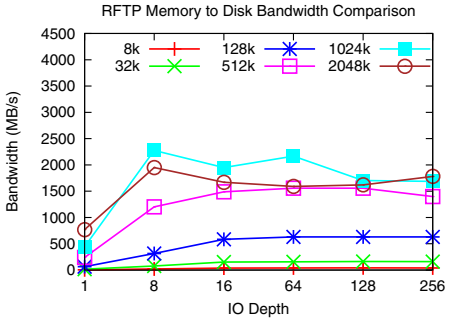

Figure 11. RFTP memory-to-disk bandwidth in MAN

VI. Conclusions

Data-intensive applications in grid and cloud computing require efficient data transfer protocols to fully utilize the capacity of advanced network infrastructure. In this paper, we have designed and implemented a RDMA-based middleware layer that provides resource abstraction and management, task synchronization and scheduling, and parallel data transfer. Our middleware utilizes the most favorable interfaces of the OFED verbs that are tightly coupled with RDMA architecture, and thus hides the heterogeneity of the underlying high-speed networks.

To demonstrate the efficiency of our middleware design, we developed a FTP application based on this middleware layer. In order to obtain the practical test data under different scenarios, we setup a platform with three different RDMA technologies, and we also tested the performance of our system over long-haul MAN links. The experiments show that our middleware achieves remarkable bandwidth performance with marginal CPU resources, and it can be a common substrate to accelerate various data transfer applications in

grid and cloud computing.

ACKNOWLEDGMENT

The authors are grateful to the facility donation of Mellanox Technologies, Inc. and Fusion-io, Inc. The authors have benefited from the numerous technical discussions with Todd Wilde from Mellanox, David McMillen from System Fabric Works, Inc., and David Strohmeyer from Intel. This work is supported by United States Department of Energy, Grant No. DE-SC0003361.

REFERENCES

[1] W. Feng, P. Balaji, C. Baron, L. N. Bhuyan, and D. K. Panda, "Performance characterization of a 10-Gigabit ethernet TOE," in *Proceedings of 13th Symposium on High Performance Interconnects (HOTI)*, August 2005.

[2] H. Jang, S.-H. Chung, and D.-H. Yoo, "Implementation of an efficient RDMA mechanism tightly coupled with a TCP/IP offload engine," in *Proceedings of International Symposium on Industrial Embedded Systems (SIES)*, June 2008.

[3] N. Bierbaum, "MPI and embedded TCP/IP Gigabit Ethernet cluster computing," in *Proceedings of 27th Annual IEEE Conference on Local Computer Networks*, Tampa, Florida, USA, November 2002, pp. 733–734.

[4] E. Yeh, H. Chao, V. Mannem, J. Gervais, and B. Booth, "Introduction to TCP/IP Offload Engine (TOE)," *10 Gigabit Ethernet Alliance (10GEA)*, October 2002.

[5] "OpenFabrics Alliance: http://www.openfabrics.org/."

[6] "Energy Sciences Network: http://www.es.net/."

[7] InfiniBand Trade Association, "InfiniBand Architecture Specification," *Release 1.2.1*, 2006.

[8] "uDAPL: User Direct Access Programming Library. http://www.datcollaborative.org/udapl_doc_062102.pdf."

[9] A. Danalis, A. Brown, L. Pollock, and M. Swany, "Introducing gravel: An MPI companion library," in *Proceedings of IEEE International Symposium of Parallel and Distributed Processing (IPDPS)*, Miami, Florida USA, April 2008.

[10] A. Danalis, A. Brown, L. Pollock, M. Swany, and J. Cavazos, "Gravel: A communication library to fast path MPI," in *Euro PVM/MPI 2008*, October 2008.

[11] P. Lai, H. Subramoni, S. Narravula, A. Mamidala, and D. K. Panda, "Designing efficient FTP mechanisms for high performance data-transfer over InfiniBand," in *Proceedings of International Conference on Parallel Processing (ICPP)*, September 2009.

[12] P. W. Frey and G. Alonso, "Minimizing the hidden cost of RDMA," in *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS)*, June 2009.

[13] N. S. V. Rao, W. Yu, W. R. Wing, S. W. Poole, and J. S. Vetter, "Wide-area performance profiling of 10GigE and InfiniBand technologies," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, November 2008.

[14] W. Yu, N. S. Rao, P. Wyckoff, and J. S. Vette, "Performance of RDMA-capable storage protocols on wide-area network," in *Proceedings of Petascale Data Storage Workshop*, November 2008.

[15] "Linux TCP Tuning: http://fasterdata.es.net/fasterdata/host-tuning/linux/."

[16] H. Subramoni, P. Lai, R. Kettimuthu, and D. K. Panda, "High performance data transfer in grid environment using gridftp over infiniband," in *Int'l Symposium on Cluster Computing and the Grid (CCGrid)*, May 2010.