Design and Performance Evaluation of NUMA-Aware RDMA-Based End-to-End Data Transfer Systems

Yufei Ren
Stony Brook University
Stony Brook, New York 11790
yufei.ren@stonybrook.edu

Tan Li
Stony Brook University
Stony Brook, New York 11790
tan.li@stonybrook.edu

Dantong Yu
Brookhaven National Laboratory
Upton, New York 11973
dtyu@bnl.gov

Shudong Jin
Stony Brook University
Stony Brook, New York 11790
shujin@stonybrook.edu

Thomas Robertazzi
Stony Brook University
Stony Brook, New York 11790
thomas.robertazzi@stonybrook.edu

ABSTRACT

Data-intensive applications place stringent requirements on the performance of both back-end storage systems and front-end network interfaces. However, for ultra high-speed data transfer, for example, at 100 Gbps and higher, the effects of multiple bottlenecks along a full end-to-end path, have not been resolved efficiently. In this paper, we describe our implementation of an end-to-end data transfer software at such high-speeds. At the back-end, we construct a storage area network with the iSCSI protocols, and utilize efficient RDMA technology. At the front-end, we design network communication software to transfer data in parallel, and utilize NUMA techniques to maximize the performance of multiple network interfaces. We demonstrate that our system can deliver the full 100 Gbps end-to-end data transfer throughput. The software product is tested rigorously and demonstrated applicable to supporting various data-intensive applications that constantly move bulk data within and across data centers.

Categories and Subject Descriptors

C.2.2 [**Computer Communication Networks**]: Network Protocols—*applications*; D.4.8 [**Operating Systems**]: Performance—*measurements*

General Terms

Design, Performance.

Keywords

Network Protocols, Storage Area Network, Remote Direct Memory Access, Multi-Core Architecture, Non-Uniform Memory Access

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
SC'13 November 17-21, 2013, Denver, CO, USA
Copyright 2013 ACM 978-1-4503-2378-9/13/11 ...$15.00.
http://dx.doi.org/10.1145/2503210.2503260.

1. INTRODUCTION

Various data-intensive applications require ultra high-speed data transfer capability, such as those in data centers, cloud-computing environments, and distributed scientific computing. They frequently need data transfer software to support true end-to-end data and file delivery, i.e., between the storage systems attached to the source and the destination hosts. Figure 1 shows our intuitive example from the Department of Energy's (DOE's) Magellan cloud data centers [4] that are interconnected by the 100 Gbps links of the DOE's Advanced Network Initiative (ANI). Such an architectural layout is often found in the DOE's National Laboratories, for example, three leadership computing facilities hosted at Argonne National Laboratory, Oak Ridge National Laboratory, and the National Energy Research Scientific Computing Center, respectively, and the tier-1 Large Hadron Collider computing facilities at Brookhaven National Laboratory and Fermilab that play a vital role in searching through petascale to exascale experimental data for scientific insights and discoveries [19]. The science programs (climate simulation, astrophysics, high-energy physics, material science, and system biology) at these DOE Laboratories frequently rely on high-performance supercomputers and server clusters, along with back-end storage systems encompassing hundreds of petabyte disk and tape storage, to run computing and data intensive applications, and to move data from experiments and simulations between computing and storage infrastructures and frequently across wide-area networks. Our primary goal is to design and deliver an efficient, extremely high-performance data transfer tool for these computing facilities that share an infrastructural layout similar to that depicted in Figure 1. To scale up data transfer to 100 Gbps and higher, we must overcome at least three different types of bottlenecks along the end-to-end paths that consist of hosts, networks, and storage systems.

First, to overcome the processing bottleneck of individual hosts, multi-core hosts often are employed for ultra high-speed data transfers. As the number of CPU sockets and cores per CPU die grows in the multi-core architecture of modern computer hosts, it becomes increasingly difficult and inefficient to have the same latency in memory access across all CPU cores. A state-of-the-art CPU architecture integrates a memory controller as a core component within

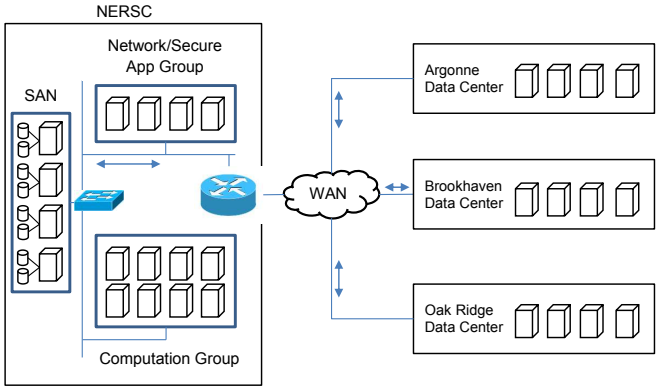


Figure 1: Data transfer and synchronization between data centers. This is an example from the Department of Energy's (DOE's) Magellan cloud data centers that are interconnected by the 100 Gbps links of the DOE's Advanced Network Initiative.

the CPU die, and discards the external memory controller hub, a component that might become a bottleneck in a multi-core architecture [3]. Memory banks in different locations of a motherboard are attached to their corresponding CPUs. Therefore, the access latencies from a specific CPU core to different memory banks are no longer same. With green computing restricting volume and power consumption, vendors turn to the Non-uniform Memory Access (NUMA) model to achieve higher resource density [12, 10, 7]. Although the high-speed connectivity between CPUs greatly facilitates arbitrary memory access, for example QuickPath Interconnect [3] and Hyper Transport [26], an application tuned for local memory access always performs much better than those that are not.

Second, advanced network technologies and protocols are employed to fully utilize the bare-metal bandwidth of ultra high-speed networks, at 100 Gbps and higher, and to eliminate network performance bottlenecks. Remote direct memory access (RDMA) [20] is one of these promising technologies because it can boost the performance of high-speed networks significantly. By enabling network adapters to transfer bulk application memory blocks to or from remote ones, and eliminating data copies in protocol stacks, RDMA achieves low latency and high bandwidth. InfiniBand [14, 13], the original RDMA implementation, dominates the technology market of intra-data center interconnections, while RDMA over Converged Ethernet (RoCE) [9] extends RDMA's capabilities to the networks between data centers that might be thousands of miles apart. Consequently, RDMA offers an opportunity to assure that large data synchronization and movement within or between data centers for applications to accomplish their routine tasks in a highly efficient manner.

Third, back-end storage systems within a server often become a severe bottleneck due to the low bandwidth of traditional magnetic disks or even recent flash solid-state disks (SSDs). One alternative to overcome this bottleneck of back-end storage systems is to build storage area networks wherein one assembles multiple storage components to provide aggregated bandwidth commensurate with a host's processing speed and its bare-metal network bandwidth. To configure and adapt high-performance RDMA networking technology into storage area networks, researchers [11] implemented an iSCSI extension for RDMA (iSER) [16], to enable SCSI commands and objects to be transferred over RDMA-based networks, such as InfiniBand and RoCE.

In this paper, we describe the design, tuning, and performance evaluation of a novel high-speed data transfer system for delivering data at 100 Gbps in an end-to-end fashion. The system utilizes a pair of multi-core front-end hosts (sender and receiver). Our research includes the follows. First, our back-end storage systems use the standard iSER protocol that is configured for high-speed data access. The protocol enables InfiniBand based data delivery from the back-end storage systems to the front-end hosts. This design allows us to eliminate the back-end storage bottleneck with the scalable InfiniBand. Second, between the front-end hosts with multiple network connections, we integrate our RDMA-based file transfer protocol, RFTP [23, 21, 22], into the end-to-end data transfer system, and optimize its performance to maximize bandwidth throughput and minimize host processing overhead. Third, for all hosts along an end-to-end path, we optimize their performance via NUMA tuning. Thus, we minimize the impact of host processing overhead. We note in the current implementation of iSER or RFTP, the NUMA factor is not considered, and we have observed the performance benefit of simple NUMA tuning in this paper. To summarize, our design is the first to achieve 100 Gbps and higher end-to-end real data file transfers between one pair of commodity hosts, and to do so, we have overcome several aforementioned bottlenecks. We evaluate our system comprehensively, using the testbeds that closely resemble the production environments, common in large national laboratories and commercial cloud computing providers. Furthermore, more tests were performed with inter-data center data transfers along long-haul high bandwidth links of over 4000 miles long.

The rest of this paper is organized as follows. In Section 2, we present the background information, and the motivations of our research. We describe our system design in Section 3, and comprehensively evaluate the entire end-to-end system in Section 4. Finally, we offer our conclusions and highlight our contributions.

2. BACKGROUND AND MOTIVATIONS

In this section, we present evidences to show that the advances in hardware technology improve bare-metal performance, but existing software is not developed to take advantage of these advances. Consequently, multiple bottlenecks and issues still exist along end-to-end data transfer paths. Among them, special efforts are needed to improve the efficiency of memory access in multi-core systems, along with techniques for hardware acceleration to maximize the capacity of network protocols. A clear understanding of these advances and a subsequent holistic approach to tackle these new issues are necessary since they are not available in the existing software systems.

2.1 Memory Access in NUMA Multi-core Systems

The stubborn speed disparity between the CPU and memory, named the "Memory Wall", common in the previous single-core architecture era, will continue to exist and even deteriorate with multi-core architecture. As detailed in [29], latency in memory access will be a major bottleneck in the

computer system. Pursuing higher CPU frequency is not sustainable due to the power wall: increasing transistor current leakage leads to uncontrollable power consumption and generates excessive heat that is hard to dissipate. From system architecture aspect, memory latency might partially negate a high CPU clock rate and the associated computing power. As a result, chip designers might well turn to exploring multi-core architectures and pack more cores into a single CPU die. Consequently, the speed imbalance between fast-growing number of CPU cores and memory will become more severe in the multi-core architecture.

The state-of-the-art NUMA architecture introduces a non-uniform hierarchy of memory latency. Most operating systems often provide only standard scheduling methods and shift to applications the burden of NUMA-related scheduling and tuning. Within this paradigm, applications with high performance requirements must be aware of the physical locations of main memory and even peripheral devices, and implement location-aware mapping functions to co-schedule CPU cores, memory, and devices for application threads with the overall goal of reducing the latency and increasing bandwidth in memory access. The NUMA architecture is not proposed to overcome the memory wall problem. However, it offers applications a hardware platform so as to improve their aggregate performance in a multi-core environment via a suitable policy of memory allocation.

2.2 Protocol Offloading

Another technological advance is the hardware protocol offloading to reduce the processing cost of network protocols in computers. For example, there are at least two memory copies for each data packet sent/received by TCP/IP applications. One is between applications and operating system (kernel), and the other is between operating system (kernel) and network interfaces. For high performance computing, data copies limit a system's overall performance due to inefficient utilization of memory bandwidth and high CPU consumption. Recently, the bandwidth for a single network adapter has reached 40, 56, or even 100 Gbps [5]. Furthermore, a high-end server is often equipped with multiple adapters for load balancing and fault tolerance. Thus, traditional TCP/IP applications may hit the memory wall problem due to the performance penalty resulted from multiple data copies long before reaching the limit of bare-metal network bandwidth. Consequently, adding network capacity does not improve the actual data transfer performance.

For end-to-end data-transfer systems, both back-end storage network and front-end data movement components must reduce data copy operations and avoid the associated performance penalties. The RDMA protocol and its zero-copy techniques efficiently satisfy this requirement since it offloads network protocol processing directly into hardware and avoids data copies from/to the kernel space. For example, to build a back-end storage system using SAN, Dalessandro *et al.* [11] implemented iSCSI extensions for RDMA (iSER) [16].

2.3 A Motivating Experiment

To illustrate the importance of the aforementioned technology advances to data transfer applications, we describe a simple experiment carried out in our testbed with multi-core NUMA technology. Two IBM X3650 M4 hosts are connected by three pairs of 40 Gbps RoCE connections (RDMA over Converged Ethernet). Each RoCE adapter is installed into an eight-lane Peripheral Component Interconnect (PCI) Express 3.0 slot. The theoretical maximum bandwidth of the bi-directional network of such a system is 240 Gbps.

First, we measured the maximum memory bandwidth of our hosts. We compiled STREAM [18], the de facto memory bandwidth benchmark, with the OpenMP option enabled to support multi-threaded test. The *Triad* function showed that the peak memory bandwidth for two NUMA nodes is 50 GB/s, or 400 Gbps. For socket-based network applications, there are two data copies for each network operation at each end of a TCP/IP session. Therefore, the maximum TCP/IP bandwidth that the system can support is 200 Gbps.

Then, we tested TCP/IP stack performance via *iperf* [1] to assess the maximum bi-directional end-to-end bandwidth offered by this testbed. With the default setting, iperf uses only a small chunk of memory, and reuses the same data in the memory chunk. Since the data is always cached within CPU, and it avoids one memory read access. Under these conditions, the result of iperf's performance matches that of RDMA-based data transfer because it has the same number of memory accesses as RDMA. However, such a test does not reflect real data transfer applications that need to continuously refill data from memory and back-end storage. To eliminate this cache effect, we purposely enlarged the sender's buffer to exceed the size of the CPU cache. Since iperf is lightweight in user space, and it only transfers memory data to or receives it from network interfaces, most CPU cycles are spent on processing the TCP/IP protocol stack in the kernel space. We captured the percentage of CPU cycles in the kernel through *perf* [6], a Linux kernel profiling tool. During a ten-minute test with the Linux default scheduling policy, the average aggregate bandwidth was 83.5 Gbps. The kernel space and the user space memory copy routines, viz., the *copy_user_generic_string*, consumed about 35% of the overall CPU usage.

For comparison, we optimized "iperf" by tuning the NUMA locality, and repeated this same test. The aggregate bandwidth increased to 91.8 Gbps, about 10% higher than the previous iperf test with the default Linux scheduler.

The experiment and results afford two observations. First, the TCP/IP protocol stack requires multiple data copies and incurs a significant amount of processing overhead that further complicates multi-core memory access, and increases the severity of the memory wall problem. Consequently, the bottleneck of an end-to-end path is host processing operations, rather than network bandwidth. Second, the NUMA memory access incurs additional hardware (CPU cores) cost; for example, latency for synchronization with remote cores can further aggravate the ensuing bottleneck.

The main objective of our work thus is to carefully design an end-to-end data transfer system to eliminate the bottlenecks along a data path. These bottlenecks can comprise transferring hosts, back-end storage systems, and front-end host-to-host network communication channels. We have designed, implemented, and evaluated our RDMA-based system for wide-area data intensive applications [23, 21]. We are aware of several other studies [17, 24, 27] on integrating the RDMA capability into data-transfer applications and evaluating the resulted systems. However, those studies have not yet been validated along the entire end-to-end path, including high performance back-end systems and wide-area network links.

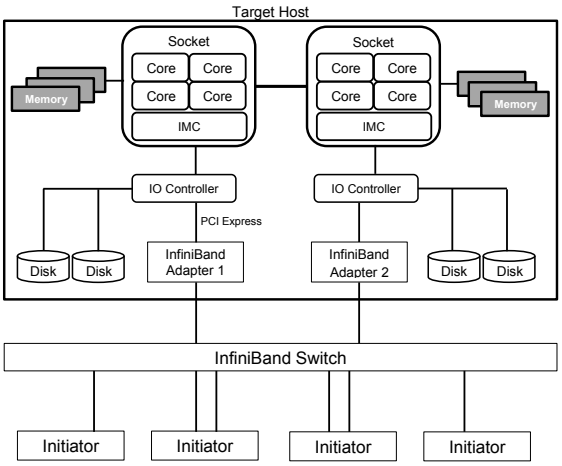


Figure 2: iSER tuning in NUMA architecture with multiple adapters.

3. CHARACTERIZATION OF SYSTEM DESIGN AND NETWORK APPLICATION

In this section, we describe the design of our end-to-end data transfer system. It encompasses one back-end storage system designed as a storage area network, one pair of sending and receiving front-end hosts, and a data transfer application over the entire infrastructure. We detail each component and analyze their performance.

3.1 Back-End Storage Area Network Design

We use the iSER protocol for data communication between a pair of the front-end client and back-end storage server within a storage area network. In this protocol, we follow the definitions in the iSCSI architecture, and call this pair of client and server "initiator" and "target", respectively. An initiator starts the data transfer process by sending I/O requests to the target that then proactively transfers the data. For example, to handle a read block I/O request sent by the initiator, the target will compose an RDMA *Write* work request to send data to the initiator, while a write I/O request triggers an RDMA *Read* from the target to fetch data from the initiator.

The default target process has a multi-threaded implementation that takes advantages of multi-core architecture to handle multiple I/O requests simultaneously to assure a high throughput. However, with the default setting, the NUMA factor and locations of the Peripheral Component Interconnect (PCI) devices are not considered. There are two possible methods to integrate the NUMA technology into a target. One is to use the *numactl* utility to bind a dedicated target process to each logical NUMA node; the other is to integrate the *libnuma* [15] programming interface into the target implementation. The former needs an explicit, static NUMA policy, while the latter relies on scheduling algorithm for each I/O request. Redesign of iSCSI with the *libnuma* API and libraries is beyond the scope of this paper. We only implement the former solution and here present its effect on NUMA.

We use Linux *tmpfs* as the back-end storage in our prototype system. By adjusting the location of the memory file

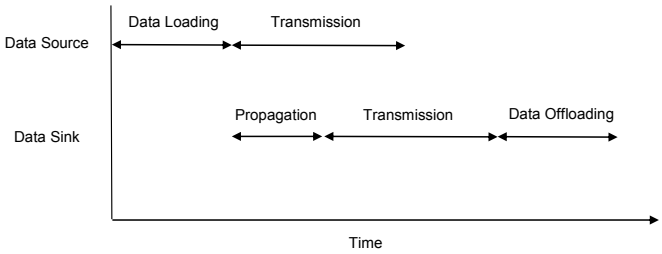


Figure 3: Data block transfer delay breakdown.

with the *mpol* and *remount* options [2], we pin each file into a specified NUMA node memory. Thereafter, we assign each NUMA node to a dedicated target process to handle local I/O requests. Therefore, all NUMA nodes have low latency in accessing local memory, and thereby, the best throughput performance. For a system with multiple adapters, as shown in Figure 2, this choice of design ensures that different I/O requests are handled via different links, hence resulting in the best aggregate performance. The effectiveness of this design will be evaluated by our experiments later.

3.2 RDMA Application Protocol: Cost Analysis and Implementation

Three major components are involved in an end-to-end data transfer application: Data loading, data transmission, and data offloading. Figure 3 shows these components at data source and data sink. Depending on the type of data storage (such as local disks and SANs) and transmission networks (LAN and WAN), the throughput and latency of each component may vary. Any one of the three components can become a bottleneck.

We use our RDMA-based file-transfer protocol, RFTP, to move data within our system. RFTP supports pipelining and parallel operations and configures itself efficiently to utilize system resources and raw network bandwidth. To confirm the benefits of RFTP's performance, we break down its cost and compare it with the traditional TCP-based data transfer protocols.

To gain insights into the performance of data transfer applications and the efficiency of protocol offloading, we undertake a five-minute test in our local test environment. The data source loads data from /dev/zero, and then transfers it over a 40 Gbps RoCE link to the data sink. The latter will dump data into /dev/null, i.e., simply discard the received data. Both RDMA-based RFTP and TCP-based iperf accomplish this task at the transfer rate of 39 Gbps. To attain RFTP's the resource usage of each data transfer thread, we call Linux `getrusage` interface on both the client side and server side. We again analyze the cost of iperf data transfer using the Linux perf tool.

As shown in Figure 4, our RDMA based solution consumes a total of 122% CPU, among which the user space protocol processing uses 56%. [1] In contrast, TCP needs a total of 642% CPU consumption; its kernel space protocol processing accounts for 311%. RDMA's data-copy overhead is 0% because of zero-copy, while TCP pays 213% on copying data between user space and kernel space. Since the data

[1]Note, we use absolute CPU time in the measurement, for example, 122% CPU consumption means the total CPU usage is equivalent to 1.22 × one fully utilized CPU core.

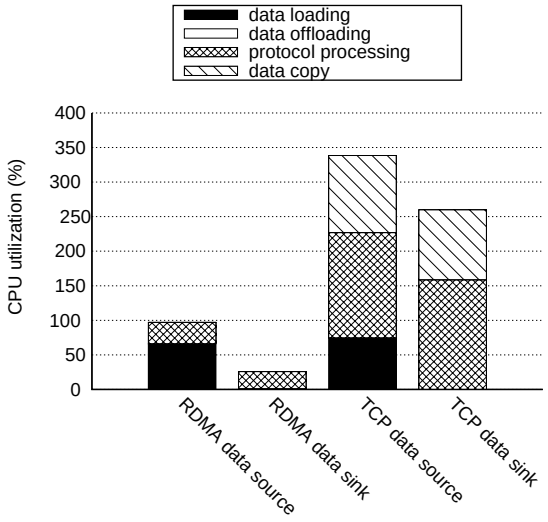


Figure 4: The breakdown of data transfer cost at 40 Gbps rate.

sources load data from /dev/zero, the kernel must flush the user memory block with 0s without involving any DMA. In both the RDMA-based and TCP cases, the data sources require about 70% CPU cycles of one core to accomplish the task. Dumping data into /dev/null involves less than 1% overhead for the RFTP, while iperf does not offload data. In both cases, the overhead from offloading can be omitted in this experiment. To summarize, through this cost evaluation, RDMA demonstrates its efficient protocol offloading and zero-copy in high-speed data transfer environment.

4. EXPERIMENTAL RESULTS

In this section, we validated the end-to-end data throughput of our software and its CPU consumption, and experimentally confirmed the effectiveness and efficiency of our innovation of NUMA awareness and RDMA hardware offloading. We undertook comprehensive experiments on both the LAN and WAN testbeds. We first describe the testbed's configurations that consist of the RDMA implementation with both InfiniBand and RoCE interconnection. We evaluated our developed system in three different scenarios: First, the back-end system's performance with NUMA-aware tuning; second, the application performance in an end-to-end LAN setting; and third, the network performance over a 40 Gbps RoCE long distance path in wide-area networks.

4.1 Testbed Setup

As shown in Figure 5, the LAN experimental system consists of back-end and front-end sections. At the back-end, each initiator or target has two Mellanox InfiniBand adapters, each of which is fourteen data rate (FDR, 56 Gbps) and connected to a Mellanox FDR InfiniBand switch. Therefore, the maximum bandwidth for loading and offloading data is 112 Gbps. At the front-end, three pairs of quad data rate (QDR) 40 Gbps RoCE network cards connect the RFTP client and server with a maximum aggregate bandwidth of 120 Gbps.

In our iSER software setup, we deployed open-iscsi utility version 2.0-872.41 at the initiator host, and SCSI target daemon with version 1.0.31 on the target host. As discussed

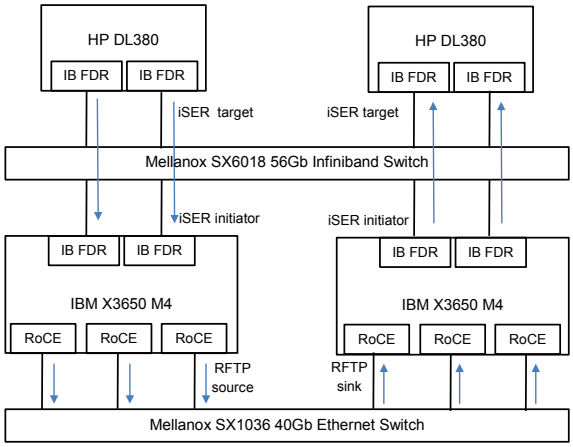


Figure 5: RDMA-based end-to-end system connectivity in LAN.

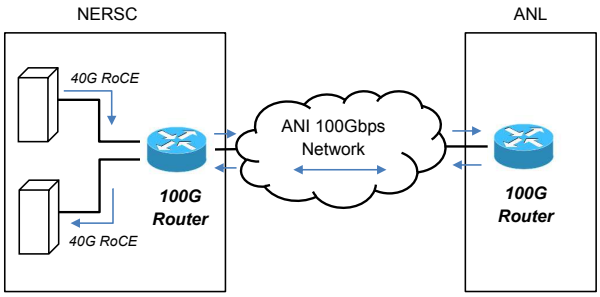


Figure 6: The DOE's ANI 40 Gbps RoCE WAN between NERSC and ANL. This 4000-mile link is a loopback network from NERSC to ANL and then back to NERSC. The RTT of the link is about 95 milliseconds.

in the previous section, the target incurs the largest fraction of the cost of the iSER protocol processing among all the hosts in our iSER configuration. We first investigated the characteristics of the processes in the target host, including the impact of the NUMA's binding configuration and the block size of I/O requests.

Initially, we attempted to set up a back-end storage based on Fusion IO's PCI-based SSD flash drives. However, we found that when applications read or wrote 100 gigabytes data or more continuously to the SSD drive, the thermal-throttling technology of SSDs proactively took actions to throttle the system's performance to prevent overheating the on-board circuits. These preventive operations degraded the I/O's performance to about 500MB/s, a severe bottleneck that made our performance evaluation impossible at the speed of 100 Gbps. Thus, in our experiments, we built back-end storage in the main memory of target hosts that dissipates heat much quickly, and so performs consistently over a wider range (from air-conditioned data centers to laboratory environment with normal temperature) of operational temperature. We created six logical units (LUNs) with on the target host, split and load-balanced all I/O requests between the two available InfiniBand links. Each LUN's size

Table 1: Testbed Configuration

	Front-end LAN	Back-end LAN	Front-end WAN
CPU * Cores	Intel Xeon E5-2660 2.20GHz 16 Cores	Intel Xeon E5-2650 2.00GHz 16 Cores	Intel Xeon E5-2670 2.90GHz 12 Cores
NUMA nodes	2	2	2
Mem(GBytes)	128	384	64
Network Adapters	40 Gbps RoCE QDR	56 Gbps IB FDR	40 Gbps RoCE QDR
OS	CentOS 6.3	CentOS 6.3	Fedora release 17
Kernel Version	2.6.32-279	2.6.32-279	3.4.3-1
OFED Version	MLNX OFED 1.5.3-3.1.0	MLNX OFED 1.5.3-3.1.0	OFED 1.5.4
TCP Congestion Control Algorithm	cubic	cubic	cubic
MTU Size	9000 (RoCE link)	65520 (IB link)	9000
RTT(ms)	0.166	0.144	95

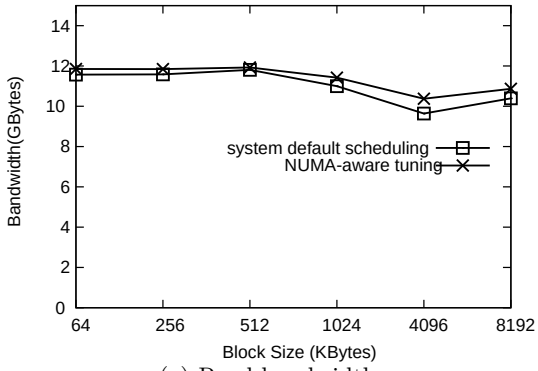
was 50 gigabytes, and the total size of the dataset was 300 gigabytes. The large size memory (we borrowed 768G byte dual in-line memory modules from HP vendors for building the high performance backend storage in our testbed) can be formatted to host any data files and represent a real-world storage solution with low latency and high throughput.

In addition to the testbed configuration within the local network, we also utilized the WAN testbed provided by the DOE's Advanced Networking Initiative (ANI). For these tests, the DOE's ANI supplied a 40 Gbps RoCE wide-area network to evaluate data transfer with RFTP over a long-haul 40 Gbps link. This 4000-mile link is a loopback network from the National Energy Research Scientific Computing Center (NERSC) in Oakland, CA to Argonne National Laboratory near Chicago, IL and then back to the NERSC. As shown in Figure 6, the two hosts are connected to wide area networks by a 100 Gbps router, the Alcatel-Lucent Model SR 7750 border. The corresponding routing records on NERSC router are configured as a loopback via ANL's 100 Gbps router [28]. Table 1 lists the detailed configurations for all the hosts, in both the LAN and WAN testbeds described.

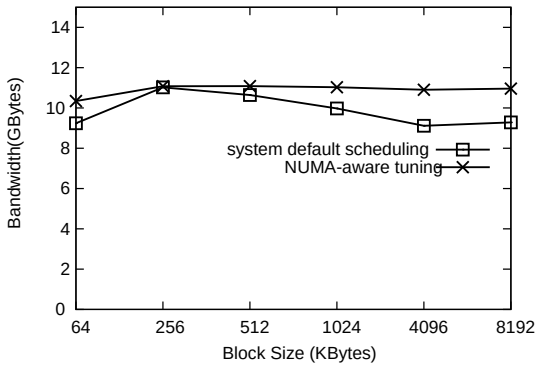
4.2 Evaluation of Memory-Based Storage System Performance

In this set of experiments, we evaluated the improvement in the iSER's performance with our NUMA-aware tuning policy and compared it with the Linux's default scheduling policy.

The iSCSI target host (the back-end storage) is based on Linux tmpfs file system created out of the system's main memory to eliminate the inefficient magnetic disk or SSD bottleneck. The system's memory is large enough (300 GB) to be comparable to a regular SAS disk in terms of the volume, whilst offering a performance a hundred of times faster than that of a magnetic disk. We created six logical units to spread parallel IO requests into different banks of the main memory. To minimize the overhead from the file system, the target exported the back-end memory file as raw devices to allow the initiator to choose any type of file systems as appropriate. We choose the flexible I/O tester [8], fio, as the benchmark software, and each test case lasted five minutes.



(a) Read bandwidth

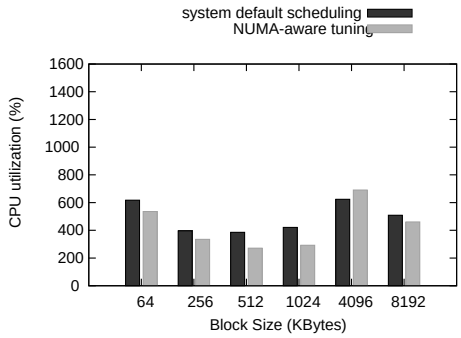


(b) Write bandwidth

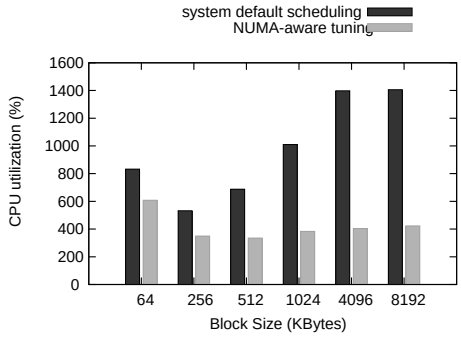
Figure 7: iSER bandwidth comparison between default scheduling and NUMA-tuning.

Figures 7 and 8, respectively, show the bandwidth and CPU consumption, in these tests. In each figure, we separately illustrate the data read and data write performance. To achieve the best performance, multiple I/O threads run simultaneously against each LUN. The gain in performance levels off once the number of threads reaches a certain threshold. Beyond that, too many I/O threads would introduce more contention, and impact the overall performance. In our testbed, we found that the optimal configuration is to use four threads for each LUN.

For read operations, the bandwidth improvement is merely 7.6% with NUMA binding, and neither is the saving on CPU consumption significant. However, for write operations, we observed an improvement in bandwidth up to 19% for the block size larger than 4 megabytes, and using the default Linux binding policy increases CPU consumption threefold. We argue that the main reason of this disparity between read and write operations is the significant overhead of cache coherency and synchronization that is needed for write operations. We note that we performed the read and write operations on a memory-based tmpfs file system. A write request essentially is a memory-write operation, and if it is executed without NUMA-aware tuning, one such operation will invalidate all other data copies in the caches at other NUMA nodes. With NUMA-aware tuning, this invalidation occurs only locally and thus, the overhead is low. When read requests are executed, with or without NUMA-aware tuning, the data copies are always "cached" or "shared" instead of



(a) Read CPU utilization



(b) Write CPU utilization

Figure 8: iSER CPU utilization comparison between default scheduling and NUMA-tuning.

"modified", and hence, the overhead from cache coherency is minimal.

We also notice that the bandwidth performance of serving read requests out of iSER is slightly better by 7.5% than that of serving write requests. We believe this reflects the better performance of RDMA Write (used by read requests) than RDMA Read (used by write requests). This difference in performance between RDMA Read and Write was revealed in a previous study [23]. When an iSER initiator sends a read request to a target, the target would use RDMA write to write (send) data directly to the memory of the initiator for the actual transfer of data; thus, the observed performance of a read request has a better bandwidth.

4.3 End-to-End Data Transfer Performance

In this section, we describe our tests and our validation of data-transfer applications in an integrated testbed environment and gather its performance with an end-to-end perspective. Our goal is to gain insight into how our application can be adapted to day-to-day real transfer scenarios.

In mimicking a realistic data transfer scenario, we adopted one wherein a transfer application first interacts with file systems via the POSIX interfaces rather than via raw devices. The details of implementing the functions of different file systems are hidden by the POSIX interfaces. Furthermore, our preliminary tests in our testbed demonstrated that the throughput differences among the raw block devices exported by target via the iSER protocol, Linux universal ext4 file system, and the XFS [25] built over the exported block devices via the iSER protocol, are comparable. Since the XFS file system particularly is efficient for parallel I/O

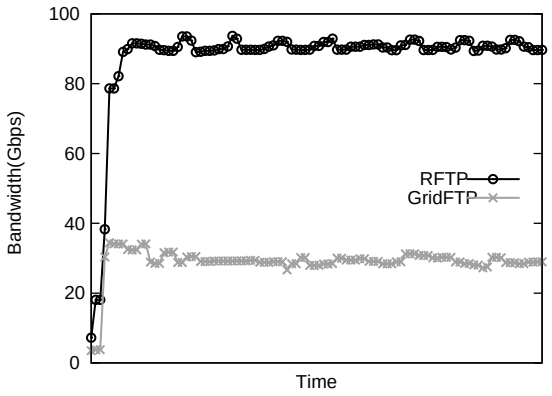


Figure 9: Throughput of end-to-end data transfer over 25 minutes.

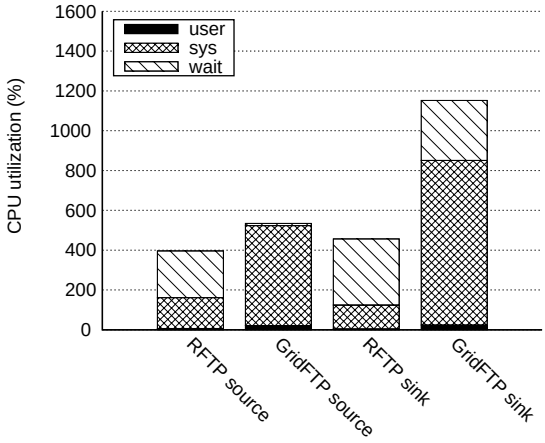


Figure 10: CPU utilization breakdown for RFTP and GridFTP.

and better aligned with our testing requirements, without losing generality, we chose XFS over other file types and formatted the exported block device with XFS from the initiator side.

We combined the back-end system and front-end system to show the end-to-end performance between RFTP and GridFTP, a widely used data transfer tool in high performance computing. To assure a fair comparison between GridFTP and RFTP, and to assure the achievement of the best performance of both applications, and minimize the penalty of remote memory access for each of them, we used numactl to bind the RFTP and GridFTP processes to a specified NUMA node. Figure 9 shows the two applications' performance during 25-minute-long tests. Our prior "fio" tests revealed that the narrowest section along the end-to-end data transfer path resides on file I/O write operation; its performance is 94.8 Gbps. Therefore, the best performance of the testbed for end-to-end data transfer is 94.8 Gbps. RFTP achieved 91 Gbps, i.e., 96% of the effective bandwidth of the overall system, while GridFTP obtained 29 Gbps, i.e., only 30% of the bandwidth for the following reasons. First, TCP stack processing incurs a substantial overhead, such as data copy between kernel space and user

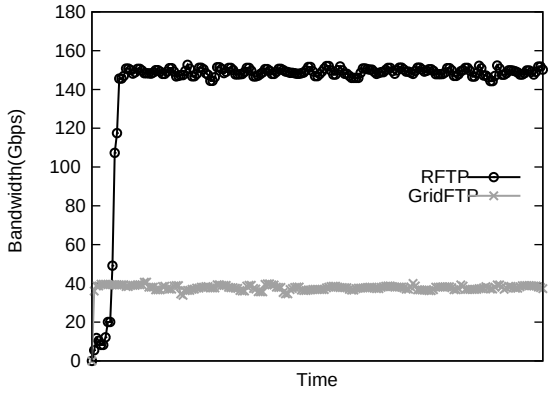


Figure 11: Throughput of bi-directional end-to-end data transfer over 50 minutes.

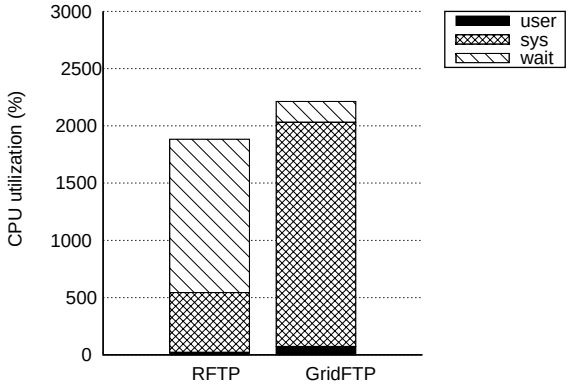


Figure 12: CPU utilization breakdown for RFTP and GridFTP bi-directional test.

space and interrupt handling for an I/O-intensive application. The high "sys" CPU in GridFTP, shown in Figure 10, reveals the high TCP/IP stack-processing cost of GridFTP. Second, GridFTP has a single-threaded design that causes the network to be in an idle state when this thread performs I/O request with long waiting time. Consequently, GridFTP is not able to fully utilize the resources of the whole I/O system. Running multiple processes simultaneously may alleviate this problem, but at the price of higher CPU consumption. Third, without support for direct I/O, GridFTP suffers the I/O cache effect at the front-end sender and receiver hosts.

To further clarify the best end-to-end host performance in terms of bandwidth and CPU consumption, we performed bi-directional data transfer tests with RFTP and GridFTP. In this experiment, we initiated data transfer simultaneously from each end of the end-to-end path to the other end. The configurations are the same as in the previous experiment. We expected that the aggregate bandwidth in the bi-directional tests would have been twice the performance in the unidirectional experiment due to the full duplex property of each component in the transfer path. However, the experimental results did not match with our expectation. Contention for resources increased for bi-directional tests because of more intensive parallel I/O requests to the back-

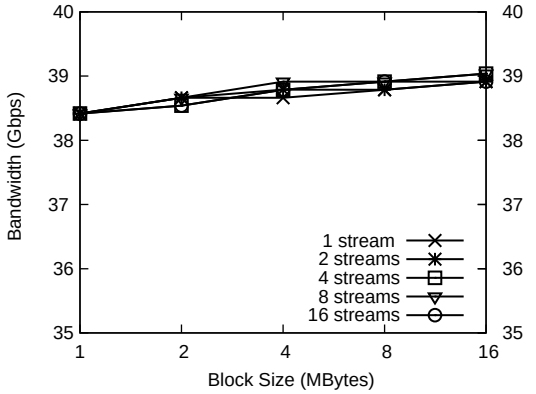


Figure 13: RFTP bandwidth with various block sizes and numbers of streams.

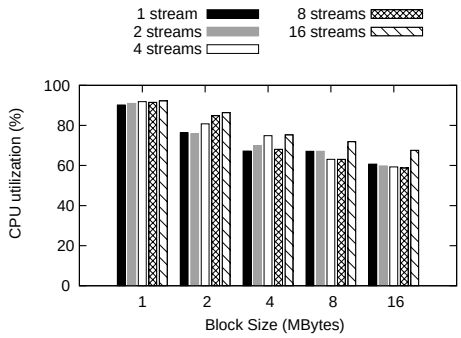
end hosts, memory copies, and higher protocol processing overhead at the front-end hosts.

As shown in Figure 11, our RFTP demonstrated an impressive 83% bandwidth improvement in the bi-directional experiments versus the unidirectional ones, and almost doubles the unidirectional performance (17% less). On the other hand, GridFTP demonstrated only about a 33% improvement due to its high CPU contention, as shown in Figure 12.

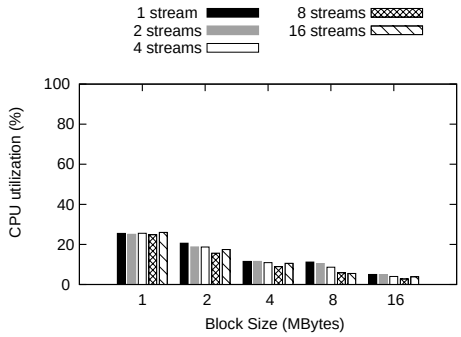
4.4 Experimental Results over 40 Gbps WAN RoCE Link

Long-haul fat links introduce much higher latency than local area networks, and have a large bandwidth delay product (BDP). It is challenging for traditional network protocols (between front-end hosts) to fill up the network pipe at the speed of 100 Gbps and beyond. Here, we evaluate the effectiveness of RFTP in eliminating this bottleneck effect. We ran RFTP over a RoCE link in DOE's ANI testbed. This link has an RTT of about 95 milliseconds, and the BDP is close to 500 megabytes. We cannot relocate our entire testbed system to the point of presence (POP) site of the DOE's ANI testbed for a suite of full WAN tests due to the restriction in testbed management and administration in a remote data center. We had to leverage the existing end systems provided by the DOE's ANI testbed. For our experiments now, we only can conduct memory-to-memory data transfers (between front-end hosts) to show that our RFTP affords a good solution to support end-to-end data transfers in the wide-area networks. We expect that if RFTP performs well over the RoCE link, then our full end-to-end data transfer system would perform equally well if it were deployed in the ANI testbed.

Figure 13 shows the bandwidth performance of RFTP when we use different numbers of parallel data streams. The x-axis shows the block size of data transferred, while the y-axis shows the actual payload data transfer bandwidth, excluding the protocol overhead. We verify that RFTP utilizes 97% of the raw bandwidth of the testbed link due to its efficient design (including the use of proactive feedbacks and asynchronous control message exchanges [23]). A small processing overhead is needed for control messages (for example creating RDMA channels and passing credit tokens), and this overhead decreases in a direct relation to increase in the message block size. The overhead reduction also is



(a) RFTP sending side CPU utilization



(b) RFTP receiving side CPU utilization

Figure 14: RFTP CPU utilization with various block sizes and numbers of streams.

reflected by the lower CPU consumption as shown in Figure 14(a) for the sender, and in Figure 14(b) for the receiver.

As we noted earlier that this wide area data transfer did not involve storage area networks due to the difficulty of deploying and managing them remotely at the NERSC. Our prior experiments show that we can get the maximum performance along each segment of end-to-end LAN and WAN data transfers, i.e., back-end data upload to the front-end system, data transfer over networks, and data offload to the back-end storage system again.

Furthermore, the performance of network data transfer is unaffected by long latency. Based on these observations, we conclude that our RFTP easily can scale up the performance that is commensurate with full-fledged end-to-end distributed testing or production infrastructures consisting of large-scale storage area networks with hundreds or thousands of target servers and long latency inter-data center network links. These infrastructures often are found at the DOE's National Laboratories and cloud data centers with intensive data transfer requirements.

5. CONCLUSIONS

Modern data centers of scientific computing must transfer and synchronize a large amount of data continuously either locally within themselves or remotely with other data centers for visualization, analysis, and disaster recovery. Accordingly, end-to-end high performance data transfer software must combine efficient design and performance tuning, to eliminate any potential bottlenecks within storage systems, at front-end hosts, and along network communication paths. In this paper we have described a novel design of such a system that integrates RDMA protocol implementation, multi-core NUMA tuning, and an optimized back-end storage area network.

To demonstrate the efficiency of our solution, we set up testbeds in both LANs and WANs. We studied the processing cost of TCP/IP stack and our RDMA based protocol. We demonstrated the performance benefits of an RDMA based solution adopted by both our RFTP and iSER. We also compared our solution with the high performance GridFTP software in various end-to-end configurations. Our performance evaluation demonstrated that our solution is three times faster than GridFTP. We also validated our protocol design in a 4000-mile network path provided by the Department of Energy's ANI testbed. These evaluations and studies verified that our solution can achieve remarkable bandwidth utilization of 97% and fully utilize the available hardware capabilities.

Acknowledgments

The authors are grateful to the facility and hardware donation of Mellanox Technologies, Inc., LSI Corp., and Fusionio, Inc. The authors benefited from numerous technical discussions with Gilad Shainer, Roi Dayan, Erin Filliater, Yaron Haviv, Bill Lee, Dudu Slama, and Todd Wilde, from Mellanox, Brian Tierney, Eric Pouyoul, and Scott Richmond from Lawrence Berkeley National Laboratory, Paul Grun and David McMillen from System Fabric Works, Inc., and Ezra Kissel and Martin Swany from Indiana University. This research is supported by United States Department of Energy, Grant No. DE-SC0003361. The ESnet Advanced Network Initiative (ANI) Testbed, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231. Both contracts are funded through the American Recovery and Reinvestment Act of 2009.

6. REFERENCES

[1] NLANR/DAST : Iperf - the TCP/UDP bandwidth measurement tool, 2003-2008. http://iperf.sourceforge.net/.

[2] NUMA memory allocation policy for tmpfs, 2006.

[3] An introduction to the Intel QuickPath Interconnect, January 2009.

[4] The Magellan report on cloud computing for science. Technical report, 2011.

[5] New mellanox interconnect to break 100g throughput, June 2012.

[6] perf : Linux profiling with performance counters, 2012.

[7] M. Annavaram, E. Grochowski, and J. Shen. Mitigating Amdahl's Law through EPI throttling. In *Proceedings of the 32nd annual international symposium on Computer Architecture*, pages 298–309, May 2005.

[8] J. Axboe. Flexible I/O Tester.

[9] D. Cohen, T. Talpey, A. Kanevsky, U. Cummings, M. Krause, R. Recio, D. Crupnicoff, L. Dickman, and P. Grun. Remote Direct Memory Access over the Converged Enhanced Ethernet fabric: Evaluating the options. In *2009 17th IEEE Symposium on High Performance Interconnects (HOTI)*, pages 123–130, 2009.

[10] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes. Cache hierarchy and memory subsystem of the AMD Opteron processor. *IEEE Micro*, pages 16–29, 2010.

[11] D. Dalessandro, A. Devulapalli, and P. Wyckoff. iSER storage target for object-based storage devices. In *Proceedings of Fourth International Workshop on Storage Network Architecture and Parallel I/Os*, September 2007.

[12] U. Drepper. What every programmer should know about memory. September 2007.

[13] IBTA. Infiniband Trade Association. http://www.infinibandta.org/, 2010.

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

[15] A. Kleen. An NUMA API for linux, August 2004.

[16] M. Ko, M. Chadalapaka, J. Hufferd, U. Elzur, H. Shah, and P. Thaler. Internet Small Computer System Interface (iSCSI) Extensions for Remote Direct Memory Access (RDMA). RFC 5046 (Proposed Standard), Oct. 2007.

[17] 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.

[18] J. D. McCalpin. STREAM: Sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, Charlottesville, Virginia, 1991-2007. A continually updated technical report. http://www.cs.virginia.edu/stream/.

[19] D. Overbye. Physicists find elusive particle seen as key to universe, July 2012.

[20] R. Recio, B. Metzler, P. Culley, J. Hilland, and D. Garcia. A remote direct memory access protocol specification. RFC 5040 (Proposed Standard), Oct. 2007.

[21] Y. Ren, T. Li, D. Yu, S. Jin, and T. Robertazzi. Middleware support for RDMA-based data transfer in cloud computing. In *Proceedings of High-Performance Grid and Cloud Computing Workshop*, May 2012.

[22] Y. Ren, T. Li, D. Yu, S. Jin, and T. Robertazzi. Design and testbed evaluation of rdma-based middleware for high-performance data transfer applications. *Journal of Systems and Software*, 86(7):1850 – 1863, 2013.

[23] Y. Ren, T. Li, D. Yu, S. Jin, T. Robertazzi, B. L. Tierney, and E. Pouyoul. Protocols for wide-area data-intensive applications: design and performance issues. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 34:1–34:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[24] 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.

[25] A. Sweeney. Scalability in the XFS file system. In *Proceedings of USENIX Annual Technical Conference*, pages 1–14, 1996.

[26] The HyperTransport Consortium. HyperTransport I/O technology overview, June 2004.

[27] Y. Tian, W. Yu, and J. S. Vetter. RXIO: Design and implementation of high performance RDMA-capable gridftp. *Computers and Electrical Engineering*, 38(3):772 – 784, 2012.

[28] B. Tierney, E. Kissel, M. Swany, and E. Pouyoul. Efficient data transfer protocols for big data. In *Proceedings of the 8th International Conference on eScience*, October 2012.

[29] W. A. Wulf and S. A. McKee. Hitting the memory wall: implications of the obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24, Mar. 1995.