A Fast Read/Write Process to Reduce RDMA Communication Latency

Li Ou, Xubin He, *Member, IEEE*
Electrical and Computer Engineering Department
Tennessee Technological University
{*lou21, hexb*}*@tntech.edu*

Jizhong Han, *Member, IEEE*
Institute of Computing Technology
Chinese Academy of Sciences
hjz@ict.ac.cn

Abstract

RDMA reduces network latency by eliminating unnecessary copies from network interface cards to application buffers, but how to reduce memory registration cost is a challenge. Previous studies use pin-down cache and batched deregistration to address this issue. In this paper, we propose a new design of communication process: Fast RDMA Read and Write Process (FRRWP), to reduce the overhead of the memory registration and message synchronization in the critical data path of RDMA operations. FRRWP overlaps memory registrations between a client and a server, and allows applications to submit RDMA write operations without being blocked by message synchronization. We use a mathematic model to calculate the overall latency of FRRWP. Compared to traditional RDMA operations, our results show FRRWP reduces the total communication latency dramatically in the critical data path.

1 Introduction

Remote Direct Memory Access (RDMA)[1, 5, 13, 3, 6] offers low latency, high throughput, and low CPU overhead communication in network storage systems. While RDMA decreases latency by eliminating unnecessary copies from network interface cards to application buffers, there are a number of challenges to be addressed. One of them is efficient communication buffer management to reduce memory registration and deregistration cost. Previous research [15, 17, 18, 14] shows that memory registration is an expensive operation since it requires pinning of pages in physical memory and accessing the on-chip memory of the network interface card. The overhead caused by memory registration dramatically degrades the performance of RDMA and increases network latency in the critical data path of I/O operations.

Basically, a RDMA operation is a two-fold process: it requires memory registrations in both clients and servers, and exchange synchronization messages to accomplish registration before the real RDMA read or write operations. The cost of a complete RDMA process includes the cost of the memory registrations in the client and server, the overhead of synchronization messages, and the cost of real RDMA read or write operations. Several attempts [15, 20, 17, 18, 4, 14, 12] have been made to reduce the overhead of memory registration directly. In general applications, a pin-down cache [15] is incorporated into the memory manager. Several cache designs for memory registration [17, 14, 12] are proposed based on the pin-down cache to take advantage of temporal locality of memory accesses of RDMA. Another way to improve the performance of RDMA is to overlap the memory registrations between the client and the server, and reduce the overhead of synchronization messages.

In this paper, we evaluate the cost of memory registration in both user and kernel space. We analyze latency of memory registration and find three main parts which contributes most to the total costs. We then propose a new communication scheme between a RDMA client and server, Fast RDMA Read and Write Process (FRRWP), to minimize the cost of memory registration in the critical data path. FRRWP re-schedules the communication process of RDMA to overlap memory registrations between the client and the server. It allows issues of RDMA operations without being blocked by the synchronization messages: the applications may submit a RDMA write immediately after they finish local memory registrations, without waiting for the confirmation of registrations from the peer node. We compare the performance of FRRWP with traditional RDMA operations using a mathematic model. The results show that compared to traditional RDMA operations, FRRWP can reduce the total communication latency in the critical data path by at least 22us.

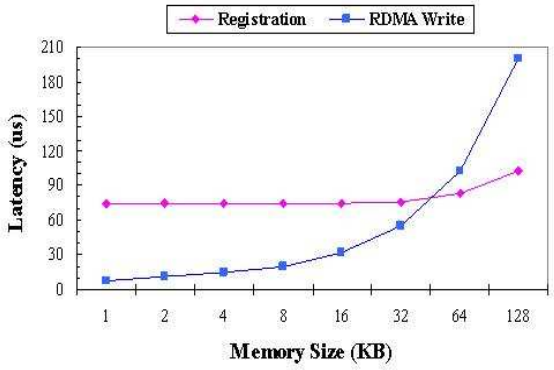


Figure 1. Comparison between latency of memory registration and RDMA write with various size of messages in user space.

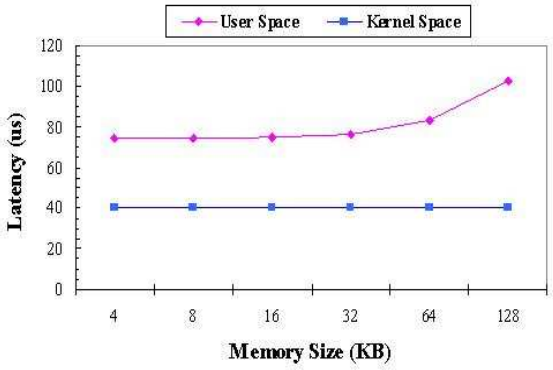


Figure 2. Comparison of memory registration latency between user space and kernel space with various size of messages.

2 Cost analysis of memory registrations

2.1 Background Review

In RDMA, a network interface card (RNIC or InfiniBand HCA) writes or reads user specified buffers directly without unnecessary copies, so before each RDMA operation, it is required to register a memory region where user buffers are located. In the process of registration, the device driver first maps the virtual memory address to the physical address, then pins the memory region to make sure that in the operations of RDMA, the memory region is not swapped out from physical memory. After map and pin, the driver reports the information of the memory region to NIC, in which a table is used to keep information of all registered memory regions. A memory region cannot be pinned forever, otherwise the effective size of physical memory used for other purpose is reduced. On the other side, the number of entries in the registration table is limited. For instance, a typical implementation of Myrinet only contains 1024 page table entries [15], and the Giganet cLan card used in [20] allows 1GB of outstanding registered buffers, which is still much smaller than the total size of physical memory that a high performance server is equipped. When the number of registered buffers exceeds this limit, the application needs to deregister memory and free resources on the NIC, which involves the operation of unpin of the memory region and remove the entry from the table. Memory registration and deregistration are time-consuming operations.

The cost of memory registration and deregistration varies with the performance of hosts. For instance, in a pretty old Pentium Pro machine (200MHz), one memory page (4KB) registration takes 26us [15], while the same operation only need 7us with a much faster Intel Xeon 2.4GHz processor [18]. Although high performance servers reduce time of memory registrations, the cost is still almost same as

Table 1. Latency of fast and ordinary memory registrations in kernel space.

Memory size (KB)	Fast MR (us)	Ordinary MR (us)
4kB	0.506637	40.08733
8kB	0.513922	40.32209
16kB	0.561122	40.18301
32kB	0.626815	40.32764
64kB	0.785199	40.22049
128kB	1.026974	40.44059

the network latency of the contemporary interconnect used by servers [15, 18]. If every RDMA operation has to be blocked by the registration and deregistration, the overhead is very large and overall latency of the communication is very high. Previous studies [15, 18] show that without any optimization, the RDMA performance is hurt by the memory registration and deregistration so much that even the traditional send and receive operations, which involve several memory copies, could outperform RDMA if the message size is small. Experiments [15, 18] show that if message size of most operations is less than 1K, RDMA with normal memory registration may not provide better performance than traditional way, and in some case, even worse.

2.2 Experimental setup

To study the cost of memory registrations in RDMA, we setup our experimental environments with two Dell servers and InfiniBand network. The server is equipped with a 2.8*GHz Intel P4* microprocessor, 1024*MB* memory, and a HCA: MT25204 (FW 1.0.8, Rate 20Gbps). Two servers are connected with a InfiniBand switch MT47396(FW 0.8.4). The operating system is Suse SLES 10 linux-2.6.13-15-smp. The InfiniBand software package is IBG2-2.0.1.

We developed a Client-Server program to test the latency of memory registration and RDMA write operation between two servers. We vary the message size from 1KB to 128KB, and compare latency in both the user space and the kernel space. For each message size, we record the average latency from multiple tests: 1000 times for small size messages in the user space, 100 times for large size messages in the user space, and 50 times in the kernel space.

2.3 Result analysis

First we compare the latency of memory registration and RDMA write with various size of messages in user space in Fig. 1. It is obvious that the cost of memory registration is so huge that it is much higher than the latency of RDMA operation itself, especially with small size messages. With such high cost, the benefit of RDMA is reduced, and furthermore, the latency of RDMA operation of small size messages, including memory registration and real RDMA write, makes it unattractive compared to traditional network protocol stack. The result is consistent with previous research [15, 18], but the difference is that in our experiments, the cost of memory registration are higher than real RDMA operations in some cases. It is reasonable because reducing memory registration cost is limited by performances of PCI bus of hosts, which improves very slowly, while the latency and bandwidth of network subsystems improve quickly.

We explain in Section 2.1 that the cost of memory registration consists of three main parts: maps the virtual address to physical address, pins the memory region, and registers to RDMA card. With such high latency of memory registrations, we want to know how those three parts contribute to whole costs. We examine the latency of memory registrations in kernel space. We use __get_free_pages to allocate memory regions and register the memory region using ib_reg_phys_mr, which is a kernel service provided by the kernel VAPI module. The memory region allocated by __get_free_pages is returned with physical address and physically contiguous, so there is no need to map address. Any memory region allocated in kernel space will not be swapped out any time, so there is no cost of pining memory. With such configurations, we expect that the cost of memory registration in kernel space only includes the latency of registering to RDMA card. Our results are presented in 2. First we find that latency in kernel space is about 40us less than that of user space, when the memory size is smaller than 32KB. Since the registration in kernel space only includes latency of registering to RDMA card and the registration in user space includes all three parts, we know that the costs of mapping address, pining memory, and crossing user-kernel interfaces count about half of the total latency, and cost of registering to physical card counts the other half. when the memory region is larger than 32KB, the latency of user space increases dramatically, but the latency of kernel space is still independent to the memory size. The reason is that in user space, the system call *malloc* dos not guarantee that allocated memory region is physically contiguous. In our experiments, we find that memory regions less than 32KB are contiguous, but it is not the case for larger regions. With separated physical memory regions, the latency of mapping address, pinning memory, and even registering to RDMA card should be higher, because kernel do those jobs in terms of physically region.

From previous experiments, we know that the latency of registering to RDMA card counts about 40us, 50% of total cost. Since other costs may be eliminated by allocating contiguous physical spaces and pre-pining, it is important to know that what is the main part of cost to register to RDMA card, and if it is possible to eliminate it. The cost of registering to RDMA card includes two parts: allocate a table in kernel memory and record phsical address of memory region, and write I/O registers of RDMA card to register memory information. With fast memory registration, user pre-allocates a table in kernel memory to record phsical address of memory region, and pre-writes I/O registers of RDMA card to register memory information, and only fills the table for phsical address of memory region during the real memory registration operations. We compare the latency of fast registration and ordinary registration in kernel space and show the results in Table 1. Amazingly, the latency of fast registration is so low that it can be almost ignored. It is obvious that the main part of latency in registering to RDMA card is the cost of communicating with I/O card and writing I/O registers.

Research in [18] showed that cost of fast memory registration in user space consists of two parts. First part is the cost of per registration, and second part is cost of per page. In [18], the cost of registering memory region is modeled as $T = a * p + b$, where a is the registration cost per page, and b is the overhead per operation, and p is the size of the memory region in pages. In their testbed, the costs of per page in registration is $0.77us$. The overhead per registration and deregistration operations is $7.42us$. With this result, we find that our design reduces the latency in the entire communication process by $T_r = 0.77 * p + 7.42$. Our results of fast memory registration in kernel space is consistent with the previous research, because the cost of kernel space fast registration is so small that the main cost of user space fast registration comes from latency of crossing user-kernel interfaces. Our results show that latency of switching environment is about $5us$, which is main part of cost per operations in previous model.

From our experimental results, we find that the latency of communicating with I/O card and writing I/O registers counts about half of the cost of memory registration, and unfortunately, unlike other parts of cost, it can not be elim-

inated by optimizing kernel and modifying software.That part of latency is still high enough, especially when compared to latency of RDMA write itself. Actually, although the cost of fast memory registration in user space is very low, compared to ordinary registrations, it is still almost same with the cost of real network operations, because of latency of switching environment. To improve RDMA performances, researchers considers several ways, such as memory registration cache [15, 20, 17, 18, 4, 14, 12]. In this paper, we reduce the overhead of memory registration and improve the performances of RDMA by overlaps memory registrations of client and server and allowing the applications submit RDMA write immediately after they finish local memory registrations.

3 Design of FRRWP

Before issuing real RDMA read/write operations, the client and server need to finish registration operations, and there are several synchronization message between the client and server to exchange peer *Rkey*. In the typical communication process, shown in Fig 3, the registration operations in both sides and the synchronization messages are totally sequential, in which both the client and server have to wait the complete of peer registrations.

In FRRWP, we change the flow of communication process to overlap the registrations, shown in Fig 4. The client first sends a synchronization message to the server to start a new RDMA transaction. Then both sides start the memory registrations. After that, one side sends a synchronization message to inform *Rkey* to the the other side where real RDMA write will be submitted. After both *Rkey* (peer memory region) and *Lkey* (local memory region) are received, the real RDMA write operation starts. In FRRWP, the registrations on both the client and server are processed in parallel, so the overall latency of RDMA is reduced.

From Fig 3, we find that the client or server is still blocked before the stage of a RDMA write, because they need to wait for the synchronization message with the *Rkey* being sent from the peer. After finish the local registration, the server or client application need poll or wait for the event of the incoming synchronization message (using RDMA receive operation). Before that, they cannot submit any RDMA write requests to the device driver. In this case, the overhead of context switching between the device driver and application is considerable. To improve the performance, we introduce a new operation, Conditional RDMA Write (CRW), in which, the RDMA write can be issued before receiving the peer *Rkey*. The device driver will hold CRW requests, until associated *Rkey* from the peer is received. Another operation, Send Tag for CRW (STCRW), is also introduced in the peer side to send the associated *Rkey* for the CRW. A CRW and a STCRW operations are coupled

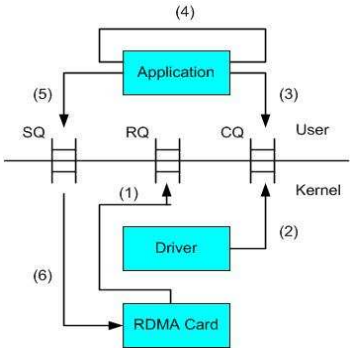


Figure 5. Traditional RDMA Operation.

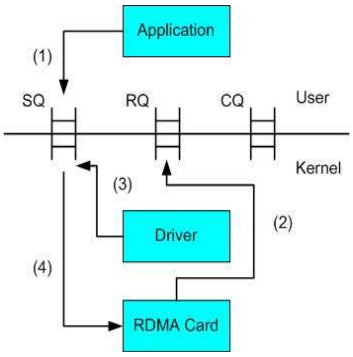
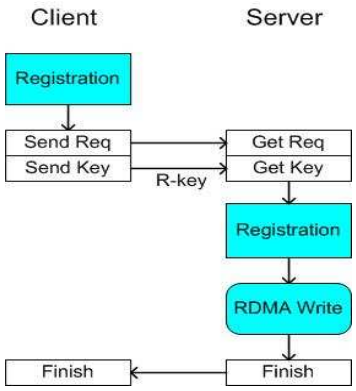


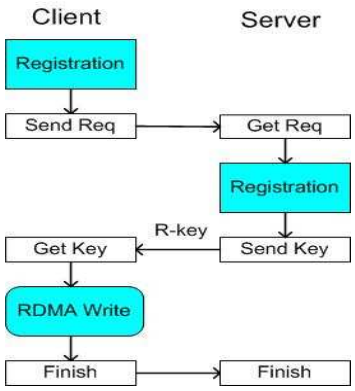
Figure 6. Conditional RDMA Write (CRW).

together by a common tag, CWTAG, which may be sent from a client to a server through a synchronization message at the beginning of the transaction. Using CRW, the client (or server) can submit a RDMA write to the device driver following the local registration without being blocked by the peer. After receiving a synchronization message containing a CWTAG from peer, the driver checks the issued CRW with the same CWTAG, and submits the real RDMA write operation along with the *Rkey* from the peer.

Fig 5 shows the interaction between the application and the kernel in traditional RDMA operations. (1) The RDMA card writes the synchronization message to the a buffer of the receive queue *(RQ)*. (2) The driver constructs a data structure to inform completion of the receive operation and insert it into the completion queue *(CQ)*. (3) The application polls the completion queue and retrieves the synchronization message. (4) The application processes the message and retrieves *Rkey*. (5) The application inserts a RDMA write request to the send queue *(SQ)* (6) The driver then submits the real RDMA write to the RDMA card. For comparison, Fig 6 shows our design of Conditional RDMA Write (CRW). (1) The application immediately inserts a conditional write request to the *(SQ)* without being blocked. Then, the application is free and the driver will take care of the following processes. (2) The RDMA card writes the synchronization message to the a buffer of the *(RQ)*. (3) The

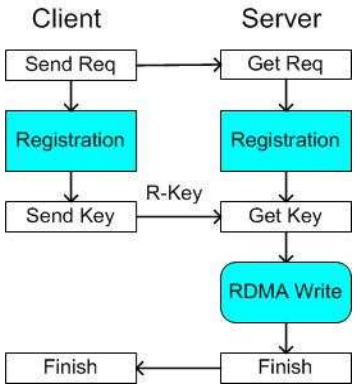


(a) Read

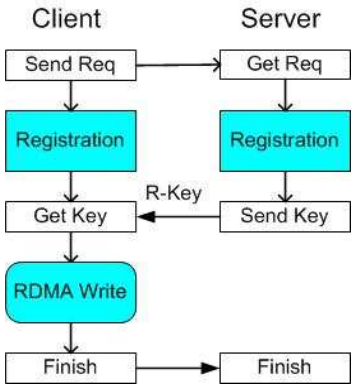


(b) Write

Figure 3. Typical RDMA Read and Write Process.



(a) Read



(b) Write

Figure 4. Read and Write Operations in FRRWP.

driver uses STCRW in the message to locate the RDMA requests with the same STCRW in the *(SQ)*. (4) The driver submits the RDMA write to the RDMA card with the *Rkey* in the message. Comparing Fig 5 and Fig 6, we find that Conditional RDMA Write (CRW) is more efficient: first, the new design removes Step 3 in the traditional RDMA process; second, the application does not have to wait for the *Rkey*.

4 Latency analysis

We expect that the FRRWP reduces the communication latency in the critical data path of RDMA operations. The benefit of FRRWP comes from two sides. First, the overlapped memory registrations between a client and a server; Second, the non-blocking Conditional RDMA Write (CRW).

Research in [18] showed that the cost of memory registration consists of two parts, the cost of each registration and the cost of each page. In [18], the cost of registering memory regions is modeled as $T = a * p + b$, where a is the registration cost per page, and b is the overhead per registration operation, and p is the size of the memory region in pages. In their testbed, the cost per page is $0.77us$. The overhead per registration is $7.42us$. With this result, we find that our design reduces the latency in the entire communication process by $T_r = 0.77 * p + 7.42$, because of overlapped memory registrations.

Comparing Fig 5 and Fig 6, we find that CRW reduces latency for following reasons. First, after receiving a synchronization message, the driver does not need to construct a data structure and insert it into the complete queue. The latency of this part is T_1. Second, the driver directly processes the message and sends a RDMA write to the card without

the participation of the application, so the latency caused by the operation that the application polls the complete queue and inserts a request to send queue is eliminated. They are T_2 and T_3, respectively. To find T_1, we use a test program which is a kernel module and performs 1000 times of constructing a completion data structure and inserting it into a queue. The program monitors the entire process and calculates the average time for each operation. The test machine in our lab is a *Dell Power Edge 420*, with a $2.8GHz$ *Intel Pentium-4* microprocessor, $2048M$ memory and a $40G$ IDE disk. The experimental result shows that T_1 is $4us$. T_2 and T_3 are the cost of context switch between the device driver and the application. We use a test program to monitor 1000 times of *getpid()* and find that the average cost per operation is $5us$. *getpid()* is a very simple system call which only returns an integer from a kernel data structure, so it reflects the minimum latency of switching environment. Actually, T_2 and T_3 should be large than $5us$, but we use it as an estimation. According to all those results, the latency T_c reduced by CRW is about $4+2*5=14us$.

Add T_r and T_c together, the cost saved by the FRRWP in the whole communication process is approximately $T = T_r + T_c = 21.42 + 0.77 * p$, where p is the size of the memory region in term of pages. We find that the minimum latency reduced by FRRWP is $22us$ per page.

5 Related Work

Previous researchers have evaluated the performances of RDMA at various platforms. In [11, 19, 9, 8, 10, 16, 17, 18], authors compare RDMA latency and bandwidth using InfiniBand. In [7, 9], performances of RDMA over IP are evaluated with RNIC. While RDMA decreases latency by eliminating unnecessary copies from network interface cards to application buffers, there are a number of challenges to be addressed. One of them is efficient communication buffer management to reduce memory registration and deregistration cost. Previous research [15, 17, 18, 14] shows that memory registration is an expensive operation. Our research in evaluating RDMA performances and memory registration costs is based on but different to previous work, because we compare cost of memory registration in both user space and kernel space, and analyze the three main parts of memory registration costs, then find where the main latency comes from.

Several studies have been done to reduce the overhead caused by memory registration and deregistration in RDMA. Tezuka *et al.* [15] propose a *pin-down cache* for Myrinet. Pin-down cache postpones deregistration of registered buffers and caches the registration information for possible future accesses to the same memory region. Zhou *et al.* [20] eliminate pinning and unpinning from registration and deregistration path by combining memory pinning and allocation together. They also demonstrate that *batched deregistration* is an efficient way to reduce average cost of memory deregistration. In [17], Wu *et al.* propose a two-level architecture, *FMRD*, for memory registration by adopting both pin-down cache and batched deregistration. Based on pin-down cache, a *lazy cache* is proposed in [14], which combines a cache of registration mapping with a lazy approach to memory deregistration. In [12], Ou *et al.* propose an effective cache scheme: Memory Registration Region Cache (MRRC), to minimize the cost of memory registration in the critical data path of RDMA operations. MRRC manages memory in terms of memory region, and replaces old memory regions according to both their sizes and recency. Both MRRC and FRRWP try to reduce the latency of RDMA operations in the critical data path, but they achieve the object through different ways. MRRC focuses on the latency of RDMA memory registration, and reduces the cost by using a novel cache. FRRWP concentrates on the entire process of RDMA communications and reduces the communication latency overlapping memory registrations between clients and servers.

In some application, memory region is predefined and can be pre-registered in the initialization phase to avoid extra cost in the critical path of data transferring. In the design of *Unifier* [19], the cache buffers are divided into two groups (ready buffers and raw buffers). The ready buffers are registered and resident in the system during the Unifier's life time. In the implementation of RDMA-Based MPI, Liu *et al.* [11] introduced a technique called *persistent buffer association*, in which buffers at both send and receiver side are allocated, registered, and associated during the initialization phase. In [4], a *firehose* algorithm is proposed for RDMA in shared memory system. The *firehose* algorithm starts by determining the largest amount of application memory that can be shared with remote machines, then all shared memory are pinned and registered, and linked to a *firehose* interface, from which remote machines can write and read shared memory at any time.

Other research focuses on reducing cost of memory registration directly. In RDMA Protocol Verbs Specifications (RDMAVS 1.0) [6] and Mellanox IB-Verbs extension (VAPI) [2], a new registration schema: Fast Memory Registration (FMR) is introduced, in which registration operations are divided into two distinct steps. In the first step, applications apply a handle and allocate resource in the NIC. This step can be done in the initialization of the application. In the second step, application issues the fast registration requests with the pre-allocated handle and the detail information of the memory region, then memory is pinned at last. The second step is finished before the RDMA read or write operations. Since the resource of NIC is pre-allocated, the overhead of FMR in the critical data path is smaller than that of the traditional memory registration operations. Experi-

mental results [14] show that the delay of memory registration is reduced 50us by using FMR with Intel Xeon 2.4GHz processors. In [18], Wu *et al.* propose an *Optimistic Group Registration* (OGR) to reduce the cost of memory registration for noncontiguous accesses. *Optimistic Group Registration* integrates multiple registrations of noncontiguous memories into one operation, and registers a large memory region containing several noncontiguous buffers.

6 Conclusions

In this paper, we evaluate the cost of memory registration in both user and kernel space. We analyze three main parts of latency and find out which part contributes most to the total costs. Based on latency analysis, we propose a new communication scheme between a RDMA client and server, Fast RDMA Read/Write Process (FRRWP), to reduce the overhead of the memory registration and synchronization messages in the critical data path. FRRWP overlaps memory registrations between RDMA clients and servers. It allows the applications to submit a RDMA write immediately after they finish local memory registrations, without waiting for the confirmation of registrations from the peer node. Our analysis show that FRRWP dramatically reduces the total communication latency in the critical data path of RDMA.

Acknowledgments

This work was supported in part by the Research Office under a Faculty Research Grant and the Center for Manufacturing Research at Tennessee Technological University. It was also partially supported by the 973 Program of China under contract No. 2004CB318202, and Faculty Research Grant at Institute of Computing Technology, Chinese Academy of Sciences. The authors would like to thank the REU student, Karthik Sankar, for conducting literature survey on RDMA. He is supported by the US National Science Foundation under a REU grant SCI-0453438. The authors would also like thank Ian Jiang for helping collect part of the experimental data.

References

[1] Infiniband trade association. infiniband architecture specification, release 1.0, october 24, 2000.

[2] Mellanox technologies. mellanox IB-Verbs API(VAPI), rev. 0.95, march 2003.

[3] RDMA consortium. architectural specifications for RDMA over TCP/IP.

[4] C. Bell and D. Bonachea. A new dma registration strategy for pinning-based high performance networks. In *17th International Parallel and Distributed Processing Symposium*, 2003.

[5] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. K. Su. Myrinet: A gigabitperSecond local area network. *IEEE-Micro*, 15(1):29C366, February 1995.

[6] J. Hilland, P. Culley, J. Pinkerton, and R. Recio. RDMA protocol verbs specification (version 1.0). Technical report, RDMA Consortium, April 2003.

[7] H. W. Jin, S. Narravula, G. Brown, K. Vaidyanathan, P. Balaji, and D. K. Panda. Performance evaluation of rdma over ip: A case study with the ammasso gigabit ethernet nic. In *Workshop on High-Performance Interconnects for Distributed Computing (at HPDC'05)*, July 2005.

[8] S. Liang, R. Noronha, and D. Panda. Exploiting remote memory in infiniband clusters using a high performance network block device. Technical report, Ohio State University.

[9] S. Liang, R. Noronha, and D. K. Panda. Swapping to remote memory over infiniband: An approach using a high performance network block device. In *IEEE International Conference on Cluster Computing (Cluster 2005)*, September 2005.

[10] J. Liu, D. K. Panda, and M. Banikazemi. Evaluating the impact of rdma on storage i/o over infiniband. In *SAN-03 Workshop*, Feb. 2004.

[11] J. Liu, J. Wu, S. Kini, P. Wyckoff, and D. K. Panda. High performance rdma-based mpi implementation over infiniband. In *ICS '03*, June 2003.

[12] L. Ou, X. He, and J. Han. Mrrc: A efficient cache for fast memory registration in rdma. In *Proc. of the NASA/IEEE Conference on Mass Storage Systems and Technologies (MSST)*, 2006.

[13] F. Petrini, W. C. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The quadrics network (QsNet): High-performance clustering technology. In *In Hot Interconnects*, 2001.

[14] M. Rangarajan and L. Iftode. Building a user-level direct access file system over infiniband. In *3rd Workshop on Novel Uses of System Area Networks*, 2004.

[15] H. Tezuka, F. OCarroll, A. Hori, and Y. I. Pindown. Pindown cache: A virtual memory management technique for zero-copy communication. In *Int. Parallel Processing Symposium*, March 1998.

[16] V. Tipparaju, G. Santhanaraman, J. Nieplocha, and D. K. Panda. Host-assisted zero-copy remote memory access communication on infiniband. In *Int'l Parallel and Distributed Processing Symposium (IPDPS 04)*, April 2004.

[17] J. Wu, P. Wyckoff, and D. K. Panda. PVFS over InfiniBand: Design and performance evaluation. In *International Conference on Parallel Processing*, Oct 2003.

[18] J. Wu, P. Wyckoff, and D. K. Panda. Supporting efficient noncontiguous access in PVFS over InfiniBand. In *Cluster 2003 Conference*, December 2003.

[19] J. Wu, P. Wyckoff, D. K. Panda, and R. Ross. Unifier: unifying cache management and communication buffer management for pvfs over infiniband. In *CCGrid '04*, April 2004.

[20] Y. Zhou, A. Bilas, S. Jagannathan, C. Dubnicki, J. F. Philbin, and K. Li. Experiences with vi communication for database storage. In *ISCA*, 2002.