Journal of Network and Computer Applications 32 (2009) 642–651

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

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

An efficient design for fast memory registration in RDMA

Li Ou[a], Xubin He[b,*], Jizhong Han[c]

[a] DELL Inc., USA
[b] Electrical and Computer Engineering Department, Tennessee Technological University, Cookeville, TN 38505, USA
[c] Institute of Computing Technology, Chinese Academy of Sciences, China

ARTICLE INFO

Article history:
Received 7 October 2007
Received in revised form
26 March 2008
Accepted 5 July 2008

Keywords:
RDMA
Memory registration
Cache
Performance evaluation

ABSTRACT

Remote Direct Memory Access (RDMA) improves network bandwidth and reduces latency by eliminating unnecessary copies from network interface card to application buffers, but the communication buffer management to reduce memory registration and deregistration cost is a significant challenge to be addressed. Previous studies use pin-down cache and batched deregistration, but only simple LRU is used as a replacement algorithm to manage cache space. In this paper, we evaluate the cost of memory registration in both user and kernel spaces. Based on our analysis, we reduce the overhead of communication buffer management in two aspects simultaneously: utilize a Memory Registration Region Cache (MRRC), and optimize the RDMA communication process of clients and servers with Fast RDMA Read and Write Process (FRRWP). MRRC manages memory in terms of memory region, and replaces old memory regions according to both their sizes and recency. 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 compare the performance of MRRC and FRRWP with traditional RDMA operations. The results show that our new design improves the total cost of memory registrations and overall communication latency by up to 70%.

1. Introduction

The advent of networking technologies facilitates the service of storage over networks. Remote Direct Memory Access (RDMA) is emerging as the central feature in modern network interconnects. It offers low latency, high throughput, and low CPU overhead communication in network storage systems. These enabling technologies eliminate or reduce costs of memory copy, network access, interrupt, and protocol processing in the network subsystem. Interconnects like InfiniBand (Infiniband Trade Association, 2000), Myrinet (Boden et al., 1995), and Quadrics (Petrini et al., 2001) have long introduced RDMA in LAN environments. RDMA over IP has been developed to extend the benefits of RDMA across the WAN/Internet. The RDMA Consortium has proposed the RDMA Protocol Verbs Specifications (RDMAVS 1.0) (Hilland et al., 2003) to standardize the efforts.

While RDMA improves network bandwidth and decreases latency by eliminating unnecessary copies from network interface cards (NICs) to application buffers, a number of challenges must be addressed. One of the most significant issues is efficient communication buffer management to reduce memory registration and deregistration costs. Previous research (Bell and Bonachea, 2003; Rangarajan and Iftode, 2004; Tezuka et al., 1998; Wu et al., 2003a, b; Zhou et al., 2002) 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 NIC, such as InfiniBand Host Control Adapter (HCA) and RNIC of RDMA over IP. Experimental results (Tezuka et al., 1998) from Myrinet and an extremely old Pentium Pro machine (200 MHz) show that one memory page transfer (4 KB) only takes 25.6 μs while the memory registration cost is approximately 26 μs. Even with a much faster configuration (InfiniBand HCA and Intel Xeon 2.4 GHz processor) (Wu et al., 2003b), the registration of a memory page still costs about 7 μs, almost the same as the transfer time for that page. The cost and overhead of memory registration dramatically degrade the performance of RDMA and increase network latency in the critical data path of I/O operations.

Several attempts (Bell and Bonachea, 2003; Rangarajan and Iftode, 2004; Tezuka et al., 1998; Wu et al., 2003a, b; Zhou et al., 2002) have been made to reduce the overhead of memory registration in RDMA. In some special environments (Bell and Bonachea, 2003; Liu et al., 2003; Wu et al., 2004), the memory region used by applications is predefined and can be preregistered in the initialization phase; thus in the critical path of data transferring, memory registration is not necessary. In general applications, since dynamic registration and deregistration cannot

* Corresponding author. Tel.: +1 931 372 3462; fax: +1 931 372 3436.
E-mail addresses: li_ou@dell.com (L. Ou), hexb@tntech.edu (X. He), hjz@ict.ac.cn (J. Han).

1084-8045/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2008.07.008

be avoided, a pin-down cache (Tezuka et al., 1998) is incorporated in the memory manager. A pin-down cache delays deregistration of registered buffers and caches their registration information for future accesses of the same memory region. Several cache designs for memory registration (Rangarajan and Iftode, 2004; Wu et al., 2003a) are proposed based on the pin-down cache to take advantage of temporal locality of memory accesses of RDMA. Current memory registration caches manage memories at the page level and only consider LRU as the replacement algorithm. Most applications using RDMA register and deregister memory regions containing multiple continuous or noncontiguous memory pages; thus, page level management for registration caches is not efficient enough. Furthermore, with multiple page memory regions, the locality of memory accesses is also changed, and the general LRU algorithm is probably not the best choice.

In this paper, we evaluate the cost of memory registration in both user and kernel spaces. We analyze latency of memory registration and find three main parts which contributes most to the total costs. Based on our analysis, we reduce the overhead of memory registration in two aspects simultaneously: utilize a memory registration cache, and optimize the RDMA communication process of clients and servers.

We propose a new cache management scheme, Memory Registration Region Cache (MRRC), to minimize the cost of memory registration and deregistration in the critical data path. MRRC manages memory in terms of memory regions, which contain one or more memory pages, and considers pipelining between RDMA operations and memory registrations. MRRC organizes the cache stack using the LRU algorithm, but divides the stack into three sections and evicts memory regions from the *eviction section* according to both the size and recency.

We then propose a new communication scheme between an 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 an RDMA write immediately after they finish local memory registrations, without waiting for the confirmation of registrations from the peer node.

The performance of MRRC is compared with traditional RDMA memory registration operations and other typical registration cache management algorithms such as pin-down cache (Tezuka et al., 1998) and FMRD (Wu et al., 2003a). The results show that compared to traditional RDMA memory registration, MRRC improves the total cost of memory registrations by up to 70%. We compare the latency of FRRWP with traditional RDMA operations using a mathematic model. The results show that FRRWP reduces the total communication latency in the critical data path by 68%.

The rest of the paper is organized as follows. Background material is presented in Section 2. Section 3 examines related work. Section 4 evaluates the cost of memory registration in both user and kernel space. Sections 5 and 6 describe the design issues of MRRC and FRRWP in detail, respectively. MRRC is compared to the previous efforts to improve RDMA performance in Section 7. The latency of FRRWP is analyzed in Section 8. Section 9 draws the conclusions.

2. Background review

In RDMA, an NIC (RNIC, RDMA NIC) 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 the 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 mapping and pinning, 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. When the number of registered buffers exceeds this limit, the application needs to deregister memory and free resources on the NIC, which involves the unpinning 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 (200 MHz), one memory page (4 KB) registration takes 26 μs (Tezuka et al., 1998), while the same operation only need 7 μs with a much faster Intel Xeon 2.4 GHz processor (Wu et al., 2003b). Although high performance servers reduce time of memory registrations, the cost is still almost same as the network latency of the contemporary interconnect used by servers (Tezuka et al., 1998; Wu et al., 2003b). If every RDMA operation is blocked by the registration and deregistration, the overhead is very large and overall communication latency is very high. Previous studies (Tezuka et al., 1998; Wu et al., 2003b) 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 (Tezuka et al., 1998; Wu et al., 2003b) show that if the message size of most operations is less than 1 K, RDMA with normal memory registration may not provide better performance than the traditional way, and in some cases, even worse.

The simple solution of pre-registering all buffers at application startup is not general and cannot be applied in most systems, since they use large caches and require large amount of memory buffers. Dynamic memory registration is not avoided if applications keep using different buffers. To improve RDMA performance, it is very important to reduce the overhead of memory registration and deregistration.

3. Related work

Several studies have improved the performance of memory registration and deregistration of RDMA. Tezuka et al. (1998) propose a *pin-down cache* for Myrinet. A pin-down cache delays the deregistration of registered buffers and caches their registration information for future accesses of the same memory region. LRU is used as a replacement algorithm in a pin-down cache to manage memory registrations. Zhou et al. (2002) eliminate pinning and unpinning from the registration and deregistration paths by combining memory pinning and allocation together. They also demonstrated that *batched deregistration* is an efficient way to reduce the average cost of deregistration memory. Wu et al. (2003a), propose a two-level architecture, *FMRD*, for memory registration by adopting both a pin-down cache and batched deregistration. *FMRD* also takes advantage of the Mellanox fast memory region registration extension in VAPI (Mellanox Technologies, 2003). Based on the pin-down cache, a *lazy cache* is proposed in Rangarajan and Iftode (2004), which combines a cache of registration mappings with a lazy approach to memory deregistration. The *lazy cache* is implemented using an

LRU list with a fixed size hash table for fast access, and allows applications to query the size of the buffer/descriptor set. This work in memory registration builds upon but is different from the previous studies because a new replacement algorithm for MRRC considers the effects of various sizes of memory regions.

In some application, memory regions are predefined and can be preregistered in the initialization phase to avoid extra cost in the critical path of data transferring. In the design of *Unifier* (Wu et al., 2004), 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. (2003) introduced a technique called *persistent buffer association*, in which buffers at both the sender and receiver sides are allocated, registered, and associated during the initialization phase. In Bell and Bonachea (2003), a *firehose* algorithm is proposed for RDMA in a 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 regions 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 directly reducing the cost of memory registration. In RDMA Protocol Verbs Specifications (RDMAVS 1.0) (Hilland et al., 2003) and the Mellanox IB-Verbs extension (VAPI) (Mellanox Technologies, 2003), 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 a resource in the NIC. This step can be done in the initialization of the application. In the second step, the application issues the fast registration requests with the pre-allocated handle and the detail information of the memory region; then the memory is pinned at last. The second step is finished before any RDMA read or write operations. Since the resources of the NIC are pre-allocated, the overhead of FMR in the critical data path is smaller than that of traditional memory registration operations. Experimental results (Rangarajan and Iftode, 2004) show that the delay of memory registration is reduced by 50 μs by using FMR with Intel Xeon 2.4 GHz processors. Wu et al. (2003b) propose an *Optimistic Group Registration* (OGR) to reduce the cost of memory registration for noncontiguous accesses. *OGR* integrates multiple registrations of noncontiguous memories into one operation, and registers a large memory region containing several noncontiguous buffers.

Caching is a common technique for improving the performance of any I/O system. Researchers have developed many algorithms to manage the buffer cache, such as LRU (Dan and Towsley, 1990), MRU (Denning, 1968), LFU, FBR (Robinson and Devarakonda, 1990), LRU-k (O'Neil et al., 1993), 2Q (Johnson and Shasha, 1995), LIRS (Jiang and Zhang, 2002), and ARC (Megiddo and Modha, 2003). Jiang et al. (2005) propose a new management scheme, DULO, to balance the temporal and spatial locality of workload. Gill and Modha (2005) use a new ordering algorithm, WOW, to resort the writing sequences of non-volatile cache by combining both spatial and temporal localities.

4. Cost analysis of memory registrations

To study the cost of memory registrations in RDMA, we set up our experimental environments with two servers and InfiniBand network. The server is equipped with a 2.8 GHz *Intel P4* microprocessor, 1024 MB memory, and an InfiniBand HCA. Two servers are connected with an InfiniBand switch.

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 1 to 128 KB, 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.

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 the previous research (Tezuka et al., 1998; Wu et al., 2003b), 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 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 Fig. 2. First we find that latency in kernel space is about half of user space, when the memory size is smaller than 32 KB. 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 32 KB, 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

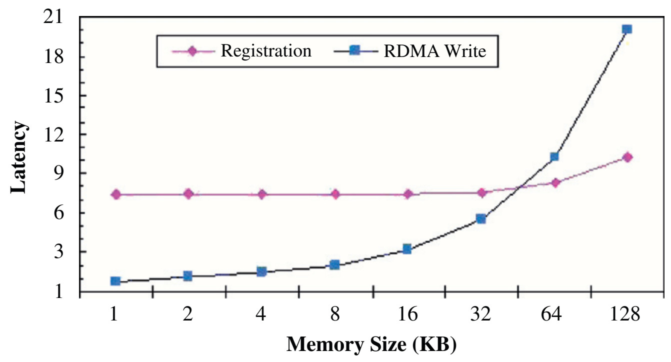

Fig. 1. Comparison between latency of memory registration and RDMA write with various size of messages in user space. The data are normalized to the latency of 1 KB RDMA write.

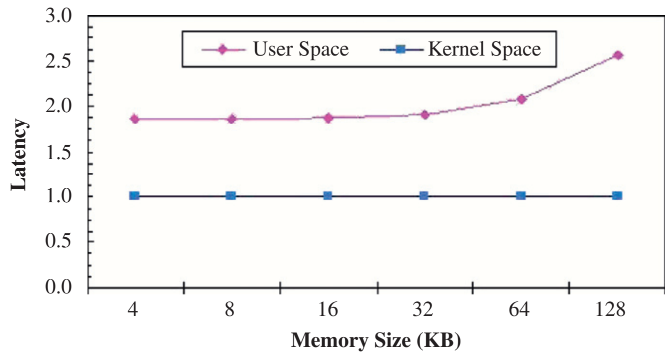

Fig. 2. Comparison of memory registration latency between user space and kernel space with various size of messages. The data are normalized to the latency of 4 KB kernel space.

region is physically contiguous. In our experiments, we find that memory regions less than 32 KB 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 the previous experiments, we know that the latency of registering to RDMA card counts about 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 physical address of memory region, and write I/O registers of RDMA card to register memory information. With FMR, user pre-allocates a table in kernel memory to record physical address of memory region, and pre-writes I/O registers of RDMA card to register memory information, and only fills the table for physical address of memory region during the real memory registration operations. We compare the latency of fast registration and ordinary registration in kernel space and present 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 Wu et al. (2003b) showed that cost of FMR in user space consists of two parts. First part is the cost of per registration, and second part is cost of per page. In Wu et al. (2003b), 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 0.77 μs. The overhead per registration and deregistration operations is 7.42 μs. 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 FMR 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 5 μs, 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 eliminated 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 FMR in user space is very low, compared to ordinary registrations, it is still almost same with the

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

Memory size (KB)	Fast MR	Ordinary MR
4	1.000	79.124
8	1.014	79.588
16	1.108	79.313
32	1.237	79.599
64	1.550	79.387
128	2.027	79.822

The data are normalized to the latency of 4 KB fast MR.

cost of real network operations, because of latency of switching environment. In this paper, we reduce the overhead of memory registration and improve the performances of RDMA by using MRRC, and FRRWP, which overlaps memory registrations of client and server and reduces the overhead of synchronization messages.

5. Memory registration region cache (MRRC)

In RDMA operation, memory is accessed in terms of region, which includes several physical pages, so memory is also registered in gratitude of memory region. A memory region could be a bulk of contiguous virtual memory, or a list of noncontiguous physical memory pages. In this design, we use an MRRC to manage information of all registered memory regions. The size of the MRRC is equal to the maximal memory pages an NIC allows to register. The deregistration of a memory region is delayed until the MRRC cache is full, and after that, a replacement algorithm, *Memory Resorting and Eviction* (MRE) is used to choose regions for batched deregistration according to their size and recency. Any RDMA operation could be issued directly without memory registration if the memory region it reads or writes has already stayed in the MRRC, otherwise, a new registration operation is required and the corresponding information is updated into MRRC.

Since memory is managed in terms of regions, management policy of MRRC is different from traditional buffer caches which only manage fixed size memory pages. There are four possibilities for the relationship between a memory region newly requested by an RDMA operation and memory regions already cached. First, the new region exactly matches a cached region; thus, no registration is needed for actual RDMA operations. Second, the new region is a subset of a cached region, and also no further registration is needed. Third, there is an intersection between the new region and a cached region. Registering the new region directly is a simple but costly solution, since not only cache space is wasted by duplicated memory pages, but also registering time is long due to the full size memory region. In our design, the new region is divided into two parts, the first part is totally matched with or a subset of a cached region, and the second part is treated as a new region which is registered immediately. This design not only saves the space of the MRRC, but also improves system performance by pipelining memory registration and RDMA operations. In the same time of registration of the second part, the RDMA operation for the first part can be issued to reduce total response time. In the last case, there is no overlap between the new region and any cached region, and a new registration operation is necessary.

The MRRC is designed by using the LRU algorithm and its data structure: the LRU stack. The MRRC stack is partitioned into two sections (shown in Fig. 3), similar to the DULO cache (Jiang et al., 2005). The top part is the *frequent reference* section used for admitting newly accessed memory regions. The lower part is the

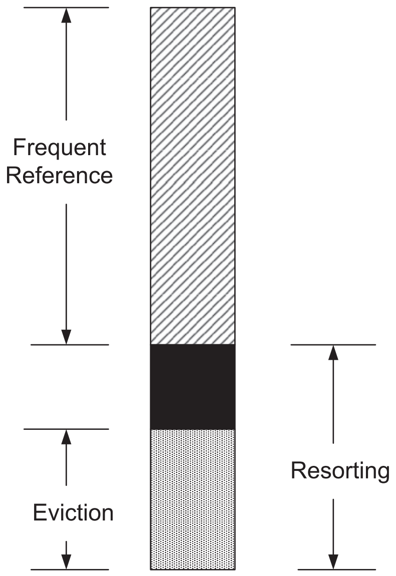

Fig. 3. LRU stack of MRRC.

Resorting section in which all regions are treated as candidate for eviction. The *Resorting* section is further divided into two segments. The lower part is the *eviction* segment. All regions in this segment are ready to be evicted from the cache and thus to be deregistered.

MRE replaces memory regions in the bottom of the stack according to both their recency and size. MRE prefers to replace large regions first, because a large region consumes more cache space, and small regions need more registration time compared to one large region with the same memory size. Research in Wu et al. (2003b) showed that cost of memory registration consists of two parts. The first part is the cost per registration, and second part is the cost per page. In Wu et al. (2003b), the cost of registering a memory region is modeled as $T = a * p + b$, where a is the registration cost per page, b is the overhead per operation, and p is the size of the memory region in pages. The same cost equation can be applied to deregister a region with different values of a and b. In their testbed, the costs of registration and deregistration per page are 0.77 μs and 0.22 μs, respectively. The overheads per registration and deregistration operations are 7.42 and 1.1 μs. From those results, it is easy to understand that registering multiple small regions is more expensive than registering large regions with the same number of pages. From this model, it is obvious that one-by-one deregistration is not efficient. In MRRC, a batched deregistration scheme is adopted.

Whenever the cache is full, MRE algorithm resorts all memory regions in the *Resorting* section, according to their *eviction factor*. The *eviction factor*, a function of size and recency of a memory region, is identified by *evictfact*(s, r). In the current design, *evictfact*(s, r) is defined as $r + 1/s$, where s is the size of a memory region and r is the system recency value. System keeps a global value of r. In the initialization phase, r is set to 0. Every time the cache is full, the r is reset to the *eviction factor* of the memory region in the bottom of the stack. The *eviction factor* of a memory region is not renewed unless it is zero, or the region is accessed again and sent to the top of the stack, in which case the *eviction factor* is set to zero again. All regions are sorted according to their *eviction factor*: the smaller of the *eviction factor*, the closer of a region to the bottom of the stack. At the end of the resorting, all regions in the *eviction* segment are deregistered in one operation and evicted from the cache space. Fig. 4 outlines the MRRC and MRE algorithms.

```
$r = 0$;

/* procedure to be invoked upon a reference to memory region b */

if b is in cache
    move b to the top of the stack;
else if b belongs to memory region c
    move c to the top of the stack;
else if b overlaps with memory region d {
    $u = b\&\&d$;
    $v = b - u$;
    move d to the top of the stack;
    register v and add v to the top of the stack;
}
else
    register b and add b to the top of the stack;

/* procedure to be invoked upon a full cache*/

/* e is the region at the bottom of the stack */

r = e.evictfact;
for each region a in resorting section
    a.evictfact = r + 1/s;  /* s is size of a region */;
Resort each region in resorting section according to evictfact;
Batched deregister all regions in Eviction Segment;
Evict all regions in Eviction Segment;
```

Fig. 4. MRRC and MRE algorithms.

6. Fast RDMA Read and Write Process (FRRWP)

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. In last section, we attempted to reduce the overhead of memory registration directly using MRRC. Another way to improve the performance of RDMA is to optimize the RDMA communication process of clients and servers, by overlapping the memory registrations between the client and the server, and reducing the overhead of synchronization messages.

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. 5, the registration operations in both sides and the synchronization messages are totally sequential, in which both the client and server have to wait for the completion of peer registrations.

In FRRWP, we change the flow of communication process to overlap the registrations, shown in Fig. 6. 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 other side where real RDMA write will be submitted. After both *Rkey* (peer memory region) and *Lkey* (local memory region)

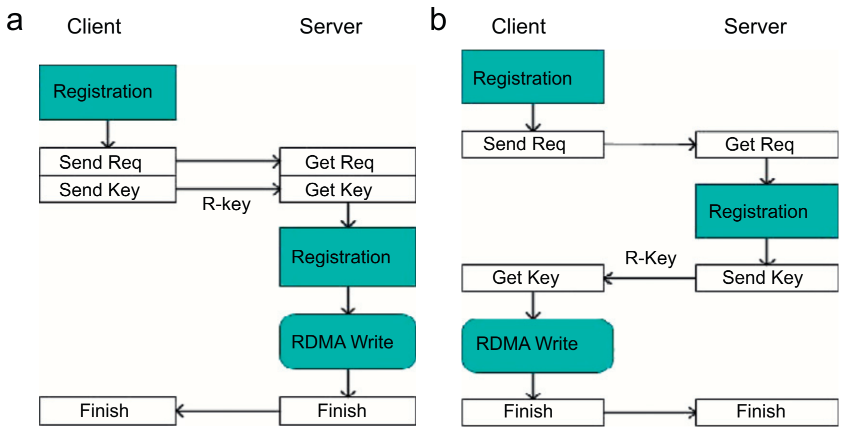

Fig. 5. Typical RDMA Read and Write Process. (a) Read and (b) write.

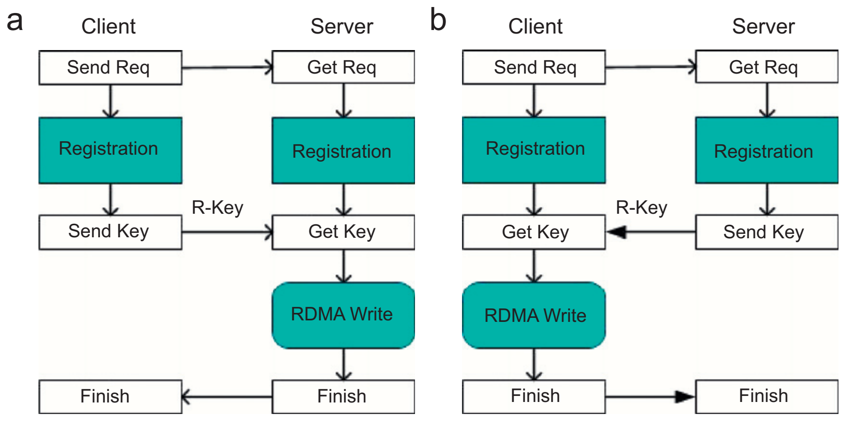

Fig. 6. Read and write operations in FRRWP. (a) Read and (b) write.

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. 5, 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 finishing of 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 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 an 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. 7 shows the interaction between the application and the kernel in traditional RDMA operations. (1) The RDMA card writes

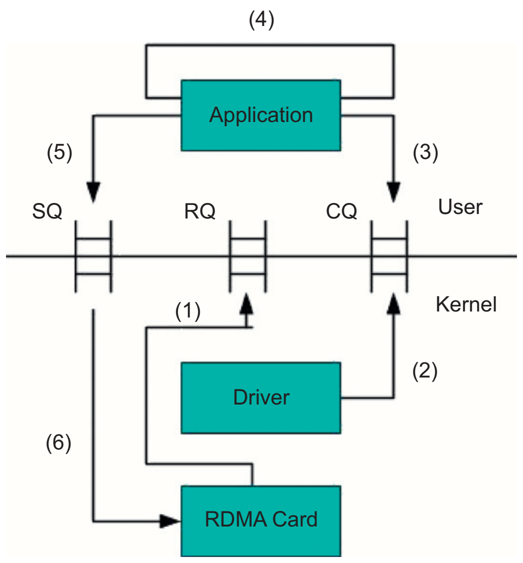

Fig. 7. Traditional RDMA operation.

the synchronization message to the 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 CQ 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. 8 shows our design of 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 buffer of the *RQ*. (3) The 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 Figs. 7 and 8, we find that 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*.

7. Simulation results

Trace-driven simulation is used to evaluate the MRRC design. A simulator is developed to simulate the cache hit ratio of memory registrations. Hit ratio here is defined as the ratio of the number of cached memory registrations to the total number of requests in the trace. In order to compare MRRC to previous efforts, the simulator implements multiple algorithms, including MRRC, and the pin-down cache (Tezuka et al., 1998). A memory page size is 4 KB.

To evaluate caching algorithms and policies, we use two buffer cache access traces. Table 2 presents the characteristics of the three traces.

The HP Cello92 trace was collected at Hewlett-Packard Laboratories in 1992 (Ruemmler and Wilkes, 1993). It captured all L2 disk I/O requests in Cello, which is a timesharing system used by a group of researchers to do simulations, compilation, editing, and e-mail, from April 18 to June 19. We use the trace collected on April 18 as the workload. The HTTPD workload was generated by a seven-node IBM SP2 parallel web server (Katz et al., 1994) serving a 524 MB data set. Multiple http servers share the same files, although they seldom read files at the same time. Based on the outputs of the simulator running the above two traces, the total time for registering all memory regions is calculated using the following formula:

$$T_{total} = \underset{i=0}{\overset{n}{sum}}(ismiss(i) * T_r(i)) + \underset{j=0}{\overset{m}{sum}}(T_{ur}(j)) \qquad (1)$$

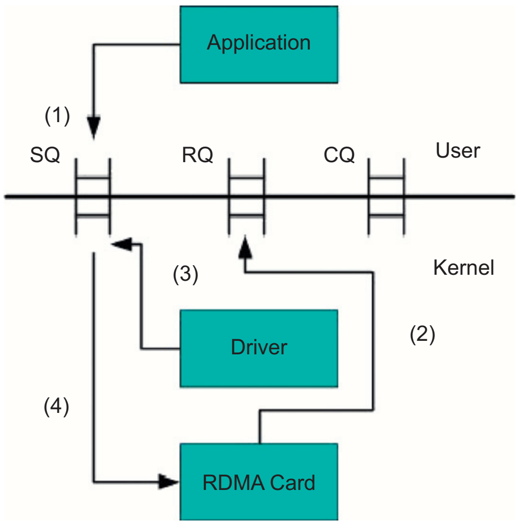

Fig. 8. Conditional RDMA Write (CRW).

Table 2
Characteristics of the two traces used in the study

Trace	Clients	Client cache	IOs (millions)	Capacity (GB)
Cello92	1	30 MB	0.5 per day	10.4
HTTPD	7	–	1.1	0.5

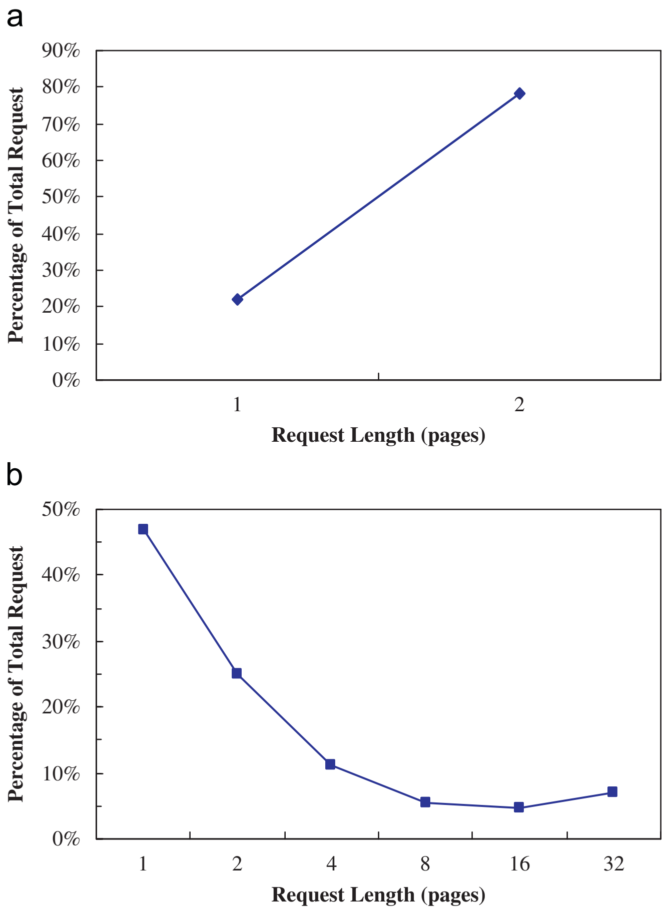

Fig. 9. Distribution of request size for (a) Cell92 and (b) HTTD traces. A point (L, P) on the curve indicates that the size of P percent of total requests is L pages.

where n is the total number of memory regions used in the trace, and m is the total number of deregistered memory regions. T_r and T_{ur} are the cost of actual memory registration and deregistration, respectively. T_r and T_{ur} are calculated based the cost model explained in Section 5. *ismiss*(i) is a Boolean function of the memory region. It outputs 1 if the corresponding region cannot be found in the cache space; otherwise, it outputs 0.

Fig. 9 shows the distribution of request sizes grouped by powers of two.[1] A request size of the two traces is expressed in terms of the number of memory pages. Most requests in the Cello92 trace are small: the maximum request size is only 2 pages. The size of requests in the HTTPD trace varies from 1 page to 32 pages. The difference comes from the environments where two traces were collected. The Cello92 trace captured all L2 disk I/O requests in Cello, in which small read or write requests were issued one by one. The HTTPD workload was generated by a seven-node IBM SP2 parallel web server, where only read requests were applied to the whole file. The distribution of the request

[1] Request sizes that are not powers of two are rounded down to the nearest power of two.

sizes of the traces impacts the results of the simulation because each request is accompanied by a memory registration operation. The request size determines the size of a memory region, and thus defines the behavior of cache replacement algorithms.

Fig. 10 compares the memory region hit ratios between MRRC and the pin-down cache with various cache sizes under the two traces. The data shows that under the HTTPD trace, MRRC has a 10% hit ratio improvement compared to the pin-down cache, but the difference is less obvious under the Cello92 trace. The previous results in Fig. 9 show that the distribution of request sizes in the two traces is totally different. This difference is the reason for resulting hit ratios. HTTPD trace has a wide distribution of request sizes and many large requests (up to 32 pages). Such character of the HTTPD trace allows MRRC to group requests by their sizes and optimize hit rates by giving small requests more chances to stay in the cache space. On the contrary, the variation of the request size in the Cello92 trace is small, and two page requests account almost 80% of the total requests, so most requests have the same size. MRRC cannot distinguish them with memory region size, so in most time, only recency is used to decide the life time of memory regions in the cache space. Thus, MRRC behaves similar to the traditional pin-down caches for the Cello92 trace.

Fig. 11 shows the total registration and deregistration time of MRRC and the pin-down cache under the two traces. The time is calculated with the formula explained in the beginning of this section. The total cost in terms of registration/deregistration time follows the trend of the hit ratios. MRRC provides a higher cache hit ratio, and thus reduces the actual number of registration operations. Moreover, the replacement algorithm of MRRC favors small requests, because registering multiple small regions is more expensive than registering large regions with the same number of pages. This small-request-weighted policy further reduces memory registration costs. The improvement of MRRC under the HTTPD trace is obvious, because of enough small requests. Although the difference of request sizes under the Cello92 trace is less obvious, MRRC still reduces the total cost with its batched registration. The maximum cost improvement is approximately 10%. As a comparison, Fig. 12 shows the improvement of MRRC over the basic RDMA operations without memory registration caches. The results clearly indicate how a cache space provided by MRRC and the related replacement algorithm could dramatically reduce RDMA memory management costs.

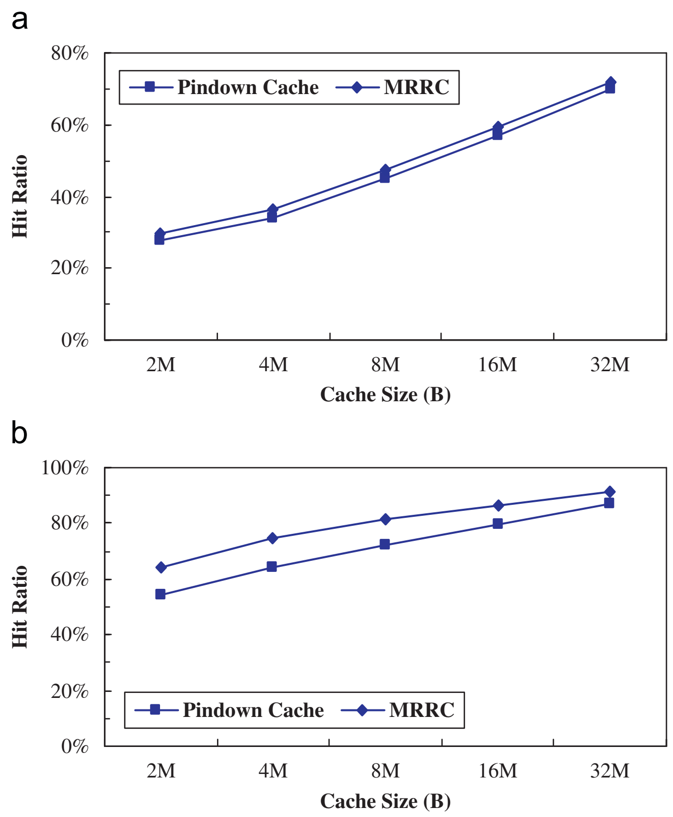

Fig. 10. Memory registration hit ratios with various cache sizes ((a) Cello92 and (b) HTTPD traces).

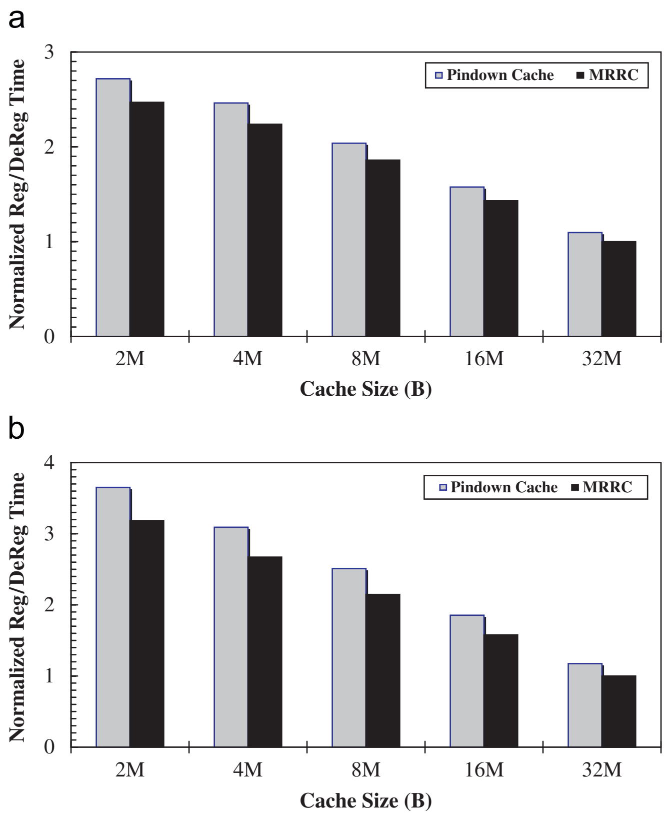

Fig. 11. Registration/deregistration costs with various cache sizes ((a) Cello92 and (b) HTTPD traces). The data are normalized to the time of MRRC with 32 MB cache size.

8. 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 CRW.

Research in Wu et al. (2003b) showed that the cost of memory registration consists of two parts, the cost of each registration and the cost of each page. In Wu et al. (2003b), 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 0.77 µs. The overhead per registration is 7.42 µs. 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 Figs. 7 and 8, 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 CQ. The latency of this part is T_1. Second, the

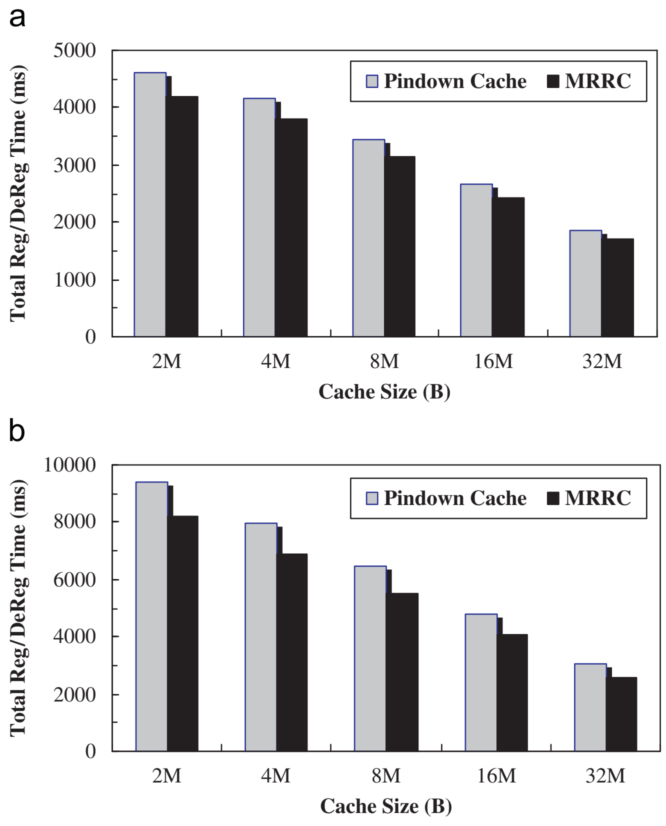

Fig. 12. Improvement of MRRC over the basic RDMA operations without the registration cache. (a) Cello92 and (b) HTTPD.

driver directly processes the message and sends an RDMA write to the card without the participation of the application, so the latency caused by the operation that the application polls the CQ and inserts a request to SQ 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 experimental result shows that T_1 is 2 μs. 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 3 μs. *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 5 μs, but we use it as an estimation. According to all those results, the latency T_c reduced by CRW is about $2 + 2 * 3 = 8$ us.

Add T_r and T_c together, the cost saved by the FRRWP in the whole communication process is approximately $T = T_r + T_c = 15.42 + 0.77 * p$, where p is the size of the memory region in term of pages. We find that the minimum latency reduced by FRRWP is 15 μs. FRRWP reduces the total communication latency in the critical data path by 68%.

9. Conclusions

In this paper, we evaluate the cost of memory registration in both user and kernel spaces. We analyze latency of memory registration and find three main parts which contributes most to the total costs. Based on our analysis, we reduce the overhead of memory registration in two aspects simultaneously: utilize a memory registration cache, and optimize the RDMA communication process of clients and servers.

We propose a new cache management scheme: *MRRC*, to improve performance of memory registration and deregistration of RDMA. *MRRC* manages memory in terms of memory region and considers pipelining between RDMA operations and memory registrations. MRRC uses MRE as a replacement algorithm, which considers both the size and recency of memory regions, because registering multiple small regions is more expensive than registering large regions with the same number of pages. To further reduce costs, MRRC adopts batched deregistration to avoid deregistering regions every replacement of the cache.

We propose a new communication scheme between an RDMA client and server, Fast RDMA Read and 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 an RDMA write immediately after they finish local memory registrations, without waiting for the confirmation of registrations from the peer node.

We have evaluated our *MRRC* and other typical registration cache designs using simulations under various workloads. The results show that *MRRC* can efficiently increase the cache hit ratios for memory registration by 10% and improves the total response time by up to 70% compared to traditional RDMA operations without optimization of memory registrations. We compare the latency of FRRWP with traditional RDMA operations using a mathematic model. The results show that FRRWP reduces the total communication latency in the critical data path by 68%.

Acknowledgments

This research was supported by the U.S. National Science Foundation under Grants CNS-0617528 and CNS-0720617 and by the TTU office of research. 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 preliminary work on *MRRC* (Ou et al., 2006b) and *FRRWP* (Ou et al., 2006a) were presented at the NASA/IEEE Conference on Mass Storage Systems and Technologies (MSST'06) and the International Workshop on Networking, Architecture, and Storages (NAS'06), respectively.

References

Bell C, Bonachea D. A new DMA registration strategy for pinning-based high performance networks. In: 17th international parallel and distributed processing symposium; 2003.

Boden NJ, Cohen D, Felderman RE, Kulawik AE, Seitz CL, Seizovic JN, Su W-K. Myrinet: a gigabit-per-second local area network. IEEE-Micro 1995;15(1): 29–36.

Dan A, Towsley D. An approximate analysis of the LRU and FIFO buffer replacement schemes. In: ACM SIGMETRICS; May 1990. p. 143–52.

Denning PJ. The working set model for program behavior. Commun ACM 1968; 1(5):323–33.

Gill BS, Modha DS. WOW: wise ordering for writes—combining spatial and temporal locality in non-volatile caches. In: FAST 2005; December 2005.

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

Infiniband Trade Association. Infiniband architecture specification, release 1.0, October 24, 2000.

Jiang S, Zhang X. LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. In: Proceedings of the ACM SIGMETRICS; 2002. p. 31–42.

Jiang S, Ding X, Chen F, Tan E, Zhang X. DULO: an effective buffer cache management scheme to exploit both temporal and spatial localities. In: FAST 2005; December 2005.

Johnson T, Shasha D. 2Q: a low overhead high performance buffer management replacement algorithm. In: Proceedings of the twentieth international conference on very large databases; 1995. p. 439–50.

Katz ED, Butler M, McGrath R. A scalable HTTP server: the NCSA prototype. Comput Networks ISDN Syst 1994;27(2):155–64.

Liu J, Wu J, Kini S, Wyckoff P, Panda DK. High performance RDMA-based MPI implementation over InfiniBand. In: ICS '03; June 2003.

Megiddo N, Modha D. ARC: a self-tuning, low overhead replacement cache. In: Proceedings of the second USENIX conference on file and storage technologies; 2003.

Mellanox Technologies. Mellanox IB-verbs API (VAPI), rev. 0.95, March 2003.

O'Neil EJ, O'Neil PE, Weikum G. The LRU page replacement algorithm for database disk buffering. In: Proceedings of the ACM SIGMOD international conference on management of data; May 1993. p. 297–306.

Ou L, He X, Han J. A fast read/write process to reduce RDMA communication latency. In: Proceedings of the international workshop on networking, architecture, and storages (IWNAS); August 2006a.

Ou L, He X, Han J. MRRC: an efficient cache for fast memory registration in RDMA. In: Proceedings of the NASA/IEEE conference on mass storage systems and technologies (MSST); May 2006b.

Petrini F, Feng WC, Hoisie A, Coll S, Frachtenberg E. The quadrics network (QsNet): high-performance clustering technology. In: In hot interconnects; 2001.

Rangarajan M, Iftode L. Building a user-level direct access file system over InfiniBand. In: 3rd workshop on novel uses of system area networks; 2004.

RDMA Consortium. Architectural specifications for RDMA over TCP/IP.

Robinson JT, Devarakonda MV. Data cache management using frequency-based replacement. In: Proceedings of the ACM SIGMETRICS conference on measurement and modeling of computer systems; 1990.

Ruemmler C, Wilkes J. Unix disk access patterns. In: Proceedings of the winter 1993 USENIX conference.

Tezuka H, O'Carroll F, Hori A, Ishikawa Y. Pindown cache: a virtual memory management technique for zero-copy communication. In: International parallel processing symposium; March 1998.

Wu J, Wyckoff P, Panda DK. PVFS over InfiniBand: design and performance evaluation. In: International conference on parallel processing; October 2003a.

Wu J, Wyckoff P, Panda DK. Supporting efficient noncontiguous access in PVFS over InfiniBand. In: Cluster 2003 conference; December 2003b.

Wu J, Wyckoff P, Panda DK, Ross R. Unifier: unifying cache management and communication buffer management for PVFS over InfiniBand. In: CCGrid '04; April 2004.

Zhou Y, Bilas A, Jagannathan S, Dubnicki C, Philbin JF, Li K. Experiences with VI communication for database storage. In: ISCA; 2002.