MRRC: an effective cache for fast memory registration in RDMA *

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 card 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, but only simple LRU is used as a replacement algorithm to manage the cache space. In this paper, we propose an effective cache scheme: Memory Registration Region Cache (MRRC), to minimize the cost of memory registration and deregistration 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. We compare the performance of MRRC with traditional RDMA memory registration operations. The results show that MRRC can dramatically reduce the total cost of memory registrations and deregistrations.

1. Introduction

Remote Direct Memory Access (RDMA)[1, 4, 9, 2, 6] 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. While RDMA improves network bandwidth and decreases latency by eliminating unnecessary copies from network interface card to application buffers, there are a number of challenges to be addressed. One of the most significant issues is efficient communication buffer management to reduce memory registration and deregistration costs. Previous research [12, 13, 14, 10] 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 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 [12, 15, 13, 14, 3, 10] have been made to reduce the overhead of memory registration in RDMA. In general applications, a pin-down cache [12] is incorporated in the memory manager. Several cache designs for memory registration [13, 10] 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 in page level and only consider LRU as 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 general LRU algorithm is probably not the best choice.

In this paper, we propose a new cache management scheme, Memory Registration Region Cache (MRRC), to minimize cost of memory registration and deregistration in the critical data path. MRRC manages memory in terms of memory region containing one or more memory pages. MRRC organizes the cache stack using LRU algorithm, but divides stack into three sections and evicts memory regions according to both the size and recency. We compare the performance of MRRC with traditional RDMA memory registration operations and other typical registration cache management algorithms such as pin-down cache [12]. The results show that compared to traditional RDMA memory registration, MRRC can reduce the total cost of memory registrations by up to 70%.

2. Design of MRRC

In RDMA operations, memory is accessed and registered in terms of region, which includes several physical pages. There are four possibilities for the relationship be-

*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.

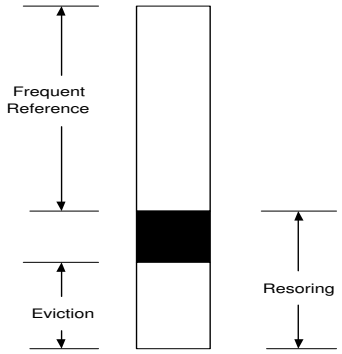


Figure 1. LRU stack of MRRC.

```
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 overlap 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 the full of the
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;
```

Figure 2. MRRC and MRE algorithms.

tween a memory region newly requested by a 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 a memory overlap between the new region and a cached 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 memory space, 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.

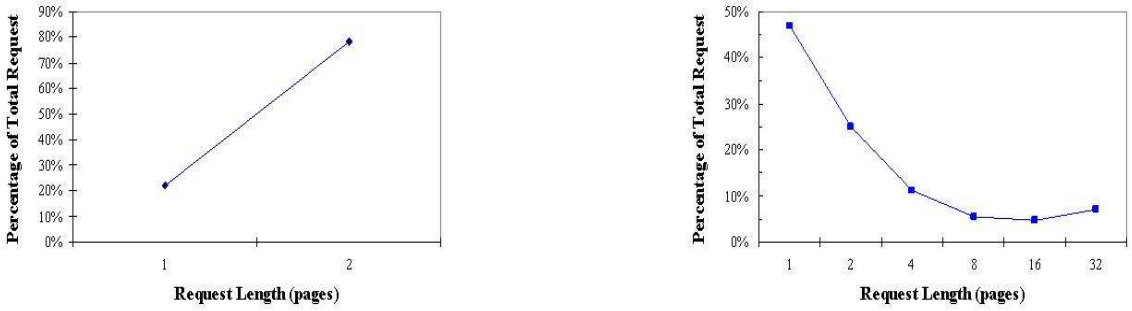
We build the MRRC by using the LRU algorithm and its data structure: the LRU stack. we partition the MRRC stack into two sections (shown in Fig. 1), similar to DULO cache [7]. The top part is the *frequent reference* section used for admitting newly accessed memory regions. The lower part is the *Resorting* section in which all regions are treated as candidate for eviction. We further divide the *Resorting* section into two segments. The lower part is *eviction* segment. All regions in this segment are ready to be evicted from the cache and thus to be deregistered.

Memory Resorting and Eviction (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 same memory size. Research in [14] showed that cost of memory registration consists of two parts. First part is the cost of per registration, and second part is cost of per page. In [14], 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. The same cost equation can be applied to deregister a region with different values of a and b. In their testbed, the costs of per page in registration and deregistration are $0.77us$ and $0.22us$, respectively. The overheads per registration and deregistration operations are $7.42us$ and $1.1us$. From to those results, it is easy to understand that registering multiple small regions is more expensive than registering large regions with same number of pages. From this model, we know that one by one deregistration is not efficient. In MRRC, a batched deregistration scheme is adopted.

Every time when the cache is full, MRE algorithm resorts all memory regions in the *Resorting* section, according to their *eviction factor*. *Eviction factor* is a function of size and recency of a memory region: $evictfact(s,r)$. In our current design, $evictfact(s,r)$ is defined as $r + 1/s$, where s is size of a memory region and r is 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. *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 *Eviction factor* is set to zero again. All regions are assorted 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. 2 outlines the MRRC and MRE algorithms.

Table 1. Characteristics of the Two Traces Used in the Study

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



(a) Cello92

(b) HTTPD

Figure 3. Distribution of request size for Cell92 and HTTD traces. A point (L, P) on the curve indicates that the size of P percent of total requests is L pages.

3. Simulation Results

We use trace-driven simulation to evaluate our MRRC design. We have developed a simulator to simulate cache hit ratio of memory registrations. We define hit ratio here as the percentage of cached memory registration on the total number of requests of the trace. In order to compare our work to previous efforts, our simulator implements multiple algorithms, including MRRC, and pin-down cache [12]. We assume a memory page size of 4KB.

To evaluate caching algorithms and policies, we use two buffer cache access traces as summarized in Table 1. The HP Cello92 trace was collected at Hewlett-Packard Laboratories in 1992 [11]. It captured all L2 disk I/O requests in Cello, a timesharing system used by a group of researchers to do simulations, compilation, editing, and e-mail. 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 [8] serving a 524 MB data set.

Based on the outputs of the simulator running the above two traces, we calculate the total time for registering all memory regions using the following formula.

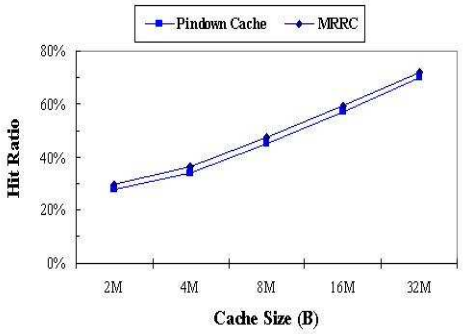
$$T_{total} = sum_{i=0}^{n}(ismiss(i) * T_r(i)) + sum_{j=0}^{m}(T_{ur}(j))$$

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 2. $ismiss(i)$ is a boolean function of the memory region. It outputs 1 if the corresponding region cannot be found in the cache space, otherwise, outputs 0.

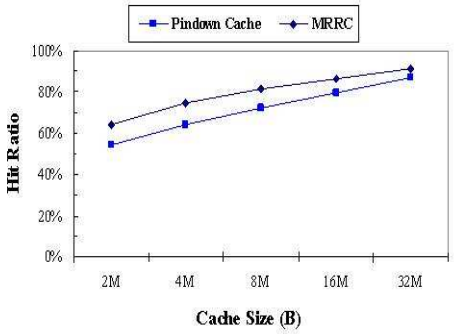
Fig. 3 shows the distribution of request size grouped by powers of two[1]. A request size of the two traces is in terms of memory pages. Most requests in the Cello92 trace are small: the maximal request size is only 2 pages. The size of requests in the HTTPD trace varies from 1 page to 32 pages. The difference comes from the environment where two traces were collected. The Cello92 trace captured all L2 disk I/O requests in Cello, in which small reading or writing requests were issued one by one. The HTTPD workload was generated by a seven-node IBM SP2 parallel web server, where reading request was applied upon the whole file every time. The distribution of the request size of the traces influences results of our simulation because each request is accompanied by a memory registration and the request size determines the size of a memory region.

We first compare memory region hit ratios between MRRC and pin-down cache with various cache sizes under the two traces. Fig. 4 shows that under the HTTPD trace, MRRC has 10% hit ratio improvement compared to pin-down cache, but the difference is not obvious under the Cello92 trace. We explained above that the distribtion of request size in the two traces is totally different. This difference is the reason for the result of hit ratios. In the HTTPD trace, there are many large size requests, so MRRC may optimize its performance by considering both size and re-

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

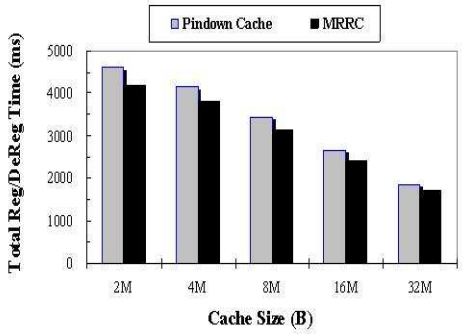


(a) Cello92

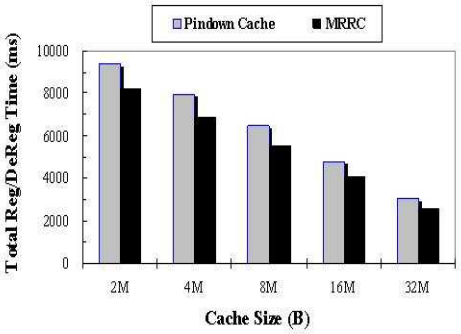


(b) HTTPD

Figure 4. Memory registration hit ratios with various cache sizes (Cello92 and HTTPD traces).

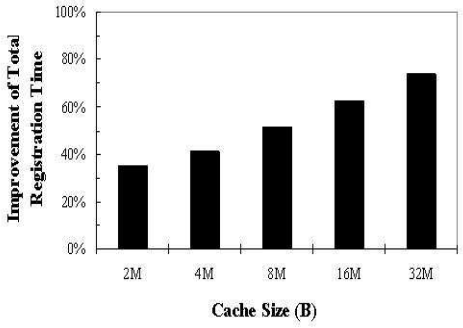


(a) Cello92

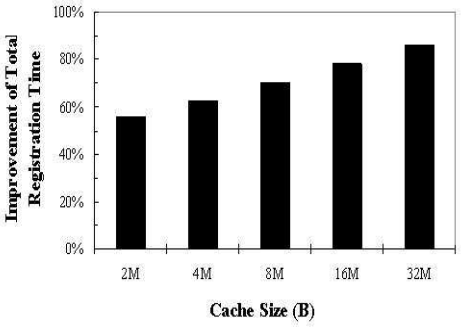


(b) HTTPD

Figure 5. Registration/Deregistration cost with various cache sizes (Cello92 and HTTPD traces).



(a) Cello92



(b) HTTPD

Figure 6. Improvement of MRRC over the basic RDMA operations without the registration cache.

cency. In the Cello92 trace, most requests have same size, so MRRC cannot distinguish them in terms of memory region size and only sorts them in the cache space according to their recency.

Fig. 5 compares total registration and deregistration time of MRRC and pin-down cache under the two traces. The total cost in term of registration/deregistration time keeps the trend of hit ratios. We find that although the difference of hit ratios under the Cello92 trace is not obvious, MRRC still reduces the total cost of registration and deregistration, because of its batched registration. The maximal improvement of cost is about 10%. Fig. 6 shows the improvement of MRRC over the basic RDMA operations without memory registration caches. It's clear that MRRC reduces the RDMA memory management cost dramatically.

4. Related Work

Several studies have been made to improve the performance of memory registration and deregistration of RDMA. Tezuka *et al.* [12] propose a *pin-down cache* for Myrinet. Pin-down cache delays deregistration of registered buffers and caches their registration information for future accesses of the same memory region. Zhou *et al.* [15] eliminate pinning and unpinning from registration and deregistration path by combining memory pinning and allocation together. They also demonstrated *batched deregistration* is an efficient way to reduce average cost of deregistration memory. In [13], 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 [10], which combines a cache of registration mapping with a lazy approach to memory deregistration.

Caching is a common technique for improving performances of I/O systems. Song *et al.* [7] propose a new management scheme, DULO, to balance temporal and spatial locality of workload. Gill *et al.* [5] use a new ordering algorithm, WOW, to resort writing sequences of Non-Volatile cache by combining both spatial and temporal locality.

5. Conclusions

In this paper, we present a new cache management scheme: *MRRC*, to improve the performance of memory registration and deregistration of RDMA. *MRRC* manages memory in terms of memory regions and uses MRE as a cache replacement algorithm, which considers both the size and recency of memory regions. To further reduce the memory registration/deregistration cost, MRRC adopts batched deregistration to avoid deregistering regions at every cache replacement.

We have evaluated our *MRRC* and other typical registration cache designs using simulations under various workloads. The results show that *MRRC* efficiently increase the cache hit ratios by 10% and reduces the total cost of memory registration and deregistration by up to 70% compared to traditional RDMA operations without optimization.

References

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

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

[3] 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.

[4] 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):29–36, February 1995.

[5] B. S. Gill and D. S. Modha. WOW: Wise ordering for writes - combining spatial and temporal locality in Non-Volatile caches. In *FAST 2005*, December 2005.

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

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

[8] E. D. Katz, M. Butler, and R. McGrath. A scalable HTTP server: The NCSA prototype. *Computer networks and ISDN systems*, 27(2):155–164, Nov 1994.

[9] 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.

[10] 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.

[11] C. Ruemmler and J. Wilkes. Unix disk access patterns. In *Proc. Winter 1993 USENIX Conf.*

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

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

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

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