

A Study on User-level Remote Memory Extension System

Shinyoung Ahn, Gyuil Cha, Youngho Kim, Eunji Lim, Seungjo Bae

Cloud Computing Department, ETRI, 218 Gajeongno, Yuseong-gu, Daejeon, 34129, South Korea

syahn@etri.re.kr, gicha@etri.re.kr, kyh05@etri.re.kr, ejlim@etri.re.kr, sbae@etri.re.kr

Abstract— The speed of memory capacity expansion of the computer system has not kept up with the speed of the increase of the memory requirement of large memory applications. Also, big memory system has been too expensive for many researchers and students. Therefore, approaches to utilize remote memory has been considered as a cost effective way to run large memory applications in the cluster environment where computing nodes are connected via high speed network. For the general users of HPC cluster system who want to run large memory application without administrator's support, we suggest a user-level remote memory extension method. We designed and implemented a remote memory extension system which mapped remote memory pages to the virtual address space of the large memory application process. The system includes three components such as remote memory consumer, integrated memory manager, and memory provider. We also developed a test program and did experiment to show how good performance the user-level remote memory extension system can achieves. Through the experimental test, we will show the measured latency and bandwidth of user-level remote memory extension system for sequential access pattern and random access pattern.

Keyword— Remote memory, Large memory application, Memory extension, Remote memory library, Page fault handling, Remote memory consumer Integrated memory manager, Memory provider

I. INTRODUCTION

Large memory applications such as IMDB(in-memory database), IMDG(in-memory data grid), denovo assembly application in the human genome sequencing area, business application acceleration, big data analytics, and large

scale scientific calculation are increasing gradually. Such large memory applications may not run or take too much time to finish their jobs in machines without enough physical memory. By the way, to run large memory applications, buying big memory system is a too expensive way.

As time passes, networking technologies have been improved. Modern networking technologies in LAN/SAN area such as Infiniband, Quadrics, and Myrinet have very low latency(a few microseconds) and high bandwidth(e.g. Infiniband FDR(Fourteen Data Rate) is 56Gbps). These technologies also support Remote Direct Memory Access(RDMA) operation mode in which CPU don't need to coordinate transferring between local memory and remote memory. The RDMA feature reduces the number of memory copy between user-level and kernel level. The feature improves the access latency and the bandwidth of the remote memory dramatically. The latency of remote memory access is a few orders of magnitudes faster than that of local disk access. Therefore, there have been many trials to use remote memory like local memory, local block, and local file system.

The approach utilizing remote memory as local memory is adding an additional level between main memory and local disk in the memory hierarchy. This makes sense because the latency of memory is faster 10,000~100,000 times rather than HDD's. If we can achieve a few micro-seconds as the access latency of remote memory which is faster 100~1,000 times than HDD, it would be very reasonable solution. In HPC cluster computing environment, the benefit of utilizing remote memory is that we can execute large memory application without additional HW cost by harvesting idle memory on remote nodes. In this paper, we don't talk about memory harvesting issues. We assume that we can use abundant idle memory.

The majority of previous approaches of extending virtual address space to remote memory have preferred implicit method which conceal the fact that users actually use remote memory when they run their large memory applications. This method requires no special re-compiling or linking with special library but requires root privilege or administrator's support because it requires kernel modification or kernel module deployment.

However, most of users of HPC clusters system can't update kernel or kernel module because they don't have administrator's privilege. Therefore, if administrator does not support, they can't use remote memory like local memory. They just can execute user-level application. Owing to this reason, the most popular programming model of HPC clusters

Manuscript received September 17, 2015. This work was supported by the ICT R&D program of MSIP/IITP. [B0101-15-0104, The Development of Supercomputing System for the Genome Analysis].

Shinyoung Ahn is with the Electronics and Telecommunications Research Institute, 218 Gajeongno, Yuseong-gu, Daejeon, 34129 South Korea (corresponding author to provide phone: +82-42-860-1256; fax: +82-42-860-6699; e-mail: syahn@etri.re.kr).

Gyuil Cha is with the Electronics and Telecommunications Research Institute, 218 Gajeongno, Yuseong-gu, Daejeon, 34129 South Korea (phone: +82-42-860-1221; fax: +82-42-860-6699; e-mail: gicha@etri.re.kr).

Youngho Kim is with the Electronics and Telecommunications Research Institute, 218 Gajeongno, Yuseong-gu, Daejeon, 34129 South Korea (phone: +82-42-860-1221; fax: +82-42-860-6699; e-mail: kyh05@etri.re.kr).

Eunji Lim is with the Electronics and Telecommunications Research Institute, 218 Gajeongno, Yuseong-gu, Daejeon, 34129 South Korea (phone: +82-42-860-1387; fax: +82-42-860-6699; e-mail: ejlim@etri.re.kr).

Seungjo Bae is with the Electronics and Telecommunications Research Institute, 218 Gajeongno, Yuseong-gu, Daejeon, 34129 Korea (phone: +82-42-860-5667; fax: +82-42-860-6699; e-mail: sbae@etri.re.kr).

is MPI(message passing interface), which use message passing for communication. On the other hands, some users are using PGAS(Partitioned Global Address Space) model by which parallel programs share their memory with others. Such programming models are based on user-level library and useful for compute intensive applications. Hence, we need a library to support large memory applications in HPC clusters.

In this paper, we designed and implemented a remote memory extension system. The system includes remote memory extension library which supports large memory application to use remote node's memory like local memory. The remainder of this paper is organized as follows. In Chapter II, we summarize and discuss the related work. In Chapter III, we show the details of remote memory extension system. Next, in Chapter IV, we show an experimental study and discuss the result. Finally, in Chapter V, we talk about our current and future work.

II. RELATED WORKS

Comer suggested remote paging mechanism based on the remote memory model which consists of several client machines and one or more dedicated remote memory server[1]. In the remote paging mechanism, clients use remote memory rather than local disk as a backing storage. Clients move memory pages to the remote memory server when the client machines exhaust their local memory[1].

Since Comer's suggestion, there have been many research works on the utilization of remote memory. Such research works have been approached from different perspectives because of the variety of research goals and network characteristics[2]. Various approaches tried to use remote memory just like a local memory[1][3-8], or as a cache for local or distributed storage[2][9-15], or as a storage[2][16].

Anderson et al. classified the approaches to utilize remote memory like local memory based on the implementation layer/methods: 1) explicit program management, 2) user-level, 3) device-driver 4) kernel-modification, and 5) Network interface[4]. Explicit program management requires the programmer to coordinate all data movement to/from the remote memory[5]. User-level implementation method requires the programmer modify their code using a new malloc for remote memory allocation[4]. Device-driver method replaces the swap device with a new device which sends the pages to remote memory[8][17]. Kernel modification method modifies the virtual memory(VM) subsystem. It shows highest performance but very lower portability[1][3]. Network Interface method is HW-level method which replaces memory controller with some sort of chip. This is most un-portable and hardest to implement.

User-level approach more portable than device driver approach and doesn't require root privilege or administrator's support. However, device driver approach doesn't require user-code modification and kernel source modification[4][8]. Even though device-driver method has been the majority of research, the method requires root privilege. If the administrator of cluster system does not support to install device driver for remote memory extension, then most of non-privileged user can't use remote memory.

To decrease the communication cost of TCP/IP networking,

RDMA enabled network technologies such as Infiniband, Quadrics, and Mirinet became to be used[8][18].

Another consideration point is about if remote memory server is dedicated. Some approaches are based on dedicated memory server. The other are based on idle memory harvesting. The latter one focused on minimizing the impact to owner of machine whose memory is harvested and improving memory utilization[5].

Unreliable memory semantic do not require the remote memory server to maintain multiple copy of data or permanent data[1], while reliable memory semantic for file system, block device and storage require some mechanisms which restore missed pages and also require security and fault tolerance[4][17].

III. REMOTE MEMORY EXTENSION SYSTEM

Fig. 1 shows an architecture of remote memory extension system and an example of remote memory allocation under the system.

The remote memory extension system requires three types of computing node: Remote memory consumer node, memory manager node, and memory provider node. Each node runs one of three components of the remote memory extension system: remote memory consumer, integrated memory manager, and memory provider.

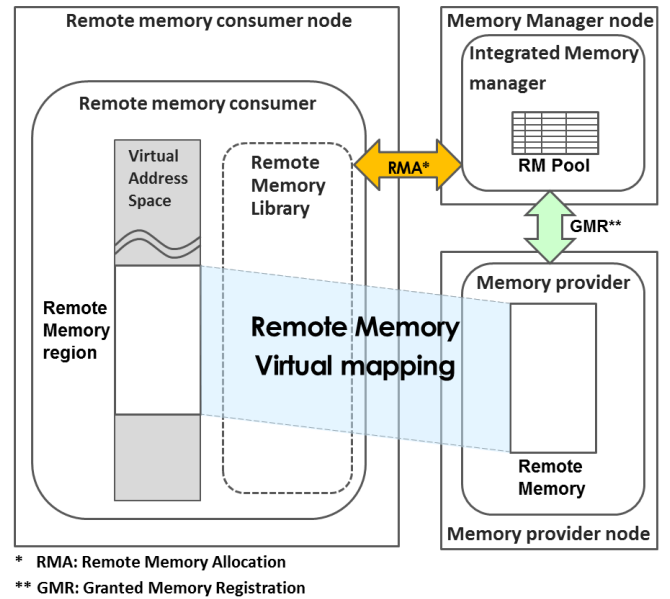


Fig. 1. Remote memory extension system and remote memory allocation under

The memory provider in the memory provider node allocates some part of its own idle memory as a granted memory. After allocation, it make the granted memory readable or writable via RDMA operation. Then, it registers the granted memory to integrated memory manager and eventually provides the granted memory block to remote memory consumer by the administration of the integrated memory manager[19].

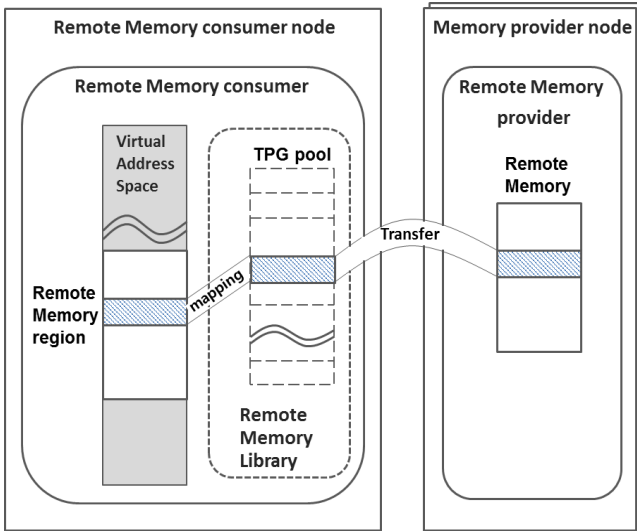
The integrated memory manager configures and builds a remote memory(RM) pool based on the granted memory which the memory provider nodes registered. It also assign the granted memory block to remote memory consumer[20]. The integrated memory manager and memory provider can be

grouped as remote memory extension server.

The remote memory consumer link a library which enables it to use a remote memory service from the remote memory extension server. The remote memory consumer requests allocation of a remote memory from the remote memory pool on the integrated memory manager through the remote memory library, which maps the allocated remote memory blocks on its virtual address space(refer Fig. 1. Remote Memory Virtual mapping). The remote memory consumer can use the remote memory like its own local memory because the remote memory library hides the details of accessing the granted remote memory.

A. The mechanism of remote memory access

To use remote memory like local memory, first, remote memory should be mapped on virtual address space of remote memory consumer process as explained in the Fig. 1. Second, user-level page fault handling for remote memory is necessary when consumer process try to access the mapped remote memory region. User-level page fault handling for remote memory is different from normal page fault handling of kernel in the way that it should obtain real data of remote memory and save them to the physical page of consumer node and map the physical page to the remote memory region by updating page table of the consumer process.



* TPG: Temporal page (physical memory)

Fig. 2. A mechanism of remote memory extension

Fig. 2. shows a mechanism of remote memory extension. As explained above, remote memory consumer process can use remote memory with the help of remote memory library and remote memory provider. Remote memory providers serve their own memory to remote memory consumer. Remote memory library handles the user-level page fault for remote memory region. It transfers real data from remote memory page to the temporary page(TPG) of consumer node and maps the TPG to virtual address space of consumer process. This remote memory library mimics page fault handling mechanism in kernel. The role of TPG is similar to that of page cache in kernel. The difference between TPG and page cache is that page cache use free memory as much as it can use,

but the size of TPG pool is restricted because we need to minimize overhead to serve remote memory extension to consumer process. Therefore, TPGs may be mapped to different addresses of remote memory region frequently. The TPGs should be recycled efficiently.

B. User-level APIs

TABLE I. shows user-level APIs for remote memory extension which the remote memory extension system provides to remote memory consumer applications. First, the remote memory consumer should initialize remote memory library before allocating remote memory and terminate the library at the end of the application via `rm_init()` and `rm_terminate()`. The `rm_init()` make a connection to integrated memory manager and register the remote memory consumer to the integrated memory manager. On the contrary, the `rm_terminate()` deregister the remote memory consumers and disconnect the connection to integrated memory manager.

TABLE I
USER-LEVEL APIs FOR REMOTE MEMORY EXTENSION

APIs	Feature
<code>int rm_init();</code>	Initialize remote memory consumer
<code>int rm_terminate();</code>	Terminate remote memory consumer
<code>void *rmalloc(size_t size);</code>	Allocate remote memory on the process's virtual address space
<code>void rtfree(void *ptr);</code>	Deallocate the allocated remote memory
<code>int rmsync(void *ptr, size_t size);</code>	Synchronize temporal pages to remote memory pages
<code>int rmflush(void *ptr, size_t size);</code>	Invalidate and recycle temporal pages by force
<code>int rmadvice(void *addr, int advice);</code>	Indicate if this memory region is allowed to prefetch

After initialization, the programmer can allocate remote memory via `rmalloc()` and may release the memory via `rtfree()` after using remote memory. For additional management of remote memory, we provide three APIs: `rmsync()`, `rmflush()`, and `rmadvise()`. The `rmsync()` synchronizes temporal pages(TPG) and remote memory page. The `rmflush()` invalidate and recycle TPGs by force. The `rmadvise()` indicates the library if prefetching remote pages is allowed.

C. User-level page fault handling

In general, page fault handling means that kernel allocate physical page and map the page to virtual address of the process by updating page table of the process where the page fault occurs. It is not allowed for user-level code to allocate physical memory page and update the page table of the process. However, to map TPG to the virtual address space of the remote memory consumer process, we need to handle page fault at not kernel-level but user-level.

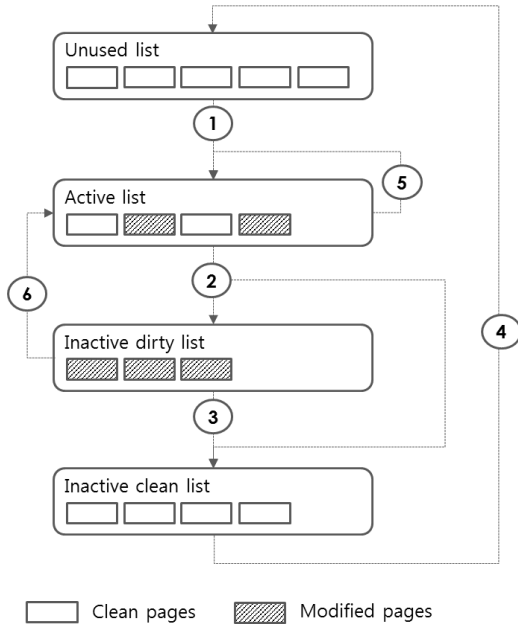
We suggested a tricky way to use signal handling mechanism of segmentation fault signal(SIGSEGV) by setting the remote memory region not readable and not writable. By doing so, we can handle the page fault at user-level by changing the action taken by the user process on receipt of SIGSEGV signal.

The page fault handler selects a TPG from TPG pool and

read real data from remote page to the TPG, and then map the TPG to the starting address where page fault occurs by calling `remap_file_pages` system call. By the way, there is no way to allocate a physical page to two different virtual addresses at user-level. We used another trick to create a TPG backing file on `tmpfs` and `mmap` the backing file to virtual address space twice: one is to remote memory region when `rmalloc()` API is called and the other is to the TPG Pool region which remote memory library is initialized[21][22].

D. The lifecycle of temporal page(TPG)

Fig. 3. Show the lifecycle of temporal pages which remote memory library operates and manages in the TPG pool. The TPG pool is composed of four double linked lists: Unused list, Active list, Inactive dirty list, and Inactive clean list. The Unused list holds all the unallocated TPGs. The Active list holds all the TPGs allocated to consumer thread and used by consumer thread. The Inactive dirty list holds all the modified TPGs out of inactivated TPGs. The Inactive clean list holds all the clean or unmodified TPGs out of inactivated TPGs.



The TPG lifecycle management operations: ① TPG Allocation, ② TPG Inactivation, ③ TPG Laundering, ④ TPG Reclaiming, ⑤ TPG Reordering, and ⑥ TPG Reactivation

Fig. 3. TPG lifecycle management diagram

The TPG lifecycle is managed through 6 operations as shown in Fig. 3. As we explained in the previous section, if remote memory consumer accesses the remote memory region, then page fault handler may be executed or not. If the accessed page of remote memory region is already mapped, then the page fault handler is not called. If not, the page fault handler is called to select unused TPG from the Unused list and store the data of remote page to the TPG and remap the TPG and finally move the TPG to the end of the Active list. These are done in the TPG Allocation operation.

If the Unused list does not have any TPGs, then the TPG Allocation operation will be delayed until the TPGs are recycled to the Unused list. Because the TPG pool has the

limited number of TPGs, TPGs should be recycled proactively. The first step of recycling is TPG Inactivation operation.

The TPG Inactivation operation inactivates some overflowed TPGs of the Active list. If the number of TPGs in the Active list is greater than the predefined maximum size of the Active list, the overflowed number of TPGs should be inactivated. The TPG Inactivation operation move the overflowed TPGs to the Inactive dirty or clean list. The overflowed TPGs are selected from the head of the Active list because the head of Active list indicates LRU page.

The TPG Laundering operation saves the data of the modified TPGs to the remote page. This operation spends considerable time because the operation should write the data of the modified pages to the remote memory page. The laundered TPGs are moved to Inactive clean list.

The TPG Reclaiming operation periodically moves all the TPGs in the Inactive clean list to the tail of the Unused list.

When remote memory consumer revisit the already mapped pages, the TPG Reactivation operation or TPG Reordering operation is executed. The TPG Reactivation operation move a revisited TPG of Inactive dirty to the Active list. The remote memory consumer may access the TPGs in the Inactive dirty list. In this case the revisited TPGs are reactivated. The TPG Reordering operation moves a revisited TPG to the tail of the Active list because the Active list should be ordered by least recently used(LRU) time.

E. Prefetching remote memory pages

The access latency of remote memory page is much greater than the one of local memory. Prefetching remote memory pages is essential mechanism to hide the latency and improve the throughput in case of sequential access pattern. We proposed prefetching algorithm which determine if the access pattern is sequential and start prefetching by managing current window and readahead window[22].

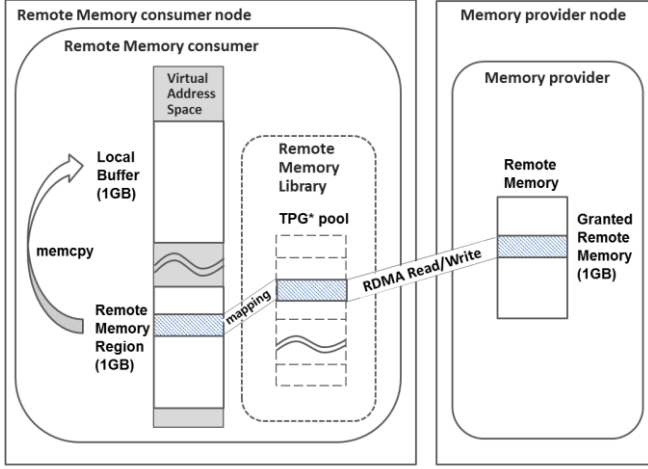
IV. AN EXPERIMENTAL STUDY

We developed the remote memory extension system which composed of integrated memory manager, memory provider, and remote memory library. The remote memory library handles user-level page fault and manage temporal page pools, provide user-level API and access remote memory pages via RDMA read/write operation. For experiment, we also wrote a test program which copy data from remote memory to local memory buffer.

A. A test program for experiment

Fig. 4. shows the test program which copies 1GB data from a remote memory region to a local buffer. It copies from the page of the remote memory of the memory provider node to the page of local buffer in the remote memory consumer node. Actually, memory copy occurs twice: one is between the granted remote memory and TPG, the other is between the TPG and local buffer. The first memory copy between the granted remote memory and TPG is done by using RDMA Read operation of Infiniband network connection. This program is for only testing how much well the remote memory extension system will work.

The size of TPG pool of the test program is 1024 pages(4MB). The size of maximum current window is 32 pages. The size of maximum readahead window is also 32 pages.



* TPG: Temporal page (physical memory)

Fig. 4. The test program which copy 1GB data from the granted remote memory to local memory buffer.

B. Experimental environment

The experiment environment are as follows. The memory manager node's HW specification is Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz with 12 core, 2 socket, 128GB RAM. The remote memory consumer node and memory provider node's HW specification is Intel(R) Xeon(R) CPU E5-2609 v2 @ 2.50GHz with 4 core, 2 socket, 64GB RAM. These nodes are connected via Infiniband FDR(Max bandwidth 56Gbps) switch. OS is Linux pure kernel 3.10.0-123.el7.x86_64.

For comparison, we measured local memory latency and bandwidth in the remote memory consumer node using Intel Memory Latency Checker(v2.3). The average idle memory access latency is about 110 ns. For measurement of memcpy throughput, we wrote a small measuring program which allocates two 1GB buffer and executes memcpy from source buffer to destination buffer. The average throughput of local memory copy is 2.7 GB/s. Generally, the measured latency of one socket machine is much smaller than two socket machine. The bandwidth is also much greater than two socket machine.

We also measured RDMA read latency(when message size is 4096 bytes and RDMA CMs is used for connection establishment management) using built in 'ib_read_lat' program. The minimum latency was 9 us, maximum latency was 25us, and average latency was 9.1 us.

C. Experimental result

We measured the access latency and the memory copy throughput of from the remote memory to local buffer. The result may include a little measuring error because we add measuring codes to the test program.

The experiment was done for sequential memory access pattern and random access pattern. The Table II shows the measured latency and the bandwidth throughput per memory access pattern.

Average latency of sequential access pattern is 11.5 us.

This is just 100 times of local memory latency(110ns). Memory copy throughput of sequential access pattern is 341MB/s. It is about 12% of the local memory copy throughput. However, the average latency of random access pattern is 41 us, which is 372 times longer than the local memory latency(110ns). The average bandwidth of random access pattern is 81MB/s and is about 3% of the local memory copy bandwidth throughput.

TABLE II
READ LATENCY AND THROUGHPUT OF THE REMOTE MEMORY

Metric	Sub metric	1	2	3	4	5	Avg
Sequential Latency (us)	Total	11.5	11.4	11.5	11.5	11.4	11.5
	Page fault handling	2.6	2.6	2.6	2.7	2.7	2.7
	Transfer	8.9	8.8	8.9	8.8	8.7	8.8
Sequential Bandwidth (MB/s)	Total	339	342	340	341	344	341
	Page fault handling	41	41.2	41.2	40.8	41.1	41.1
Random Latency (us)	Page fault handling	3.6	3.5	3.5	3.6	3.5	3.6
	Transfer	37.4	37.7	37.7	37.2	37.6	37.5
	Total	81	81	81	80	81	81
Random Bandwidth (MB/s)		81	81	81	80	81	81

The reason the latency and the throughput for sequential access pattern is better than that of random access pattern is because the test program does prefetching memory pages from the remote memory for the sequential access pattern.

$$Latency = T(page\ fault\ handling) + T(transfer) \quad (1)$$

The latency is composed of the page fault handling time and the page transfer time from remote memory. From the above experiment, in the case of random access pattern, the page fault handling time is 3.6us(random) and transfer delay is 37.5 us. Because the latency hiding effect occurs in the sequential access pattern, so we talk about the latency of random access pattern. From the latency value, transfer delay is much about 10 times than page fault handling time.

Even though the page fault handling time is much decreased(20.8 us $\rightarrow$ 3.6 us) compared to the our emulation result of our previous paper[22], it is not still enough. What's even worse, the transfer delay is much bigger than what we measured using ib_read_lat(9.1us).

V. CONCLUSIONS

In this paper, we designed and implemented a remote memory extension system. We also wrote a test program for demonstrating the performance of remote memory extension system. The system can't be used for real application yet. It needs more debugging for a real large memory application and more performance tuning(latency and bandwidth).

The remote memory extension system provides user-level API for remote memory consumer applications, user-level page fault handling, dynamic TPG management functions for recycling the very limited number of TPGs, reading/writing data from/to the remote memory, and prefetching functions.

As we discussed in the previous section, we can conclude our system may be a cost-effective solution for large memory

application with sequential memory access pattern. However, the network transfer delay is not yet acceptable. We need to decrease the network transfer delay to about 10 us because the measured average latency of RDMA Read on Infiniband network is 9.1 us.

Fortunately, we have two attack points to improve our memory extension system's performance if we consider kernel level implementation. One is to develop page fault handling in kernel level. It might decrease the page fault handling time to about 1~2 us. The other is to customize the Infiniband RDMA operations in kernel layer. It would improve the expected latency to under 5us for sequential access pattern or 10 us for random access pattern. We also need to upgrade temporal page management algorithms for performance improvement.

Our system may not be a good solution for latency sensitive application which require large memory. Our suggestion might be a good candidate for someone who need large data sharing among cluster machines because remote memory can be very fast shared memory.

ACKNOWLEDGMENT

This work was partly supported by the ICT R&D program of MSIP/IITP. [B0101-15-0104, The Development of Supercomputing System for the Genome Analysis]

REFERENCES

- [1] Douglas Comer, "A New Design for Distributed Systems: The Remote Memory Model", in *Proc. the USENIX Summer Conference*, Ahaheim, California. Pp127-135, 1990
- [2] V. Roussev, G. G. Richard III, and D. Tingstrom, "dRamDisk: Efficient RAM sharing on a commodity cluster," in *Proc. 25th IEEE International Performance, Computing, and Communications Conference (IPCCC 2006)*, Phoenix, Arizona, Apr. 10-12, 2006, pp. 193-198.
- [3] L. Iftode, K. Petersen, and K. Li, "Memory Servers for Multicomputers," in *Proc. IEEE COMPCON'93 Conference*, February 1993.
- [4] E. Anderson and J. Neefe. An Exploration of Network RAM. *Technical Report CSD-98-1000*, UC Berkley, 1998.
- [5] S. Koussih, A. Acharya, and S. Setia. Dodo: A User-Level System for Exploiting Idle Memory in Workstation Clusters. In *Proc. 8th IEEE International Symposium on High Performance Distributed Computing*, 1999.
- [6] McDonald, I.: Remote paging in a single address space operating system supporting quality of service. *Tech. Report, Dept. of Comp. Science*, Univ. of Glasgow (1999)
- [7] Y. Jeegou. Implementation of Page Management in Mome, a User-Level DSM. In *Proc. 3rd IEEE International Symposium on Cluster Computing and the Grid (CCGRID '03)*, Tokyo, Japan, May 2003.
- [8] S. Liang, R. Noronha, and D. K. Panda, "Swapping to remote memory over infiniband: an approach using a high performance network block device," in *Proc. IEEE Cluster Computing*, 2005.
- [9] M. Dahlin, R. Wang, T. E. Anderson, and D. A. Patterson, "Cooperative caching: Using remote client memory to improve file system performance," in *Proc. Operating Systems Design and Implementation*, 1994.
- [10] Ming Zhao, Dynamic policy disk caching for storage networking. *IBM Research Report*. Publication Date: Nov 2006.
- [11] J. Liu, W. Huang, B. Abali, and D. K. Panda, "High performance VMM-bypass I/O in virtual machines," in *Proc. the annual conference on USENIX '06 Annual Technical Conference (USENIX ATC '06)*, 2006.
- [12] Haogang Chen, Xiaolin Wang, Zhenlin Wang†, Xiang Wen, Xinxin Jin, Yingwei Luo, Xiaoming Li, REMOCA: Hypervisor Remote Disk Cache, in *Proc. IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp. 162-169, 2009
- [13] J. Oleszkiewicz, L. Ziao, and Y. Liu, "Parallel network RAM: Effectively utilizing global cluster memory for large data-intensive parallel programs," in *Proc. IEEE 2004 International Conference on Parallel Processing (ICPP'04)*, 2004.
- [14] L. Wang, J. Zhan, and W. Shi, "In Cloud, Can Scientific Communities Benefit from the Economies of Scale?" *IEEE Transactions on Parallel and Distributed Systems*, vol. 99, no. PrePrints, 2011.
- [15] Hyuck Han, Young Choon Lee, Member, IEEE, Woong Shin, Hyungsoo Jung, Heon Y. Yeom, Member, IEEE, and Albert Y. Zomaya Fellow, IEEE, "Cashing in on the Cache in the Cloud" in *IEEE Trans. On Parallel and Distributed Systems*, Vol. 23 no. 8, pp. 1387-1399, Aug. 2012
- [16] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, "The case for RAMclouds: scalable high-performance storage entirely in DRAM," *SIGOPS Oper. Syst. Rev.*, vol. 43, pp. 92-105, January 2010.
- [17] T. Newhall, D. Amato, and A. Pshenichkin, "Reliable adaptable network ram," in *Proc. of IEEE Cluster'08*, 2008.
- [18] J. Liu, J. Wu, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. *International Journal of Parallel Programming*, 32(3):167-198, 2004.
- [19] Eun-Ji Lim, Young-Ho Kim, Shin-Young Ahn, Gyu-Il Cha. Distributed Memory Integration System for High Performance Data Processing, In *Proc. of the International Conference on Future Web*, 2014
- [20] Young-Ho Kim, Shin-Young Ahn, Eun-Ji Lim and Gyu-Il Cha. Design of Granted Memory Management for Distributed Memory Node with multi-HCAs and Large Capacity Memory. In *Proc. of the 6th International Conference on Internet*. 2014
- [21] Shinyoung Ahn, Gyuil Cha, Youngho Kim, Eunji Lim. Exploring feasibility of user-level memory extension to remote node, In *Proc. of the 6th International Conference on Internet*. 2014
- [22] Shinyoung Ahn, Gyuil Cha, Youngho Kim, Eunji Lim. Design and Implementation of User-level Remote Memory Extension Library, in *Proc. 18th International conference on Advanced Communication Technology*, pp. 716-721, 2015



Shinyoung Ahn was born in South Korea in 1974. He received the B.E., M.E. degree in information engineering from SungKyunKwan University, Seoul, Korea, in 1997, 1999, respectively. He also received the M.E. degree in software engineering from Carnegie Mellon University, Pittsburgh, USA, in 2005.

He joined Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea, in 1999. Since 1999, he has been with the cloud computing department, where he is currently a senior researcher. His main areas of research interest are high performance computing, cloud computing, workflow scheduling, and software architecture.

Mr. Ahn is a member of Korea Information Processing Society.



Gyuil Cha was born in South Korea in 1970. He received the B.S., M.S. degree in Computer Science from Korea University, Seoul, Korea, in 1998, 2000, respectively.

He joined Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea, in 2000. Since 2011, he has been with High-Performance Computing Research Section, where he is currently a senior research member of engineering staff. His main areas of research interest are High Performance Computing (HPC), System Architecture, and Kernel software.



Youngho Kim was born in South Korea in 1973. He received the B.E., M.E. degree in Information and Communication Engineering from Chungbuk National University, Korea, in 1999 and 2001 respectively.

He joined ETRI (Electronics and Telecommunications Research Institute) in 2001. Since 2001, he has been working as a senior researcher at the Cloud Computing Department. His current research interests include High Performance Computing, Cloud Computing, and Parallel and Distributed Systems. Mr. Kim is a member of Korea Information Processing Society.



Eunji Lim received the B.E., M.E. degree in Computer Science from Pusan National University, Busan, Korea, in 1999, 2001, respectively. Since 2001, she has been with Cloud Computing Department in Electronics and Telecommunications Research Institute(ETRI), Korea, where she is currently a senior researcher. Her main areas of research interest are Distributed System and High Performance Computing



Seung-Jo Bae received his MS degree in Computer Science and Ph.D. degree in Computer & Information Science from Syracuse University in 1992 and 1997, respectively. He is a principal researcher at Electronics and Telecommunications Research Institute (ETRI) in Korea. His research interests are in the area of High Performance Computing & Parallel Computing.