Design and Implementation of User-level Remote Memory Extension Library

Shinyoung Ahn, Gyuil Cha, Youngho Kim, Eunji Lim

Electronics and Telecommunications Research Institute

{syahn, gicha, kyh05, ejlim}@etri.re.kr

Abstract— **The increase of memory capacity has not kept up with the continuous increase of large memory applications. Therefore, approaches to utilize remote memory like a local memory has been considered as a cost effective way to run large memory application in the cluster environment where computing nodes are connected via high speed network. For the users of HPC cluster 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 library model which extends the virtual address space of the large memory application process to remote memory. It includes user-level API and page fault handling mechanism, temporal page pool management and remote page prefetching algorithm. We also developed a performance test program to show if the user-level remote memory extension library works well. From the experimental test, we found that user-level remote memory extension library works well for applications with sequential access pattern.**

Keywords— **Remote memory, Large memory application, Memory Extension, remote memory library**

I. INTRODUCTION

Large memory applications such as In-memory database, human genome sequencing, business application acceleration, big data analytics, and large scale scientific calculation are increasing gradually. Such large memory applications may not run in machines without enough physical memory. Even if we can run these applications, we will experience thrashing of the system. However, to run large memory applications, constructing big memory system is a too expensive way.

As time passes, networking technologies have been improved. Modern networking technologies such as Infiniband, Quadrics, and Myrinet have very low latency(a few microseconds) and high bandwidth(max 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. 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.

Utilizing remote memory is adding an additional level between main memory and local disk in the memory hierarchy. In HPC cluster computing environment, the benefit of utilizing remote memory like local memory is that we can execute large memory application without additional HW cost by harvesting idle memory on remote nodes.

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 development.

However, most of users of HPC clusters(or supercomputers) 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. For this reason, the most popular programming model of HPC clusters 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 design and implement a remote memory 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 discuss the related work. In Chapter III, we present our ideas on the remote memory library for user-level remote memory extension. Next, Chapter IV shows an experimental study and result. Finally, Chapter V discusses our 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.

TABLE 1. COMPARISON BETWEEN USER-LEVEL APPROACH AND DEVICE DRIVER APPROACH

	User-level approach	Device driver approach
Advantage	1) More portable 2) No root privilege or administrator's support	1) No user-code modification 2) No kernel source modification
Disadvantage	1) Pages are still subject to disk paging, 2) Overhead of page fault handling, 3) Not application transparency(only benefit using the library)	1) OS page can be sent to remote memory, 2) The overhead of context switching and copy pages in and out of kernel (when user-level process is used for communicating with remote memory server[2])

Table 1. shows comparison between user-level approach and device driver approach [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

To use remote memory like local memory, first, remote memory should be mapped on virtual address space of remote memory consumer process. 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.

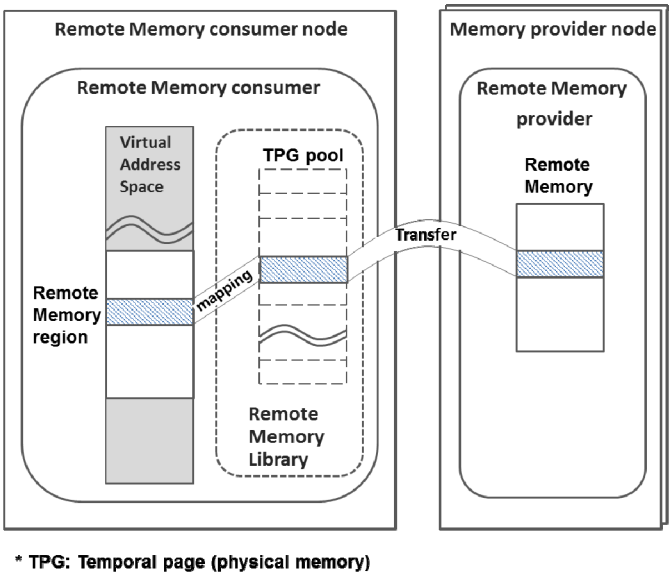

Figure 1. A mechanism of remote memory extension

Figure 1. 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.

A. User-level API for remote memory management

The remote memory library provides next user-level API for remote memory management.

- int rminit();
- int rmterminate();
- void *rmalloc(size_t size);
- void rmfree(void *ptr);
- int rmsync(void *ptr,size_t size);
- int rmflush(void *ptr,size_t size);
- int rmadvise(void *addr, int advise);

First, the programmer who develop large memory application should initialize remote memory library before allocating remote memory and terminate the library at the end of the application via rminit() and rmterminate(). After initialization, the programmer can allocate remote memory via rmalloc() and may release the memory via rmfree() after using remote memory. For more 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.

B. User-level page fault handling

In general, page fault handling means that kernel allocate physical page and updates 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 consumer process, we need to handle page fault at not kernel-level but user-level.

The only tricky way for user-level page fault handling is to use signal handling mechanism for segmentation fault signal(SIGSEGV). If user-level process tries to access one of pages of the memory region where access is not allowed, then CPU will raise the SIGSEGV signal and kernel should send the signal to the user process. We can handle the page fault at user-level by changing the action taken by the user process on receipt of SIGSEGV signal.

SIGSEGV signal handler selects a TPG from TPG pool(this mimics allocating physical memory page) and store real data from remote page to the TPG, and then map the TPG to the starting address of the page where page fault occurs. We use two system call in SIGSEGV signal handler: mprotect() and remap_file_pages(). The mprotect() makes the range of remote memory region readable, writable or inaccessible and the remap_file_pages() update pages table indirectly.

By the way, there is no way to allocate a physical page to two different virtual addresses at user-level. For user-level page fault handling, TPGs should be accessed by both consumer thread and user-level page fault handler. This means physical memory pages used as TPG Pool should be mapped to virtual address space of consumer process twice. One is TPG pool region for user-level page fault handler of remote memory library and the other is remote memory region for consumer's main thread. To make both consumer thread and user-level page fault handler access same physical temporal pages(TPG), we use a trick to create a TPG backing file on tmpfs and open the file, next call mmap system call with file descriptor of the file at remote memory library. A TPG backing file is mapped twice. First, the TPG backing file is mapped to the TPG Pool region which remote memory library manage. Second, the TPG backing file is mapped to the remote memory region of consumer process when rmalloc() API is called.

When consumer thread access the remote memory region, the page fault handler of the remote memory library will dynamically map TPGs by calling remap_file_pages systm call. Before all the TPGs are allocated and mapped to remote memory region, remote memory library will recycle the TPGs because TPG pool is very restricted resources.

C. Temporal page(TPG) pool management

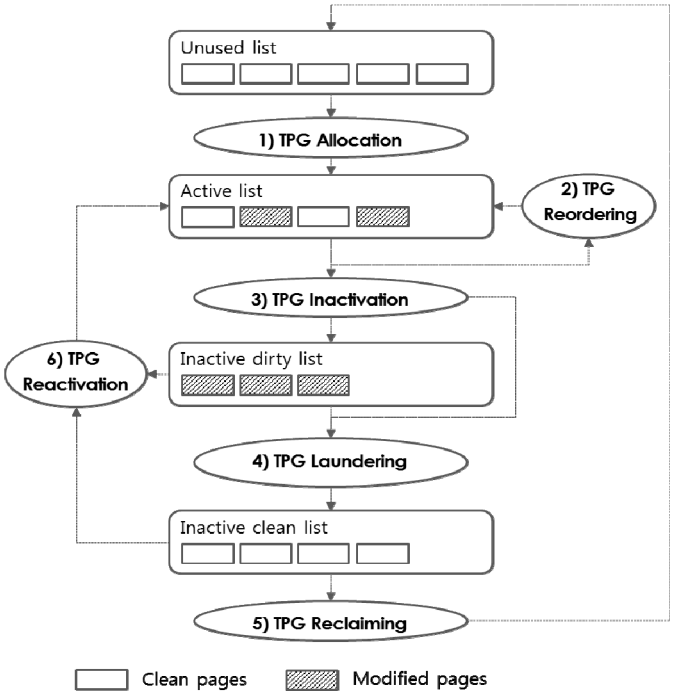

Figure 2. TPG pool management diagram

Figure 2. depicts how the remote memory library operates and manages temporal pages of 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 operation of TPG pool consists of 6 steps: 1) TPG Allocation, 2) TPG Reordering, 3) TPG Inactivation, 4) TPG Laundering, 5) TPG Reclaiming, and 6) TPG Reactivation.

As we explained in the previous section, if consumer thread accesses remote memory region, then page fault handler may be executed or not. If the accessed page of remote memory region is already mapped to TPGs, then the page fault handler is not called. If not, the page fault handler 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 step 1) TPG Allocation.

If the Unused list does not have any TPGs, then the TPG Allocation will be delayed until the TPGs is 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 3) TPG Inactivation.

The 2) TPG Reordering step moves some revisited TPGs to the tail of the Active list because the Active list should be ordered by least recently used(LRU) time.

The 3) TPG Inactivation step 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 move the overflowed TPGs to the Inactive dirty or clean list. The overflowed TPGs is selected from the head of the Active list because the head of Active list is LRU page.

The 4) TPG Laundering step saves the data of the modified TPGs to the remote page. This step spends considerable time because the step should send the data of the modified pages to the remote memory provider and should receive acknowledgement from it. The confirmed TPGs are moved to Inactive clean list.

The 5) TPG Reclaiming step periodically moves all the TPGs in the Inactive clean list to the tail of the Unused list.

The 6) TPG Reactivation step move some revisited TPGs of Inactive dirty or clean list to the Active list. Remote memory consumer may access the TPGs in the Inactive list. In this case the revisited TPGs are reactivated.

The period of TPG Inactivation is adjusted in reverse proportion to the activated(allocated) number of TPGs because static period of TPG Inactivation may cause the lack of unused TPGs in the Unused list. If the number of activated TPGs increase, the period should become shorter. If the number decrease, then the period should become longer.

D. Prefetching remote memory pages

The access latency of remote page is much greater than the one of local memory. Prefetching remote memory pages is essential mechanism to improve the latency and the throughput. Figure 3. shows the algorithm to prefetch remote pages in case of sequential access pattern. We defined three variables for prefetching algorithm: RA_flag, Current window, RA window. RA_flag indicate if readahead is ongoing. Current widnow represent already prefetched remote memory region. RA window means the remote memory region under prefetching.

First, in step 1), the algorithm determine if the access pattern is sequential. If the previously accessed page and currently accessed page are consecutive, or if the previously accessed page is the first page of Current window and the currently accessed page is the first page of RA window, then the algorithm determines the pattern is sequential.

If access pattern is sequential, the algorithm proceeds to the step 2). The step 2) checks if RA_flag is true, which means that readahead is ongoing. If RA_flag is false(now readahead is not being started), the algorithm proceeds to the step 3). The step 3) sets RA_flag true and initialize Current window and RA window. If RA_flag is true in the step 2), the algorithm checks if the requested page is in the RA Window in the step 6). If it is, then the algorithm updates Current window and RA window.

If access pattern is not sequential in step 1), then proceed to the step 4). The step 4) checks again if RA_flag is true. If it is true, the algorithm set RA_flag false and reset Current window/RA window in the step 5).

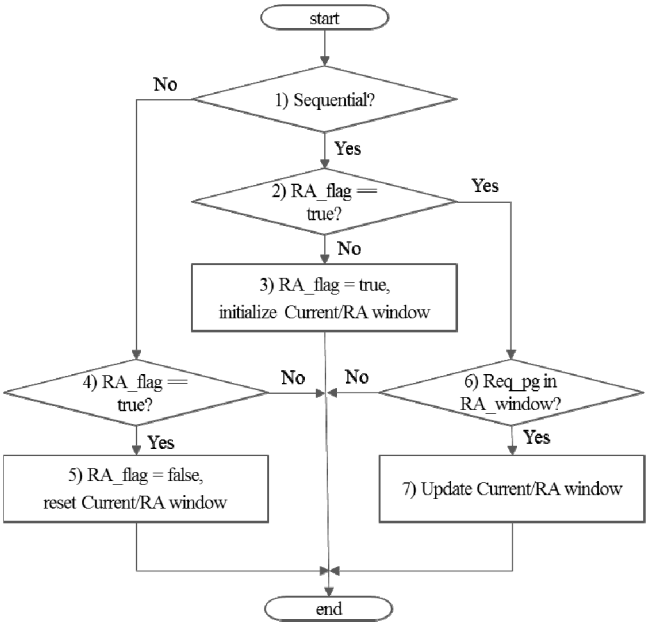

Figure 3. The algorithm of prefetching remote page for sequential access pattern

IV. AN EXPERIMENTAL STUDY

We wrote a Remote Memory Library which handle user-level page fault and manage temporal page pools and provide user-level API. We also wrote an emulation program which emulate remote memory extension using the Remote Memory Library.

A. An emulation program for experiment

Figure 4. shows the emulation program which copies 1GB data from (emulated) remote memory region to local buffer. It copies from the page of the emulated remote memory to local buffer's page. Actually, memory copy occurs twice: one is between the Emulated Remote Memory and TPG, the other is between the TPG and Local Buffer in Figure 4. The Emulated Remote Memory is actually allocated in local memory because we do not implement memory provider yet. The first memory copy between the Emulated Remote Memory and

TPG emulates the memory transfer from memory provider. This program is for only testing how much well the Remote Memory Library will work.

The size of TPG pool of the emulator program is 1024(4MB). The size of maximum Current window is 32 pages. The size of maximum RA window is also 32 pages.

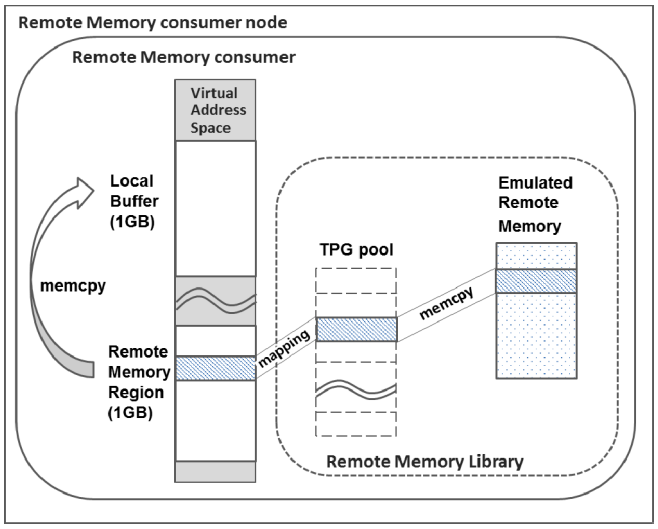

Figure 4. The emulation program copy 1GB data from the emulated remote memory to local memory.

B. Experimental result

The experiment environment are as follows. The server's HW specification is Xeon® CPU E5-2620, 2 socket 6 core, 128GB memory. For comparison, we measured memory latency of the above server using Intel Memory Latency Checker(v2.1). The average memory access latency is about 120 ns(local memory node: 90ns, the other memory node: 150 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 1189MB/s.

TABLE 2. LATENCY AND THROUGHPUT OF THE EMULATED REMOTE MEMORY

Access pattern	Metric	1	2	3	4	5	Avg.
Sequential	Latency (us)	1.3	1.18	1.17	0.7	0.93	1.05
	Throughput (MB/s)	817	819	822	853	819	826
Random	Latency (us)	19.9	21.4	19.6	21.5	21.5	20.8
	Throughput (MB/s)	165	173	144	138	158	155.6

The experiment was done for sequential memory access pattern and random access pattern. Above Table 2. shows the measured latency and the throughput per memory access pattern of the emulation program.

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

Average latency of sequential access pattern is 1.05 us. This is just 9 times of local memory latency(120ns). Memory copy throughput of sequential access pattern is 826MB/s. It is about 70% of the local memory copy throughput. However, the average latency of random access pattern is 20.8 us, which is 170 times longer than the local memory latency(120ns). The average throughput of random access pattern is 156MB/s and is 13% of the local memory copy throughput. The reason the latency and the throughput for sequential access pattern is not bad is because the emulation program does prefetching memory pages from the Emulated Remote Memory for the sequential access pattern. On the contrary, we observed very poor latency and throughput for the random access pattern because prefetching does not work in case of random access.

V. CONCLUSIONS

In this paper, we designed and implemented a remote memory extension library without communication feature with remote memory provider. The library emulates remote memory locally. We also wrote an emulation program for demonstrating the performance of remote memory extension library. The library can't be used for real application yet. It needs communication feature with remote memory provider, and also need remote memory provider as shown in Figure 1 for a real large memory application.

The remote memory library 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, and prefetching functions reading/writing data from/to emulated remote memory.

Through the experiment, we showed 9 times longer latency and almost 70% throughput rather than local memory in case of sequential access pattern, but 170 times longer latency and 13% throughput for random access pattern. From this result, we found that prefetching is very effective, but processing delay of page fault handling and page management is more increased. When we compared this result with our previous work which emulated very simple page fault handling without TPG management and prefetching features, the latency of random access pattern increase 6 times rather than the previous work's latency which was 3 microseconds.

If we develop remote memory provider additionally, we should consider network transmission. It will increase the access latency to remote memory and decrease throughput of memory copy to/from remote memory. The latency might increase and the throughput might decrease. Fortunately, there is very good networking technology. Many HPC systems use Infiniband network to interconnect computing nodes and storage nodes. Latency and throughput of Infiniband FDR network is not bad(latency: 1~2 micro seconds, throughput is almost 6GB/s). Infiniband also supports RDMA(Remote Direct Memory Access) feature which enables application to read/write remote memory without intervention of kernel. If

we use Infiniband network, we may not experience more latency delay or throughput degradation.

The mechanism we suggest might not be cost effective for someone who frequently runs large memory application and require very fast response time. Even though it has overhead of user-level page fault handling, it may be a silver bullet for someone who has cluster system and need to run large memory application sometimes. It would be a good solution for someone who develops applications with shared memory among cluster machines.

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] and the 'Cross-Ministry Giga KOREA Project' of the Ministry of Science, ICT and Future Planning, Republic of Korea (ROK). [GK14P0100, Development of Tele-Experience Service SW Platform based on Giga Media]

REFERENCES

[1] Douglas Comer, "A New Design for Distributed Systems: The Remote Memory Model", in Proc. Of 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 25th IEEE International Per-formance, 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 IEEE COMPCON'93 Conference, Febuary 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 Proceedings of the Eighth 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. of the 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 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 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 Proceedings of 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 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 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. Mazi`eres, 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 Proceedings 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.

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