A Remote Memory System for High Performance Data Processing

Hyun-Hwa Choi, Kangho Kim, and Seung-Jo Bae

Abstract—**Application's memory footprints are growing exponentially due to an increase in their data set and additional software layer. Modern RDMA capable networks such as InfiniBand and Myrinet with low latency and high bandwidth provide us a new vision to utilize remote memory. Remote idle memory can be exploited to improvement performance of memory intensive applications on individual nodes. Network swapping will be faster than traditional swapping to local disk. In this paper, we design a remote memory system for remote memory utilization in InfiniBand clusters. We present the architecture, communication method and algorithm of InfiniBand Block Device (IBD), which is implemented as loadable kernel module for version 3.5.0-45 of the Linux kernel. Especially, we discuss design issues transfer pages to remote memory. Our experiments show that IBD can bring more performance gain for applications whose working sets are larger than the local memory on a node but smaller than idle memory available on the cluster.**

Index Terms—**Remote memory, distributed memory, swapping, cluster system, InfiniBand.**

I. Introduction

Application's memory footprints are growing exponentially due to an increase in their data set and additional software layer. This memory requirement outpaces the growth in the capacity of current memory modules. Traditionally magnetic disks are used as the backing store for virtual memory. However the overall performance of applications is degraded due to the low speed of disks when applications need more memory than is physically available. In addition, since processor performance improves at a higher rate than disk seek latency, the cost of a disk access continues to increase with time [1].

Modern networking technologies such as Infiniband and Myrinet with low latency of a few microseconds and high bandwidth of up to 10 Gbps provide us a new vision to utilize remote memory for local system performance improvement. It is clear that paging to idle memory is faster than paging to disk because network ram eliminates the physical seek time and the bandwidth of network connections is increasing faster than the bandwidth to disk .

Several techniques have been proposed to use remote memory. These techniques include two approaches: remote memory mapping [2], [3] and remote memory swapping. Remote memory mapping techniques deal with the remote memory as an extension to the local memory space. These techniques require inflexible malloc-like APIs and recompilation of the existing applications. Remote memory swapping techniques deal with the remote memory as a swap device. The aim of these techniques is partially to fill the performance gap between local memory and hard disk without modifying the OS or the running applications.

Especially cluster systems, that encapsulate hundreds or even thousands of independent computing nodes within a single platform, have an imbalance of memory usage across different computing nodes. Therefore large amounts of idle cluster memory are almost always available for remote paging [4], [5]. Remote paging adds remote memory between main memory and disk in the local memory hierarchy. This caching technology provides an efficient way to boost application performance.

In order to improve cluster throughput at the main memory usage, we propose a remote memory system for clustered architectures. Our remote memory system, called IBD (InfiniBand Block Device), is a block device driver implemented as a loadable kernel module. Applications can take advantage of IBD without having to re-compile or link with special libraries. IBD can be used as disk device and be added as a swap device easily. In particular, we have the following contributions:

- We design a remote memory system exploiting the efficient low-latency high-bandwidth feature of InfiniBand.
- We demonstrate the feasibility and benefit of our remote memory system by developing a prototype and evaluating the performance of application programs.
- We give insights into how to take advantage of InfiniBand to significantly speed up the remote memory access.

The rest of the paper is organized as followings. Section II reviews the related work. Section III describes the design of the proposed remote memory systems and the study to improve the performance of remote memory access. Section IV provides the results from our evaluations and analyzes the findings. Finally Section V concludes the work and discusses the future works.

II. Related Work

There have been several previous projects that attempt to use remote memory in clustered systems. These remote memory systems can be either user-level or kernel-level implementation.

User-level implementation allows user applications to be

Manuscript received October 2, 2014; revised December 9, 2014. This work was supported by the ICT R&D program of MSIP/IITP [10038768, The Development of Supercomputing System for the Genome Analysis].

The authors are with the Cloud Computing Research Department, Electronics and Telecommunications Research Institute (ETRI), Daejeon 305-350, Republic of Korea (e-mail: {hyunwha, khk, sbae}@ etri.re.kr).

built on top of a user-level memory management library. This results in a lower performance due to increased context switching between user-space and kernel-space.

Dodo [4] requires application-level modification and provides remote memory as read-only caching. Writes to remote memory are propagated to disk in parallel to being sent to the remote node.

DLM (Distributed Large Memory System) [2] is user-level software as a swap device and offers better performance by tuning its parameters independently from kernel swap parameters.

Kernel-level implementations have better performance since all memory management is performed in the kernel. Therefore users have no conception of the use of remote memory.

ACMS (Autonomous collaborative Memory System) [5] proposes a memory acquisition protocol to perform autonomous memory collaboration across multiple nodes within a cluster. ACMS classify nodes into memory client node, memory server node and memory neutral node according to their memory usage. The nodes perform dynamic memory collaboration using the memory acquisition protocol over Ethernet.

HPBD (High Performance network Block Device) [6] is a kernel-level swap device using native InfiniBand communication verbs. The client is a block device driver and the server is a RamDisk based user space program. The client send requests for to the remote memory servers, then the remote memory servers perform RDMA read and RDMA write operations based on request type. The execution time of quick sort using HPBD is up to 21 times faster than local disk.

LocaSwap [7] provides remote memory based on Ethernet network in kernel space called LocaBus [8]. LocaSwap is the virtual RamDisk implemented as a normal block disk driver. During the load of the client module, the client sends remote memory servers messages requesting it to reserve a specified amount of memory for remote access. The reserved memory is managed by sector granularity.

Nswap [9] has no centralized server. Each node independently can make swapping decision based on partial information about the state of the cluster. Nswap supports migration of remotely swapped pages between the cluster nodes, and supports dynamic growing and shrinking of each nodes' Nswap cache.

R2MS [1] is a kernel-level reliable remote memory system to validate whether a remote memory system is efficient for large memory data processing. R2MS transfers all pages into remote memory with Remote Direct Memory Access operations over Infiniband and provides a fast crash/recovery mechanism.

III. IBD Implementation

In this section, we describe the proposed InfiniBand Block Device (IBD), including architecture, communication protocol and algorithm.

A. IBD Architecture

IBD consists of a memory client node and memory server nodes as shown Fig. 1.

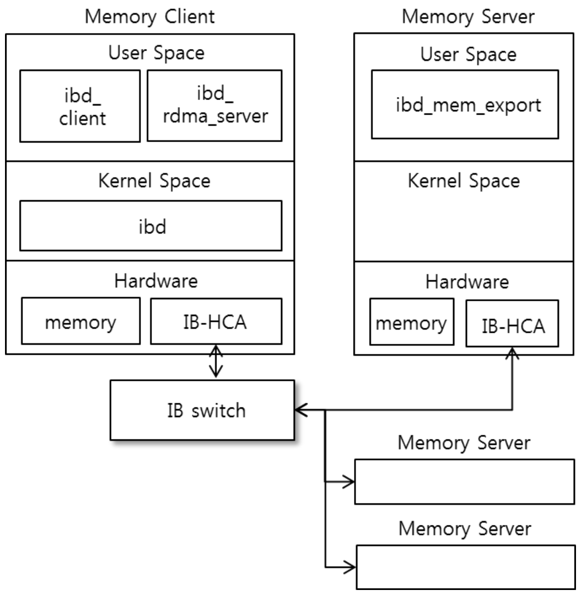

Fig. 1. IBD architecture.

A memory client node performs computation, accesses its memory space. When it exhausts its local memory, it will free pages by using memory server nodes. A memory server acts as fast backing store for the memory client node. In other words, the pages are stored on the memory server nodes and are brought back into local memory on the memory client node by demand.

The memory client and the memory server nodes are interconnected through InfiniBand with a scalable, high bandwidth. In an InfiniBand network, nodes are connected to the switch-based network fabric by Host Channel Adapters (HCA). The data transfer between the local memory of a memory client node and the remote memory of a memory server node is performed with Remote Direct Memory Access (RDMA) over InfiniBand.

IBD has four components such as ibd, ibd_client, ibd_rdma_server and ibd_mem_export.

The ibd, which is a Linux kernel module, allows remote memory to appear as block device on a memory client node. When the ibd is loaded, a dev file, /dev/ibd0, is created. The ibd block device can be accessed as either a swap partition or as a normal file system. We easily construct a file system on it or map it on Linux's swap mechanism using standard system tools such as mkfs.ext2, mkswap, mount and swapon.

The ibd_client configures the available memory servers and how much memory they have. The size of the ibd block device is less or equal to the amount of remote memory requested from the memory server nodes. In addition, the ibd_client is used to terminate IBD.

The ibd_rdma_server on a memory client node is used for page transfer. I/O operations are performed in sector granularity. Using the range value of the sector, the ibd_rdma_server determines which memory server has a requested sector and where the sector is stored on the memory server's memory.

The ibd_mem_export is executed on a memory server node. When it is loaded, it will wait for a connection request from the ibd_rdma_server. Once a request is received, the reserved memory is allocated for the ibd_rdma_server.

B. Communication Method

IBD uses two types of communication channels: Ethernet and InfiniBand, as shown in Fig. 2. It uses TCP/IP sockets based on Ethernet for synchronization messages between the memory client and memory server to exchange peer rkey. Otherwise, it sends the kernel's paging requests to the remote memory servers using native InfiniBand communication verbs.

We implemented the remote memory detecting, allocation and deallocation protocol by communication between the ibd_rdma_server and the ibd_mem_export which are connected via Ethernet NICs. The ibd_mem_export on a memory server node donates a portion of its memory space. Once ibd_rdma_server runs on a memory client node, it exchanges communication information such as the unique node ID, information about the message transfer engine of InfiniBand architecture called queue pair (qp) and rkey to resolve memory mapping. The key exchange operations are performed over TCP/IP socket connections.

To transfer real data page, we exploit the InfiniBand architecture as a backbone network. Communication over InfiniBand requires message buffers to be registered with the OS. Memory registration is based on virtual memory addresses to pin down pages for DMA operations with the HCA. This allows data delivery to remote memory buffers directly with zero-copy along the communication path. The RDMA read/write operations can lead performance improvement of the IBD.

C. Data Transfer Policy

As defined above, an IBD provides as a backing store memory exported by multiple memory servers. In multiple memory server scenario, there is a design issue of load balancing such as data striping and request multiplexing. The data striping and request multiplexing are traditional ways to exploit parallelism of a single request. Because of the high bandwidth feature of InfiniBand, we determine not to split a single request to multiple one. The IBD allocates sector values on reserved memory in order of memory server names taken by user. Then the IBD sends a page-out request to the corresponding memory server according to the sector value.

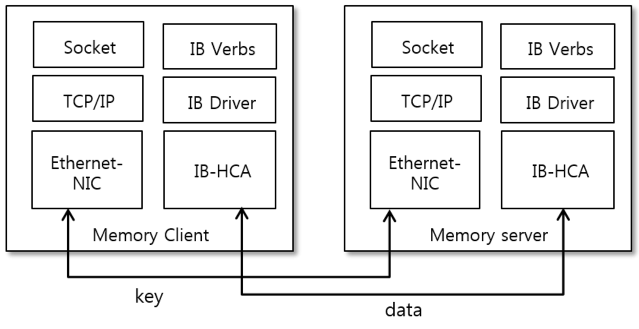

Fig. 2. Communication method.

IV. Data Transfer Design

There are many issues in designing and implementing an IBD. This section discusses design issue related to the data transfer path on a memory client node. Fig. 3 shows different data transfer models implemented by us.

A. User Level Data Transfer

A key of InfiniBand Architecture gives every application direct access to the messaging service. Direct access means that an application need not rely on the operating system to transfer data. The InfiniBand driver on user space can interact directly with the hardware by writing to a segment of mapped memory. This is involved in functions calls such as ibv_post_send and ibv_post_recv. Avoiding kernel traps is one way to decrease the overall latency of send/receive operations.

Fig. 3 (a) shows a path on a memory client node to send paging request. Linux virtual memory manager manages physical memory on memory client node. When free pages available to the virtual memory manager fall below a threshold, page-out requests are triggered by the kernel thread kswapd to swap pages out to the backing store on swap devices. An ibd kernel module serves the swap requests as normal block I/O requests. The ibd_rdma_server gets the I/O requests from the ibd to utilize zero-copy data transfer and starts data passing using communication buffer registered with the HCA. At this time, memory copy occurs from kernel space to user space, and context switch of the ibd_rdma_server needs two times between user space and kernel space. The cost of memory copy and context switch is considerable.

B. User and Kernel Shared Data Transfer

The POSIX mmap() interface offers a viable way to map file or device into memory. Using such an interface, the ibd_rdma_server on user space can share paging request memory allocated by the ibd module on kernel space. As shown Fig. 3 (b), this model removes memory copy for paging request from kernel space to user space, thus provides potentials for performance improvement.

C. Kernel Level Data Transfer

Fig. 3 (c) shows the ibd_rdma_server implemented on kernel space. This model is the model without memory copy and context switch of user level data transfer. We will report the benefit of the kernel level data transfer in Section V.

V. Performance Evaluation

This section describes a set of performance experiments done to determine the effectiveness of IBD. The main goal of the IBD is to use remote memory for high performance. Therefore the experiments are designed to test the performance of using remote memory as swap device.

The experiments are conducted on a cluster of dual Intel Xeon E5-2650 2.67 GHz nodes. Each node has 128 GB physical memory. All nodes are connected to InfiniBand network using Mellanox SX6036 FDR IB switch with 36 port and Mellanox FDR InfiniBand HCA (ConnectX-3). Each node has a 1TB SATA3 hard disk. The operating system is Ubuntu with a Linux kernel version 3.5.0-45.

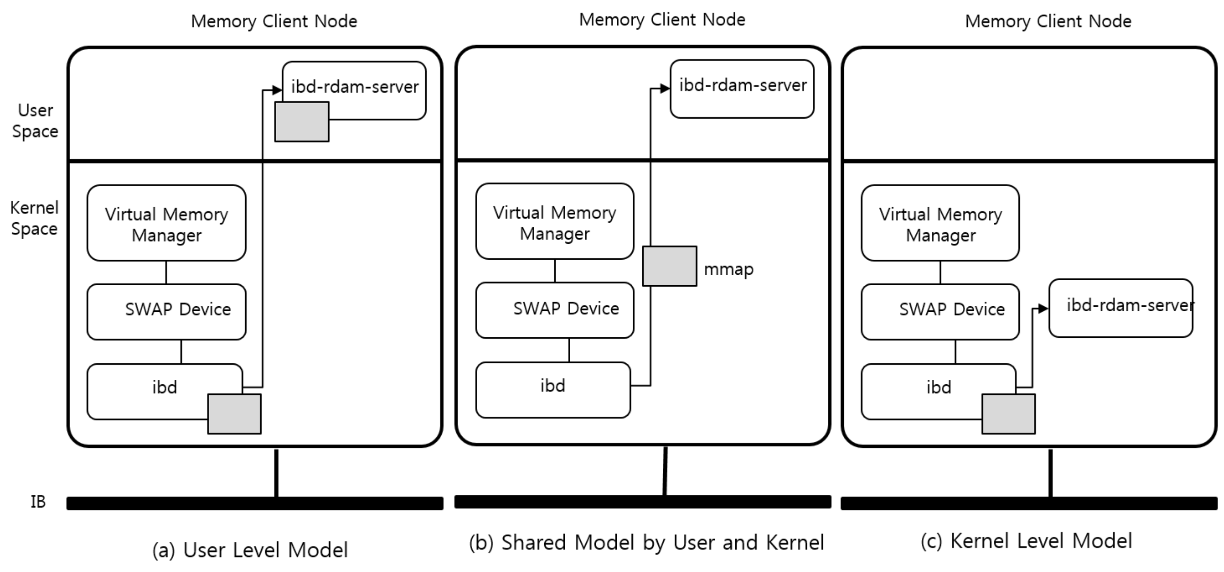

Fig. 3. Data transfer model.

To compare the performance impact of remote paging with local memory performance and study the impact of network performance on remote paging, we change the total local memory size available to the OS.

We first evaluated the performance with enough local memory, swapping over IBD, and local disk. We use the performance of applications running in-memory as the base line for evaluation. The experiment is performed by a simulation program written in C called allocating-memory. The simulation program includes memory allocation and initialization.

Fig. 4 shows the time required for the memory allocation and initialization for various amounts of requested memory. Each experiment is repeated ten times. For comparison of the usual hard disk swapping and IBD swapping, we limit the local memory size that is available to 10GB.The experiments are performed by allocating memory of 20, 30, 40, 50, and 60 Gigabytes. Thus in all experiments, allocating-memory allocates more memory than physically available, forcing the operating system to swap memory. The allocating-memory allocated five times more memory than physical memory available.

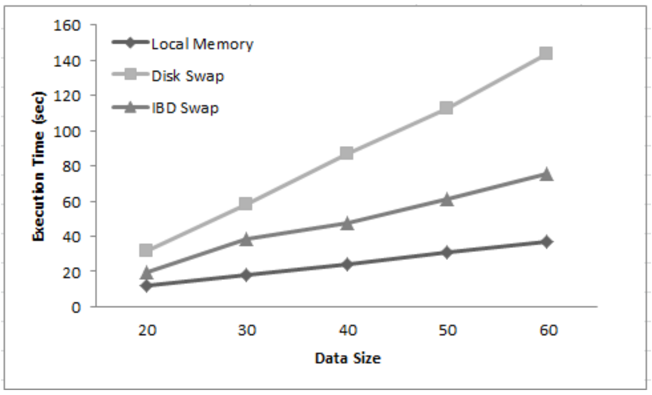

Fig. 4. Impact on IBD.

Because the result of the execution time without local memory experiments includes the time required to swap out inactive processes, the execution times by IBD swap and hard disk swap are longer than for local memory. But, the time taken by IBD swap to do the memory allocation and initialization experiment is shorter than the required for hard disk. The performance gaps between the IBD swap and the hard disk swap increase as the amount of used swap space increases. Even though the Linux kernel tries to re-order disk I/O requests sequentially to minimize the disk head movement, IBD performs significantly better than the hard disk. For 60GB experiment, local memory execution time is 36.73 seconds, while IBD takes 75.5 seconds and hard disk takes 143.64 seconds. Thus memory is only 2 times faster than IBD, and IBD swapping is 2 times faster than local disk.

This is caused by the fact that the order in which pages are swapped out is not the same as the order in which pages will be swapped back in. In other words, the swap mechanism causes the random disk head movement. The IBD is good at transferring data on the random nature of access patterns.

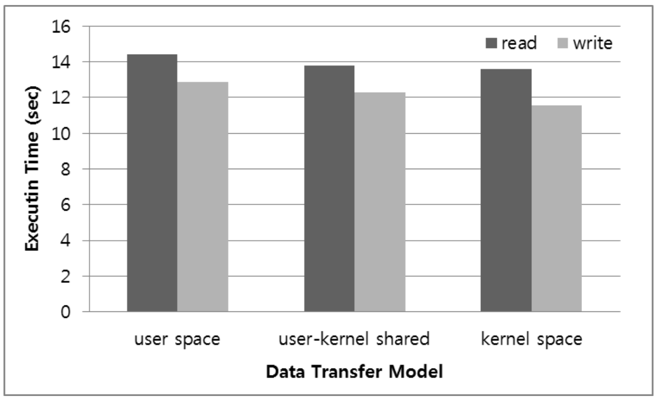

Fig. 5. Impact on data transfer model.

We measured the performance of the IBD data transfer models. We performed sequential read and write experiments using Linux utility dd. Fig. 5 shows the execution time of data read from or written to the remote memory. Clearly, the IBD implemented on kernel space results in higher performance of read and receive operation. By avoiding memory copies and kernel context switches, the shared data transfer model between user space and kernel space improves data read and write operation performance by about 4%, and the kernel data transfer model improves it by about 10%. This means that the host overhead along the path for data transfer becomes an important performance factor.

VI. Conclusion

This paper investigated the potential benefits of a remote memory system. Considering the speed gap between DRAM and networks, the performance results show that the remote memory system as swap device has better performance than we expected. The IBD outperforms hard disk based swapping by up to 2 times.

We evaluated the improvements that can be achieved by avoiding the memory copy and context switch. Our results show that data transfer path on host node is the important factor affecting application performance on IBD.

In our future work, we plan to investigate the scalability of our IBD and the ways of minimizing host overhead on the swapping critical path. Plans also are in place to provide a method migrating remote pages between memory servers.

Acknowledgment

This work was supported by the ICT R&D program of MSIP/IITP. [10038768, The Development of Supercomputing System for the Genome Analysis]

References

[1] H. Han, H. Jung, S. Kang, and H. Y. Yeom, "Performance evaluation of a remote memroy system with commodity hardware for large-memory data processing," *Cluster Comput*, vol. 14, no. 4, 2011

[2] H. Midorikawa, M. Kurokawa, R. Himeno, and M. Sato, "DLM: A distributed large memory system using remote memory swapping over cluster nodes," in *Proc. the IEEE International Conference on Cluster Computing*, 2008, pp. 268-273.

[3] N. Wang, X. Linu, J. He, J. Han, L. Zhang, and Z. Xu, "Collaborative memory pool in cluster system ," in *Proc. the International Conference on Parallel Processing*, 2007, p. 17.

[4] S. Koussih, A. Acharya, and S. Setia, "Dodo: A user-level system for expoiting idle memory in workstation cluster," in *Proc. the IEEE International Symposium on High Performance Distributed Computing*, 1999.

[5] A. Samih, R. Wang, C. Maciocco, T. Y. C. Tai, and Y. Solihin, "A collaborative memory system for high-[erformance and cost-effective clusterd architectures," presented at Workshop on Architectures and systems for Big Data, 2011.

[6] S. Liang, R. Noronha, and D. K. Panda, "Swapping to remote ,emory over InfiniBand: an approach using a high performance network block device," in *Proc. the IEEE International Conference on Cluster Computing*, 2005.

[7] P. Werstein, X. Jia, and Z. Huang, "A remote memory swapping system for cluster computers," in *Proc. the International Conference on Parallel and Distribured Computing*, pp. 75-81, 2007.

[8] P. Werstein, M. Pethick, and Z. Huang, "LocalBus: A kernel to kernel communicaiton channel for cluster computing," in *Proc. the International Conferences on Parallel and Distribuetd Computing, Aplication and Technologies*, 2004, pp. 497-504.

[9] T. Newhall, S. Finney, K. Ganchew, and M. Spiegel, "Nswap: A network swapping module for Linux clusters," *Euro-Par*, pp. 1160-1169, 2003.

Hyun-Hwa Choi received the B.S. degree in computer engineering from Chungnam National University in 2000, and the M.S. degree in computer engineering from Pohang university in 2002. She received the Ph.D. degreee in computer engineering from Chungnam National University in 2013.

She joined ETRI (electronics and teleconnumications research institute), Daejeon, Korea in 2002. From 2002 to 2003, she was involved in development of a large scale data management system. From 2004 to 2006, she was involved in development of the smart object processing framework that provides the real time processing of large sensor data stream. From 2007 to 2009, she was involved in development of the distributed data processing system. From 2010 to 2012, she developed a workflow management system for high performance computing. Since 2013, she has currently developed a block device driver supporting remote memory system for data intensive applications.

Dr. Choi's research interests include parallel and distributed processing, high-dimensional query processing, and data stream management system

Kangho Kim received the B.S. degree and the M.S. degree in computer engineering from KyungBuk National University in 1993 and 1996, respectively.

He joined SERI (software engineering research insttitute) and ETRI (electronics and teleconnumications research institute), Daejeon, Korea in 1995 and 2000 respectively. From 2000 to 2008, he was involved in development of Linux and cluster management system. Since 2009, he has developed system virtualization. Mr. Kim's research interests include operating system and next generation memory.

Seung-Jo Bae received the B.S. degree in computer engineering from Yonsei University in 1987. He received the M.S. degree and the Ph.D. degree in computer and information science form Syracuse University in 1992 and 1997, respectively.

He joined ETRI (electronics and teleconnumications research institute), Daejeon, Korea in 1997. Since 1997, he was involved in development of execution engine for high performance computing. Dr. Bae's research interests include parallel and distributed computing and execution engine.