Swapping to Remote Memory over InfiniBand: An Approach using a High Performance Network Block Device*

Shuang Liang Ranjit Noronha Dhabaleswar K. Panda

Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{*liangs,noronha,panda*}*@cse.ohio-state.edu*

Abstract

Traditionally, operations with memory on other nodes (remote memory) in cluster environments interconnected with technologies like Gigabit Ethernet have been expensive with latencies several magnitudes slower than local memory accesses. Modern RDMA capable networks such as InfiniBand and Quadrics provide low latency of a few microseconds and high bandwidth of up to 10 Gbps. This has significantly reduced the latency gap between access to local memory and remote memory in modern clusters. Remote idle memory can be exploited to reduce the memory pressure on individual nodes. This is akin to adding an additional level in the memory hierarchy between local memory and the disk, with potentially dramatic performance improvements especially for memory intensive applications. In this paper, we take on the challenge to design a remote paging system for remote memory utilization in InfiniBand clusters. We present the design and implementation of a high performance networking block device (HPBD) over InfiniBand fabric, which serves as a swap device of kernel Virtual Memory (VM) system for efficient page transfer to/from remote memory servers. Our experiments show that using HPBD, quick sort performs only 1.45 times slower than local memory system, and up to 21 times faster than local disk. And our design is completely transparent to user applications. To the best of our knowledge, it is the first work of a remote pager design using InfiniBand for remote memory utilization.

*This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506, and National Science Foundation's grants #CCR-0204429, and #CCR-0311542.

1 Introduction

According to Moore's law, the computing power of modern CPUs doubles approximately every 18 months. Similar trends apply to the capacity of modern memory and disk systems. This allows modern systems to process large amounts of data "in-memory" with an improved throughput. It also enables developers to design and implement algorithms previously considered impractical to exploit the rich resources of these systems.

However, even with the dramatic increase in memory capacities, modern applications are quickly keeping pace with and even exceeding the resources of these systems. For example, modern databases typically maintain millions of records. Keeping the working set in memory for database transactions demands a high volume of memory space, and could potentially strain memory resources. In these situations, modern systems with virtual memory management start to swap memory regions to and from the disk. Swapping to disk may severely impinge on the overall performance for applications.

Modern networking technologies such as InfiniBand, Myrinet and Quadrics [9, 17, 7] with their low-latency of a few micro-seconds and high throughput of up to 10 Gbps, especially the Remote Direct Memory Access (RDMA) operations featuring low CPU utilization provide us a new vision to utilize remote resources for local system performance improvement. In this paper, we take on the challenge to design a system that can utilize remote memory for local memory hierarchy enhancement in InfiniBand based cluster systems. We aim to achieve the following goals:

- Design a remote memory system exploiting the efficient low-latency high-bandwidth feature of InfiniBand, which can deliver a comparable performance to local memory system.

- Evaluate the network performance impact on our remote memory system.
- Enable applications to benefit from remote memory transparently.

Some previous works have investigated methods to take advantages of remote memory [1, 2, 4, 8, 11, 15]. In this paper, we take the approach of remote memory paging to improve system performance. There are several reasons for this choice:

1. Remote paging automatically adds remote memory between main memory and disk in the local memory hierarchy. Thus all the sophisticated caching techniques evolved from the past for the memory hierarchy are still in place for performance optimization.
2. Remote paging capability allows processes to take the benefits of remote memory transparently.
3. Even though parallelizing applications is another way to utilize cluster resources including memory resources, it may not be an easy task to deliver best performance for applications with mostly fine-grained parallelism. Additionally, parallelization may not be cost effective in some cases.

Remote paging provides an efficient way to boost application performance in these cases. We implement our design of HPBD - an optimized remote pager using native InfiniBand communication, and evaluate the performance using micro-benchmarks and applications. Our experiment results show that using HPBD, quick sort performs only 1.45 times slower than using abundant local memory, and up to 21 times faster than using local disk as the swap device.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 provides the relevant background. Section 4 and 5 present the design and implementation of the remote memory pager. An experimental evaluation of the remote memory pager is discussed in Section 6. We conclude this paper in Section 7.

2 Related Work

Several works have studied the utilization of remote memory in the context of cluster environment for different purposes. we broadly differentiate these works by four criteria: a) simulation or implementation based; b) global resource management system or resource sharing tools; c) user-level designs or kernel-level designs; and d) User-Level Protocol(ULP) or TCP/IP based.

As shown in Table 1, the simulation based studies include "Job Migration and Network RAM" (JMNRM) [23], "Parallel Network RAM" (PNR) [18] and "Cooperative Caching" (COCA) [4]. In JMNRM and PNR, Xiao et al. studied the impact of combining network memory and job migration for system scalability and throughput improvement, PNR is later proposed to utilize global memory for parallel scientific programs. In COCA, Dahlin et al. studied the performance benefits of cooperative file caching using client memory, and evaluated several cooperative caching algorithms using trace driven simulation.

		Global Management	Kernel Level Design	TCP/IP Based	ULP Based
Simulation Based	COCA[4]	Y	N/A	N/A	N/A
	PNR[17]	Y	N/A	N/A	N/A
	JMNRM[24]	Y	N/A	N/A	N/A
Implementation Based	NRAM[5]	N	N	Y	N
	NRD[12]	N	Y	Y	N
	RRMP[14]	N	Y	Y	N
	MOSIX[3]	Y	Y	Y	N
	GMM[7]	Y	Y	Y(UDP)	N
	DoDo[10]	Y	N	Y	Y
	HPBD	N	Y	N	Y

Table 1. Modern work in designing remote memory system

Studies on MOSIX [3] and the Global Memory Management (GMM) [8] address the problem of remote memory utilization by cluster-wide resource management. Both are based on kernel level implementation. MOSIX is a load balancing system with transparent job migration. It uses a memory ushering algorithm [2] to choose target nodes and avoid disk swapping. GMM is a global memory system for OSF/1 workstation clusters. It is designed as a kernel module to function together with the node's VM system, page-out daemon and unified buffer cache. S. Koussih et al. designed a run time system DoDo [11] to use remote memory from a user level perspective. It is implemented on top of a U-NET [21] communication architecture and provide a socket interface for portability.

Network RAM Disk (NRD) [13] and Reliable Remote Memory Pager (RRMP) [15] focus on reliability studies of remote memory utilization. E. Anderson and J. Neefe studied design issues of Network RAM and proposed a user-level signal handling based implementation [5]. All these three works are based on TCP/IP.

Another related work is GNBD/VIA [10], in which K. Kim et. al. proposed a kernel level socket interface over VIA KVIPL library for GNBD (A Network Block Device for GFS [19]) to exploit the raw performance provided by Virtual Interface Architecture(VIA) for file system performance improvement. It focuses on matching communication semantics between socket and VIA design for file block transfer.

Our work is different from the previous works in that we focus on the study of network performance impact on remote paging. We propose an optimized design to use the

InfiniBand features such as RDMA operations and asynchronous event handling, which can deliver a performance comparable to local memory system. It is a kernel level design, and is completely transparent to user applications.

3. Background

3.1. InfiniBand Overview

The InfiniBand Architecture (IBA) [9] is an open specification designed for interconnecting compute nodes, I/O nodes and devices in a system area network. It defines a communication architecture from the switch-based network fabric to transport layer communication interface for inter-processor communication. In an InfiniBand network, compute nodes are connected to the fabric by Host Channel Adapters (HCA). HCA exposes a queue-pair based transport layer interface to the hosts. The send queue keeps control information for outgoing messages, while the receive queue keeps descriptions for incoming messages. Communication requests are submitted to the queues through descriptors in a non-blocking fashion. Completion of requests are reported through Completion Queues (CQs), which can be shared among different queue pairs.

Communication over InfiniBand requires message buffers to be registered with the OS. This allows message delivery to application buffers directly with *zero-copy* along the communication path. This is a salient feature for modern interconnects which enables the high bandwidth low latency capabilities of InfiniBand, especially for large messages. Yet for system memory management, *zero-copy* can't be exploited directly. Memory registration is based on virtual memory addresses to pin down pages for DMA operations with the HCA. While for remote memory exploitation, pages needs to be swapped in/out to remote nodes. Thus if we choose to use the *zero-copy* feature for page transfer, registration on-the-fly is needed. More details are discussed in Section 4.1 for the trade-offs.

Two communication semantics are supported in IBA: channel semantics with traditional send/receive operations and memory semantics with RDMA operations. RDMA operations allow one side of the communication parties to exchange information directly with the remote memory without the involvement of the remote host. This enables better computation and communication overlap, thus provide potentials for performance improvements.

To meet the needs of QOS, several service levels are provided in InfiniBand, such as Reliable Connection based service(RC), Unreliable Connection based service(UC), Reliable Datagram based service(RD), Unreliable Datagram based service(UD) and RAW Datagram. To reduce the complexity of reliability issues for paging requests, we focus on RC in this paper.

InfiniBand supports IP emulation IPoIB. IP based application can run directly over the same InfiniBand fabric. Figure 1 shows the basic latency performance for *memcpy*, RDMA write operation, IPoIB and GigE with data segment of size up to 128K. It shows that RDMA_WRITE latency between two nodes is quite comparable to local *memcpy* latency. Thus using RDMA operations provides the potential of significant performance improvement for remote paging.

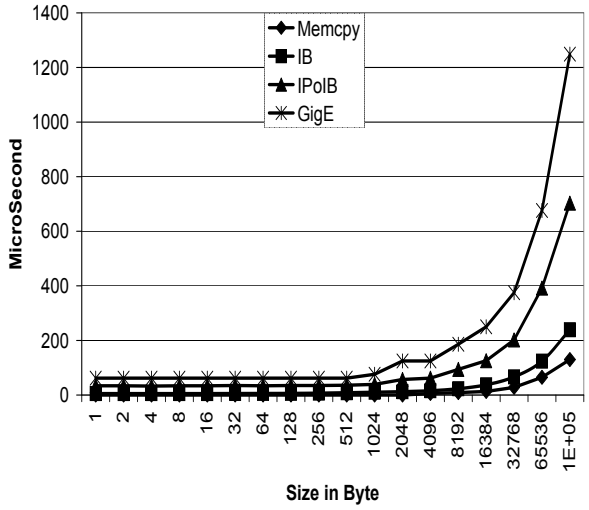

Figure 1. Latency Comparison of Different Networks and Memcpy Up to 128K

3.2. Linux Swapping Mechanism

Paging is an important function of Linux virtual memory system (VM). VM manages all physical memory resources. When free pages available to VM fall below a threshold, page-out requests are triggered by the kernel thread *kswapd* to swap pages out to the back-store on swap devices. Page-in requests are invoked on demand as page faults occur. Multiple swap devices are supported by Linux kernel. Page-out data are placed to these devices based on their priorities. The device driver for each swap device serves the swap requests as normal I/O requests and deals with device specific operations. Thus, designing a block device driver which supports I/O requests to/from remote nodes is a viable solution for remote pager design. This mechanism puts remote memory between local memory and local disk system in the memory hierarchy with the caching mechanisms enabled at no additional costs. It is completely transparent to applications, and can be beneficial to overall system performance. In this paper, we use this approach to design a native InfiniBand-capable network block device driver for remote paging.

3.3. Network Block Device

Network Block Device is a software emulation for local block storage using remote resources at the block level. The idea is to provide a local block level interface to upper OS management layer, while allocate and deallocate remote resources over network. It is widely used in file systems for data storage such as in [19]. NBD [14] is a Linux implementation of Network Block Device over TCP/IP using kernel-level socket interface for network communication. It is available in the Linux kernel source code.

As motivated by the network performance impact studies on remote paging, we compare the performance of HPBD with NBD. We activate NBD over GigE and IPoIB for our experiments. As of Linux-2.4.18 kernel, a single NBD device can only be served by a single remote server. We note that although we are able to use NBD as a swap device in our experiment, deadlock is reported [14] because of memory allocation in TCP networking.

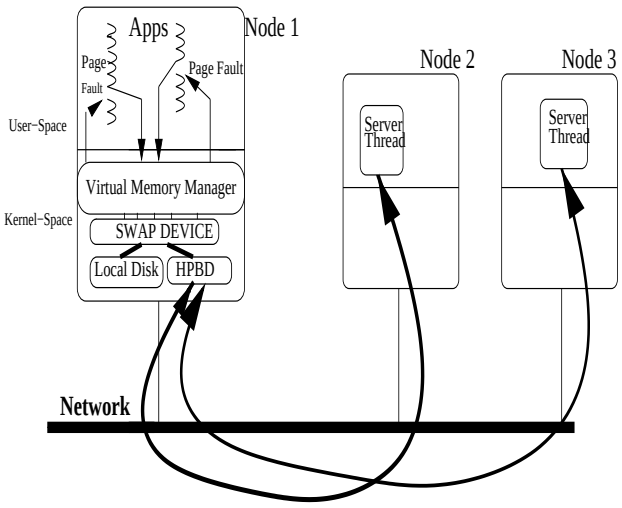

Figure 2. Remote Memory System Architecture

4. Proposed Design

In this section, we analyze the design issues and present our design of HPBD.

4.1 Design Issues

- Kernel level vs. user level design:

  To implement a remote pager, two general approaches can be considered: a) kernel level design or b) user level design. The kernel level approach of a device driver design is introduced in Section 3.2. For user level designs, the general idea is to implement a modified dynamic memory allocator for the run-time library system and provide remote memory allocation capabilities, such as [5, 11]. Although it is relatively easy to implement, there are several inherent disadvantages for a user level design: a) pages are still subject to disk paging by the underlying OS; b) the memory protection mechanism as the basic implementation technique involves high overhead; and c) user space design is not completely application transparent and can only benefit applications using the library.

  With the device driver approach, we can avoid the above problems. Additionally, portability across different platforms can be achieved with a kernel module implementation. Classic kernel mechanisms for performance optimization such as page replacement algorithms are also in place. Figure 2 presents the system architecture of our remote paging system.

- Memory registration and buffer management:

  As other high performance interconnects, InfiniBand depends on Network Interface (NI) aware DMAable buffers to implement *zero-copy* data transfer. It requires that communication buffers must be registered with the HCA before message passing can start.

  As shown in Figure 3, memory registration operation is a costly operation. Thus in most designs, applications allocate a large memory buffer pool and pre-register it, avoiding the repetitive registration cost on the critical path to minimize the overhead. This can be done by a user application in a transparent way as in [12], which implements a *malloc* hook to keep track of registered buffer entries, and cache registered buffers on deallocation for future use. Though this method is viable with user space application, it can't be used for remote paging. Paging requests can potentially come from anywhere in the paged memory system. The same physical page could be associated with different virtual address between requests. Therefore, no persistent address association exists to allow registered page caching. While registration on-the-fly remains a choice, it is very costly compared with copy cost as shown in Figure 3, especially within the range of 4K - 127K where the page requests reside.

  We design a pre-registered memory pool allocator, and copy pages to/from this area for communication. For most applications, the average page request size is much smaller than 127K, thus the benefit of the copying solution is more significant.

- Asynchronous communication:

  In a client-server architecture, the swapping process

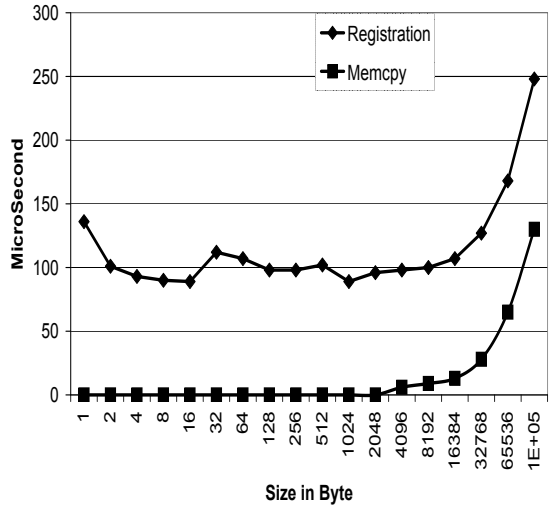

Figure 3. Memory Registration vs. Memcpy Cost

sends out paging requests to remote memory server and waits to be served. In user space InfiniBand design, this can be accomplished by simply polling the completion queue(CQ). With a kernel based approach, it is not practical as most OSes are not preemptive in kernel mode(Linux is not preemptive until 2.5 version). Asynchronous communication must be supported in this scenario. At the server side, asynchronous mode can also significantly decrease the host CPU usage.

- Thread safety:

  As a device driver, HPBD client is a shared resource, thread safety must be ensured. Since we take advantage of the thread safety feature of VAPI [16], the verb interface provided by our InfiniBand stack, our focus will be mainly on exclusion of the internal device data structures, such as internal request queues and buffer management primitives.

- Reliability and error handling:

  Reliability is an important issue for swap device design. Failure in page handling can adversely impact system stability and even crash the system. We choose RC service as our network transport, it excludes most of the reliability issues from network and message signature is used to validate requests and responses. In [6] and [13], other reliability techniques such as mirroring and parity are studied. These issues are out of the scope of this paper.

4.2. Designing HPBD

HPBD is based on client-server architecture, as shown in Figure 2. It serves the kernel's paging requests by communicating with remote memory servers using native InfiniBand communication verbs. The client is a block device driver, which serves I/O requests stream from the VM system by sending requests to the remote memory servers. The server is a RamDisk based user space program, which provides it own local memory for paging store and push/pull pages from client using RDMA operations.

4.2.1 RDMA Operations and Remote Server Design

In HPBD, there are two types of messages: control message and data message. Control messages are used to send page requests and acknowledge request completions. Data messages are for actual page transfers. In our design, we use both RDMA read and RDMA write operations for data message traffic. The remote memory server decides the type of RDMA operations based on the request type. As shown in Figure 4, RDMA read operation is used for swap-out paging requests to pull data out of the client, and RDMA write operation for swap-in requests to push data into client. By allowing multiple outstanding RDMA operations, RDMA and *memcpy* overlap is supported, which improves server side CPU utilization.

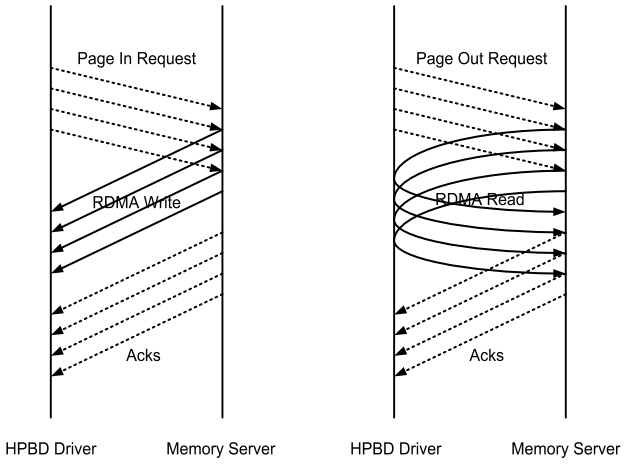

Figure 4. RDMA Design for Remote Memory Server

We choose to make the server initiate the RDMA operation for several reasons:

In our current design, RamDisk is used as a memory provider. Since RamDisk is exposed by a file system interface, we can't directly obtain the memory address of the server for the client to initiate RDMA operations.

Second, with this architecture the server can potentially provide any device attached for page store instead of using main memory only, thus more flexibility is allowed for future work.

Third, as we plan to provide a more flexible server design, which can provide idle memory dynamically. For similar reasons discussed in Section 4.1, pre-registration is not a feasible here. Thus, the server can't export memory address as a priori for client initiated RDMA operations.

4.2.2 Registration Buffer Pool Management

Registration buffer pool is a pre-registered memory area for data message transfer. It is initialized at device load time with a default pool size of 1MB. Memory buffers are allocated from the pool by a first-fit algorithm.

Buffer allocation failure must be carefully dealt with, since swap request failure will potentially crash applications or even the entire system. A memory allocation wait queue is used to accommodate the allocation requests that can not be filled temporarily. Deallocation of data buffers will wake up any threads that is in the queue.

One problem with the allocation algorithm is external fragmentation of the registration buffer pool. This can cause lots of the complexities in the implementation, and may cause multiple *memcpy* operations for a single request, thus negatively impact our system's performance. To solve the problem, a merging algorithm is used at buffer deallocation time. The algorithm checks with neighbor regions of the current buffer and merges with them if they are free. This algorithm ensures contiguous buffer allocation for page requests. Its simplicity incurs little overhead.

4.2.3 Event Based Asynchronous Communication

The client side performs asynchronous communication using two threads. One thread is in charge of sending requests to servers as soon as they are issued by the kernel. The other thread is in charge of receiving replies from servers. The receiver works in a bursty manner. It sleeps until a receive completion event is triggered. When it wakes up, it processes all the replies that are available and goes back to sleep for the next event. By this way, the overhead of repetitive event triggering for clustered replies is avoided.

The server works in a similar way. It processes requests and issues RDMA operations asynchronously. When all outstanding RDMA operations and replies are completed, the server goes to sleep after idling for 200 μsec.

4.2.4 Flow Control

The user-level networking[21] idea of InfiniBand requires pre-posted receive buffers for control messages. This introduces the problem of flow control, which is not an issue for TCP based design for its stream semantics.

Here we use a water-mark to represent available receive buffer credits. A client is allowed to send requests to servers only if the outstanding request number is less than a threshold (which means the water-mark is above it). If water-mark falls below, requests will be queued until credits are available.

4.2.5 Multiple Server Support

Multiple server support allows multiple nodes to export their memory for page store demands. It also enables a larger address space for the client that can be accommodated in the remote memory level of its memory hierarchy.

In multiple servers scenario, load balancing is a new design issue. Data striping and request multiplexing are traditional ways to exploit parallelism. In our case, it could potentially be used to reduce the impact of copy cost at the server side. But the 128K bound of a single request size limits the benefit of such parallelism, as for a single outstanding RDMA operation, multiple *memcpy* can be completed with the overlap capability provided. Also because of the high bandwidth feature of InfiniBand, splitting a single request to multiple ones may offset the benefit as well. Thus we choose non-striping scheme in our design, and distribute the swap area across the servers in a blocking pattern.

5. Implementation

In this section, we discuss the implementation details of the HPBD client driver and the server program.

In the HPBD client driver, we associate each minor device an IBA context, which contains the IBA communication specific information, such as HCA information, completion queue, shared registered memory pool and queue pair arrays. The completion queues are shared among queue pairs connecting different servers. A socket interface is created at the initialization phase for queue pair information exchange. Each device maintains a request queue, which contains the outstanding requests to servers. A single request in the queue may represent multiple physical requests to different servers depending on the address range and size of the request. A request is successfully served when each physical request is replied with successful acknowledgment.

To implement the event based asynchronous handling of replies, an event handler is associated with the receive completion queue using the VAPI interface EVAPI_set_comp_eventh. The handler, when invoked, can wake up the reply processing kernel thread. To support this mechanism, the server needs to set the solicitation control

field of the send descriptor, thus the HCA driver at the client can execute the correspondent handler.

The mutual exclusion of accesses to request queue and pre-registered memory pool are ensured using locking mechanisms.

The server is a typical daemon program. It is able to serve multiple clients using different swap areas. For each server process, receive queue is checked periodically for requests. The requests are served by manipulating RamDisk based files and RDMA operations. RDMA operation completions are checked asynchronously to support *memcpy* overlap. Finally, a timer is used to count the server idle time. The server yields the CPU after idling for 200 μsec, a similar VAPI interface described above is used to wake up the server and notify the new incoming requests.

6. Performance Experiments and Evaluation

6.1. Experiment Setup

The experiments are conducted on a cluster of dual Intel Xeon 2.66 GHz nodes. Each node has 512 KB L2 cache, 2GB physical memory and PCI-X 133 MHz bus. All nodes are connected to InfiniBand network using Mellanox 144-port switch (MTS 14400) and Mellanox MT23108 HCA. Each node has a 40GB ST340014A ATA/ATAPI-6 hard disk. The operating system is RedHat 9.0 Linux.

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 and vary the swapping devices.

Two testing scenarios are used in our evaluation. In each scenario, we test with "enough" local memory, swapping over HPBD, NBD and local disk. We use the performance of applications running "in-memory" as the baseline for evaluation. For NBD, only one server case is reported, since as of Linux-2.4 kernel, a single NBD device can only be served by a single remote server directly.

For both test scenarios, we use all of the 2GB memory physically available for local memory performance test.

- Single server test swap area setup

  In this scenario, we set the local memory size as 512MB and 1GB RamDisk at remote server as swap area. We use this setup both for micro-benchmark and application benchmarks.

- Multiple server test swap area setup

  We use this scenario for application benchmark tests, we set the local memory size as 512MB. For single application execution instance, the 512MB total swap area is evenly distributed among the servers; for multiple application execution instances, each memory server is configured with 512MB swap area.

We use 3 different test programs. One is a micro-benchmark "testswap", which allocates a 1GB array and sequentially write integers into this array. Second is an implementation of a quick-sort algorithm[20], which sorts 256M random generated integers, whose data set is around 1GB on our IA-32 platform as well. We choose this application, because quick-sorting is a frequently used algorithm for various applications. Improvements for sorting over HPBD may provide a ground for benefits of other applications. Another application is "Barnes", which is an application in the Stanford SPLASH-2 suite [22]. It implements the Barnes-Hut method to simulate the interaction of a system of bodies. We simulate the interaction between 2097152 bodies. For this configuration, the memory usage of this application incrementally increases with a largest size of 516MB observed. For each of the tests, we run these applications multiple times and report the average performance number.

6.2 Micro-benchmark Performance Results

As shown in Figure 5, the execution time of testswap in local memory is around 5.8 seconds; HPBD is 8.4 seconds. Thus local memory is only 1.45 times faster than HPBD, while HPBD is 2.2 times faster than the disk. At the same time HPBD performs 1.45 times better than GigE, and 1.29 times better than IPoIB.

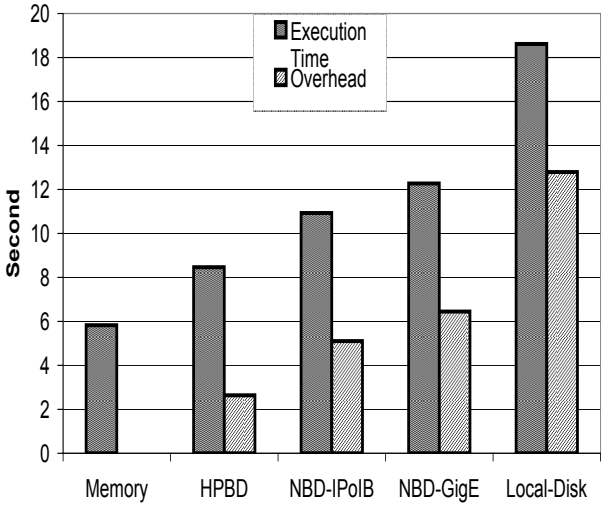

Figure 5. Testswap Execution Time

These results show that network performance has a significant impact on the remote pager and that as network speed approaches what the memory system can deliver, the host overhead along the path for swapping becomes an important performance factor. The performance improvement

of HPBD over IPoIB by 1.29 times shows that simply using TCP/IP protocol over high performance network can not benefit from the low latency feature efficiently. And TCP/IP stack processing becomes an important overhead on the critical path.

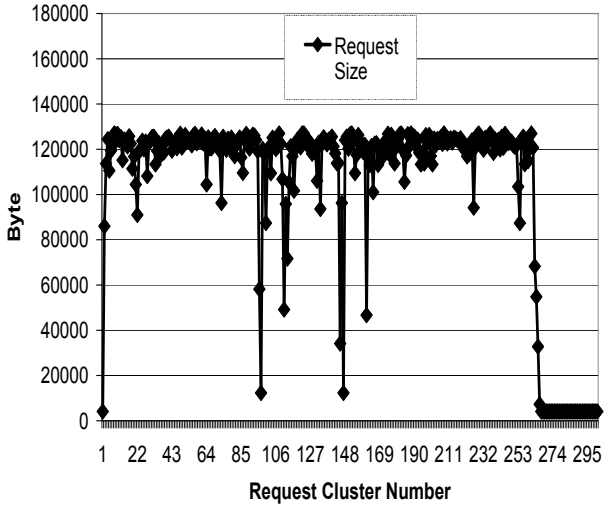

Figure 6. Testswap Average Request Size for Each Request Cluster

Since NBD-IPoIB and NBD-GigE follow identical code path above the IP protocol layer, and according to our profiling for "testswap" shown in Figure 6, "testswap" involves mostly with messages around 120K. By applying Amdahl's law, we find out that network overhead is about 48% percent of the overhead of GigE and only 34.5% for IPoIB.

We can not make a direct comparison with HPBD using Amdahl's law, because: a) HPBD does not go through the TCP/IP stack, which means the host overhead for network processing is less; b) HPBD does some optimization to overlap the copy overhead and the RDMA time at the server side, while NBD simply uses blocking mode transfer for each request and response. Due to the asynchrony of different components in the system, such as page prefetching and flushing, an accurate measurement of the network latency on the critical path is not possible without thorough analysis of the swapping mechanism of the kernel and each run case, which is beyond the scope of this paper. But a rough estimate with Amdahl's law would show that with HPBD, the network cost is less than 30%, thus host overhead is more dominant for the performance.

6.3 Application Performance Results

In this section, we present the performance result for application tests.

6.3.1 Single Server Performance

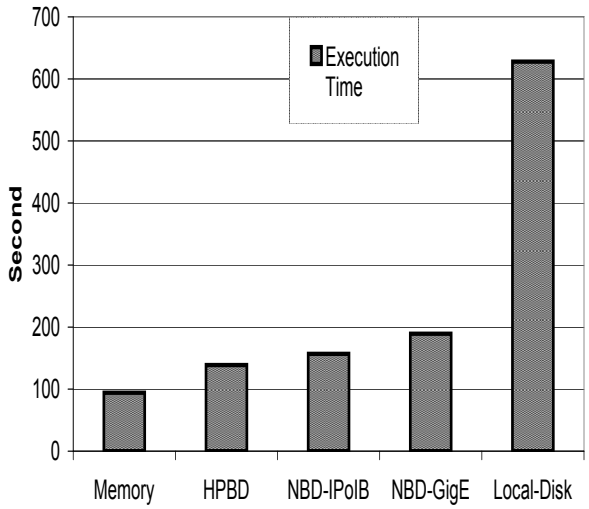

Figure 7. Quick Sort Execution Time

For quick sort test, as shown in Figure 7, local memory execution time is 94 seconds, while HPBD delivers 138 seconds. Thus memory is only 1.47 times faster than HPBD, and HPBD is 4.5 times faster than local disk. Among different remote pagers, HPBD is 1.36 times faster than NBD GigE and 1.13 times faster than IPoIB.

For Barnes shown in Figure 8, similar trends are observed. Since Barnes does not perform an intensive swapping activity as quick sort for its relatively small memory usage, the improvement is less evident.

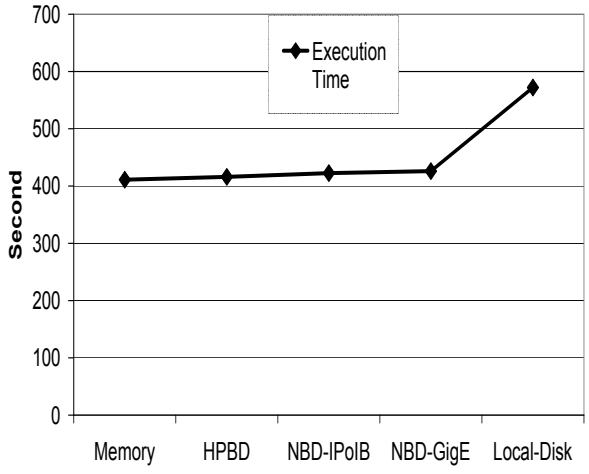

Figure 8. Barnes Execution Time

6.3.2 Multiple Server Performance

Multiple server support allows applications to take advantages of more idle memory from multiple servers. This is

important when memory resources are scarce on a compute node and contention is intensive. Figure 9 shows the case, where two quick sort applications sorting 256M integers respectively are run on a single node in a dual processor system, thus CPU contention is not an issue. It shows that with HPBD, both applications are able to give reasonable performance compared with the 2GB local memory case. When 50% of local memory is available, HPBD performs only 1.7 times slower, when only 25% of local memory is available, HPBD performs 2.5 times slower. While with only disk paging, the execution time is tremendously high, which is 36 times of local memory case.

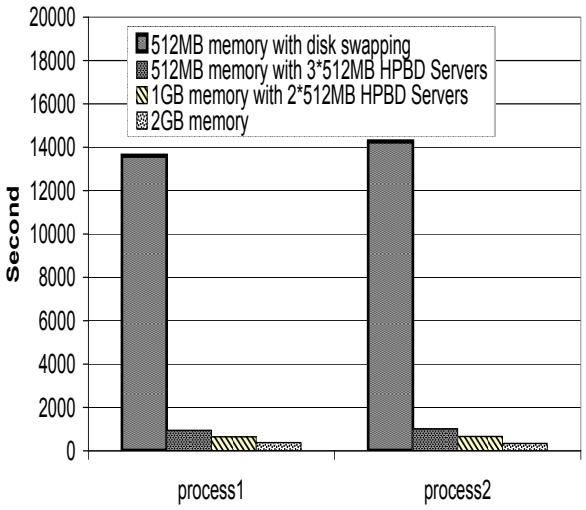

Figure 9. Quick Sort Execution Time for Two Concurrent Execution Instances

Dealing with multiple servers involves some extra overhead to maintain multiple connections. Figure 10 presents the execution time up to 16 servers for quick sort. The number shows HPBD performs similarly up to 8 servers. For 16 nodes server there is some degradation. This is due to the HCA design for multiple queue pair processing.

7. Conclusion and Future Work

In this paper, we study the design issues involved in utilizing remote memory in InfiniBand based high-performance clusters. We take the approach of remote memory paging to enhance local memory hierarchy, and analyze the pros and cons between a kernel level design and a user level design. We propose HPBD, a high performance network block device for InfiniBand networks and implement it as a kernel module for Linux 2.4. Our experiment results show that using HPBD for remote paging, quick sort runs only 1.45 times slower than local memory system, and up to 21 times faster than swapping using local disk. We

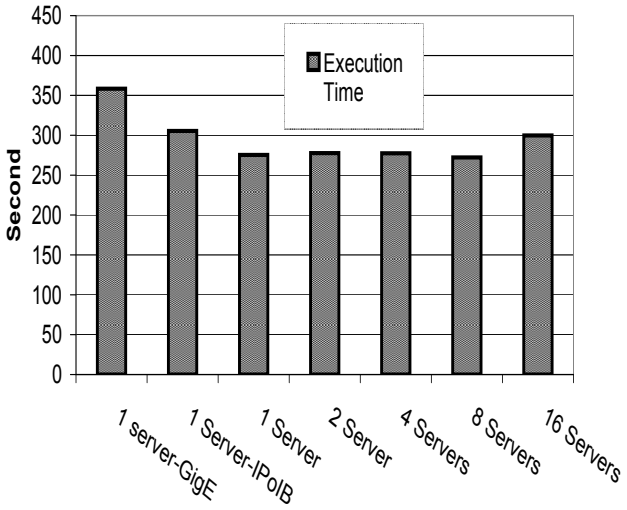

Figure 10. Quick Sort Execution Time With Multiple Servers

also identify that host overhead is a key issue for further performance improvement for remote paging over high performance interconnects clusters.

In our future work, we plan to investigate ways of minimizing host overhead on the swapping critical path and enable HPBD to utilize cluster wise idle memory in a dynamic and cooperative manner. We also intend to investigate designs that can eliminate copy cost and fully utilize the *zero-copy* feature of RDMA operations.

References

[1] A. Acharya and S. Setia. Availability and utility of idle memory in workstation clusters. In *Proceedings of the 1999 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 35–46, 1999.

[2] A. Barak and A. Braverman. Memory ushering in a scalable computing cluster. In *Proceedings of IEEE Third International Conference on Algorithms and Architecture for Parallel Processing*, 1997.

[3] A. Barak, S. Guday, and R. G. Wheeler. *The MOSIX Distributed Operating System: Load Balancing for UNIX*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1993.

[4] M. Dahlin, R. Wang, T. E. Anderson, and D. A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proceeding of the First Symposium on Operating Systems Design and Implementation*, pages 267–280, 1994.

[5] E. Anderson and J. Neefe. An Exploration of Network RAM. Technical Report CSD-98-1000, UC Berkley, 1998.

[6] E. Felten and J. Zahorjan. Issues in the implementation of a remote memory paging system. Technical Report 91-03-09, University of Washington, 1991.

[7] Fabrizio Petrini and Eitan Frachtenberg and Adolfy Hoisie and Salvador Coll. Performance Evaluation of the Quadrics Interconnection Network. *Journal of Cluster Computing*, 6(2):125–142, April 2003.

[8] M. J. Feeley, W. E. Morgan, E. P. Pighin, A. R. Karlin, H. M. Levy, and C. A. Thekkath. Implementing global memory management in a workstation cluster. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 201–212, 1995.

[9] InfiniBand Trade Association. The InfiniBand Architecture. http://www.infinibandta.org/specs.

[10] K. Kim, J.-S. Kim, and S.-I. Jung. GNBD/VIA: A Network Block Device over Virtual Interface Architecture on Linux. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 163, 2002.

[11] 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.

[12] 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.

[13] M. Flouris and E. Markatos. The Network RamDisk: Using remote memory on heterogeneous NOWs. *Journal of Cluster Computing*, 2(4):281–293, 1999.

[14] P. Machek. Network Block Device (TCP version). http://nbd.sourceforge.net/.

[15] E. Markatos and G. Dramitinos. Implementation of a Reliable Remote Memory Pager. In *Proceedings of the USENIX 1996 Annual Technical Conference*, pages 177–190, 1996.

[16] Mellanox Technologies. Mellanox VAPI Interface, July 2002.

[17] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W. Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, 1995.

[18] J. Oleszkiewicz, L. Xiao, and Y. Liu. Parallel Network RAM: Effectively Utilizing Global Cluster Memory for Large Data-Intensive Parallel Programs. In *Proceedings of the 33rd International Conference on Parallel Processing*, pages 353–360, 2004.

[19] S. R. Soltis, T. M. Ruwart, and M. T. O'Keefe. The Global File System. In *Proceedings of the Fifth NASA Goddard Conference on Mass Storage Systems*.

[20] R. L. R. Thomas H. Cormen, Charles E. Leiserson and C. Stein. *Introduction to Algorithms*. The MIT Press, 55 Hayward Street,Cambridge, MA 02142-1315, 2001.

[21] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 40–53, 1995.

[22] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22th International Symposium on Computer Architecture*, pages 24–36, Santa Margherita Ligure, Italy, 1995.

[23] L. Xiao, X. Zhang, and S. A. Kubricht. Incorporating Job Migration and Network RAM to Share Cluster Memory Resources. In *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing*, page 71, 2000.