Exploiting Remote Memory in InfiniBand Clusters using a High Performance Network Block Device (HPBD)

SHUANG LIANG, RANJIT NORONHA, AND DHABALESWAR K. PANDA

Technical Report
OSU-CISRC-5/05-TR36

Exploiting Remote Memory in InfiniBand Clusters using a High Performance Network Block Device (HPBD)*

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, remote memory accesses in cluster systems are very expensive operations, which perform 20-100 times 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 made remote memory much closer to the local memory system. Using remote idle memory to enhance local memory hierarchy thus becomes an attractive choice, especially for data intensive applications in cluster environment. In this paper, we take 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), 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.

1 Introduction

Moore's law dictates that the computing power of a modern CPU doubles approximately every 18 months. Similar trends apply to the capacity of modern memory and disk systems. This allows modern systems to quickly process large amounts of data "in-memory", allowing for increased throughput. It also allows application developers to design and implement algorithms previously considered impractical to exploit the resource rich nature of these systems.

*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.

However, even with the dramatic increase in memory capacities, modern applications have been quickly keeping pace with and even exceeding the resources of these systems. For example, modern databases typically maintain millions or even billions of records and are ever increasing. To keep the working set in memory for database operations demands a high volume of memory space, and may exceed what the computer systems can provide. Swapping to disk in these cases may severely impinge on the performance of these systems.

Modern networking technologies like InfiniBand, 10 GigE, Myrinet, and Quadrics [6, 14, 16] provide improved performance to the end-application users. This is both in terms of low-latency of a few micro-seconds and high throughput of up to 10 Gbps. In addition, they allow the user to use modern communication techniques like Remote Direct Memory Access (RDMA), atomic operations, and hardware multicast. RDMA allows access to a remote computer's memory space without involvement from the remote computer's CPU. This makes it possible for the application developers to design efficient communication protocols.

With the low latencies of networking technologies, it is natural to ask whether we can take advantage of RDMA primitives to access remote memory efficiently and enhance the memory hierarchy. If yes, how can such a memory system be designed, so that its performance is close to that of main memory locally available to a processor? Additionally, how can such a system be designed so that the impact of remote memory accesses can have a minimal impact on the remote node?

In this paper, we take on the challenge of designing such a system. We aim at achieving the following goals:

- Design a modern remote memory system exploiting efficient low-latency high-bandwidth communication, which can deliver comparable performance to local memory system.

- Evaluate the different design trade-offs such as a kernel level design vs. a user level design, and study the network performance impact on our remote memory system.
- Enable applications to benefit from our systems transparently.

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 system. An experimental evaluation of the remote memory system is discussed in section 6. We conclude this paper in section 7.

2 Related Work

There have been several investigations into using remote memory for different purposes. These studies, as shown in Figure 1, may broadly be classified into *simulation* based or *implementations* based; *global resource management* or *loosely cooperative system*; *user-level* design or *kernel-level* design, and finally *User-Level Protocol(ULP)* or *TCP/IP* based.

The simulation based studies include the study of Job Migration and Network RAM (JMNRM) [22], Parallel Network RAM (PNR) [15], Cooperative Caching (COCA) [2] and Remote Paging System (RPS) [4]. In JMNRM and PNR, impact of combining network memory and job migration for improved system scalability and throughput are studied, and PNR is proposed to utilize global memory for parallel scientific programs. COCA focuses on global file caching to improve the file cache hit ratio and reduce response time. RPS proposed the idea of dynamically using remote workstation's memory for paging store, and a queue model is used to predict performance.

Studies on MOSIX [1] and the Global Memory Management (GMM) [5] address the problem of utilizing remote memory resources by global management from a kernel perspective. MOSIX is a software tool design for Unix-like operating system to migrate jobs transparently from heavily loaded servers to lightly loaded servers. It is implemented as a set of adaptive resource sharing algorithms in the kernel. GMM is a global memory system for OSF/1 workstation clusters. It is designed as a module to function together with the node's VM system, page-out daemon and unified buffer cache. S. Koussih et. al. studied utilizing cluster idle memory from a user level perspective and designed a run time system DoDo [8] for remote memory exploitation. It is implemented on top of a U-NET [19] communication architecture and provide a socket interface for portability.

Network RAM Disk (NRD) [10] and Reliable Remote Memory Pager (RRMP) [12] focus on reliability studies of remote memory utilization, either from the perspective of ramdisk or remote pager. E. Anderson and J. Neefe studied design issues of Network RAM and proposed a user-level signal handling based implementation [3] .

Another related work is GNBD/VIA [7], in which K. Kim et. al. proposed a kernel level socket interface over VIA KVIPL library for GNBD (A Network Block Device used by GFS [17]) to utilize the performance provided by VIA, a user-level protocol, for file system performance improvement. It focuses on using user-level networking for remote file transfer.

Our work is different from the previous work in that we focus on the study of network performance impact on utilizing remote memory for paging, and propose an optimized design to fully utilize the InfiniBand features such as RDMA operations and deliver a performance comparably to local memory systems. It is a kernel level design, and completely transparent to user applications.

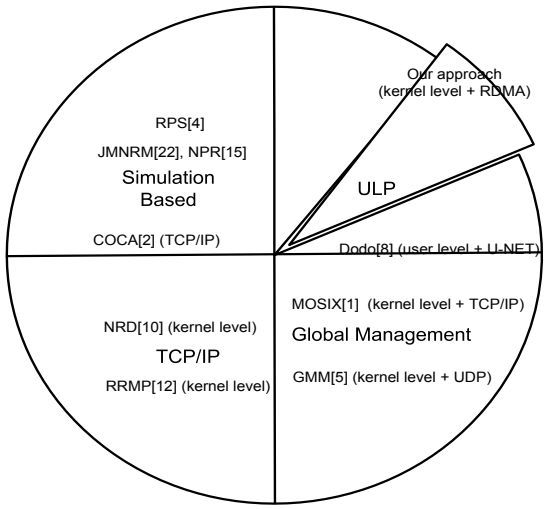

Figure 1. Modern work in designing remote memory system

3. Background

In this section, we provide the background of our work. Our work focuses on using low latency and high bandwidth network to design a remote memory system, so that it can perform comparable as local memory system. Before we present the design, an overview of InfiniBand, Linux swapping mechanism and Network Block Device is introduced.

3.1. Infiniband Overview

The InfiniBand Architecture (IBA) [6] is a specification designed for interconnecting processing nodes, I/O nodes and devices in a system area network. It defines a communication architecture from the switched network fabric to transport layer communication interface for inter-processor

communication and I/O. In an InfiniBand network, processing nodes and I/O nodes are connected to the fabric by Host Channel Adapters (HCA). HCAs expose a queue-pair based transport layer interface. The send queue keeps control information for outgoing messages, while the receive queue keeps descriptions for incoming message placement information. Communication requests are submitted to the queues through descriptors in a non-blocking mode. Completion of requests are reported through Completion Queues (CQs), which can be shared among different queue pairs. Memory registration for communication buffer is required for communication using InfiniBand. Memory registration pins down pages in the memory and registers them with the HCA, and thus can facilitate buffer address translation.

Several service levels are provided in InfiniBand to meet the needs of QOS. In this paper, we focus on RC (Reliable Connection based service) for both reliability and performance considerations.

For the communication operations, IBA supports both channel and memory semantics. Therefore, messages can be delivered using send/receive operations as well as RDMA operations. In this paper, we take advantage of these RDMA features for page transfer and offload the client for request management, thus improving the overall system performance.

InfiniBand also supports IP emulation IPoIB, with which IP based application can run directly over InfiniBand transparently. Figure 2 shows the basic latency numbers for *memcpy*, RDMA_WRITE operation, IPoIB and GigE. We can see that RDMA_WRITE latency is very close to *memcpy* latency, and provide the potential of significant performance improvement for remote paging.

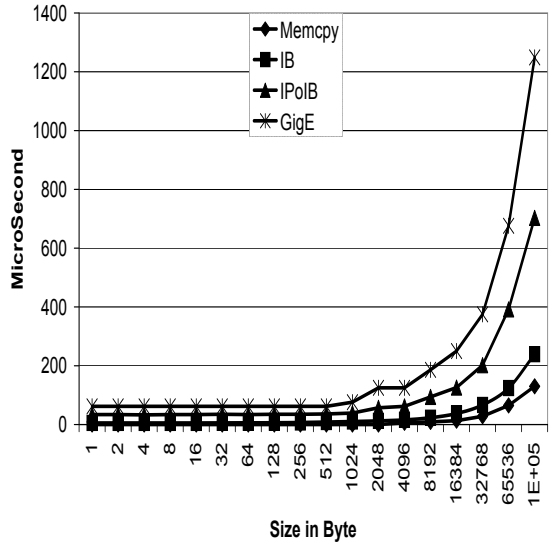

Figure 2. Latency Comparison of Different Networks and Memcpy Up to 128K

3.2. Linux Swapping Mechanism

Paging is an important part of the kernel's virtual memory system (VM). VM manages all system memory resources and sends out page-in requests to swap devices when there is a page fault. And when free pages available to VM fall below a threshold, page-out requests will be triggered by the kernel thread *kswapd*. Multiple prioritized swap devices are supported by the kernel. The kernel chooses to place the page-out data based on the priority of each swap device. The driver for the corresponding swap device serves the swap requests from kernel as normal I/O requests and deal with device specific operations.

Thus, designing a block device driver which supports I/O requests to/from remote nodes becomes a good choice for remote pager design. In this paper, we take this approach. It is completely transparent to applications, and can be beneficial to overall system performance. More details will be discussed in the section 4.1.

3.3. Network Block Device

Network Block Device [11] is a software abstraction of local block storage at the block device layer. The idea is to simulate a local device interface to the upper layer, while allocate and deallocate resources over the network for remote resource utilization. It is widely used in file system for data replication [17]. NBD is a Linux implementation of such device over TCP/IP using kernel-level socket interface for network communication; and it is available in the Linux source tree.

As motivated by the network performance impact studies on remote paging, we compare the performance of HPBD with NBD. We activate this device over GigE and IPoIB for our experiments. As of Linux-2.4 kernel, a single NBD device can only be served by a single remote server. Though we are able to use NBD as a swap device in our experiment, deadlock is reported [11] because of memory allocation problem in TCP networking.

4. Proposed Design

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

4.1. Design Alternatives

To implement a remote pager, several approaches could be considered. The device driver approach taken by this paper is introduced in section 3.2. Another approach is to modify the dynamic memory allocation library implementation [3, 8], and provide remote memory allocation capabilities for requests that can't be filled by local memory system. This approach is feasible, but several disadvantages are inherent for the design.

- Pages are still subject to disk paging by the underlying OS unless the relevant page-out requests in the kernel are intercepted and passed to the modified allocator, which usually runs at user space. By doing so, the allocator essentially has to implement part of the kernel's swapping mechanism.

- To implement the page-in request for the swapped out pages of the address space, memory pages must be protected and segmentation fault signal is used for page fault trapping. These are very expensive operations. On our testing platform, a single page fault operation with dummy fault handling incurs an overhead of 11 μsec.

- User space design is not completely application transparent. Either the application needs to use a new allocator explicitly in the program or it has to be compiled against a new library. Thus legacy applications can't run without recompilation.

With the device driver approach, we implement our driver as a kernel module. Thus, no modification to the OS is needed, and it can be easily ported to other Linux based systems. With this approach, the kernel will be in charge of making page-out decisions based on system wise page aging with an approximate LRU algorithm. Also, caching mechanism is in place to facilitate page sharing, swap buffering and race condition handling. Our experience shows that it is an efficient solution and is running successfully on both 32bit and 64bit systems. Figure 3 presents the system architecture of our remote paging system.

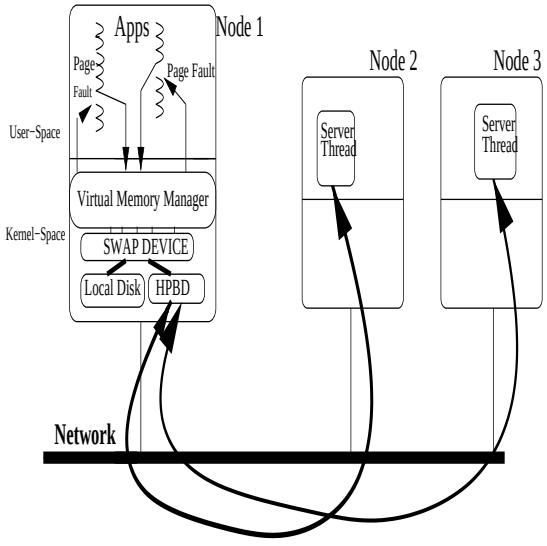

Figure 3. Remote Memory System Architecture

4.2. Design Challenges

With the device driver approach in mind, our design challenge is to propose a design that can leverage the InfiniBand technology for paging requests in kernel space that can deliver performance. Most work on InfiniBand designs provide solutions for user-space communication systems such as MPI and PVFS [9, 21]. Design of HPBD must address several additional issues for swap device kernel module design:

- Memory registration and buffer management: InfiniBand, as all the other high performance interconnects depends on Network Interface (NI) aware DMAable buffers to implement data transfer. In InfiniBand, communication buffers must be registered with the HCA before message passing can start. As shown in Figure 4, 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 to minimize the overhead. This can be done by a user application in a transparent way as in [9], which implements a *malloc* hook for pre-registered buffer allocation and deallocation. Though this method is viable with user space application, it can't be used directly for remote paging. Page requests can potentially come from anywhere in the paged memory system and be associated with any application's address space. Therefore, pre-registration is not feasible. While registration on-the-fly remains a choice, it will be very costly as shown in Figure. 4.

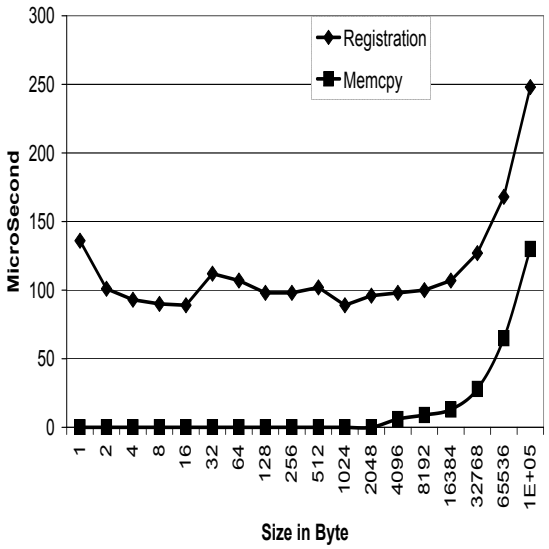

Figure 4. Memory Registration vs. Memcpy Cost

We design a pre-registered memory pool allocator and copy pages to/from this area for communication. As page swapping requests are within the range of 4K -

127K in size. And for most applications, the average request size is much smaller than 127K, thus the copy cost is much cheaper. As the memory pool is a shared resource, an appropriate size must be specified to avoid the bottleneck effect.

- Asynchronous communication: In a client-server architecture, the swapping process sends out paging requests to remote memory server and wait 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 feasible, as most OS are not preemptive in kernel mode. Asynchronous communication must be supported in this scenario. Also at the server side, asynchronous mode can significantly decrease the host CPU usage by allowing the server to sleep when no requests are coming.

- Thread safety: As a device driver, HPBD is shared among all the processes that need to use this device, thread safety must be ensured. As we take advantage of the thread safety feature of VAPI [13], 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: Because failure in swapping can easily crash the whole system. Thus reliability is an important issue for swap device design. In [4] and [10], several techniques such as mirroring and parity are studied for reliability issues. As we choose RC service as our network transport, it excludes most of the reliability issues from network. We do not focus on this issue for this paper.

4.3. Designing HPBD

HPBD is based on client-server architecture, as shown in Figure 3. It serves the kernel's paging requests by communicating with remote memory servers using native InfiniBand communication verbs. The client side is a block swap device driver, which servers I/O requests stream from the VM system. The server is a ramdisk based user space program, which provides local memory for paging store.

4.3.1 RDMA operations and Remote Server Design

As one of our motivations for this work, we design HPBD to show the performance benefit of protocol off-loading network. Thus performance impact of design choices is a major issue. We propose to use RDMA operations for performance optimizations, and support overlap between RDMA and memory-copy operations for request processing at the memory server.

In HPBD, there are two types of messages: control message and data message. Control messages are used to send paging request descriptions 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 5, A RDMA_READ operation is used for swap-out paging request to pull data out of the client, and RDMA_WRITE operation for swap-in request to push data into client. We support RDMA and *memcpy* overlap by allowing multiple outstanding RDMA operations, thus host CPU can be utilized for ramdisk operations while network transfers are in progress.

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 reason discussed in section 4.2, pre-registration is not a feasible solution. Thus, the server can't export memory address as a priori for client initiated RDMA operations.

4.3.2 Registration Cache Management

Registration cache is a pre-registered buffer pool, which is initialized at device load time. The default cache size is 1MB. Memory buffers are allocated by a first-fit algorithm.

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 designed to accommodate the allocation requests which can not be filled temporarily. Deallocation of data buffer will wake up any threads that is on 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, and 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, and its simplicity incurs little overhead.

4.3.3 Event Based Asynchronous Communication

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 bursting 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 will go to sleep after idling for 200 μsec.

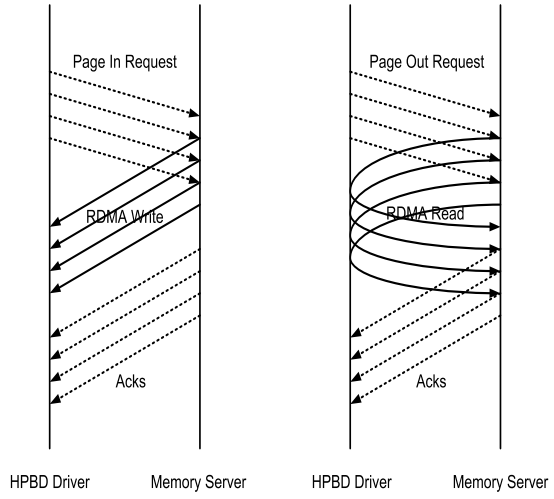

Figure 5. RDMA Design for Remote Memory Server

4.3.4 Flow control

Following the idea of user-level networking [19], InfiniBand is designed to realize zero-copy message passing. Such a design would require pre-posted receive buffers to make message send operation complete successfully. 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 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.

4.3.5 Multiple Server Support

Multiple server support is important to remote pager in a cluster environment, because only by allowing multiple nodes to export their memory, cluster-wise idle memory can be fully utilized. And from the perspective of client, multiple servers also enable a larger address space that can be accommodated in the remote memory level of its memory hierarchy.

In multiple servers scenario, load balancing is a new design issue. And using data striping or request multiplexing is a traditional way 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, since for a single outstanding RDMA operation, multiple *memcpy* can be completed with the overlap capability provided. Thus server bottleneck is not an issue in our case. 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 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 initialize 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 will be 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 will wake up a 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 invoke the handler correspondingly.

The request queue and pre-registered memory pool are shared resources among multiple instances of the driver and different queue pairs, thus mutual exclusion of accesses must be ensured.

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, and RDMA operations are issued accordingly. 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 200 μsec idle period, a similar VAPI interface described above is used to wake up the server and notify the new incoming request.

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 and 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 device.

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

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 512M and 1G ramdisk at remote server as swap area.

- multiple server test swap area setup

  In this scenario, we set the local memory size as 512M and 512M total swap area is evenly distributed among the servers. Here we also include the single server results for this configuration as comparison base.

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 a quick-sort algorithm [18], which sorts 256M random generated integers, whose data set is around 1G on our ia32 platform as well. We choose this application, because quick-sorting is a frequently used algorithm for various applications. Improvement for sorting over HPBD will provide a ground for benefits of other applications. Another application is "Barnes", which is an application in the Stanford Splash2 suite [20]. 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 516M observed. For each of the tests, we run these applications multiple times and report the average performance number.

6.2 Microbenchmark Performance Results and Analysis

In this section, we provide the microbenchmark testswap's performance results for single memory server test. A network overhead analysis is also presented.

As Figure 6 shows, 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.

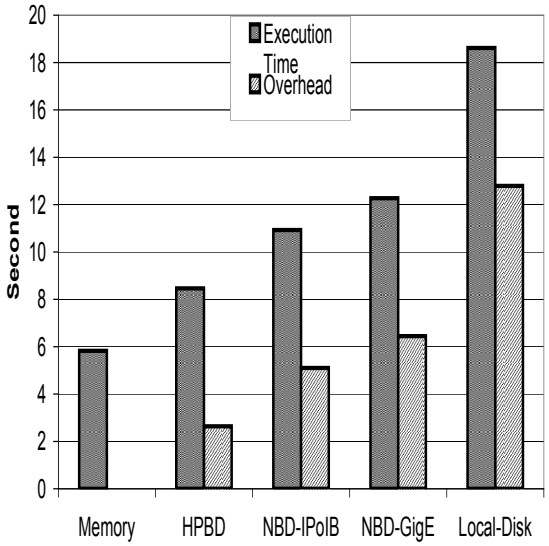

Figure 6. Testswap Execution Time

These results show that network performance has a significant impact on the remote pager, and HPBD performs the best among the three.

The results also show that as network speed approaches what the memory system can deliver, the host overhead along the path for swapping becomes an important performance factor. From Figure 6, we can see HPBD significantly reduce the extra overhead compared to IPoIB, which shows that simply using TCP/IP protocol over high performance network can not truly benefit from the latency features. And TCP/IP host overhead becomes an important overhead on the critical path.

Since NBD-IPoIB and NBD-GigE follow exactly the same code path above the IP protocol layer, and according to our profiling for testswap shown in Figure 7, "testswap" involves messages mostly of size around 120K. By applying

Amadahl's law, we find out that network overhead is only 48% percent of the overhead of GigE and 34.5% for IPoIB.

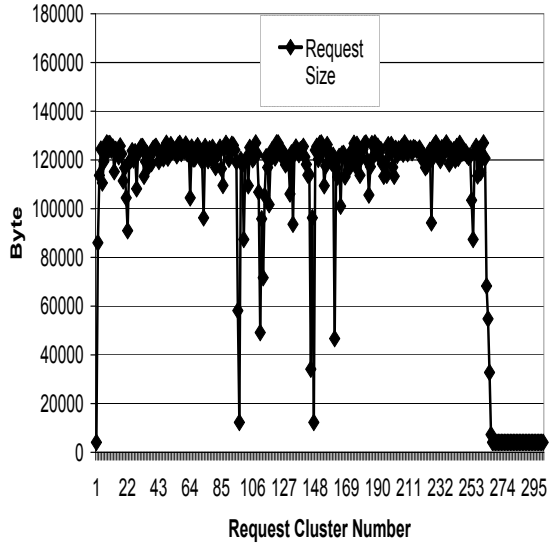

Figure 7. Testswap Average Request Size for Each Request Cluster

We can not make a direct comparison with HPBD using Amadahl's law, because of: 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 Linux asynchronous I/O for swapping and prefetching for pages, an accurate measurement of the network latency on the critical path is not possible either 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 Amadahl's law would show that with HPBD, the network cost is less than 30%, thus host overhead is much 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

For quick sort test, as shown in Figure 8, 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.

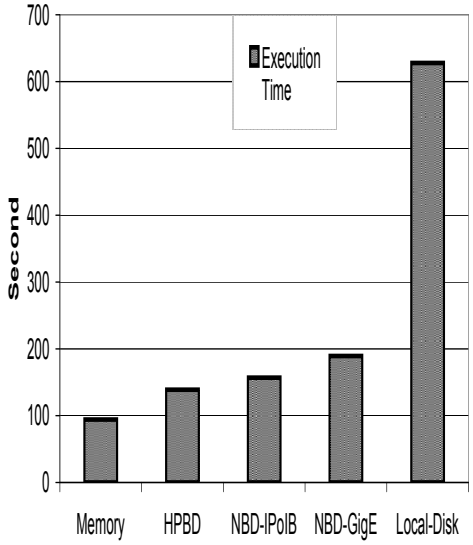

Figure 8. Quick Sort Execution Time

For Barnes shown in Figure 9, 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.

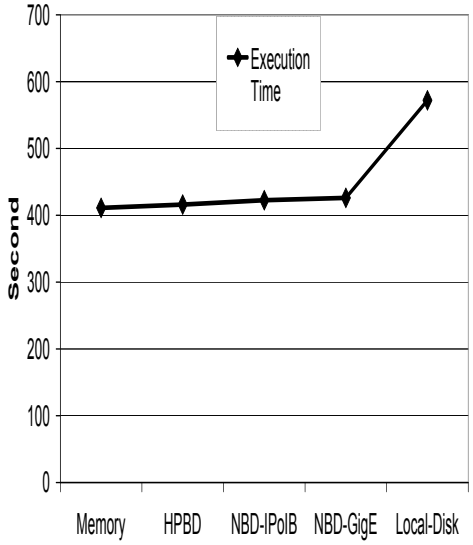

Figure 9. Barnes Execution Time

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 10 shows the case, where two quick sort applications sorting 256M integers respectively are run on a single node in a dual processor system, where CPU contention is not an issue. It shows

that with HPBD, both applications are able to give reasonable performance compared with the 2G local memory case. When 50% of local memory is avaiable, 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.

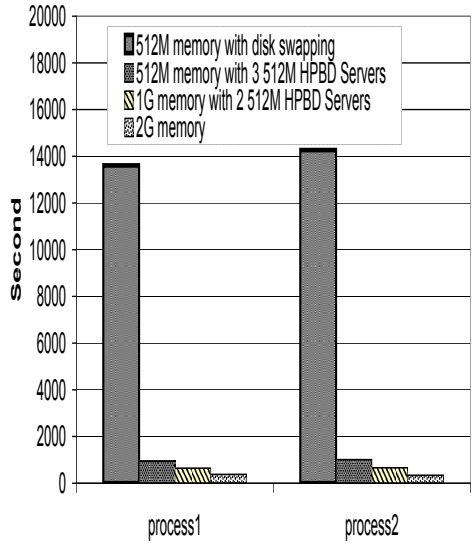

Figure 10. Quick Sort Execution Time for Two Concurrent Execution Instances

Dealing with multiple servers involves some extra overhead, since multiple connections have to be maintained. Figure 11 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 to use remote memory for InfiniBand based high-performance clusters. We analyze the pros and cons of different design alternatives such as kernel level design vs. user level design and compare different design trade offs. We propose HPBD, a high performance network block device, and present its design and implementation. Our experiments 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 also identify that host overhead is a key component 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.

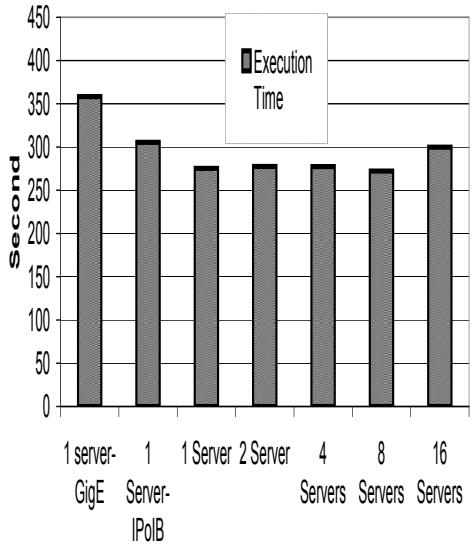

Figure 11. Quick Sort Execution Time With Multiple Servers

References

[1] A. Barak, S. Guday, and R. G. Wheeler. *The MOSIX Distributed Operating System: Load Balancing for UNIX*, volume 672. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1993.

[2] 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*, pages 267–280, 1994.

[3] E. Anderson and J. Neefe. An exploration of network RAM. Technical Report CSD-98-1000, UC Berkley, 1998.

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

[5] 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 *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 201–212, New York, NY, USA, 1995. ACM Press.

[6] Infiniband Trade Association. The infiniband architecture. http://www.infinibandta.org/specs.

[7] K. Kim, J.-S. Kim, and S.-I. Jung. Gnbd/via: A network block device over virtual interface architecture on linux. In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 163, Washington, DC, USA, 2002. IEEE Computer Society.

[8] S. Koussih, A. Acharya, and S. Setia. Dodo: A user-level system for exploiting idle memory in workstation clusters. In *Proc. of the Eighth IEEE Int'l Symp. on High Performance Distributed Computing (HPDC-8)*, 1999.

[9] 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.

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

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

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

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

[14] Myricom, Inc. Myrinet Products Specifications. http://www.myri.com/myrinet.

[15] 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. IEEE Computer Society, 2004.

[16] Quadrics Ltd. Quadrics Documentation. http://www.quadrics.com.

[17] 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*, pages 319–342, College Park, MD, 1996. IEEE Computer Society Press.

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

[19] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-net: A user-level network interface for parallel and distributed computing. In *SOSP*, pages 40–53, 1995.

[20] 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.

[21] J. Wu, P. Wyckoff, and D. K. Panda. Pvfs over infiniband: Design and performance evaluation. In *ICPP*, pages 125–132, 2003.

[22] L. Xiao, X. Zhang, and S. A. Kubricht. Incorporating job migration and network ram to share cluster memory resources. In *HPDC '00: Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC'00)*, page 71, Washington, DC, USA, 2000. IEEE Computer Society.