PVFS over InfiniBand: Design and Performance Evaluation [*]

Jiesheng Wu[†] Pete Wyckoff[‡] Dhabaleswar Panda[†]

[†]Computer and Information Science
The Ohio State University
Columbus, OH 43210
{wuj, panda}@cis.ohio-state.edu

[‡]Ohio Supercomputer Center
1224 Kinnear Road
Columbus, OH 43212
pw@osc.edu

Abstract

I/O is quickly emerging as the main bottleneck limiting performance in modern day clusters. The need for scalable parallel I/O and file systems is becoming more and more urgent. In this paper, we examine the feasibility of leveraging InfiniBand technology to improve I/O performance and scalability of cluster file systems. We use Parallel Virtual File System (PVFS) as a basis for exploring these features.

In this paper, we design and implement a PVFS version on InfiniBand by taking advantage of InfiniBand features and resolving many challenging issues. We design the following: a transport layer customized for PVFS by trading transparency and generality for performance; buffer management for flow control, dynamic and fair buffer sharing, and efficient memory registration and deregistration.

Compared to a PVFS implementation over standard TCP/IP on the same InfiniBand network, our implementation offers three times the bandwidth if workloads are not disk-bound and 40% improvement in bandwidth in the disk-bound case. Client CPU utilization is reduced to 1.5% from 91% on TCP/IP. To the best of our knowledge, this is the first design, implementation and evaluation of PVFS over InfiniBand. The research results demonstrate how to design high performance parallel file systems on next generation clusters with InfiniBand.

1. Introduction

In modern day clusters, I/O is quickly emerging as the main bottleneck limiting performance. The need for scalable parallel I/O and file systems is becoming more and more urgent. As well, the use of standards in the hardware components and in the software used in the cluster systems is also becoming not just convenient but a necessity to ensure software reuse.

There has been a significant amount of work on parallel and cluster file systems, which has repeatedly demonstrated that a viable infrastructure consists of *commodity storage units connected with commodity network technologies*, to provide high performance and scalable I/O support in cluster systems [2, 4, 18, 12, 17, 21, 22]. The PVFS (Parallel Virtual File System) [4] is a good example of such an architecture and a leading cluster file system for parallel computing in cluster systems. It addresses the need of high performance I/O on low-cost Linux clusters.

However, the performance of network storage systems is often limited by overheads in the I/O path, such as memory copying, network access costs, and protocol overhead [1, 11, 15]. Emerging network architectures such as InfiniBand Architecture [9] create an opportunity to address these issues without changing fundamental principles of production operating systems. Two common features shared by these networks are: *user-level networking* and *remote direct memory access* (RDMA).

InfiniBand has been recently standardized by industry to design next generation high-end clusters for both data-center and high performance computing. In this paper, we examine the feasibility of leveraging InfiniBand technology to improve I/O performance and scalability of cluster file systems. We use PVFS as a basis for exploring these features and focus on a number of challenging issues that are important for cluster file systems, including PVFS software architecture which can take full advantage of InfiniBand features, efficient transport layer to support PVFS protocols, and buffer management. We implement PVFS over InfiniBand by taking advantage of user-level networking and RDMA. We evaluate our implementation using PVFS and MPI-IO benchmarks and applications. We compare its performance with that of unmodified PVFS over IBNice [13], a TCP/IP implementation on InfiniBand.

This work contains several research contributions. Primarily, it takes the first step toward understanding the role of the InfiniBand architecture in next-generation cluster file systems. Our research shows that:

1. The capabilities of InfiniBand user-level communication and RDMA can improve all performance aspects of PVFS, including bandwidth, access time, and CPU utilization.
2. A transport layer based on InfiniBand user-level programming interface requires careful design regarding aspects of communication strategy selection and vari-

[*]This research is supported in part by Sandia National Laboratory's contract #30505, Department of Energy's Grant #DE-FC02-01ER25506, and National Science Foundation's grants #EIA-9986052 and #CCR-0204429.

ous optimizations in itself and interactions with other software components.

3. Memory registration and deregistration for networks with remote DMA capabilities adds a new dimension to transport issues for I/O intensive applications. They pose challenges on cluster file systems and require careful management of buffer resources.
4. Compared to a PVFS implementation over TCP/IP on the InfiniBand network, our implementation offers a factor of three improvement in throughput. CPU utilization decreases from 91% with IBNice to 1.5% in our native implementation.

The rest of the paper is organized as follows. We first give a brief overview on InfiniBand in section 2. Section 3 presents the architecture of PVFS over InfiniBand. Sections 4 and 5 describe the design of the PVFS transport layer and buffer manager over InfiniBand, respectively. The performance results are presented in section 6. Finally we examine related work in section 7 and draw our conclusions and discuss future work in section 8.

2. Overview of InfiniBand

The InfiniBand Architecture (IBA) [9] defines a System Area Network (SAN) for interconnecting both processing nodes and I/O nodes. It provides a communication and management infrastructure for inter-processor communication and I/O. InfiniBand Architecture has built-in QoS mechanisms which provide virtual lanes on each link and define service levels for individual packets.

A queue-based transport layer is provided in IBA. A Queue Pair (QP) consists of two queues: a send queue and a receive queue. The completion of requests is reported through Completion Queues (CQs). Both channel and memory semantics are supported in the IBA transport layer. In channel semantics, send/receive operations are used for communication. A receiver must explicitly post a descriptor to receive messages in advance. In memory semantics, RDMA write and RDMA read operations are used.

3. Proposed PVFS Architecture

In this section, we first give a brief overview of PVFS. Then we define a general software architecture of PVFS based on InfiniBand.

3.1. PVFS Overview

PVFS is a leading parallel file system for Linux cluster systems. It was designed to meet increasing I/O demands of parallel applications in cluster systems. In PVFS, a number of nodes in a cluster system can be configured as I/O servers and one of them is also configured to be the metadata manager. It is possible for a node to host computations while serving as an I/O node.

PVFS stripes files across a set of I/O server nodes to achieve parallel accesses and aggregate performance. PVFS uses the native file system on the I/O servers to store individual file stripes. An I/O daemon runs on each I/O node and services requests from compute nodes, particularly read and write requests. Thus, data is transferred directly between I/O servers and compute nodes. More details about PVFS can be found in [4].

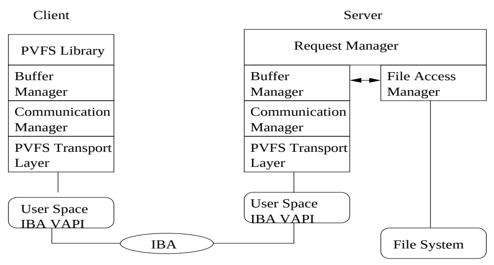


Figure 1. Proposed PVFS Software Architecture on InfiniBand Network.

3.2. Proposed PVFS Software Architecture

The orginal PVFS was designed over TCP/IP in a monolithic manner. Sockets are used for transfering messages. TCP/IP stream semantics is taken into account to avoid any buffer management. Since there are significant differences in both semantics and functionality between sockets and the IBA user-level interface, we believe a modular architecture is helpful to better address design issues and to achieve an efficient implementation.

Figure 1 shows our proposed PVFS software architecture over the InfiniBand network. Since the metadata server is a simpler case of the I/O server, we only show the architecture of the client and the I/O server here.

There are six modules in the PVFS architecture. A buffer manager, a communication manager, and a PVFS transport layer reside on both the client and server sides. The PVFS library is used by the client to generate requests. A request manager and a file access manager exist on the server side to process client requests.

The transport layer transfers data using user-level InfiniBand primitives. The buffer manager supplies the transport layer buffers and also supplies buffers to the file access manager for file accesses. The request manager receives requests and decides in what order to service requests, using information supplied by the file access manager. The communication manager chooses communication mechanisms and schedules data transfers.

InfiniBand network offers much more flexible design space for PVFS compared to other networks. Communication manager is responsible for choosing an appropriate communication mechanism for each message. It also schedules data communication to reduce network congestion and avoid delaying other traffic in the network. It is capable of applying a service level to each message which marks its priority as it moves through the network.

In this paper, we focus on the transport layer and buffer manager, which become more complicated when designing PVFS over InfiniBand as compared to the original design of PVFS over TCP/IP. Communication manager is also unique over InfiniBand, however, due to the space limitation, we do not cover it in details in this paper.

4. Designing PVFS Transport Layer

The PVFS transport layer provides data, metadata, and control channels between PVFS compute nodes, I/O server nodes, and the metadata manager. In this section, we first analyze the characteristics of various types of messages in

PVFS. Second, we make appropriate communication strategy selection for them, including communication choices, message transfer mechanisms and event handling. Then we propose optimized small data transfers and pipelined bulk data transfers to further optimize the PVFS transport layer.

4.1. Messages and Buffers in PVFS

Messages in PVFS can be categorized as *request messages, reply messages, data messages, and control messages*. A request message is sent by the compute node to the server (I/O server node or the metadata manager server) to direct it to initiate operations such as read, write, and lookup. The manager node also uses a request message to inform the I/O server node of metadata management operations if needed. A reply message is sent by a server to inform the request initiator of completion of a request. Data messages are used to transfer payload for file reads and writes. Control messages are internal messages in the PVFS system, such as flow control messages.

There are two types of buffers: *Internal buffers and RDMA buffers*. Internal buffers are allocated by the PVFS system. They are pinned when a connection is established and remain active for a long period of time. On the servers they can be used to service multiple clients. RDMA buffers are used to achieve zero-copy data transfer between the compute nodes and the I/O server nodes. On the client side, RDMA buffers are provided by the application when it initiates read and write operations. On the I/O server side, RDMA buffers are allocated to stage data in memory before it moves to the disk or to the network.

4.2. Communication Choices

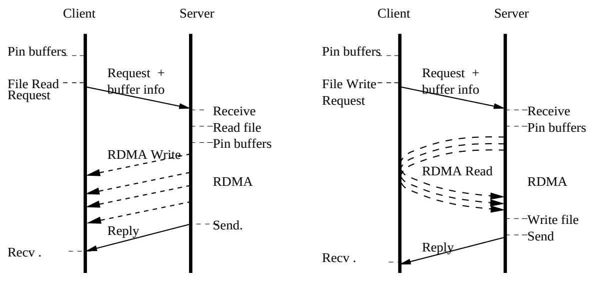
InfiniBand provides both reliable and unreliable connection and datagram services. Since PVFS requires a reliable transport layer, we focus only on the reliable connection service.

In reliable connection service, InfiniBand offers Send/Recv operations and both read and write RDMA operations. For each operation, the initiator can choose whether to generate a completion event or not. Send/Recv operations and RDMA Write with Immediate data operations consume receive descriptors and result in Solicited or Unsolicited completion on the receive side [9]. These features provide a flexible design space and the opportunity to optimize performance. Design choices should be made to achieve a better fit for particular message types according to how well they align with the characteristics of the corresponding communication operations.

We choose send/recv operations for request, reply, and control messages. Details about this choice can be found in [20]. For data messages, the decision pertaining whether to use RDMA Write or Read is also critical and discussed in section 4.3. For small data messages, a tradeoff can be made between the use of zero-copy RDMA data transfers and non zero-copy transfers. We discuss the details of this choice in section 4.5.

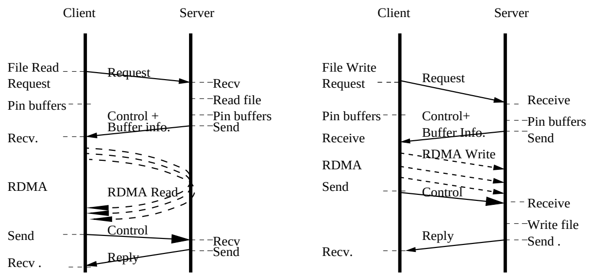
4.3. Message Transfer Mechanisms

There are four basic transfer mechanisms for PVFS messages: *Send/Recv*, *server-based RDMA*, *client-based RDMA*, and *hybrid RDMA*. We elaborate these mechanisms below and show how to map PVFS operations to them.



(a) PVFS read (b) PVFS write

Figure 2. Server-based RDMA Mechanism.



(a) PVFS read (b) PVFS write

Figure 3. Client-based RDMA Mechanism.

In Send/Recv mechanism, messages are sent from send internal buffers to receive internal buffers. Request and control messages are sent by this mechanism. Data messages also can be sent using this mechanism, at the cost of some memory copies. Flow control issues related to Send/Recv message transfer are described in section 5.1.

In server-based RDMA mechanism, RDMA operations are initiated only by the I/O servers. The clients are responsible for providing RDMA buffer information. Figures 2(a) and 2(b) show the operations involved in read and write transfers, respectively. Since client RDMA buffer information can be provided along with the request messages, the I/O servers can initiate RDMA operations asynchronously according to when they can be scheduled.

Figures 3(a) and 3(b) show the operations involved to perform reads and writes when initiated using RDMA operations from the client. Generally speaking, the client-based RDMA mechanisms require the server to send a control message containing its RDMA buffer information before data transfer can begin. It also requires that the client notify the servers when RDMA operations are finished. It can be seen that more control messages are usually needed in the client-based RDMA mechanism, compared to the server-based RDMA mechanism.

RDMA read is a round-trip operation and its performance is usually lower than that of RDMA Write. The details of RDMA Write and Read performance comparison can be found in [20]. Therefore, one can consider a hybrid RDMA mechanism, wherein only RDMA Write operations are used. In the hybrid mechanism, a PVFS read is designed with server-based RDMA Write as shown in Figure 2(a) and a PVFS write is designed with client-based RDMA Write as shown in Figure 3(b).

4.4. Polling or Interrupt on Events

InfiniBand provides a single structure, Completion Queues (CQ), to notify and deliver events for a large number of connections. There are two basic methods to catch an event in a CQ. One is that applications explicitly poll the associated CQ. Another one is to invoke pre-registered event handlers to notify applications of events by interrupts. In this method, applications can sleep and relinquish CPU when waiting for an event.

Important goals when designing PVFS over InfiniBand are to minimize CPU overhead on the client side, minimize response latency for short transfers, and maximize throughput for large transfers. In our design, notification of completion of sending request messages on the client side is done using polling and notification of completion of incoming reply and control messages with interrupts. On the server side, all event notification is done with polling, as is appropriate for a dedicated machine.

4.5. Transport Layer Optimizations

We consider two schemes to optimize small data transfers: Inline and Fast RDMA Write. For bulk data transfers, pipelining communication and I/O is also considered.

4.5.1 Inline Data Transfer

Zero-copy data transfers require that application buffers be registered before data transfer and may be deregistered after data transfer. For small data messages, the performance benefit of zero-copy transfer may not offset the cost of memory registration and deregistration. In *Inline data transfer* scheme, data is first copied into internal buffers which are pre-registered and then transferred by Send/Recv mechanism. If data can fit in an internal buffer with the request (for write) or the reply message (for read), they are sent in one message. This technique has been used elsewhere [6].

4.5.2 Fast RDMA Write

There is a significant performance difference between RDMA Read and RDMA Write when the transfer size is not large. This implies that using RDMA Write for small data transfers is preferable if the benefit can offset the overhead of doing so. *Fast RDMA Write* is mainly used to optimize PVFS write operations. However, it is also used to optimize PVFS read operations by avoiding application buffer registration and deregistration.

To optimize small writes, the client does RDMA Write to transfer data to the I/O server. However, as shown in Figure 3(b), two additional control messages are needed. To avoid the first control message, a small set of RDMA buffers (called *Fast RDMA buffers*) are allocated and registered when a connection is established. The buffer information is cached on the peer side. Thus, the client can RDMA write data directly into the Fast RDMA buffers on the server. We use RDMA Write with Immediate data to avoid the second control message.

4.5.3 Pipelined Bulk Data Transfer

There are two major phases in each I/O path: communication phase, where data is transfered between client buffers and server buffers, and I/O phase, where data is moved from server buffers to disk. Overlap between these two phases is necessary for high performance in the case of large write (or read) requests. One way to achieve communication and I/O overlap is to split large transfers into multiple smaller transfers. Pipelining communication and I/O also reduces memory pressure in I/O servers. The I/O server can use double buffering to service concurrent requests.

5. Designing Buffer Manager

A buffer manager provides buffers to the PVFS transport layer and the file access manager. Buffers are either internal buffers or RDMA buffers. There are three main tasks in a buffer manager. First, flow control on internal buffers is to ensure that every message sent by a Send operation has a receive buffer posted on the receiver side. Second, it should provide efficient memory registration and deregistration operations for RDMA buffers. Third, a buffer manager should provide fair and dynamic sharing to buffer consumers. This task is particularly important in the I/O server. We focus on these issues below.

5.1. Flow Control on Internal Buffers

Internal buffer management is a well-discussed issue in the literature. A small set of internal buffers are allocated and pinned on both sides of a connection. Each connection has a separate pool of internal receive buffers. To ensure that an incoming message can be put in an internal receive buffer, a credit-based flow control mechanism is deployed on a per-connection basis. At the beginning, some number of receive descriptors, each associated with an internal receive buffer, are posted for each connection. Then, the number of currently posted receive buffers is advertised by flow control updates, which can be piggybacked on other messages or sent as control messages. This information can also be exchanged implicitly in the flow of matched request and response message pairs.

5.2. Server RDMA Buffer Management

Server RDMA buffers are used to receive data from clients and to read data from files. These buffers are effectively used to bridge the performance gap between network and disk. Due to highly concurrent requests and possible large request sizes, a significant portion of the total memory must be allocated as RDMA buffers on a dedicated server. Clearly, the server can reuse these buffers for different requests. Thus, all these regions can be pre-registered at startup. The I/O server then keeps using them to service client requests. Other options of allocating and registering buffers, including a dynamic scheme, are discussed in [20]. Even with some dynamics, it can be expected that the frequency of memory registration and deregistration is low in the I/O server side. Thus, efficient memory registration and deregistration is not a huge issue.

The more important function for a server buffer manager is to provide a fair and dynamic buffer sharing among all clients. In the PVFS transport layer based on InfiniBand, data is transferred as whole messages, not as bytes in a stream. Buffers are also supplied explicitly. Message transfers are thus atomic, and data placement and data arrival are not separated as they are in TCP/IP. Therefore, unlike PVFS over TCP/IP, explicit buffer assignment is needed in PVFS over InfiniBand.

Another issue is that transfer sizes for requests could be different. This variability can offer better performance, while it requires that the buffer manager be able to supply different sizes of virtually contiguous buffers. Avoiding fragmentation is important in this scenario.

The server buffer manager in our design works as follows. First, all RDMA buffers are allocated and organized in zones, where each zone has a list of buffers of the same size. Given a particular transfer size, we first look at the corresponding zone list to try to get a contiguous buffer. If there is no buffer available, the buffer may be chosen from a bigger zone list. If there is no bigger buffer available, the transfer will be chopped into small transfers using smaller RDMA buffers. By this way, there is no dynamic fragmentation on RDMA buffers and it is usually possible to transfer data with a given transfer size.

5.3. Client RDMA Buffer Management

The client buffer manager is primarily responsible for efficient registration and deregistration of PVFS application memory regions. Memory registration and deregistration are expensive operations. Thus, they impact performance significantly when they are performed dynamically. On the other hand, PVFS I/O applications require a large number of I/O buffers which may be allocated no earlier than when the request is issued, it is not possible to pre-register all I/O buffers. Therefore, dynamic registration and deregistration are not easily avoided.

To reduce the cost of dynamic registration and deregistration, a pin-down cache [7] is incorporated in the buffer manager. Pin-down cache delays deregistration of registered buffers and caches their registration information. When these buffers are reused, their registration information can be retrieved from pin-down cache. This technique is quite effective when the amount of buffer reuse is high.

However, I/O intensive applications which PVFS mainly targets use a large number of different I/O buffers. The buffer reuse ratio may be low. This poses a challenge on approaches such as pin-down cache which work well only in the case where applications keep using a moderate number of buffers. In the next subsection, we propose a *two-level architecture* to support efficient memory registration and deregistration for I/O intensive applications.

5.4. Fast Memory Registration and Deregistration

Dynamic buffer registration is not avoided if applications keep using different buffers. To reduce its cost, InfiniBand software and adapters are expected to provide efficient registration operation. There are some optimization on buffer deregistration in the literature. Zhou *et al.* [22] demonstrated *batched deregistration* is an efficient way to reduce the average cost of deregistering memory for database applications.

We propose a two-level architecture: *pin-down cache plus Fast Memory Registration component (termed as FMR) and Deregistration component (termed as FMD)*. We refer to this two-level architecture as Fast Memory Registration

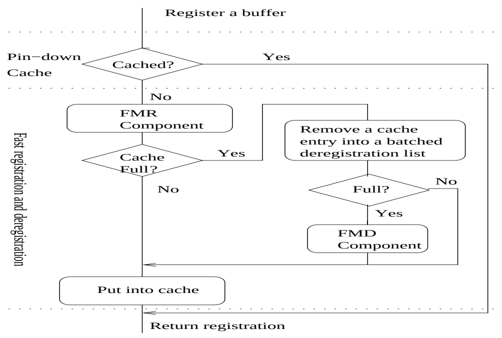


Figure 4. Fast Memory Registration and Deregistration (FMRD).

and Deregistration (*FMRD*) scheme in the rest of this paper. This architecture offers advantages from both pin-down cache and batched deregistration.

As shown in Figure 4, when a buffer is to be registered, first, it checks if its registration is cached; if yes, information is returned immediately. Otherwise, FMR is invoked to register the user buffer. The registration information is inserted into the cache. If there is no space left in the cache, one entry is evicted from the cache and put into a deregistration list. FMD is invoked to deregister all buffers in the deregistration list when the number of entries in the list reaches a threshold.

When a buffer is to be unregistered, only some information such as reference count of the buffer is modified in the cache. Real deregistration is delayed. Deregistration occurs later in a batched fashion during registration.

The fast memory registration component also takes advantage of Mellanox fast memory region registration extension in VAPI [14]. More details about this architecture are discussed in [20].

6. Performance Results

We have implemented PVFS on our InfiniBand testbed with designs described in Sections 4 and 5. Our implementation is based on PVFS version 1.5.6. The InfiniBand interface is VAPI [14], which is a user-level programming interface developed by Mellanox and compatible with the InfiniBand Verbs specification. This section presents performance results from a range of benchmarks on our implementation of PVFS over InfiniBand. First, we quantify that PVFS can take full advantages of InfiniBand features to achieve high throughput, low CPU utilization, and high scalability by comparing performance of our implementation with that of PVFS over IBNice [13], a TCP/IP implementation for InfiniBand. We use both PVFS and MPI-IO micro-benchmarks as well as applications to carry out the comparison. Then we quantify the impact of different buffer management schemes on performance. Due to space limitation, the impacts of optimizations in the transport layer on performance are not shown in this paper. Details are discussed in [20]. Unless stated otherwise, the unit megabytes (MB) in this paper is an abbreviation for 2^{20} bytes, or 1024×1024 bytes.

6.1. Experimental setup

Our experimental testbed consists of a cluster system consisting of 8 nodes built around SuperMicro SUPER P4DL6 motherboards which include 64-bit 133 MHz PCI-X interfaces. Each node has two Intel Xeon 2.4 GHz processors with a 512 kB L2 cache and a 400 MHz front side bus. The machines are connected with Mellanox InfiniHost MT23108 DualPort 4x HCA adapter through an InfiniScale MT43132 Eight 4x Port InfiniBand Switch. The Mellanox InfiniHost HCA SDK version is thca-x86-0.0.6-rc1-build-002. The adapter firmware version is fw-23108-1.16.0000_5-build-001. Each node has a Seagate ST340016A, ATA 100 40 GB disk. We used the Linux RedHat 7.2 operating system.

6.2. Network and File System Performance

Table 1 shows the raw 4-byte one-way latency and bandwidth of VAPI and IBNice. The benchmark we used for this purpose is *ttcp*, version 1.12-2, with a large socket buffer size of 256 kB to improve IBNice performance. The VAPI Send/Recv and RDMA Write performance is measured using the Mellanox *perf_main* benchmark. The VAPI RDMA Read performance is measured using our own program which is constructed similarly to *perf_main*.

Table 2 compares the read and write bandwidth of an *ext3fs* file system on the local 40 GB disk against bandwidth achieved on a memory-resident file system, using *ramfs*. The *bonnie* [8] file-system benchmark is used.

Table 1. Network performance

	Latency (μs)	Bandwidth (MB/s)
IBNice	40.1	185
VAPI Send/Recv	8.1	825
VAPI RDMA Write	6.0	827
VAPI RDMA Read	12.4	816

Table 2. File system performance

	Write (MB/s)	Read (MB/s)
ext3fs	25	20
ramfs	556	1057

It can be seen that there is a large difference in bandwidth realizable over the network compared to that which can be obtained to a disk-based file system. However, applications can still benefit from fast networks for many reasons in spite of this disparity. Data is frequently in server memory due to file caching and read-ahead when a request arrives. Also, in large disk array systems, the aggregate performance of many disks can approach network speeds. Caches on disk arrays and on individual disks also serve to speed up transfers. Therefore, the following experiments are designed to stress the network data transfer independent of any disk activities. We mainly focus on experiments on a memory-resident file system. Results on *ramfs* are representative of workloads with sequential I/O on large disk arrays or random-access loads on servers which are capable of delivering data at network speeds. We also show some results on *ext3fs* to quantify the impact of CPU utilization on the scalability of I/O server.

6.3. PVFS Concurrent Write Bandwidth

The test program used for concurrent write performance is *pvfs-test*, which is included in the PVFS release package. We followed the same test method as described in [4]. In all tests, each compute node writes and reads a single contiguous region of size $2N$ MB, where N is the number of I/O nodes in use.

Figure 5 shows the write performance with the original impmentation on IBNice and our implementation of PVFS over VAPI, respectively. With IBNice, the bandwidth increases at a rate of approximately 160 MB/s with each additional compute node when there are sufficient I/O nodes to carry the load. With VAPI, our implmentation offers a bandwidth increase of roughly 360 MB/s with each additional compute node. Similar results are attained for PVFS read and can be found in [20].

6.4. MPI-IO Micro-Benchmark Performance

The same test as in the previous subsection was modified to use MPI-IO calls rather than native PVFS calls. The number of I/O nodes was fixed at four, and the number of compute nodes was varied from one to four. Figure 6 shows the performance of MPI-IO over PVFS on VAPI and IBNice, for both memory-based and disk-based file systems. On *ramfs* file system, Figure 6 shows that PVFS native over VAPI offers about three times better performance than PVFS over IBNice. Even on a disk file system, *ext3fs*, it can be seen that although each I/O server is disk-bound, a significant performance improvement, 15–42%, is still achieved. This is because the lower overhead of PVFS-VAPI leaves more CPU cycles free for I/O servers to process concurrent requests.

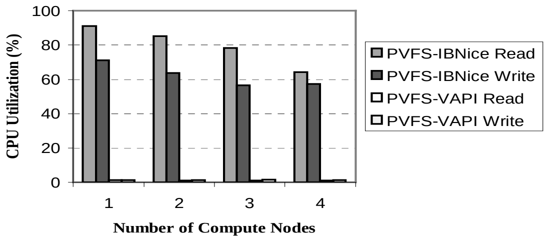
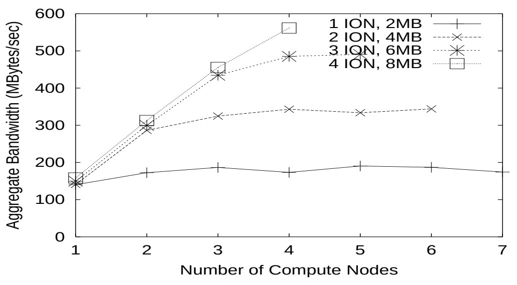


Figure 7. CPU Utilization of MPI-IO

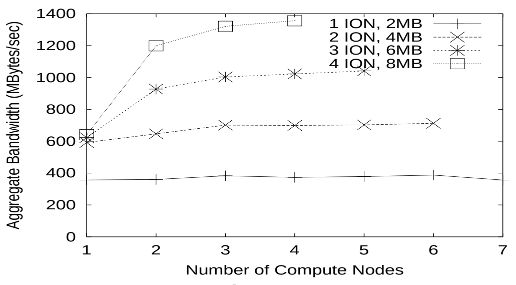
Figure 7 shows CPU utilization on the compute nodes when the same program runs with four I/O servers on *ramfs*. It can be seen that the CPU overhead of compute nodes is as high as 91% in PVFS-IBNice. In contrast, the CPU overhead in PVFS over VAPI is as low as 1.5%. This demonstrates potential for greater scalability to a large number of compute node clients.

6.5 Fast Memory Registration and Deregistration

We run pvfs-test program again with three different memory registration and deregistration schemes. Results are presented in Figure 8. The first one is to dynamically register and deregister I/O buffers per each I/O operation, noted as *Dynamic* in the plot. The second one is to use pin-down cache only, noted as *Pin-down cache*. The third one is to use FMRD, noted as *FMRD*. The test program performs 1000 I/O operations, in which I/O buffers are from

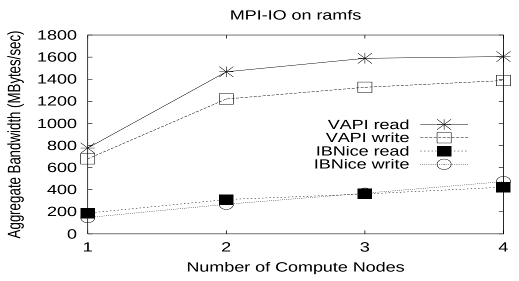


(a) IBNice



(b) VAPI

Figure 5. PVFS write performance comparision between IBNice and VAPI.



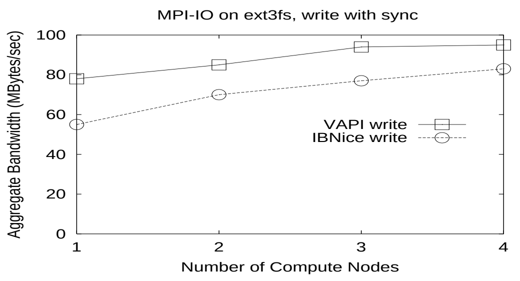


Figure 6. MPI-IO Performance

a buffer pool with 1000 different buffers. We control pin-down cache hit ratio explicitly. We choose 20% and 80% cache hit as representatives of low buffer reuse and high buffer reuse cases, respectively. The cache size is 100, which allows us to take deregistration into account.

Figure 8 shows PVFS write bandwidth with different schemes. Note that these results are normalized to the results of the case where there is not any buffer registration and deregistration. We make three observations. First, memory registration and deregistration have a significant impact on performance. Up to 35% decrease is seen in the dynamic scheme. Second, significant improvement on performance with pin-down cache and FMRD is achieved. Particularly, if the buffer reuse ratio is 80%, pin-down cache increases bandwidth by about 24%, while FMRD increases bandwidth by about 28%. Third, FMRD works much better than pin-down cache in cases where buffer reuse ratio is low. There is about 9% improvemnt compared to pin-down cache when buffer reuse ratio is 20%.

6.6 Performance of the Tiled I/O Benchmark

The test application *mpi-tile-io* [16] implements tiled access to a two dimensional dense dataset. This type of workload is seen in visualization applications and in some numerical applications. For our tests, we used four compute nodes and four I/O server nodes. Each compute node renders to one of a 2×2 array of displays, each with 1024×768 pixels. The size of each element is 24 bytes, leading to a file size of 72 MB.

The access pattern in this test is noncontiguous in file space but contiguous in memory. This is a good candidate to exercise PVFS list I/O [5]. We test two versions of *mpi-tile-io*: one is to use multiple contiguous I/O operations to achieve noncontiguous file accesses ("Without list I/O"), the other uses PVFS list I/O to make a single noncontiguous access.

Figure 9 shows the results for both PVFS-VAPI and PVFS-IBNice. Compared to the performance of PVFS-VAPI and PVFS-IBNice, with list I/O, PVFS-VAPI offers 2.7 and 2.2 times the bandwidth on read and write, respectively. Without list I/O, the improvement is 79% and 93%, respectively. The improvement difference between using list I/O and not using it is because the access size is larger for each pair of request and reply messages with list I/O and can yield more improvement from the VAPI layer.

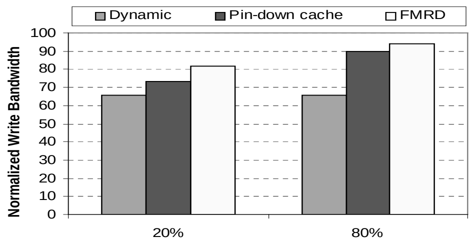


Figure 8. Effects of Memory Registration and Deregistration

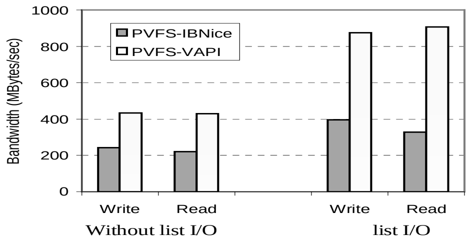


Figure 9. Performance of tiled I/O.

7. Related Work

Various user-level communication protocols have been used for network storage in the past. Zhou *et al.* [22] present their experiences with VIA networks for database storage. Magoutis *et al.* [12] explore DAFS performance characteristics, also on VIA. Our work is based on the InfiniBand architecture.

Zahir [21] described a storage-networking transport layer for the Lustre file system based on VI-like networks. Carns [3] proposed a Buffer Message Interface (BMI) as a transport layer for the next generation PVFS on both TCP/IP and Myrinet/GM. Liu *et al.* [10] proposed a client/server communication middleware over system area networks. Our transport layer differs them in many ways, especially in the select of communication mechanisms and the cooperation between buffer management and communication management to deal with particular issues in I/O intensive applications.

Research work in [19, 7, 22] have proposed different approches to reduce memory registration and deregistration overheads, such as pin-down cache and batched deregistration. Our work, the two-level architecture, is indeed a combination of the pin-down cache and batched deregistration.

8 Conclusions and Future Work

In this paper, we study how to leverage the emerging InfiniBand technology to improve I/O performance and scalability of cluster file systems. We designed and implemented a version of PVFS that takes advantage of InfiniBand features. Our work shows that the InfiniBand network and its user-level communication and RDMA features can improve all aspects of PVFS, including throughput, access time, and CPU utilization. However, InfiniBand network also poses a number of challenging issues to I/O intensive applications which PVFS targets. We addressed these issues in this paper by designing: a transport layer customized for the PVFS protocol by trading transparency and generality for performance, buffer management for flow control, dynamic and fair buffer sharing, and efficient memory registration and deregistration. Inline, Fast RDMA Write, and Pipelined Bulk data transfers were designed and implemented in the transport layer. Our results show that these techniques bring significant performance gains. We also demonstrated that our proposed two-level memory registration and deregistration architecture works better than other schemes and offers efficient memory registration and deregistration in the I/O intensive environment.

As of this writing, a major rewrite of PVFS is in active development. Our work is directly applicable to this next generation PVFS over networks with user-level access and RDMA capabilities. We are working with the PVFS team to incorporate our design into the next generation PVFS and to implement it on InfiniBand.

Acknowledgments: We would like to thank the PVFS team at Argonne National Laboratory and Clemson University for giving us the access to the latest version of PVFS implementation and for providing us with crucial insights into the implementation. We are also thankful to Jiuxing Liu and Sushmitha P. Kini form our research group for many discussions with us.

References

[1] D. Anderson, J. Chase, S. Gadde, A. Gallatin, K. Yocum, and M. Feeley. Cheating the I/O Bottleneck: Network Storage with Trapeze/Myrinet. In *Proceedings of the Usenix Technical Conference. New Orleans, LA.*, 1998.

[2] M. Bancroft, N. Bear, J. Finlayson, R. Hill, , R. Isicoff, and H. Thompson. Functionality and Performance Evaluation of File Systems forStorage Area Networks (SAN). In *the Eighth NASA Goddard Conference on Mass Storage Systems and Technologies*, 2000.

[3] P. Carns. Design and Analysis of a Network Transfer Layer for Parallel File Systems. Master thesis. http://parlweb.parl.clemson.edu/techreports/.

[4] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A Parallel File System for Linux Clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, Atlanta, GA, 2000. USENIX Association.

[5] A. Ching, A. Choudhary, W. keng Liao, R. Ross, and W. Gropp. Noncontiguous I/O through PVFS. In *Proc. of the IEEE Int. Conf. on Cluster Computing*, 2002.

[6] DAFS Collaborative. Direct Access File System Protocol, V1.0, August 2001.

[7] H. Tezuka and F. O'Carroll and A. Hori and Y. Ishikawa. Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication. In *12th Int. Parallel Processing Symposium*, March 1998.

[8] http://www.textuality.com/bonnie/. Bonnie: A File System Benchmark.

[9] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.0, October 24, 2000.

[10] J. Liu, M. Banikazemi, B. Abali, and D. K. Panda. A Portable Client/Server Communication Middleware over SANs: Design and Performance Evaluation with InfiniBand. In *SAN-02 Workshop (in conjunction with HPCA)*, Feb. 2003.

[11] C. Lever and P. Honeyman. Linux NFS Client Write Performance. In *Proceedings of the Usenix Technical Conference, FREENIX track, Monterey*, June 2001.

[12] K. Magoutis, S. Addetia, A. Fedorova, M. Seltzer, J. Chase, A. Gallatin, R. Kisley, R. Wickremesinghe, and E. Gabber. Structure and performance of the direct access file system. In *Proc. of USENIX 2002 Annual Technical Conference, Monterey, CA*, pages 1–14, June 2002.

[13] Mellanox Technologies. Mellanox InfiniBand InfiniHost Adapters, July 2002.

[14] Mellanox Technologies. Mellanox IB-Verbs API (VAPI), Rev. 0.95, March 2003.

[15] V. S. Pai, P. Druschel, and W. Zwaenepoel. IO-Lite: A Unified I/O Buffering and Caching System. *ACM Transactions on Computer Systems*, 18(1):37–66, 2000.

[16] R. B. Ross. Parallel I/O Benchmarking Consortium. http://www-unix.mcs.anl.gov/ rross/pio-benchmark/html/.

[17] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *First USENIX Conference on File and Storage Technologies*.

[18] Storage Networking Industry Association. Shared Storage Model. www.snia.org/tech_activities/shared_storage_model.

[19] M. Welsh, A. Basu, and T. von Eicken. Incorporating Memory Management into User-Level Network Interfaces. In *Proc. Hot Interconnects V*, August 1997.

[20] J. Wu, P. Wyckoff, and D. K. Panda. PVFS over InfiniBand: Design and Performance Evaluation. Technical Report, OSU-CISRC-04/03-TR (http://nowlab.cis.ohio-state.edu/projects/mpi-iba/index.html) , April 2003.

[21] R. Zahir. Lustre Storage Networking Transport Layer. http://www.lustre.org/docs.html.

[22] Y. Zhou, A. Bilas, S. Jagannathan, C. Dubnicki, J. F. Philbin, and K. Li. Experiences with VI communication for database storage. In *Proceedings of the 29th annual international symposium on Computer architecture*, pages 257–268. IEEE Computer Society, 2002.