

# PVFS2 over Quadrics: Design, Implementation and Performance Evaluation

Weikuan Yu   Shuang Liang   Dhabaleswar K. Panda

Technical Report  
OSU-CISRC-12/04-TR64

# PVFS2 over Quadrics: Design, Implementation and Performance Evaluation \*

Weikuan Yu Shuang Liang Dhabaleswar K. Panda

Network-Based Computing Laboratory  
Dept. of Computer Sci. & Engineering  
The Ohio State University  
{yuw,liangs,panda}@cse.ohio-state.edu

## Abstract

*Parallel I/O needs to keep pace with the ever-increasing computing power of high performance computers for the real world applications to perform well. Exploiting high-end interconnect technologies such as Quadrics to reduce the network access cost and scale the aggregated bandwidth is one of the ways to increase the performance of storage systems. In this paper, we explore the challenges of supporting parallel file system with modern features of Quadrics, including user-level communication and RDMA operations. We then design and implement a Quadrics-capable version of a parallel file system (PVFS2) by overcoming Quadrics static communication model and providing an efficient transport layer over Quadrics, which includes an optimized non-contiguous IO support using multiple zero-copy RDMA operations with a chained event. Experiment results indicates that, with Quadrics user-level protocols and RDMA operations, the performance of PVFS2 is significantly improved in terms of both data transfer and management operations. With four IO server nodes, our implementation improves PVFS2 aggregated read bandwidth by up to 140% compared to PVFS2 over TCP/IP. Moreover, it delivers significant performance improvement in terms of IO access to application benchmarks such as mpi-tile-io [24] and BT-IO [26]. To the best of our knowledge, this is the first work in the literature to report the design of a high performance parallel file system over Quadrics user-level communication protocols.*

Keywords: *Parallel IO, Parallel File System, RDMA, Zero-Copy, Quadrics*

## 1. Introduction

Parallel computing architecture has recently evolved into systems with thousands of processors. In the mean time, the gap between computer processing power and disk throughput is becoming wider as the growth of the latter continuously lags behind that of the former [21]. Large I/O-intensive applications on these platforms demand increasingly higher I/O throughput. Correspondingly, scalable parallel I/O needs to be available for these real world applications to perform well. Both commercial [13, 15, 9] and research projects [19, 12, 1] have been developed to provide parallel file systems for I/O accesses on such architectures. Among them, the Parallel Virtual File System 2 (PVFS2) [1] has been created with the intention of addressing the needs of next generation systems using low cost Linux clusters with commodity components.

On the other hand, high performance interconnect technologies such as Myrinet [4], InfiniBand [14], and Quadrics [3] not only have been deployed into large commodity component-based clusters to provide higher computing power, but also have been utilized in commodity storage systems to achieve scalable parallel I/O support. For example, the low-overhead high-bandwidth user-level communication provided by VI [30], Myrinet [20], and InfiniBand [27] has been utilized to parallelize I/O accesses to storage servers and increase the performance of parallel file systems.

One of the leading interconnect technologies, Quadrics Interconnects [23, 3], provides very low latency ( $\leq 2\mu s$ ) and high bandwidth. It also supports many of the cutting-edge communication features, such as OS-bypass user-level communication, remote direct memory access (RDMA), as well as hardware atomic and collective operations. Moreover, Quadrics network interface provides a programmable network co-

---

\*This research is supported in part by a DOE grant #DE-FC02-01ER25506 and NSF Grants #EIA-9986052 and #CCR-0204429.

processor, which offloads much of the communication processing down to the network interface and contributes greatly to its efficient point-to-point and collective communication. These salient features and their performance advantages of Quadrics have not been leveraged to support scalable parallel IO throughput at the user-level, though some of these modern features, like RDMA, are exploited in other interconnects, such as Myrinet [20] and InfiniBand [27]. Currently, some distributed file systems that exploit the advantages of Quadrics are developed on top of Quadrics kernel communication library, e.g., Lustre [8]. But this approach incurs higher network access overhead because the operating system is included in the communication path. In addition, as a distributed file system Lustre is designed to scale the aggregated bandwidth for accesses to files on different servers, while parallel file accesses from a single parallel job cannot directly take its maximum benefits. For example, concurrent writes from multiple processes in a single parallel job cannot benefit with Lustre. A typical platform may utilize a parallel file system such as PFS [15] to export scalable bandwidth to a single job by striping the data of a single parallel file system over multiple underlying file systems such as Lustre. However, the extra multiplexing process adds more to the cost in the path of IO accesses.

In this paper, we examine the feasibility of supporting parallel file systems with Quadrics user-level communication and RDMA operations. PVFS2 [1] is used as a parallel file system in this work. We first characterize the challenges of supporting PVFS2 on top of Quadrics interconnects, focusing on: (a) constructing a client-server model over Quadrics at the user-level, (b) mapping an efficient PVFS2 transport layer over Quadrics, and (c) optimizing the performance of PVFS2 over Quadrics such as efficient non-contiguous communication support. Accordingly, we implement PVFS2 over Quadrics by taking advantage of Quadrics RDMA and event mechanisms.

We then evaluate the implementation using PVFS2 and MPI-IO [18] benchmarks. The performance of our implementation is compared to that of PVFS2 over TCP. Quadrics IP implementation on top of its interconnect is used in the TCP implementation to avoid network differences. Our work demonstrates that: (a) a client/server process model necessary for file system communication is feasible with Quadrics interconnects; (b) the transport layer of a parallel file system can be efficiently layered on top of Quadrics; and (c) the performance of PVFS2 can be significantly improved with Quadrics user-level protocols and RDMA capabilities. Compared to a PVFS implementation over TCP/IP over Quadrics, our implementation increases the aggregated read performance of PVFS2 by 140%. It is also able to deliver significant performance improvement in terms of IO access to ap-

plication benchmarks such as `mpi-tile-io` [24] and `BT-IO` [26]. To the best of our knowledge, this is the first work in the literature to report the design of a high performance parallel file system over Quadrics user-level communication protocols.

The rest of the paper is organized as follows. In the next section, we provide overviews of Quadrics and PVFS2, and the challenges to design PVFS2 over Quadrics. Section 3 provides the design of client/server model over Quadrics. Sections 4 and 5 discuss the design of the PVFS2 transport layer over Quadrics communication mechanisms. The implementation is provided in Section 6, followed by the performance evaluation in Section 7. Section 8 gives a brief review of related works. Section 9 concludes the paper.

## 2. Challenges in Designing PVFS2 over Quadrics/Elan4

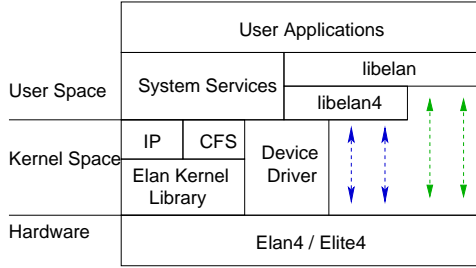
Quadrics interconnects [3] and its parallel programming libraries, `libelan` and `libelan4` [23], are widely used to support high performance computing. However little is known about how to leverage Quadrics high performance user-level communication to support high performance parallel file system. This section provides a brief overview of Quadrics/Elan4 and PVFS2, and discusses the challenging issues in designing PVFS2 over Quadrics/Elan4.

### 2.1. Overview of Quadrics/Elan4

Quadrics [22, 23] has recently released its second generation network, `QsNetII` [3]. This new release provides very low latency, high bandwidth communication with its two building blocks: the Elan-4 network interface card and the Elite-4 switch, which are interconnected in a fat-tree topology. As shown in Fig. 1, Quadrics provides two communication libraries: `libelan` and `libelan4` in the user space, and a kernel communication library on top of its Elan4 network [23]. While the kernel communication library provides communication support to Lustre (CFS) [8] and IP protocols, the user-level communication libraries (`libelan` and `libelan4`) can provide OS-bypass communication and Remote Directed Message Access (RDMA) directly to parallel user applications.

### 2.2. Overview of PVFS2

PVFS2 [1] is the second generation parallel file system from the Parallel Virtual File System (PVFS) project team. It incorporates the design of the original PVFS [20] to provide parallel and aggregated I/O performance. A client/server architecture is designed in

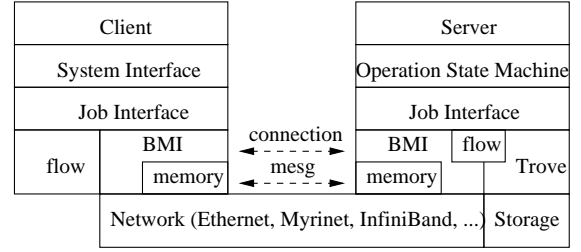


**Fig. 1. Quadrics/Elan4 Communication Architecture**

PVFS2. Both the server and client side libraries can reside completely in user space. Clients communicate with one of the servers for file data accesses, while the actual file IO is striped across a number of file servers. Metadata accesses can also be distributed across multiple servers. Storage spaces of PVFS2 are managed by and exported from individual servers using native file systems available on the local nodes. More information about PVFS2 can be found in [1].

### 2.3. Challenges

PVFS2 provides a network abstraction layer to encapsulate all the functionalities needed for communication support. The resulting component is called Buffered Message Interface (BMI), which interacts with other components in the software architecture to support low-level IO accesses. Fig. 2 shows a diagram of PVFS2 components on both the client side and the server side. As shown in the figure, BMI functionalities can be further classified into three categories: connection management between processes, message passing activities for interprocess communication (IPC) and the memory management needed for IPC. In particular, Quadrics user-level programming libraries has a unique design for running Higher Performance Computing (HPC) applications. All parallel jobs over Quadrics need to start from a static pool of application processes [23]. This is rather incompatible to the needs of file systems, which start servers first and deliver IO services to incoming clients. In addition, PVFS2 interprocess communication between servers and clients needs to be properly layered over Quadrics communication mechanisms to expose the best capability of Quadrics hardware. In this work, we take on the following issues to design PVFS2 over Quadrics: (a) constructing a client/server communication model in terms of connection management, (b) designing PVFS2 basic transport protocol to appropriate Quadrics communication mechanisms for message transmission, and (c) optimizing PVFS2 performance over Quadrics.



**Fig. 2. The Architecture of PVFS2 Components**

## 3. Designing a Client/Server Communication Model Over Quadrics/Elan4

As described in Section 2.2, PVFS2 is designed as a client/server architecture. In contrast, a parallel job over Quadrics libraries runs as a static pool of application processes [23]. All of these processes join or leave the Quadrics network in a synchronized manner. In addition, to facilitate this process, Quadrics requires a resource management framework such as RMS [23] to launch the parallel applications. To provide a PVFS2 client/server architecture over Quadrics, it is necessary to break the model of static process pool used in Quadrics parallel jobs and eliminate the need of a resource management framework.

### 3.1. Allocating a Dynamic Pool of Processes over Quadrics

Each process must acquire a unique Virtual Process ID (VPID) and use it as an identity for network addressing before the communication starts over Quadrics. VPID is an abstract representation of Quadrics capability, which describes the network node ID and context ID owned by a process, and the range of network nodes and the range of contexts all processes have. Typically, Quadrics utilizes RMS [23] to allocate appropriate capabilities for all application processes before launching a parallel job. The capabilities from all processes share the same range of network nodes and the same range of contexts. Together with the network node ID and a context ID, each process can determine its VPID based on the capability. In this way, a static pool of application processes is launched over Quadrics network.

To allocate a dynamic pool of processes over Quadrics, we change the aforementioned allocation scheme. First, we expand the range of nodes to include every node in the network. Second, a large range of contexts is provided on each node. Table 1 shows the format of Elan4 capability for all PVFS2 processes. On each node, the first context is dedicated to the server process, if present, and the rest of the contexts are left for the client processes. The VPID is needed to identify an elan4 process is calculated with this formula:

$node\_id * (j - i + 1) + (ctx - i)$ . A client process obtains the corresponding parameters from the PVFS2 fstab entry as shown on the third row of Table 1. Clients connect to a server on a dynamic basis, and notify the server when they leave. Servers allocate communicating resources as new clients join in, and deallocate when they disconnect or timeout. There are no restrictions for processes to synchronize memory allocation and synchronized startup.

**Table 1. Elan4 Capability Allocation for Dynamic Processes**

| Setting    | Value                                 |
|------------|---------------------------------------|
| Capability | $node\{0..N\}ctx\{i..j\}$             |
| VPID       | $node\_id * (j - i + 1) + (ctx - i)$  |
| fstab      | elan4://server_id:server_ctx/pvfs2-fs |

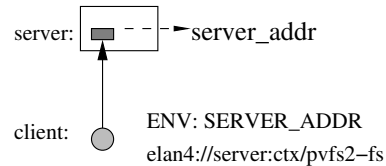
### 3.2. Fast Connection Management

A process over Quadrics needs to know both the VPID and an exposed memory location of a remote process before sending a message. Parallel jobs built from default Quadrics libraries, typically use a global memory address to initiate communication because the memory allocation is synchronized and a global virtual memory [23] is available. Without the use of a global memory, we design two different schemes for clients to initiate communication to PVFS2 servers over Quadrics/Elan4. Initially, a basic scheme utilizes TCP/IP-based socket. A server opens a known TCP port and polls for incoming communication requests from time to time. Clients connect to this known TCP port and establish a temporal connection to exchange VPID and memory addresses.

Because establishing and tearing down the connections between clients and servers is so common for file I/O services, it is desirable to design a fast connection management scheme to achieve scalable IO access. In another scheme, we use native communication over Quadrics for communication initiation. All servers start with a known node ID and context ID, which together determine the VPID according to the allocation scheme described earlier in this section. A set of receive queue slots are also allocated, which start at a unified memory address across all the servers. Servers then poll on this receive queue for new connection requests (and also IO service), using a Queue-based Direct Memory Access model. The QDMA model is described in more detail in Section 4.1. Because this memory for the receive queue (a portion of the NIC memory mapped to the host address space) is allocated dynamically at run time in the current Quadrics implementation, one constraint here is

that the server needs to report its address at the startup time. We pass this address to clients as an environmental parameter. Further investigation will study the feasibility and impact of mapping Quadrics NIC memory to a fixed memory space.

As shown in Fig. 3, a client process that is initiating the connection with a server. The client obtains the VPID of the server based on the pvfs2 fstab file and the memory address of the server’s receive queue through an environmental variable, `SERVER_ADDR`. Using the known memory address and the known VPID, a client can initiate a message to the server, which includes its own VPID and address of its exposed memory location. When a connection is initiated, the corresponding network addressing information is recorded into a global address list. Lists to record all the outstanding operations are also created. This address information and associated resources are removed when a connection is finalized as if no connection has been established earlier.



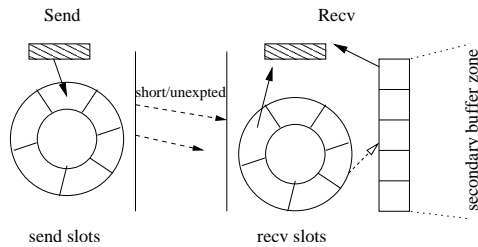
**Fig. 3. Connection Initiation over Native Elan4 Communication**

## 4. Designing PVFS2 Basic Transport Layer over Quadrics/Elan4

All PVFS2 [1] network message transmission functionalities are included in the BMI interface [6]. Two models of message transmission, matched and unexpected, are included. All network operations are designed in a nonblocking manner to allow multiple of them in service concurrently. Several test APIs are specified for the completion of outstanding messages. Quadrics provides two basic interprocess communication models: Queue-based Direct Memory Access (QDMA) and Remote Direct Memory Access (RDMA) [23]. QDMA can only transmit messages up to 2KB. The other model, RDMA read/write, supports transmission of arbitrary messages over Quadrics network. Using these two models, the transport layer of PVFS2 over Quadrics/Elan4 is designed with two protocols, eager and rendezvous, to handle different size messages.

#### 4.1. Short and Unexpected Messages with Eager Protocol

The QDMA model allows a process to check incoming QDMA messages posted by any process into its receive queue. An eager protocol is designed with this model to transmit short and unexpected messages. As mentioned in Section 3.2, this QDMA model is used in initiating dynamic client/server connection scheme with Quadrics native communication.



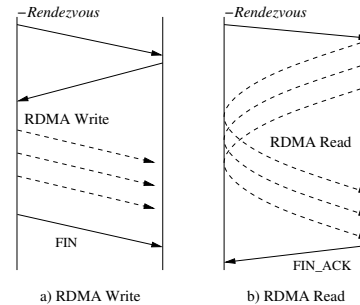
**Fig. 4. Eager Protocol for Short and Unexpected Messages**

As shown in Fig. 4, in the eager protocol, a number of sender buffers are allocated on the sender side to form a send queue, and a fixed number of receive queue slots are created on the receiver side to form a receive queue. In addition, a secondary receive buffer zone is created with another set of receive buffers. The number of receive buffers in the secondary zone can grow or shrink on an on-demand basis. In this eager protocol, a new message is first copied into a sender queue slot, sent over the network, and eventually received into a receive queue slot. If the message is an unexpected message, it is then copied into a receiver buffer immediately without waiting for a matching receive operation to be posted. The receive queue slot is then recycled to receive new messages. For a message that needs to be matched, it remains in the receive queue slot until a matching receive operation is posted. This can save an extra message copy if the operations is posted in time. However, if the number of receive queue slots becomes low under various situations, these messages are copied into the receive buffers in the secondary buffer zone to free up receive slots for more incoming messages. When the messages are eventually matched, the receive buffers are also recycled into the secondary buffer zone. If there are relatively a large number of free receive buffers in the secondary zone, they are deallocated to reduce the memory usage.

#### 4.2. Long Messages with *Rendezvous* Protocol

Quadrics RDMA (read/write) communication model can transmit arbitrary size messages [23]. A *rendezvous*

protocol is designed with this model for long messages. Two schemes are proposed to take advantage of RDMA read and write, respectively. As shown in Fig. 5 left diagram, RDMA write is utilized in the first scheme. A *rendezvous* message is first initiated from the sender to the receiver in both schemes. The receiver returns an acknowledgment to the sender when it detects a *rendezvous* message. The sender then sends the full message with a RDMA write operation. At the completion of RDMA write, a control fragment, typed as FIN, is sent to the receiver for the completion notification of the full message. The right diagram in Fig. 5 shows the second scheme with RDMA read. When the *rendezvous* message arrives at the receiver, instead of returning an acknowledgment to the sender, the receiver initiates RDMA read operations to get the data. When these RDMA read operations complete, a different control message, typed as FIN\_ACK, is sent to the sender, both for acknowledging the arrival of the earlier *rendezvous* fragment and notifying the completion of the whole message.



**Fig. 5. *Rendezvous* Protocol for Long Messages**

### 5. Optimizing the Performance of PVFS2 over Quadrics

To improve the basic design discussed in Section 4, we have explored several further design issues including zero-copy non-contiguous network IO access, adaptive *rendezvous* message transfer with RDMA read/write and optimization on completion notification.

#### 5.1. Adaptive *Rendezvous* with RDMA Read and RDMA Write

As discussed in Section 4.2, RDMA read and write are both utilized in the *rendezvous* protocol. This achieves zero-copy transmission of long messages. File systems, such as DAFS [10], also take advantage of similar RDMA-based message transmission. Typically the server decides to use RDMA read or write based on

whether the client is performing a read or write operation: a read operation is implemented as RDMA write from the server, and a write operation as a RDMA read. Thus a server process can be potentially overloaded with a large number of outstanding RDMA operations, which can lead to suboptimal performance due to the bandwidth drop-off [5]. Therefore a basic throttling mechanism is needed to control the number of concurrent outstanding RDMA operations. We introduce an adaptive throttling mechanism to regulate the number of RDMA operations at any given time. Under a light load, the sender initiates a long message operation to the receiver, which in turn pulls the message from the sender through RDMA read. If either side has a large number of outstanding RDMA operations, the receiver gathers this knowledge from its local communicate state and the information passed from the sender in the initial *rendezvous* packet. When the receiver is heavily loaded, it notifies the sender accordingly and the sender completes the operation through RDMA write. For the reason of fairness to multiple clients, if both the sender and the receiver are heavily loaded, the priority is given to the server and have the client carry out the RDMA operations. It does not matter whether it is a sender or a receiver. Table 2 provides a sample receiver’s decision table for the RDMA *rendezvous* protocol.

**Table 2. Receiver’s Decision Table for Adaptive RDMA *Rendezvous* Protocol**

| load: sender <> receiver | loaded server? | RDMA  |
|--------------------------|----------------|-------|
| greater                  | yes            | write |
| greater                  | no             | read  |
| equal                    | yes            | write |
| equal                    | no             | read  |
| less                     | yes            | write |
| less                     | no             | read  |

## 5.2. Optimizing Completion Notification

Event mechanisms that enable both local and remote completion notification are available in Quadrics communication models. In particular, this mechanism can be used to enable notification of message completion along with RDMA read/write operations. In the *rendezvous* protocol, so long as the control information contained in the last control message is available to the remote process, the completion of a full message can be safely notified through an enabled remote event. We have designed this as an optimization to the *rendezvous* protocol. A sender process allocates a completion event and encodes the address of this event in the first *rendezvous* message. When the receiver pulls the message via RDMA read, it

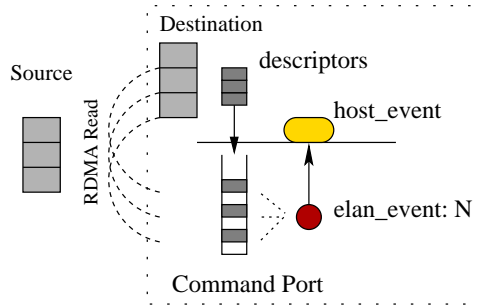
also triggers a remote event to the sender using the provided event address. Similarly, in the case of RDMA write, the receiver provides the address of such an event in its acknowledgment to the sender. The receiver detects the completion of a full message through the remote event triggered by a RDMA write operation. In either case, both sides notice the completion of data transmission without the need of an extra control message.

## 5.3. Zero-Copy Non-Contiguous IO

Non-contiguous I/O access is the main access pattern in scientific applications. Thakur et. al. [25] also noted that it is important to achieve high performance MPI-IO to have native noncontiguous access support in file systems. PVFS2 provides list network I/O interface in its BMI interface to support communication over non-contiguous memory. The corresponding communication operations on the sender and receiver sides do not have to exactly match in terms of offset, length or the number of units, so long as the receiver is providing enough memory for the incoming data. Basic list IO non-contiguous communication support usually resorts to memory packing and unpacking to convert non-contiguous communication to contiguous. An alternative is to use multiple contiguous operations, this would require multiple send and receiver operations on both the sender and receiver sides. The former leads to an unnecessary memory copy on both the sender and receiver sides; the latter leads to more processing and smaller communication operations. Both would result in performance degradation.

Quadrics provides non-contiguous communication operations in the form of *elan\_putv* and *elan\_getv*. However, these operations are specifically designed for the shared memory programming model (SHMEM) over Quadrics. They cannot be utilized for efficient support of PVFS2 non-contiguous IO operations because they are built on top of a global memory space, which is only available to the group of processes that are started under Quadrics static process model. In addition, the final placement of the data will still require a memory copy from the global memory to the application destination memory. We propose to utilize a single event chained to multiple RDMA to support zero-copy non-contiguous communication. An example of zero-copy list IO operation is shown in Fig. 6.

In the example shown in Fig. 6, the receiver side has collected the information about list IO fragments on both the sender and receiver sides. It first determines the number of required contiguous RDMA read operations,  $N$ . Then it constructs the same number of RDMA descriptors in the host memory and writes them together into the Quadrics Elan4 command port (a command queue to the NIC formed by a memory-mapped



**Fig. 6. Zero-Copy Non-Contiguous Communication with RDMA Read and Chained Event**

user accessible NIC memory) through programmed IO. An Elan4 event is created to wait on the completion of  $N$  RDMA. As this event is triggered, the completion of list IO operation is detected through a host-side event. The remote side is notified through a separated chained message as described in Section 4.2 or simply a remote event as described in Section 5.2. Note that, using this approach, multiple RDMA operations are issued without calling extra sender or receiver routines. Zero-copy is achieved by directly addressing the source and destination memory. If RDMA write is chosen based on the adaptive *rendezvous* protocol, similar zero-copy non-contiguous support can be achieved by issuing multiple RDMA write operations, all being chained with another Elan4 event of count  $N$ .

## 6. Implementation

With the design of client/server connection model and the transport layer over Quadrics communication mechanisms, we have implemented PVFS2 over Quadrics/Elan4. The implementation is based on the recent release of PVFS2-1.1-pre1. Due to the compatibility issue of PVFS2 and Quadrics RedHat Linux kernel distribution, we have utilized a patched stock kernel linux-2.4.26-4.23qsnnet. Our implementation is layered on top of Quadrics library, libelan4, and completely resides in the user space. We include the following choices in our implementation: short messages are transmitted in the eager protocol along with the chained control message; long messages are transmitted through the adaptive *rendezvous* protocol using zero-copy RDMA read and write; zero-copy non-contiguous IO is supported using multiple RDMA and a single chained event; a throttling mechanism is enforced to regulate the number of concurrent RDMA read and write operations.

## 7. Performance Evaluation

In this section, we describe the performance evaluation of our implementation of PVFS2 over Quadrics/Elan4. The experiments were conducted on a cluster of eight SuperMicro SUPER X5DL8-GG nodes: each with dual Intel Xeon 3.0 GHz processors, 512 KB L2 cache, PCI-X 64-bit 133 MHz bus, 533MHz Front Side Bus (FSB) and a total of 2GB PC2100 DDR-SDRAM physical memory. All eight nodes are connected to a QsNet<sup>II</sup> network [23, 3], with a dimension one quaternary fat-tree [11] QS-8A switch and eight Elan4 QM-500 cards. Each node has a 40GB, 7200 RPM, ATA/100 hard disk Western Digital WD400JB. The operating system is RedHat 9.0 Linux. To minimize the impact in network capacity, we used the TCP implementation of PVFS2 as a comparison. As mentioned in Section 2.1, Quadrics provides an IP implementation on top of its kernel communication library.

### 7.1. Performance Comparisons of Different Communication Operations

Table 3 shows the comparisons of the latency and bandwidth between TCP/IP over Quadrics and Quadrics native communication operations, including QDMA and RDMA read/write. Quadrics IP implementation is often referred to as EIP based on the name of its Ethernet module. The performance of TCP stream over Quadrics is obtained using the netperf [2] benchmark. The performance of Quadrics native operations is obtained using microbenchmark programs, `ppping` and `qping`, available from standard Quadrics releases [23].

**Table 3. Network Performance over Quadrics**

| Operations          | Latency       | Bandwidth  |
|---------------------|---------------|------------|
| TCP/EIP             | 23.92 $\mu$ s | 482.26MB/s |
| Quadrics RDMA/Write | 1.93 $\mu$ s  | 910.1MB/s  |
| Quadrics RDMA/Read  | 3.19 $\mu$ s  | 911.1MB/s  |
| Quadrics QDMA       | 3.02 $\mu$ s  | 368.2MB/s  |

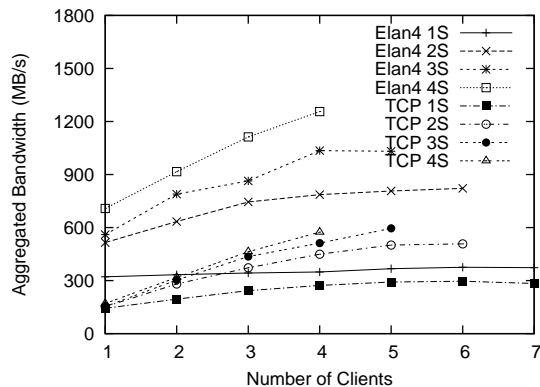
As shown in the table, Quadrics native operations provide better performance in terms of both latency and bandwidth compared to the performance of TCP over Quadrics. Moreover, the host CPU has less involvement in the communication processing when using Quadrics RDMA operations because of its zero-copy message delivery. More CPU cycles can be used to handle computation in other components and contribute to better overall file system performance. To demonstrate the potential and effectiveness of leveraging Quadrics capabilities, we focus on the following aspects: the performance of bandwidth-bound data transfer operations, the per-



formance of the latency-bound management operations, and the performance benefits to application benchmarks, such as MPI-Tile-IO [24] and BT-IO [26].

## 7.2. Performance of Data Transfer Operations

To evaluate the data transfer performance of PVFS2 file system, we have used a parallel program that iteratively performs the following operations: create a new PVFS2 file, concurrently write data blocks to disjoint regions of the file, flush the data, concurrently read the same data blocks back from the file, and then remove the file. MPI collective operations are used to synchronize application processes before and after each I/O operation. In our program, each process writes and then reads a contiguous 4MB block of data at disjoint offsets of a common file based on its rank in the MPI job. At the end of each iteration, the average time to perform the read/write operations among all processes is computed and recorded. Seven iterations are performed, and the lowest and highest values are discarded. Finally, the average values from the remaining iterations are taken as the performance for the read and write operations.

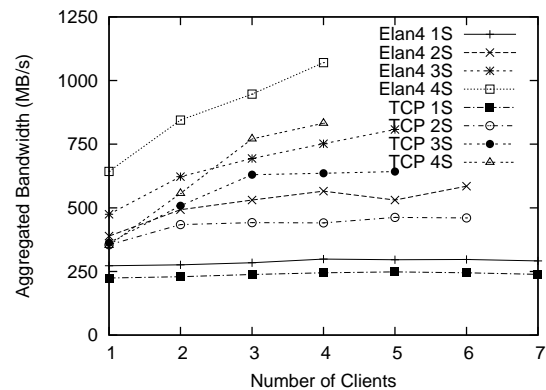


**Fig. 7. Performance Comparisons of PVFS2 Concurrent Read**

We have divided the eight-node cluster into two groups: servers and clients. Up to four nodes are configured as PVFS2 servers, and the remaining nodes are running as clients. Experimental results are labeled as *NS* for a configuration with *N* servers. Fig. 7 shows the read performance of PVFS2 over Elan4 compared to the PVFS2 over TCP. PVFS2 over Elan4 improves the aggregated read bandwidth by more than 140% compared to that of PVFS2 over TCP. This suggests that the read performance of PVFS2 is much limited by the network communication and can significantly benefit from the improvement in the network performance.

We have also performed experiments to evaluate the write performance of PVFS2/Elan4. We have observed

less than 10% performance improvement compared to PVFS2/TCP (data not shown). This is because the network bandwidth of both Elan4 and TCP are more than 350MB/s, which is much higher than the performance of the local IDE disk in the order of 40MB/s. Instead, we have used a memory-resident file system, ramfs, to avoid the bottleneck of disk access. This is shown in Fig.8. With varying numbers of clients concurrently writing to the file system, PVFS2 over Elan4 improves the aggregated write bandwidth by up to 82% compared to that of PVFS2 over TCP. This suggests that PVFS2 write bandwidth can also benefit from Quadrics communication mechanisms, though it is relatively less bounded by the network communication compared to the read performance.



**Fig. 8. Performance Comparisons of PVFS2 Concurrent Write**

## 7.3. Performance of Management Operations

PVFS2 parallel file system is designed to provide scalable parallel IO operations that match MPI-IO semantics. For example, management operations, such as `MPI_File_open` and `MPI_File_set_size`, are shown to be very scalable in [16]. These management operations typically do not involve massive data transfer. To evaluate the benefits of Quadrics low latency communication to these management operations, we have performed the following experiments using a microbenchmark program available in the PVFS2 distribution.

With the eight-node cluster, a PVFS2 file system is configured with two servers, both act as metadata and IO servers. The first experiment measures the average time to create a file using collective `MPI_File_open` with different numbers of clients. The second experiment measures the average time to perform a resize operation using collective `MPI_File_set_size` with different numbers of clients. As shown in Table 4, our PVFS2 implementation over Elan4 improves the time

**Table 4. Comparison of the Scalability of Management Operations**

| No. of clients        | TCP    | Elan4  |
|-----------------------|--------|--------|
| Create (milliseconds) |        |        |
| 1                     | 28.114 | 27.669 |
| 2                     | 28.401 | 28.248 |
| 3                     | 28.875 | 28.750 |
| 4                     | 28.892 | 28.710 |
| 5                     | 29.481 | 29.123 |
| 6                     | 29.611 | 29.410 |
| Resize (milliseconds) |        |        |
| 1                     | 0.192  | 0.141  |
| 2                     | 0.248  | 0.187  |
| 3                     | 0.330  | 0.201  |
| 4                     | 0.274  | 0.180  |
| 5                     | 0.331  | 0.226  |
| 6                     | 0.338  | 0.213  |

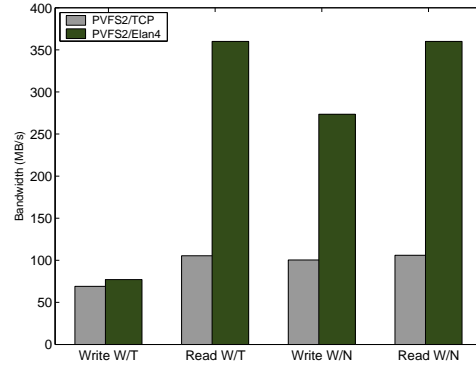
to resize a file by as much as  $125\mu s$  (37%) for up to 6 clients. However, the improvement on the time to create a file is just marginal compared to the total time. This is because the time in allocating the storage spaces at the PVFS2 server for the new file, though small, still dominates over the communication between the client and the server. On the other hand, once the file is created, the time for the operations that update the file metadata, as represented by the resize operation, can be reduced by the PVFS2 implementation over Elan4. Therefore PVFS2 implementation over Elan4 is also beneficial to the scalability of MPI-IO management operations.

#### 7.4. Performance of MPI-Tile-IO

MPI-Tile-IO [24] is a tile reading MPI-IO application. It tests the performance of tiled access to a two dimensional dense dataset, simulating the type of workload that exists in some visualization applications and numerical applications. Four of eight nodes are used as server nodes and the other four as client nodes running MPI-tile-IO processes. Each process renders a  $2 \times 2$  array of displays, each with  $1024 \times 768$  pixels. The size of each element is 32 bytes, leading to a file size of 96MB.

PVFS2 provides two different modes for its IO servers: trovesync and notrovesync. The former is the default mode in which IO servers perform `fsync` operations to flush its underlying file system buffer cache; the latter allows the IO servers to take the cache effects of the local file system for better performance. We have evaluated both the read and write performance of mpi-tile-io over PVFS2/Elan4 under both modes. As shown in Fig. 9, compared to PVFS2/TCP, PVFS2/Elan4 improves MPI-Tile-IO write bandwidth by 170% with

server side caching effects (under notrovesync mode, W/N), and 12% without caching effects (under trovesync mode, W/T). On the other hand, MPI-Tile-IO read bandwidth is improved by about 240% with or without server side caching effects. These results indicate our implementation is indeed able to leverage the performance benefits of Quadrics mechanisms into PVFS2: when the server disk access is a bottleneck, it improves the write performance with its zero-copy user-level communication which competes less with the disk access for CPU time; when the server disk access is not a primary bottleneck, it improves both the read and write bandwidth significantly.



**Fig. 9. Performance of MPI-Tile-IO Benchmark**

#### 7.5. Performance of NAS BT-IO

The BT-IO benchmarks are developed at NASA Ames Research Center based on the Block-Tridiagonal problem of the NAS Parallel Benchmark suite. These benchmarks test the speed of parallel IO capability of high performance computing applications. The entire data set undergoes complex decomposition and partition, eventually distributed among many processes, more details available in [26]. We have used four of eight nodes used as server nodes and the other four as client nodes. The BT-IO problem size class A is evaluated. Table 5 shows the BT-IO performance of PVFS2/Elan4 and PVFS2/TCP, along with the performance of BT benchmark without IO access. Compared to pure BT benchmark, the BT-IO benchmark has only 2.12 seconds extra IO time when accessing a PVFS2/Elan4 file system, but 5.38 seconds IO time when accessing a PVFS2/TCP file system. With PVFS2/Elan4, the IO time of BT-IO is reduced by 60% compared to PVFS2/TCP.

**Table 5. Performance of BT-IO Benchmark (seconds)**

| Type        | Duration | IO Time |
|-------------|----------|---------|
| No IO       | 61.71    | ---     |
| BT/IO Elan4 | 63.83    | 2.12    |
| BT/IO TCP   | 67.09    | 5.38    |

## 8. Related Work

Previous research have studied the benefits of using user-level communication protocols to parallelize IO access to storage servers. Zhou et. al. [30] have studied the benefits of VIA networks in database storage. Wu et. al. [27] have described their work on InfiniBand over PVFS1 [20]. DeBergalis et. al. [10] have further described a file system, DAFS, built on top of networks with VIA-like semantics. Our work is designed for Quadrics Interconnects over PVFS2 [1].

Models to support client/server communication and provide generic abstractions for transport layer over different networks have been described in [29, 17, 6]. Our work explores the ways to overcome Quadrics static process/communication model and optimize the transport protocols with Quadrics event mechanisms. Ching et. al [7] have implemented list IO in PVFS1 and evaluated its performance over TCP/IP. Wu et. al [28] have studied the benefits of leveraging InfiniBand hardware scatter/gather operations to optimize non-contiguous IO access in PVFS1. Our work exploits a communication mechanism with a single event chained to multiple RDMA to support zero-copy non-contiguous network IO over Quadrics.

## 9. Conclusions

In this paper, we have examined the feasibility of designing a parallel file system over Quadrics [23] to take advantage of its user-level communication and RDMA operations. PVFS2 [1] is used as the parallel file system platform in this work. The challenging issues in supporting PVFS2 on top of Quadrics interconnects are identified. Accordingly, strategies have been designed to overcome these challenges, such as constructing a client-server connection model, designing the PVFS2 transport layer over Quadrics RDMA read and write, and providing efficient non-contiguous network IO support. The performance of our implementation is compared to that of PVFS2/TCP over Quadrics IP implementation. Our experimental results indicate that: the performance of PVFS2 can be significantly improved with Quadrics user-level protocols and RDMA capabilities. Compared to a PVFS2 implementation over TCP/IP over Quadrics,

our implementation improves the aggregated read performance by more than 140%. It is also able to deliver significant performance improvement in terms of IO access to application benchmarks such as mpi-tile-io [24] and BT-IO [26]. To the best of our knowledge, this is the first high performance design and implementation of a user-level parallel file system, PVFS2, over Quadrics interconnects.

In future, we intend to leverage more features of Quadrics to support PVFS2 and study their possible benefits to different aspects of parallel file system. For example, we intend to study the feasibility of off-loading PVFS2 communication-related processing into Quadrics programmable network interface to free up more host CPU computation power for disk IO operations. We also intend to study the benefits of integrating Quadrics NIC memory into PVFS2 memory hierarchy, such as data caching with client and/or server-side NIC memory.

## Acknowledgment

We gratefully acknowledge Dr. Pete Wyckoff from Ohio Supercomputing Center and Dr. Jiesheng Wu from Ask Jeeves, Inc for many technical discussions. We would like to thank members from the PVFS2 team for their technical help. Furthermore, We also would like to thank members Drs Daniel Kidger and David Addison from Quadrics, Inc for their valuable technical support.

## References

- [1] The Parallel Virtual File System, version 2. <http://www.pvfs.org/pvfs2>.
- [2] The Public Netperf Homepage. <http://www.netperf.org/netperf/NetperfPage.html>.
- [3] J. Beecroft, D. Addison, F. Petrini, and M. McLaren. QsNet-II: An Interconnect for Supercomputing Applications. In *the Proceedings of Hot Chips '03*, Stanford, CA, August 2003.
- [4] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, 1995.
- [5] D. Bonachea, C. Bell, P. Hargrove, and M. Welcome. GASNet 2: An Alternative High-Performance Communication Interface, Nov. 2004.
- [6] P. H. Carns, W. B. Ligon III, R. Ross, and P. Wyckoff. BMI: A Network Abstraction Layer for Parallel I/O, 2004.
- [7] A. Ching, A. Choudhary, W. Liao, R. Ross, and W. Gropp. Noncontiguous I/O through PVFS. In *Proceedings of the IEEE International Conference on Cluster Computing*, Chicago, IL, September 2002.
- [8] Cluster File System, Inc. Lustre: A Scalable, High Performance File System. <http://www.lustre.org/docs.html>.

- [9] A. M. David Nagle, Denis Serenyi. The Panasas ActiveScale Storage Cluster – Delivering Scalable High Bandwidth Storage. In *Proceedings of Supercomputing '04*, November 2004.
- [10] M. DeBergalis, P. Corbett, S. Kleiman, A. Lent, D. Noveck, T. Talpey, and M. Wittle. The Direct Access File System. In *Proceedings of Second USENIX Conference on File and Storage Technologies (FAST '03)*, 2003.
- [11] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach*. The IEEE Computer Society Press, 1997.
- [12] J. Huber, C. L. Elford, D. A. Reed, A. A. Chien, and D. S. Blumenthal. PPFS: A High Performance Portable Parallel File System. In *Proceedings of the 9th ACM International Conference on Supercomputing*, pages 385–394, Barcelona, Spain, July 1995. ACM Press.
- [13] IBM Corp. IBM AIX Parallel I/O File System: Installation, Administration, and Use. Document Number SH34-6065-01, August 1995.
- [14] Infiniband Trade Association. <http://www.infinibandta.org>.
- [15] Intel Scalable Systems Division. Paragon System User's Guide, May 1995.
- [16] R. Latham, R. Ross, and R. Thakur. The impact of file systems on mpi-io scalability. In *Proceedings of the 11th European PVM/MPI Users' Group Meeting (Euro PVM/MPI 2004)*, pages 87–96, September 2004.
- [17] 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 *In SAN-02 Workshop (in conjunction with HPCA)*, February 2003.
- [18] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, Jul 1997.
- [19] N. Nieuwejaar and D. Kotz. The Galley Parallel File System. *Parallel Computing*, (4):447–476, June 1997.
- [20] P. H. Carns and W. B. Ligon III and 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, October 2000.
- [21] D. A. Patterson, G. Gibson, and R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, Chicago, IL, 1988.
- [22] F. Petrini, W.-C. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics Network: High Performance Clustering Technology. *IEEE Micro*, 22(1):46–57, January-February 2002.
- [23] Quadrics, Inc. Quadrics Linux Cluster Documentation.
- [24] R. B. Ross. Parallel i/o benchmarking consortium. <http://www-unix.mcs.anl.gov/rross/pio-benchmark/html/>.
- [25] R. Thakur, W. Gropp, and E. Lusk. On Implementing MPI-IO Portably and with High Performance. In *Proceedings of the 6th Workshop on I/O in Parallel and Distributed Systems*, pages 23–32. ACM Press, May 1999.
- [26] P. Wong and R. F. Van der Wijngaart. NAS Parallel Benchmarks I/O Version 2.4. Technical Report NAS-03-002, Computer Sciences Corporation, NASA Advanced Supercomputing (NAS) Division.
- [27] J. Wu, P. Wychoff, and D. K. Panda. PVFS over InfiniBand: Design and Performance Evaluation. In *Proceedings of the International Conference on Parallel Processing '03*, Kaohsiung, Taiwan, October 2003.
- [28] J. Wu, P. Wychoff, and D. K. Panda. Supporting Efficient Noncontiguous Access in PVFS over InfiniBand. In *Proceedings of Cluster Computing '03*, Hong Kong, December 2004.
- [29] R. Zahir. Lustre Storage Networking Transport Layer. <http://www.lustre.org/docs.html>.
- [30] 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.