Sockets Direct Protocol over InfiniBand in Clusters: Is it Beneficial?

PAVAN BALAJI, SUNDEEP NARRAVULA, KARTHIKEYAN VAIDYANATHAN, SAVITHA KRISHNAMOORTHY, JIESHENG WU AND DHABALESWAR K. PANDA

Technical Report
OSU-CISRC-10/03-TR54

Sockets Direct Procotol over InfiniBand in Clusters: Is it Beneficial? *

P. Balaji S. Narravula K. Vaidyanathan S. Krishnamoorthy J. Wu D. K. Panda

Computer and Information Science,
The Ohio State University,
2015 Neil Avenue,
Columbus, OH43210
{balaji, narravul, vaidyana, savitha, wuj, panda}@cis.ohio-state.edu

Abstract

InfiniBand has been recently standardized by the industry to design next generation high-end clusters for both data-center and high performance computing domains. Though InfiniBand has been able support low latency and high bandwidth, traditional sockets based applications have not been able to take advantage of this; this is mainly attributed to the multiple copies and kernel context switches associated with the traditional TCP/IP protocol stack. The Sockets Direct Protocol (SDP) had been proposed recently in order to enable sockets based applications to take advantage of the enhanced features provided by InfiniBand Architecture.

In this paper, we study the benefits and limitations of an implementation of SDP. We first analyze the performance of SDP based on a detailed suite of micro-benchmarks. Next, we evaluate it on two different real application domains: (1) A multi-tier Data-Center environment and (2) A Parallel Virtual File System (PVFS). Our micro-benchmark results show that SDP is able to provide up to 2.7 times better bandwidth as compared to the native sockets implementation over InfiniBand (IPoIB) and significantly better latency for large message sizes. Our experimental results also show that SDP is able to achieve a considerably higher performance (improvement of up to 2.4 times) as compared to IPoIB in the PVFS environment. In the data-center environment, SDP outperforms IPoIB for large file transfers inspite of currently being limited by a high connection setup time. However, this limitation is entirely implementation specific and as the InfiniBand software and hardware products are rapidly maturing, we expect this limitation to be overcome soon. Based on this, we have shown that the projected performance for SDP, without the connection setup time, can outperform IPoIB for small message transfers as well.

Keywords: *Sockets Direct Protocol (SDP), InfiniBand, Data-Center, PVFS*

1 Introduction

Cluster systems are becoming increasingly popular in various application domains mainly due to their high performance-to-cost ratio. Out of the current Top 500 Supercomputers, 149 systems are clusters [20]. Cluster systems are now present at all levels of performance, due to the increasing performance of commodity processors, memory and network technologies.

During the last few years, the research and industry communities have been proposing and implementing user-level communication systems to address some of the problems associated with the traditional networking protocols. The Virtual Interface Architecture (VIA) [14, 4, 17] was proposed earlier to standardize these efforts. InfiniBand Architecture (IBA) [9] has been recently standardized by the industry to design next generation high-end clusters.

Earlier generation protocols such as TCP/IP [26, 29] relied upon the kernel for processing the messages. This caused multiple copies and kernel context switches in the critical message passing path. Thus, the communication latency was high. Researchers have been looking at alternatives to increase the communication performance delivered by clusters in the form of low-latency and high-bandwidth user-level protocols such as FM [22] and GM [18] for Myrinet [13], EMP [24, 25] for Gigabit Ethernet [19], etc.

These developments are reducing the gap between the performance capabilities of the physical network and that obtained by the end users. While this approach is good for developing new applications, it might not be so beneficial for the already existing sockets applications which were de-

*This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506, and National Science Foundation's grants #EIA-9986052, #CCR-0204429, and #CCR-0311542.

veloped over a span of several years. A number of applications have been developed on kernel-based protocols such as TCP/UDP using the sockets interface. To support such applications on high performance user-level protocols without any changes to the application itself, researchers have come up with a number of techniques. These techniques include user-level sockets layers over high performance protocols [10, 21, 23, 11].

Sockets Direct Protocol (SDP) [6] is an InfiniBand Architecture specific protocol defined by the InfiniBand Trade Association. SDP was proposed along the same lines as the user-level sockets layers; to allow a smooth transition to deploy existing sockets based applications on to clusters connected with InfiniBand while sustaining most of the performance provided by the base network.

At this point, the following open questions arise:

- What kind of benefits can be expected from the current Sockets Direct Protocol implementation?
- What are the trade-offs associated with such an implementation?

In this paper, we study the benefits and limitations of an implementation of SDP. We first analyze the performance of SDP based on a detailed suite of micro-benchmarks. Next, we evaluate it on two different real application domains:

- A multi-tier Data-Center environment
- A Parallel Virtual File System (PVFS)

Our micro-benchmark results show that SDP is able to provide up to 2.7 times better bandwidth as compared to the native sockets implementation over InfiniBand (IPoIB) and significantly better latency for large message sizes. Our experimental results also show that SDP is able to achieve a considerably high performance (improvement of up to a factor of 2.4) compared to the native sockets implementation in the PVFS environment. In the data-center environment, SDP outperforms IPoIB for large file transfers in-spite of currently being limited by a high connection setup time. However, this limitation is entirely implementation specific and as the InfiniBand software and hardware products are rapidly maturing, we expect this limitation to be overcome soon. Based on this, we have shown that the projected performance for SDP, without the connection setup time, can outperform IPoIB for small message transfers as well.

The remaining part of the paper is organized as follows: Section 2 provides a brief background about InfiniBand, the modern user-level sockets implementations and the Sockets Direct Protocol (SDP) implementation. In Section 3 we discuss the software infrastructure we used; in particular about Multi-Tier Data-Center environments and Parallel Virtual File System (PVFS). Section 4 deals with the evaluation of a number of micro-benchmarks to evaluate the ideal case benefits of SDP over the native sockets implementation over InfiniBand. In Section 5, we discuss the Multi-Tier Data-Center environment we used in more detail. We present the evaluation of PVFS in Section 6 and conclude the paper in Section 7.

2 Background

In this section we provide a brief background about InfiniBand Architecture, the modern user-level sockets implementation and the Sockets Direct Protocol (SDP) implementation.

2.1 InfiniBand Architecture (IBA)

InfiniBand Architecture (IBA) is an industry standard that defines a System Area Network (SAN) to design clusters offering low latency and high bandwidth. A typical IBA cluster consists of switched serial links for interconnecting both the processing nodes and the I/O nodes. The IBA specification defines a communication and management infrastructure for both inter-processor communication as well as inter and intra node I/O. IBA also defines built-in QoS mechanisms which provide virtual lanes on each link and define service levels for individual packets.

In an InfiniBand network, processing nodes and the I/O nodes are connected to the fabric by Host Channel Adapters (HCA) and Target Channel Adapters (TCA). HCAs are associated with processing nodes and their semantic interface to consumers is specified in the form of InfiniBand Verbs. TCAs connect I/O nodes to the fabric and have interfaces to consumers that are implementation specific and are not defined in the IBA specifications. Channel Adapters usually have programmable DMA engines with protection features.

IBA mainly aims at reducing the system processing overhead by decreasing the number of copies associated with a message transfer and removing the kernel from the critical message passing path. The InfiniBand communication stack consists of different layers. The interface presented by Channel Adapters to consumers belongs to the transport layer. A Queue Pair (QP) based model is used in this interface.

Each Queue Pair is a communication endpoint. A Queue Pair consists of a send queue and a receive queue. Two QPs on different nodes can be connected to each other to form a logical bi-directional communication channel. An application can have multiple QPs. Communication requests are initiated by posting Work Queue Requests (WQRs) to these queues. Each WQR is associated with one or more pre-registered buffers from which data is either transferred (for a send WQR) or received (receive WQR). Further, the application can either choose to be signaled on the completion

of a WQR using the Signaled (SG) request or alternatively choose an Unsignaled (USG) request. When the HCA completes the processing of a signaled request, it places an entry in the Completion Queue (CQ) called as the Completion Queue Entry (CQE).

The consumer application can poll on the CQ associated with the work request to check for completion. There is also the feature of triggering event handlers whenever a completion occurs. For Unsignaled requests, no completion event is returned to the user. However, depending on the implementation, the driver cleans up the Work Queue Request from the appropriate Queue Pair on completion.

2.1.1 IBA Communication Models

IBA supports two types of communication semantics: Channel Semantics (Send-Receive communication model) and Memory Semantics (RDMA communication model).

In channel semantics, each send request has a corresponding receive request at the remote end. Thus there is a one-to-one correspondence between every send and receive operation. Failure to post a descriptor on the remote node results in the message being dropped and if the connection is reliable, it might even result in the breaking of the connection.

In memory semantics, Remote Direct Memory Access (RDMA) operations are used. These operations are transparent at the remote end since they do not require a receive descriptor to be posted. In this semantics, the send request itself contains both the virtual address for the local transmit buffer as well as that for the receive buffer on the remote end. The RDMA operations are available with the Reliable Connection (RC) service type.

The IBA specifications does not provide different primitives for the channel semantics and the memory semantics. It is the "opcode" entry in the WQR, which distinguishes between the channel semantics and the memory semantics. Most other entries in the WQR are common for both the Send-Receive model as well as the RDMA model, except an additional remote buffer virtual address (and other related entries) which has to be specified for RDMA operations.

There are two kinds of RDMA operations: RDMA Write and RDMA Read. In the RDMA Write model, the initiator directly writes data into the remote node's memory. Similarly, in the RDMA Read model, the initiator directly reads data from the remote node's memory.

2.1.2 IBA Communication Protocols

InfiniBand is an emerging standard intended as an interconnect for processor and I/O systems and devices. There are three main types of traffic over IBA fabrics. The first type of traffic is IBA native protocols such as VAPI [5], IBAL (InfiniBand Access Layer) [1], uDAPL (User-Level Direct Access Transport APIs) [7] and kDAPL (Kernel-Level Direct Access Transport APIs) [8]. These interfaces are low-level APIs for IBA. Applications can be programmed using these APIs to take full advantage of IBA user level networking and RDMA features. IP is one type of traffic (and a very important one) that could use this interconnect. InfiniBand would benefit greatly from a standardized method of handling IP traffic. IPoIB provides standardized IP encapsulation over IBA fabrics as defined by the IETF Internet Area IPoIB working group [2]. The IPoIB project [3] implements this proposed standard as a layer-2 Linux network driver. The primary responsibilities of the driver are performing address resolution to map IPv4 and IPv6 addresses to InfiniBand Unreliable Datagram (UD) address vectors, the management of multicast membership, and the transmission and reception of IPoIB protocol frames. Sockets Direct Protocol (SDP) is another type of traffic over IBA fabrics. Details about SDP are discussed in Section 2.3.

2.2 User-Level Sockets

With the advent of high performance interconnects like Myrinet, GigaNet, Quadrics and InfiniBand, the communication overhead has shifted to the software message passing system at the sender and the receiver side. Traditional communication architecture based on TCP/IP are unable to take advantage of these developments mainly due to the protocol processing overhead which involve multiple copies, kernel context switches and interrupts.

To avoid these processing overheads, researchers came up with user level protocols like VIA, FM, GM, EMP. The goal of user level protocols is to reduce the bottleneck in the software message passing, by removing the kernel from the critical path of communication. User Level Protocols achieve high performance by gaining direct access to the network interface in a protected manner. While User level protocols are beneficial for new applications, existing applications written using the sockets interface, have not been able to take advantage of these protocols. In order to allow these applications achieve the better performance provided by these networks, researchers came up with a number of solutions including user level sockets.

The basic idea of a user level sockets is to create a pseudo sockets-like interface to the application. This sockets layer is designed to serve two purposes: a) to provide a smooth transition to deploy existing application on to clusters connected with high performance networks and b) to sustain most of the performance provided by the high performance protocols.

2.3 Sockets Direct Protocol (SDP)

Sockets Direct Protocol (SDP) is an IBA specific protocol defined by the Software Working Group (SWG) of the

InfiniBand Trade Association [9]. The design of SDP is mainly based on two architectural goals:

- Maintain traditional sockets SOCK_STREAM semantics as commonly implemented over TCP/IP. Issues include graceful closing of connections, ability to use TCP port space, IP addressing (IPv4, IPv6), Connecting/Accepting connect model, Out-of-Band data (OOB) and support for common socket options

- Support for byte-streaming over a message passing protocol, including kernel bypass data transfers and zero-copy data transfers

The SDP specifications focuses specifically on the wire protocol, finite state machine and packet semantics. Operating system issues, etc can be implementation specific. It is to be noted that SDP supports only SOCK_STREAM or Streaming sockets semantics and not SOCK_DGRAM (datagram) or other socket semantics.

2.3.1 SDP Overview

SDP's Upper Layer Protocol (ULP) interface is a byte-stream that is layered on top of InfiniBand's Reliable Connection (RC) message-oriented transfer model. The mapping of the byte stream protocol to InfiniBand message-oriented semantics was designed to enable ULP data to be transfered by one of two methods: through intermediate private buffers (Bcopy) or directly between ULP buffers (Zcopy).

A mix of InfiniBand Send and RDMA mechanisms are used to transfer ULP data. Zcopy uses RDMA reads or writes, transferring data between RDMA buffers (which typically belong to the ULP). Bcopy uses InfiniBand sends, transferring data between send and receive private buffers.

SDP has two types of buffers:

Private Buffers: Used for transmission of all SDP messages and ULP data that is to be copied into the receive ULP buffer. The Bcopy data transfer mechanism is used for this traffic.

RDMA Buffers: Used when performing Zcopy data transfer. ULP data is intended to be RDMAed directly from the Data Source's ULP buffer to the Data Sink's ULP buffer.

An implementation dependent parameter defined as the Bcopy Threshold is used to abstractly define the results of the policy decision. For the Bcopy implementation, SDP relies on a flow control mechanism similar to the TCP Sliding Window protocol, i.e., the sender keeps sending data till the window is full. When the application reads data from the socket buffer, the data sink sends a control message back to the data source updating its window size.

Figure 1 shows SDP in relation to the other Architecture layers in InfiniBand.

SDP specifications also specify two additional control messages known as "Buffer Availability Notification" messages.

Sink Avail Message: If the data sink has already posted a receive buffer and the data source has not sent the data message yet, the data sink does the following steps: (1) Registers the receive user-buffer (for large message reads) and (2) Sends a "Sink Avail" message containing the receive buffer handle to the source. The Data Source on a data transmit call, uses this receive buffer handle to directly RDMA write the data into the receive buffer.

Source Avail Message: If the data source has already posted a send buffer and the available SDP window is not large enough to contain the buffer, it does the following two steps: (1) Registers the transmit user-buffer (for large message sends) and (2) Send a "Source Avail" message containing the transmit buffer handle to the data sink. The Data Sink on a data receive call, uses this transmit buffer handle to directly RDMA read the data into the receive buffer.

2.3.2 SDP Implementation

The current implementation of SDP follows most of the specifications provided above. There are two major deviations from the specifications in this implementation.

- **Buffer Availability Notification:** The current implementation does not support "Source Avail" and "Sink Avail" messages.

- **Zcopy implementation:** The current implementation does not support "Zcopy". All data transfer is done through the Bcopy mechanism. This limitation can also be considered as part of the previous limitation, since they are always used together.

3 Software Infrastructure

We have carried out the evaluation of SDP on two different software infrastructures: Multi-Tier Data Center environment and the Parallel Virtual File System (PVFS). In this section, we discuss each of these in more detail.

3.1 Multi-Tier Data Center environment

More and more people are using web interfaces for a wide range of services. Scalability of these systems is a very important factor.

A typical Multi-tier Data-center has as its first tier, a cluster of nodes known as the edge nodes. These nodes can be thought of as switches (up to the 7th layer) providing load balancing, security, caching etc. The main purpose of this tier is to help increase the performance of the inner tiers.

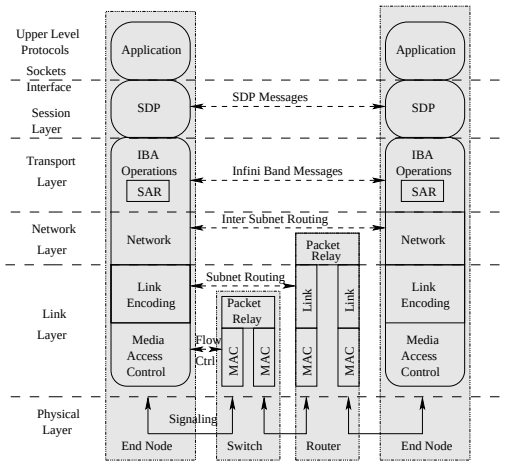


Figure 1. Sockets Direct Protocol Architecture (Courtesy: InfiniBand Specifications Volume I)

The next tier usually contains the web-servers and application servers. These nodes apart from serving static content, can fetch dynamic data from other sources and serve that data in presentable form. The last tier of the Data-Center is the database tier. It is used to store persistent data. This tier is usually I/O intensive. Figure 2 shows a typical data-center setup.

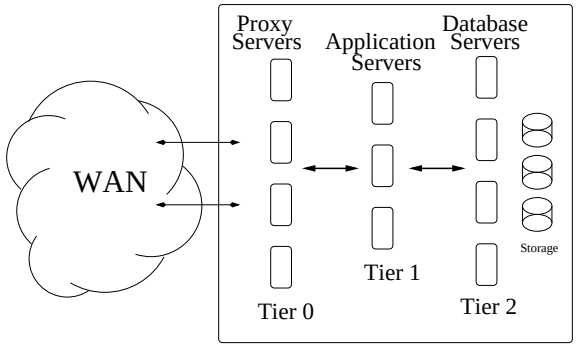


Figure 2. A Typical 3-Tier Data-Center

A request from a client is received by the edge servers. If this request can be serviced from the cache, it is. Otherwise, it is forwarded to the Web/Application servers. Static requests are serviced by the web servers by just returning the requested file to the client via the edge server. This content may be cached at the edge server so that subsequent requests to the same static content may be served from the cache. The Application tier nodes handle the Dynamic content. The type of applications this tier includes range from mail servers to directory services to ERP software. Any request that needs a value to be computed, searched, analyzed or stored uses this tier. The back end database servers are responsible for storing data persistently and responding to queries. These nodes are connected to persistent storage systems. Queries to the database systems can be anything ranging from a simple seek of required data to performing joins, aggregation and select operations on the data.

3.2 Parallel Virtual File System (PVFS)

Parallel Virtual File System (PVFS) [15] is one of the leading parallel file systems for Linux cluster systems today. It was designed to meet the increasing I/O demands of parallel applications in cluster systems. Figure 3 demonstrates a typical PVFS environment. As demonstrated in the figure, a number of nodes in the cluster system can be configured as I/O servers and one of them (either an I/O server or an different node) as a metadata manager. It is possible for a node to host computations while serving as an I/O node.

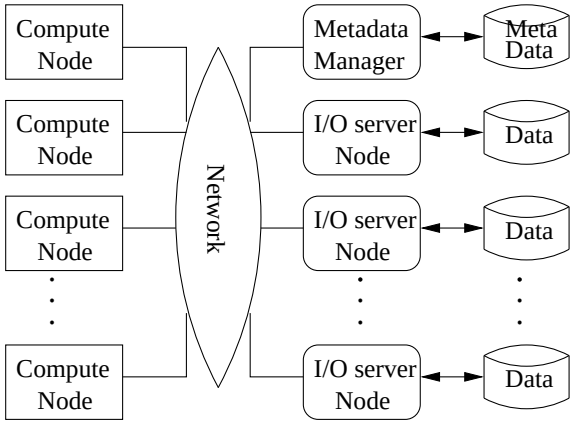


Figure 3. A Typical PVFS Setup

PVFS achieves high performance by striping files across a set of I/O server nodes allowing parallel accesses to the data. It 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 the compute nodes, in particular the read and write requests. Thus, data is transferred directly between the I/O servers and the compute nodes.

A manager daemon runs on a metadata manager node. It handles metadata operations involving file permissions, truncation, file stripe characteristics, and so on. Metadata is also stored on the local file system. The metadata manager provides a cluster-wide consistent name space to applications. In PVFS, the metadata manager does not participate in read/write operations.

PVFS supports a set of feature-rich interfaces, including support for both contiguous and non-contiguous accesses to both memory and files [16]. PVFS can be used with multiple APIs: a native API, the UNIX/POSIX API,

MPI-IO [27], and an array I/O interface called the Multi-Dimensional Block Interface (MDBI). The presence of multiple popular interfaces contributes to the wide success of PVFS in the industry.

4 SDP Micro-Benchmark Results

In this section, we compare the ideal case performance achievable by SDP and the native sockets implementation over InfiniBand (IPoIB) using a number of micro-benchmarks tests. In Sections 5 and 6, we study the performance achieved by SDP and IPoIB in the Data-Center and PVFS environments respectively.

For all our experiments we used 2 clusters whose descriptions are as follows:

Cluster1: A cluster system consisting of 8 nodes built around SuperMicro SUPER P4DL6 motherboards and GC chipsets 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-x86-0.2.0-build-001. The adapter firmware version is fw-23108-rel-1_17_0000-rc12-build-001. We used the Linux RedHat 7.2 operating system.

Cluster2: A cluster consisting of 16 Dell Precision 420 nodes connected by Fast Ethernet. Each node has two 1GHz Pentium III processors, built around the Intel 840 chipset, which has four 32-bit 33-MHz PCI slots. These nodes are equipped with 512MB of SDRAM and 256K L2-level cache.

We used Cluster 1 for all experiments in this section.

4.1 Latency and Bandwidth

Figure 4a shows the one-way latency achieved by IPoIB, SDP and Send-Receive and RDMA Write communication models of native VAPI for various message sizes. SDP achieves a latency of around 28μs for 2 byte messages compared to a 30μs achieved by IPoIB and 7μs and 5.5μs achieved by the Send-Receive and RDMA communication models of VAPI. Further, with increasing message sizes, the difference between the latency achieved by SDP and that achieved by IPoIB tends to increase.

Figure 4b shows the uni-directional bandwidth achieved by IPoIB, SDP, VAPI Send-Receive and VAPI RDMA communication models. SDP achieves a throughput of up to 471Mbytes/s compared to a 169Mbytes/s achieved by IPoIB and 825Mbytes/s and 820Mbytes/s achieved by the Send-Receive and RDMA communication models of VAPI. We see that SDP is able to transfer data at a much higher rate as compared to IPoIB using a significantly lower portion of the host CPU. This improvement in the throughput and CPU is mainly attributed to the NIC offload of the transportation and network layers in SDP unlike that of IPoIB.

4.2 Multi-Stream Bandwidth

In the Multi-Stream bandwidth test, we use two machines and N threads on each machine. Each thread on one machine has a connection to exactly one thread on the other machine. Thus, N connections are established between these two machines. On each connection, the basic bandwidth test is performed. The aggregate bandwidth achieved by all the threads together within a period of time is calculated as the multi-stream bandwidth. Performance results with different numbers of streams are shown in Figure 5. We can see that SDP achieves a peak bandwidth of about 500Mbytes/s as compared to a 200Mbytes/s achieved by IPoIB.

The CPU Utilization for a 16Kbyte message size is also presented. The benchmark reveals that SDP can not only achieve high aggregated bandwidth, but also reduce the overall CPU utilization in most cases. The CPU utilization shown in this figure is the total CPU seen by all the threads.

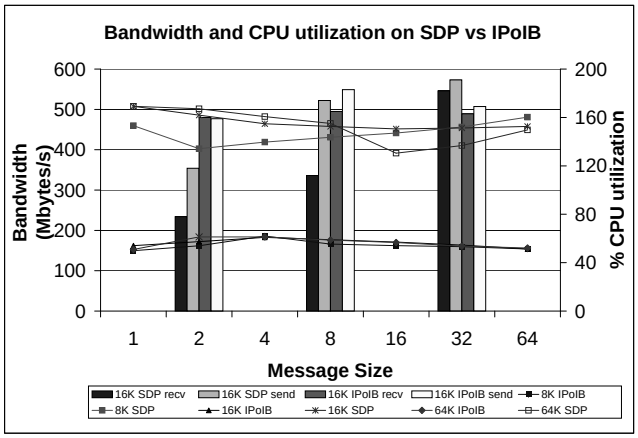
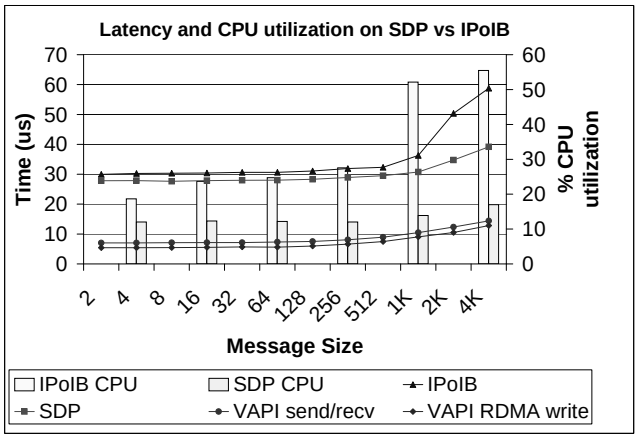


Figure 5. Multi-Stream Bandwidth

4.3 Hot-Spot Test

In the Hot-Spot test, multiple clients communicate with the same server. All clients and the server run on different machines. The communication pattern between any client and the server is the same pattern as in the basic latency test. That is, the server needs to receive messages from all the clients and sends messages to all clients as well. Thus, a hot-spot occurs on the server side. Figure 6 shows the one-way latency of IPoIB and SDP when communicating with a hot-spot server, for different numbers of clients. The server CPU utilization for a 16Kbyte message size is also



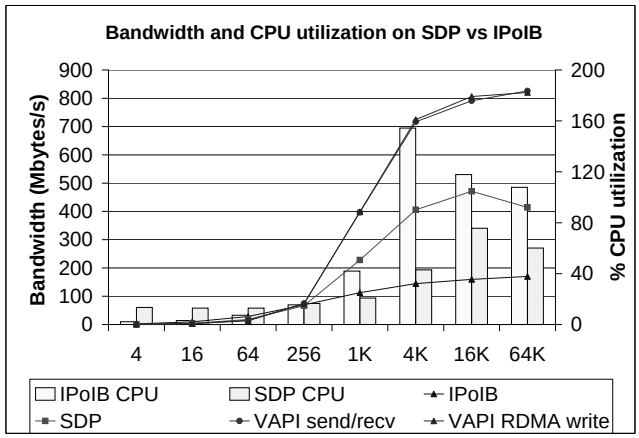


Figure 4. Micro-Benchmarks: (a) Latency, (b) Bandwidth

presented. We can see that as SDP scales well with the number of clients; its latency increasing by only a 138μs compared to 456μs increase with IPoIB for a message size of 16Kbytes. For the CPU utilization, we find that as number of nodes increase we get a tremendous improvement of more than a factor of 2, in terms of CPU utilization for SDP over IPoIB.

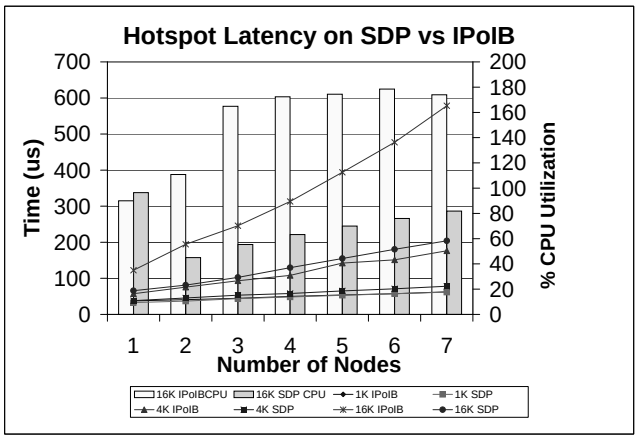


Figure 6. Hot-Spot Latency

4.4 Fan-in and Fan-out

In the Fan-in test, multiple clients from different nodes stream data to the same server. Similarly, in the Fan-out test, the same server streams data out to multiple clients. Figures 7a and 7b show the aggregate bandwidth observed by the server for different number of clients for the Fan-in and Fan-out tests respectively. We can see that for the Fan-in test, SDP reaches a peak aggregate throughput of 687Mbytes/s compared to a 237Mbytes/s of IPoIB. Similarly, for the Fan-out test, SDP reaches a peak aggregate throughput of 477Mbytes/s compared to a 175Mbytes/s of IPoIB. The server CPU utilization for a 16Kbyte message size is also presented. Both figures show similar trends in CPU utilization for SDP and IPoIB as the previous tests i.e., SDP performs about 60-70% better than IPoIB in CPU requirements.

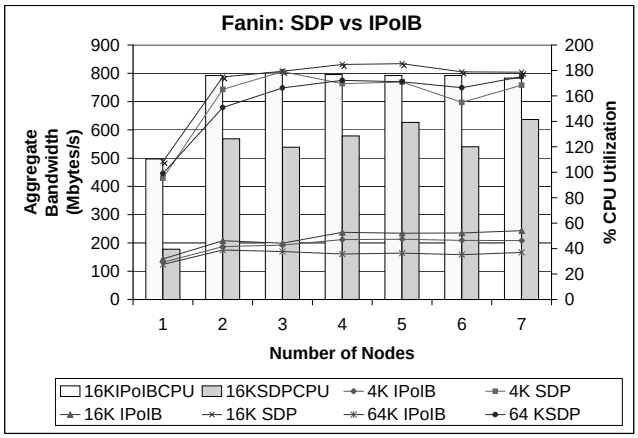
5 Data-Center Performance Evaluation

In this section, we analyze the performance of a 3-tier data-center environment over SDP while comparing it with the performance of IPoIB. For all experiments in this section, we used nodes in Cluster 1 (described in Section 4) for the data-center tiers. For the client nodes, we used the nodes in Cluster 2 for most experiments. We'll notify the readers at points in this paper when other nodes are used as clients.

5.1 Evaluation Methodology

In this section, we evaluate the amount of time taken by each request sent by the client. We analyze the time taken by the request in each tier of the data-center and nail down the bottlenecks of the data-center environment. We also show the potential benefits derivable from using SDP in this environment.

We set up a three-tier data-center test-bed to determine the performance characteristics of using SDP and IPoIB over InfiniBand. The first tier consists of the front-end proxies. For this we used the proxy module of *Apache 1.3.12*. The second tier consists of the web server and PHP application server modules of Apache, in order to service static and dynamic requests respectively. The third tier consists of the Database servers. For this, we used *MySQL servers* with master and slave server components to serve dynamic database queries. All the three tiers in the data-center reside on an InfiniBand network; the clients are connected to the



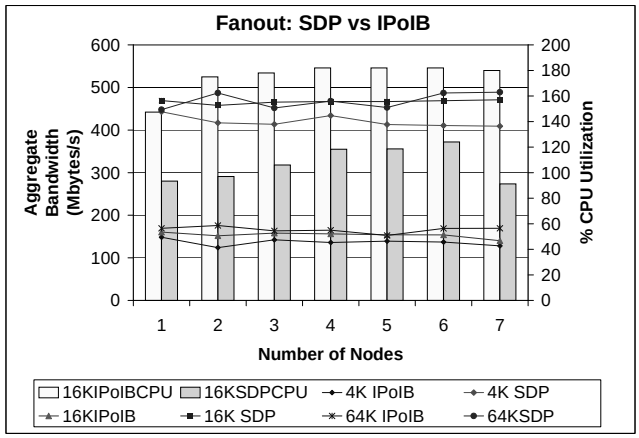


Figure 7. Micro-Benchmarks: (a) Fan-in, (b) Fan-out

data-center using a Fast Ethernet connection.

We evaluate the response time of the data-center using *Openload*, an open source client workload generator which sends requests (both static and dynamic) to the proxy at the first tier. We use a 20000 request subset of the world-cup trace [28] for our experiments. To generate requests amounting to different average file sizes, we scale the file sizes in the given trace linearly, while keeping the access pattern intact.

In our experiments, we evaluate two scenarios: requests from the client consisting of 100% static content (involving only the proxy and the web server) and requests from the client consisting of 100% dynamic content (involving all the three tiers in the data-center). "Openload" allows firing a mix of static and dynamic requests. However, the main aim of this paper was the analysis of the performance achievable by IPoIB and SDP. Hence, we only focused on these two scenarios (100% static and 100% dynamic content) to avoid dilution of this analysis with other aspects of the data-center environment such as workload characteristics etc.

For evaluating the scenario with 100% static requests, we used a test-bed with one proxy at the first tier and one web-server at the second tier. The client would fire requests one at a time, so as to evaluate the ideal case response time for the request.

For evaluating the scenario with 100% dynamic page requests, we set up the data center with the following configuration: Tier 1 consists of 3 Proxies, Tier 2 contains 2 servers which act as both web servers as well as application servers (running PHP) and Tier 3 with 3 MySQL Database Servers (1 Master and 2 Slave Servers).

The typical access pattern used for workload generation followed the transactional web benchmark TPC-W [12] as shown in Table 1. The benchmark is based on a business model that portrays the scenario of a whole-sale supplier. Some of the activities include entering, querying and checking the status of orders, monitoring various products by category, suppliers, region and territories and recording payments. We used this distribution for generating our dynamic request access pattern.

In the data-center set up for dynamic requests, we create a database consisting of a number of tables with varying sizes, attributes and relationship integrity based on the TPC-W specifications (Table 2). We also generate dynamic pages (PHP) with database accesses and updates. At the client end, we create a trace file with a series of database queries that consist of a mix of dynamic requests, some of which might be cached by the database server for a specified period of time.

Web Interaction	Shopping Mix
Browse	80%
Home	16%
New Products	5%
Best Sellers	5%
Product Details	17%
Search Requests	20%
Search Results	17%
Orders	20%
Shopping Cart	11.6%
Customer Registration	3%
Order Confirmation	3%
Cancel Orders	1%

Table 1. TPC-W Web traffic pattern

Table Name	Cardinality	Typical Row Length
Customers	1890	760
Orders	1531	220
Products	100	150
Categories	10	100
Suppliers	50	450
Territories	65	90

Table 2. TPC-W Database Specifications

5.2 Experimental Results

5.2.1 Analysis of Static Requests

We used a 20,000 request subset of the worldcup trace to come up with our base trace file. The weighted average of the file size was calculated as the ratio of the total amount of data requested in the trace file to the total number of requests. So, if the trace file contained 4 entries, with 3 of them requesting file "f1" of size "s1" and one of them requesting file "f2" of size "s2", the weighted average would be (3s1 + s2)/4.

As discussed earlier, to generate multiple traces with different average file sizes, we scale each file size with the ratio of the requested average file size and the current file size. For example, if the weighted average of the current trace is "a1" bytes and the requested average is "a2" bytes, the size of each file in the trace is increased (a2/a1) times.

Figure 8 shows the response times seen by the client for various average file sizes requested over IPoIB and SDP. As seen in the figure, the benefit obtained by SDP over IPoIB is quite minimal. In order to analyze the reason for this, we found the break-up of this response time in the proxy and web servers.

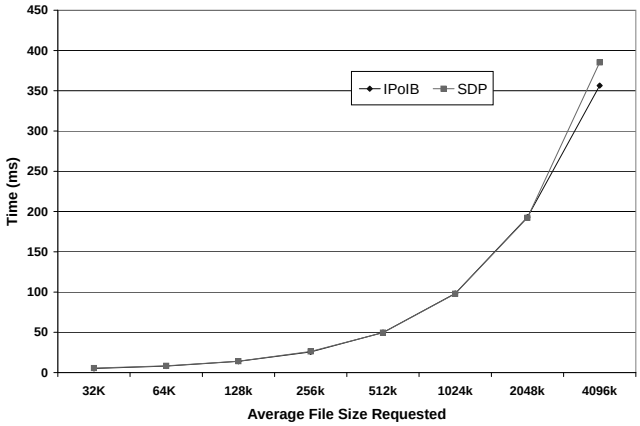


Figure 8. Client over Fast Ethernet: Response Time

Figure 9 shows the break-up of the response time for average file size requests of 64K and 128K. The "Web-Server Time" shown in the graph is the time duration for the back-end web-server to respond to the file request from the proxy. The "Proxy-Time" is the difference between the times spent by the proxy (from the time it gets the request to the time it sends back the response) and the time spent by the web-server. This value denotes the actual overhead of the proxy tier in the entire response time seen by the client. Similarly, the "Client-Time" is the difference between the times seen by the client and by the proxy.

We see that, the total response time perceived by the client is similar for both IPoIB and SDP. However, we see some benefit in the time taken at the web server when the communication within the the data-center is over SDP as compared to IPoIB. We analyzed the web server response times for various file sizes for both IPoIB and SDP.

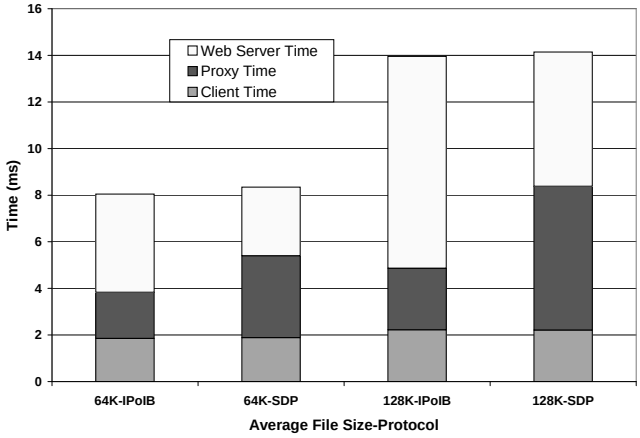


Figure 9. Response Time Split Up for Client over Fast Ethernet

Figure 10 shows the web server request servicing time for varying average file sizes. We observe that the web server over SDP is consistently better than IPoIB, implying that the web server over SDP can deliver better throughput. Further, this also implies that SDP can handle a given server load with lesser number of back-end web-servers as compared to an IPoIB based implementation due to the reduced "per-request-time" spent at the server.

Since the client connects to the data-center over fast ethernet, a possible reason for the comparability in the response times (of IPoIB and SDP) might be that the slow interconnect becomes the bottleneck for the transfer of the response message, i.e., the client is unable to accept the response at the rate at which the server is able to send the data.

To validate this hypothesis, we conducted experiments using our data-center test-bed with faster clients. Such clients may themselves be on high speed interconnects such as In-

finiBand or may become available due to Internet proxies, ISPs etc.

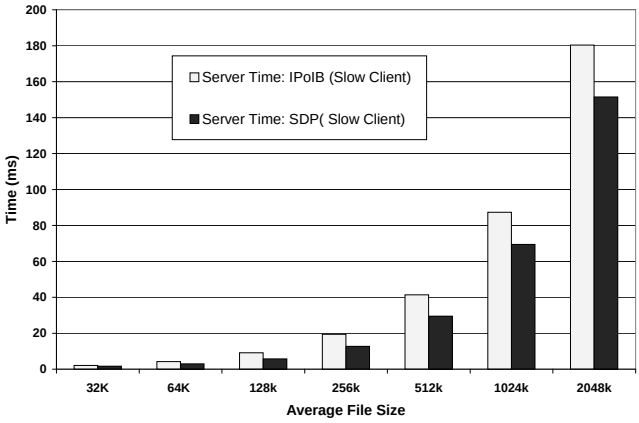


Figure 10. Response Times at the Web Server Tier for Slow Client

Figure 11 shows the client response times that is achievable using SDP and IPoIB in this new scenario which we emulated by having the clients request files over IPoIB (using InfiniBand). This figure clearly shows a better performance for SDP, as compared to IPoIB for large file transfers above 128K. To understand the lack of performance benefits for small files, we took a similar split up of the response time perceived by the client.

Figure 12 shows the splitup of the response time seen by the faster clients. We observe the same trend as seen with clients over Fast Ethernet. The "web-server time" reduces even in this scenario (Figure 13). However, it's quickly apparent from the figure that the time taken at the proxy is higher for SDP as compared to IPoIB. For a clearer understanding of this observation, we further evaluated the response time within the data-center by further breaking down the time taken by the proxy in servicing the request.

Figures 14a and 14b show a comprehensive breakup of the time spent at the proxy over IPoIB and SDP respectively. A comparison of this splitup for SDP with IPoIB shows a significant difference in the time for the the proxy to connect to the back-end server. This high connection time of the current SDP implementation, (about 500μs higher than IPoIB), makes the data-transfer related benefits of SDP imperceivable for low file size transfers.

The current implementation of SDP has inherent lower level function calls during the process of connection establishment, which form a significant portion of the connection latency. In order to hide this connection time overhead, researchers are proposing a number of techniques including persistent connections from the proxy to the back-end, allowing free connected Queue Pair (QP) pools, etc. Further, since this issue of connection setup time is completely im-

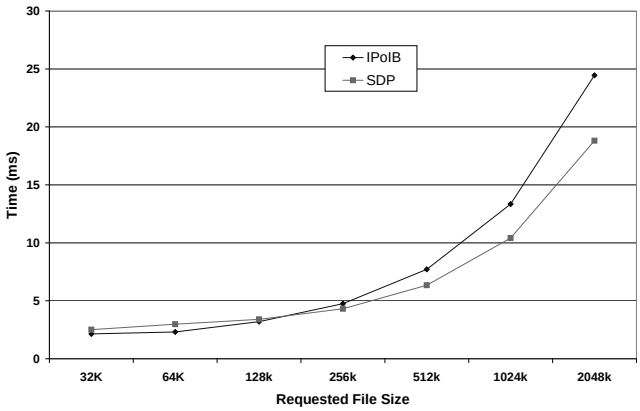


Figure 11. Fast Client Response Time

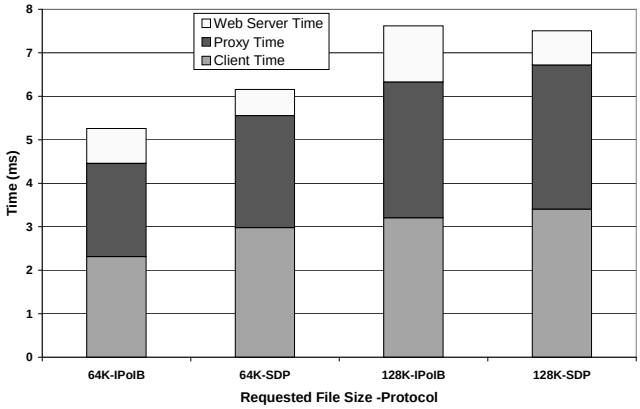


Figure 12. Response Time Split Up Times for Fast Clients

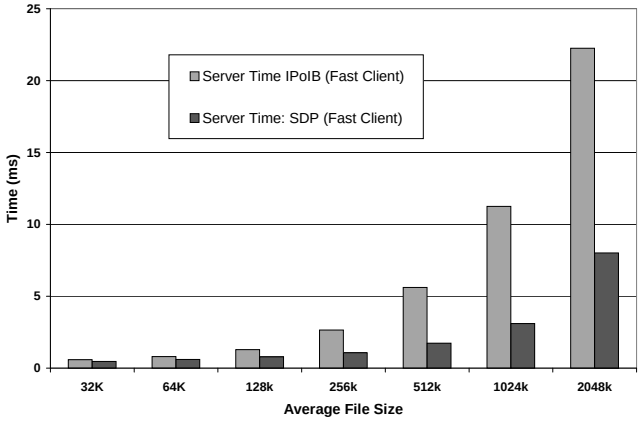


Figure 13. Response Times at the Web Server Tier for Fast Client

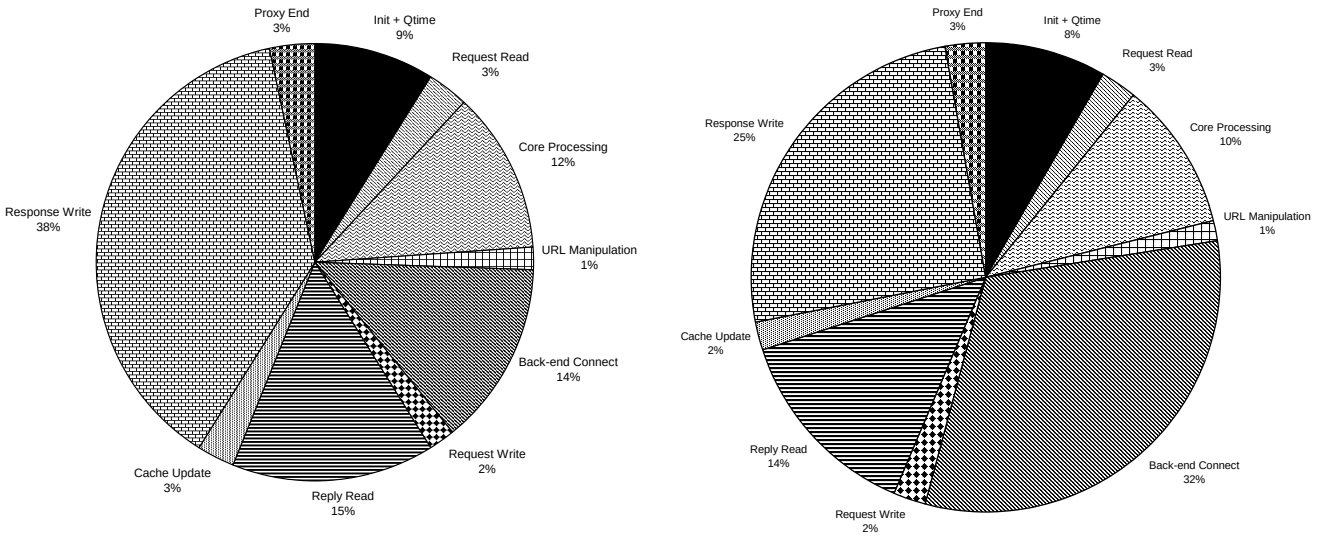


Figure 14. Proxy Split-up times: (a) IPoIB, (b) SDP

plementation specific, we tried to estimate the (projected) performance SDP can provide if the connection time bottleneck was resolved.

Figure 15 shows the projected response times of the fast client, without the connection time overhead. Assuming a future implementation of SDP with lower connection time, we see that SDP is able to give significant response time benefits as compared to IPoIB even for small file size transfers.

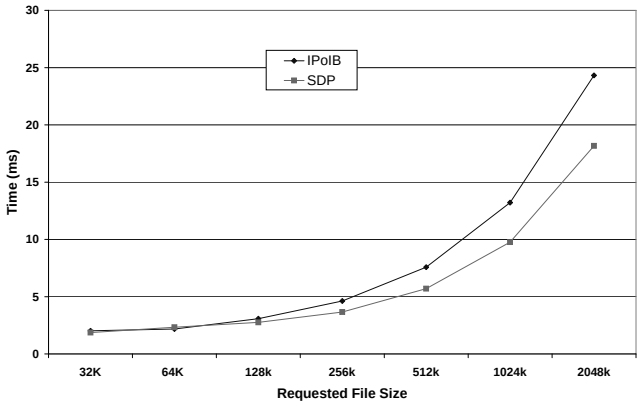


Figure 15. Fast Client Response Time without Connection Time

5.2.2 Analysis of Dynamic Requests

Figure 16 shows the processing and communication overhead seen by each of the tiers across various sets of dynamic requests over IPoIB and SDP. There is no significant benefit from SDP over IPoIB in terms of the response time. However, the interesting observation is that, consistently across all mixtures, the back-end (MySQL) time consumes most of the request processing time.

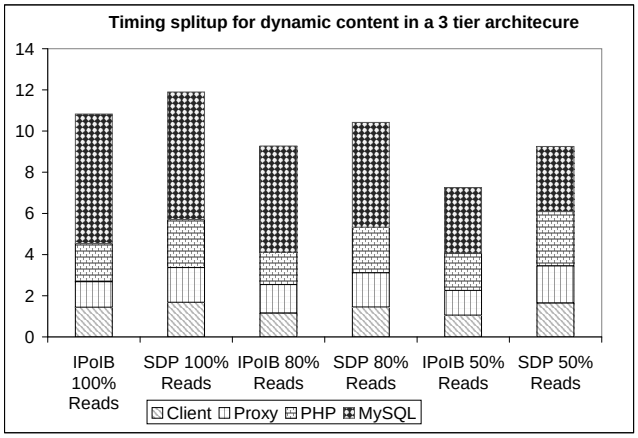


Figure 16. Timing Splitup for dynamic content (Slow Client)

When we move from 100% reads case to 80% reads to 50% reads, the response time seen by the client decreases. For SDP, the service time of the back-end database server is always less than IPoIB. This is similar to the trend observed with Web-Servers for static file requests.

To evaluate the impact of Fast Ethernet on the lack of performance benefits for SDP, we have carried out this test using a fast client (emulated over IPoIB as in the static case). Figure 17 shows the processing and communication overhead seen by each of the tiers.

It can be seen in the figure that there is little benefit for

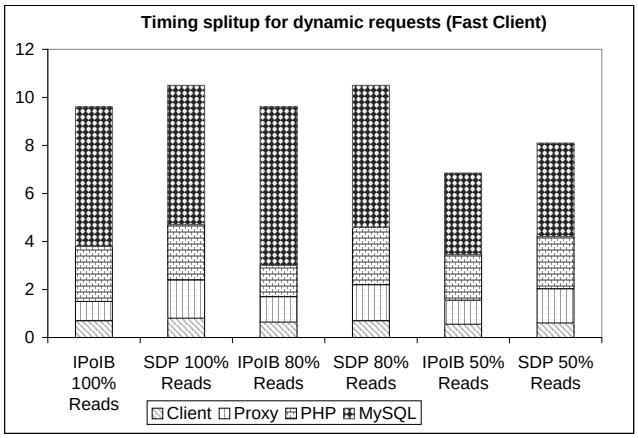


Figure 17. Timing Splitup for dynamic content (Fast Client)

SDP over IPoIB even for faster clients. This lack of benefit is attributed to the low message transfers involved in dynamic requests, which get shadowed by the high connection time for this implementation of SDP.

6 PVFS Performance Evaluation

In this section, we compare the performance of the Parallel Virtual File System (PVFS) over IPoIB and SDP with the original PVFS implementation [15]. We also compare the performance of PVFS on the above two protocols with the performance of our previous implementation of PVFS over InfiniBand [30]. All experiments in this section have been performed on Cluster 1 (mentioned in Section 4).

6.1 Evaluation Methodology

Figure 4b shows that both IPoIB and SDP can offer a bandwidth of several hundred bytes. The native InfiniBand interface (VAPI) [5] offers an even higher bandwidth of up to 830 Mbytes per second. There is a large difference between the bandwidth realized by different protocols over the InfiniBand network and that which can be obtained on a disk-based file system in most cluster systems. 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, we designed two types of experiments. The first type of experiments are based on a memory-resident file system, *ramfs*. These tests are designed to stress the network data transfer independent of any disk activity. Results of these tests 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. The second type of experiments are based on a regular disk file system, *ext3fs*. Results of these tests are representative of disk-bounded workloads. In these tests, we focus on how the difference in CPU utilization for these protocols can affect the PVFS performance.

6.2 Experimental Results

6.2.1 PVFS Concurrent Read and Write on ramfs

We used the test program, *pvfs-test* (included in the PVFS release package), to measure the concurrent read and write performance. We followed the same test method as described in [15], i.e., each compute node simultaneously reads or writes a single contiguous region of size $2N$ Mbytes, where N is the number of I/O nodes in use. For example, if the number of I/O nodes is 4, the request size is 8 Mbytes. Each compute node accesses 2 Mbytes data from each I/O node.

Figure 18 shows the read performance with the original implementation of PVFS over IPoIB and SDP and an implementation of PVFS over VAPI [30], previously done by our group. The performance of PVFS over SDP depicts the peak performance one can achieve without making any changes to the PVFS implementation. On the other hand, PVFS over VAPI depicts the peak performance achievable by PVFS over InfiniBand. We name these three cases using the legends *IPoIB*, *SDP*, and *VAPI*, respectively. With IPoIB, the bandwidth increases at a rate of approximately 140 Mbytes/s with each additional I/O node when there are sufficient compute nodes to carry the load. With SDP, the bandwidth increases at a rate of approximately 310 Mbytes/s with each additional I/O node. Note that in our 8-node InfiniBand cluster system (Cluster 1), we cannot place the PVFS manager process and the I/O server process on the same physical node since the current implementation of SDP does not support socket-based communication between two processes on the same physical node. Therefore, we have one compute node lesser in all experiments with SDP. The PVFS implementation on VAPI offers a bandwidth increase of roughly 380 Mbytes/s with each additional I/O node.

Figure 19 shows the write performance of PVFS over IPoIB, SDP and VAPI. With IPoIB, the bandwidth increases at a rate of approximately 130 Mbytes/s with each additional I/O node when there are sufficient compute nodes to carry the load. With SDP, the bandwidth increases at a rate of approximately 210 Mbytes/s with each additional I/O node. The PVFS implementation on VAPI offers a bandwidth increase of roughly 310 Mbytes/s with each ad-

ditional I/O node.

Overall, compared to PVFS on IPoIB, PVFS on SDP has a factor of 2.4 improvement for concurrent reads and a factor of 1.5 improvement for concurrent writes. The cost of writes on ramfs is higher than that of reads, resulting in a lesser improvement for SDP as compared to IPoIB. Compared to PVFS over VAPI, PVFS over SDP has about 35% degradation. This degradation is mainly attributed to the copies on the sender and the receiver sides in the current implementation of SDP. With a future zero-copy implementation of SDP, this gap is expected to be further reduced.

6.2.2 PVFS Concurrent Write on ext3fs

We also performed the above mentioned test on a disk-based file system, *ext3fs* on a Seagate ST340016A, ATA 100 40 GB disk. The write bandwidth for this disk is 25 Mbytes/s. In this test, the number of I/O nodes are fixed at three, and the number of compute nodes four. We chose PVFS *write with sync* to avoid any cache effects. Figure 20 shows the performance of PVFS write with sync with the original implementation on IPoIB and SDP and an implementation of PVFS over VAPI, respectively. It can be seen that, although each I/O server is disk-bound, a significant performance improvement of 9% is achieved by PVFS over SDP as compared to PVFS over IPoIB. This is because the lower overhead of SDP as shown in Figure 4b leaves more CPU cycles free for I/O servers to process concurrent requests.

7 Concluding Remarks and Future Work

The advent of 10-Gigabit networks such as InfiniBand has challenged the performance achievable by the traditional TCP/IP stack. Though InfiniBand has been able to support low latency and high bandwidth, traditional sockets based applications have not been able to take advantage of this; this is mainly attributed to the multiple copies and kernel context switches associated with the traditional kernel based TCP/IP protocol stack.

The Sockets Direct Protocol had been proposed recently in order to enable traditional sockets based applications to take advantage of the enhanced features provided by the InfiniBand Architecture including Remote Direct Memory Access, Solicited Events, etc. The main idea of this protocol is to provide a pseudo sockets-like API which internally utilizes InfiniBand's advanced features to provide a significantly higher performance for sockets based applications.

In this paper, we study the benefits and limitations of an implementation of SDP. We first analyze the performance of SDP based on a detailed suite of micro-benchmarks. Next, we evaluate it on two different real application domains: (1) A multi-tier Data-Center environment and (2) A Paral-

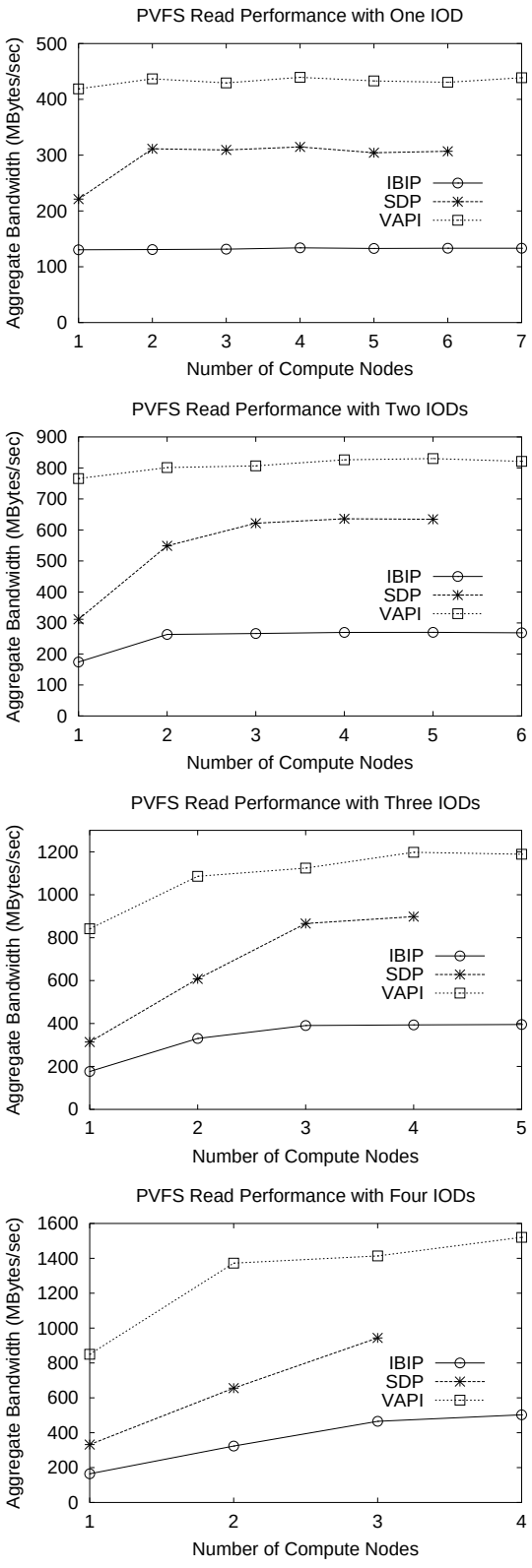


Figure 18. PVFS Read Performance Comparison.

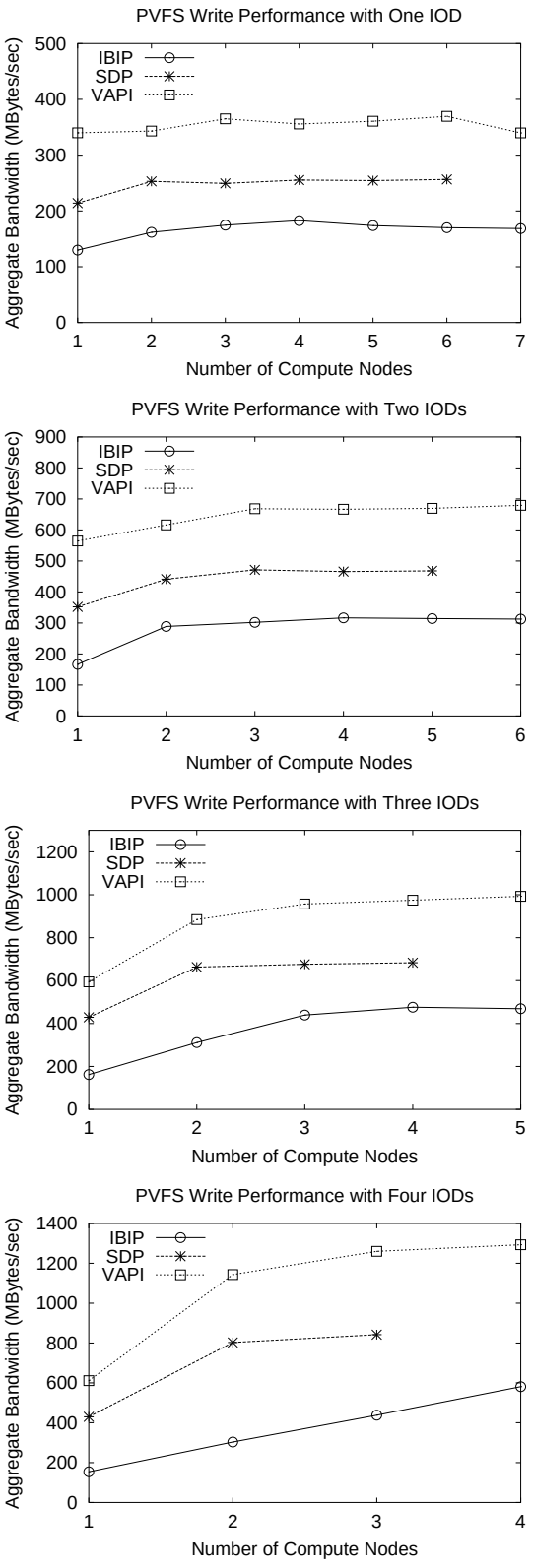


Figure 19. PVFS Write Performance Comparison.

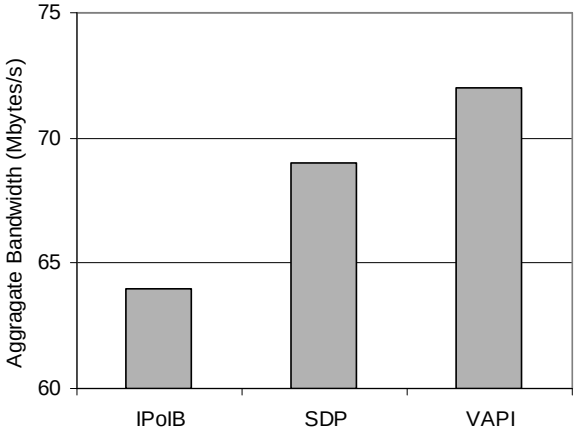


Figure 20. Performance of PVFS Write with Sync on ext3fs.

lel Virtual File System (PVFS). Our micro-benchmark results show that SDP is able to provide up to 2.7 times better bandwidth as compared to the native sockets implementation over InfiniBand (IPoIB) and significantly better latency for large message sizes. Our experimental results also show that SDP is able to achieve a considerably higher performance (improvement of up to 2.4 times) as compared to IPoIB in the PVFS environment. In the data-center environment, SDP outperforms IPoIB for large file transfers inspite of currently being limited by a high connection setup time. However, this limitation is entirely implementation specific and as the InfiniBand software and hardware products are rapidly maturing, we expect this limitation to be overcome rapidly. Based on this, we have shown that the projected performance for SDP can perform significantly better than IPoIB in all cases. These results provide profound insights into the efficiencies and bottlenecks associated with High Performance socket layers for 10-Gigabit networks. These insights have strong implications on the design and implementation of the next generation high performance applications.

We are currently working in two broad aspects with respect to SDP. Firstly, the connection time is a huge requirement for environments such as the Data-Center, where connections are established and tore down dynamically. We are currently looking at using dynamic registered buffer pools and connected Queue Pair (QP) pools to optimize SDP for such applications. The second direction we are currently working on is Power-Law networks. It has been shown that despite its apparent randomness, the Internet in reality tends to form a number of highly connected clusters together with a number of other nodes connected randomly. These clusters form central hubs for most of the data-transfer; more interestingly, most of these data transfers are intra-cluster communications, forming an ideal scenario for utilizing the capabilities of SDP.

8 Acknowledgments

We would like to thank Gali Zisman, Andy Hillaker, Erez Strauss, Yaron Haviv and Yaron Segev from Voltaire for providing us with the details of their SDP implementation. We would also like to thank Adam Wagner for all the help he provided with the Data-Center component of this paper. Lastly, we would like to thank the PVFS team at the Argonne National Laboratory and Clemson University for giving us access to the latest version of the PVFS implementation and for providing us with crucial insights into the implementation details.

References

[1] IBAL: InfiniBand Linux SourceForge Project. `http://infiniband.sourceforge.net/IAL/Access/IBAL`.

[2] IP over InfiniBand Working Group. `http://www.ietf.org/html.charters/ipoib-charter.html`.

[3] IPoIB: InfiniBand Linux SourceForge Project. `http://infiniband.sourceforge.net/NW/IPoIB/overview.htm`.

[4] M-VIA: A High Performance Modular VIA for Linux.

[5] Mellanox Technologies. `http://www.mellanox.com`.

[6] Sockets Direct Protocol. `http://www.infinibandta.com`.

[7] The DAT Collaborative. `http://www.datcollaborative.org/udapl.html`.

[8] The DAT Collaborative. `http://www.datcollaborative.org/kdapl.html`.

[9] Infiniband Trade Association. http://www.infinibandta.org.

[10] P. Balaji, P. Shivam, P. Wyckoff, and D.K. Panda. High Performance User Level Sockets over Gigabit Ethernet. In *Cluster Computing*, September 2002.

[11] Pavan Balaji, Jiesheng Wu, Tahsin Kurc, Umit Catalyurek, Dhabaleswar K. Panda, and Joel Saltz. Impact of High Performance Sockets on Data Intensive Applications. In *the Proceedings of the IEEE International Conference on High Performance Distributed Computing (HPDC 2003)*, June 2003.

[12] TPC-W Benchmark. http://www.tpc.org.

[13] 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. http://www.myricom.com.

[14] P. Buonadonna, A. Geweke, and D. E. Culler. BVIA: An Implementation and Analysis of Virtual Interface Architecture. In *Proceedings of Supercomputing*, 1998.

[15] Philip H. Carns, Walter B. Ligon III, Robert B. Ross, and Rajeev 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.

[16] Avery Ching, Alok Choudhary, Wei keng Liao, Robert Ross, and William Gropp. Noncontiguous I/O through PVFS. In *Proceedings of the IEEE International Conference on Cluster Computing*, 2002.

[17] GigaNet Corporations. cLAN for Linux: Software Users' Guide.

[18] Myricom Corporations. The GM Message Passing System.

[19] H. Frazier and H. Johnson. Gigabit Ethernet: From 100 to 1000Mbps.

[20] http://www.top500.org. Top 500 supercomputer sites.

[21] J. S. Kim, K. Kim, and S. I. Jung. SOVIA: A User-level Sockets Layer over Virtual Interface Architecture. In *Proceedings of Cluster Computing*, 2001.

[22] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM). In *Proceedings of Supercomputing*, 1995.

[23] H. V. Shah, C. Pu, and R. S. Madukkarumukumana. High Performance Sockets and RPC over Virtual Interface (VI) Architecture. In *Proceedings of CANPC workshop*, 1999.

[24] Piyush Shivam, Pete Wyckoff, and D.K. Panda. EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing. In *Proceedings of Supercomputing*, 2001.

[25] Piyush Shivam, Pete Wyckoff, and D.K. Panda. Can User-Level protocols take advantage of Multi-CPU NICs? In *Proceedings of International Parallel and Distributed Processing Symposium*, 2002.

[26] W. Richard Stevens. *TCP/IP Illustrated, Volume I: The Protocols*. Addison Wesley, 2nd edition, 2000.

[27] Rajeev Thakur, William Gropp, and Ewing 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.

[28] Internet Traffic Archive Public Tools. http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[29] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated, Volume II: The Implementation*. Addison Wesley, 2nd edition, 2000.

[30] Jiesheng Wu, Pete Wyckoff, and Dhabaleswar K. Panda. PVFS over InfiniBand: Design and Performance Evaluation. In *Proceedings of the 2003 International Conference on Parallel Processing (ICPP 03)*, Oct. 2003.