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
{balaji, narravul, vaidyana, savitha, wuj, panda}@cis.ohio-state.edu

Abstract

The Sockets Direct Protocol (SDP) has 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 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 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.

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 [10]. 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) [7] was proposed earlier to standardize these efforts. InfiniBand Architecture (IBA) [2] has been recently standardized by the industry to design next generation high-end clusters.

Earlier generation protocols such as TCP/IP 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. 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. 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 [4, 11, 13, 5].

Sockets Direct Protocol (SDP) [1] 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.

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 real application domains: (a) a Multi-tier Data-Center and (b) 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.

2 Background

In this section we provide a brief background about the Sockets Direct Protocol (SDP) implementation. Background information about InfiniBand and other existing high performance user-level sockets implementations has been skipped due to space constraints and can be found at [3].

Sockets Direct Protocol (SDP) is an IBA specific protocol defined by the Software Working Group (SWG) of the InfiniBand Trade Association [2]. The design of SDP is mainly based on two architectural goals: (a) Maintain traditional sockets SOCK_STREAM semantics as commonly im-

*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

plemented over TCP/IP and (b) Support for byte-streaming over a message passing protocol, including kernel bypass data transfers and zero-copy data transfers.

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 (using a buffer copy) or directly between ULP buffers (zero copy). A mix of InfiniBand Send and RDMA mechanisms are used to transfer ULP data.

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) Sends 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.

The current implementation of SDP follows most of the specifications provided above. There are two major deviations from the specifications in this implementation. First, it does not support "Source Avail" and "Sink Avail" messages. Second, it does not support a zero-copy data transfer between user buffers. All data transfer is done through the buffer copy mechanism. This limitation can also be considered as part of the previous ("Source Avail"/"Sink Avail") 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

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. 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.

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. A more detailed explanation of the typical data-center environment can be obtained in [3].

3.2 Parallel Virtual File System (PVFS)

Parallel Virtual File System (PVFS) [8] 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. Typically, a number of nodes in the cluster system are 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.

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 [9]. PVFS can be used with multiple APIs: a native API, the UNIX/POSIX API, MPI-IO [14], and an array I/O interface called 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 micro-benchmark level performance achievable by SDP and the native sockets implementation over InfiniBand (IPoIB). For all our experiments we used 2 clusters:

Cluster 1: An 8 node cluster 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

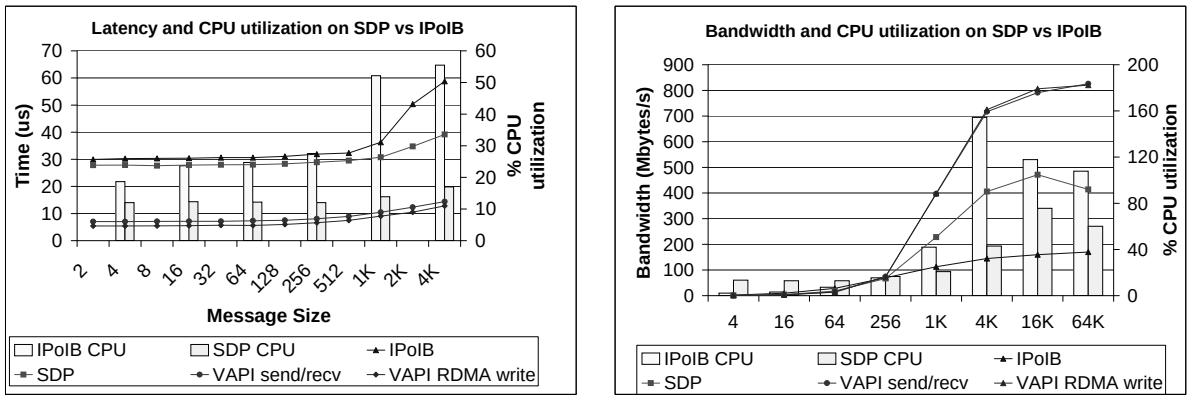


Figure 1. Micro-Benchmarks: (a) Latency, (b) Bandwidth

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 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.

Cluster 2: A 16 Dell Precision 420 node cluster 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 1a 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 1b 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 and 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 2a. 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.

4.3 Hot-Spot Test

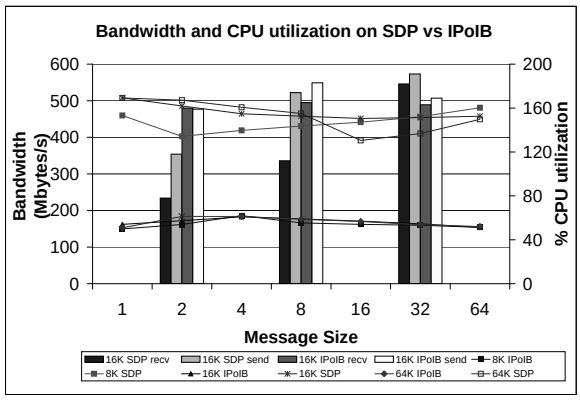
In the Hot-Spot test, multiple clients communicate with the same server. The communication pattern between any client and the server is the same pattern as in the basic latency test, i.e., the server needs to receive messages from all the clients and send messages to all clients as well, creating a hot-spot on the server. Figure 2b 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 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. Further, we find that as the number of nodes increases we get an improvement of more than a factor of 2, in terms of CPU utilization for SDP over IPoIB.

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 3a and 3b 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 appropriate points in this paper when other nodes are used as clients.



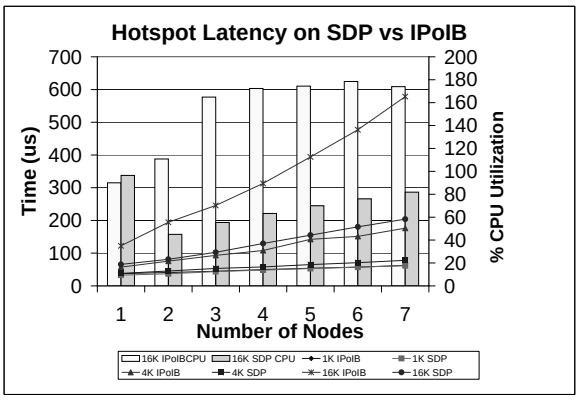
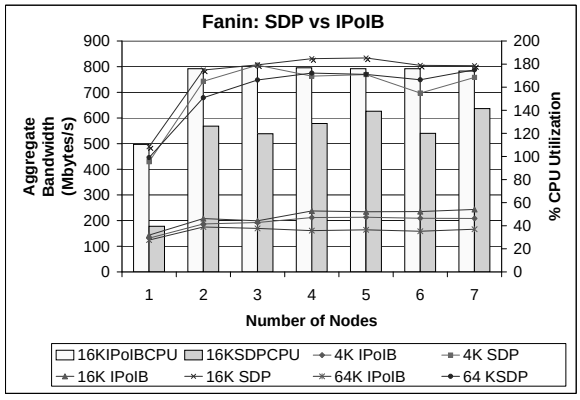


Figure 2. (a) Multi-Stream Bandwidth, (b) Hot-Spot Latency



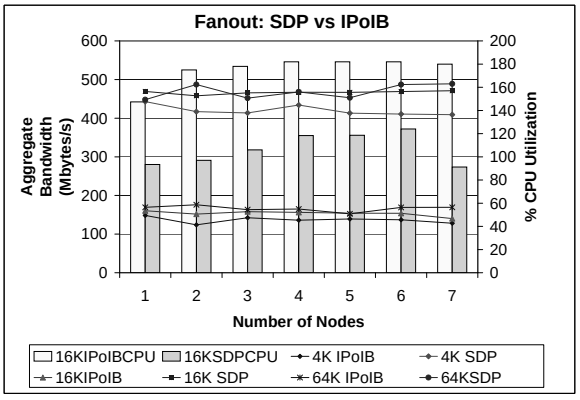


Figure 3. Micro-Benchmarks: (a) Fan-in, (b) Fan-out

5.1 Evaluation Methodology

As mentioned earlier, we used a 3-tier data-center model. Tier 1 consists of the front-end proxies. For this we used the proxy module of *apache-1.3.12*. Tier 2 consists of the web server and PHP application server modules of Apache, in order to service static and dynamic requests respectively. Tier 3 consists of the Database servers running *MySQL* to serve dynamic database queries. All the three tiers in the data-center reside on an InfiniBand network; the clients are connected to the data-center using Fast Ethernet. We evaluate the response time of the data-center using *Openload*, an open source client workload generator. We use a 20000 request subset of the world-cup trace [15] 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 is 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). We used the TPC-W transactional web benchmark [6] for generating our dynamic request access pattern (further details about the database used can be obtained in [3]).

5.2 Experimental Results

We used a 20,000 request subset of the world-cup trace to come up with our base trace file. 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 weighted average (weighted by the frequency of requests made to the given file) of the base trace file.

Figure 4a 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 4b 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 moment it gets the

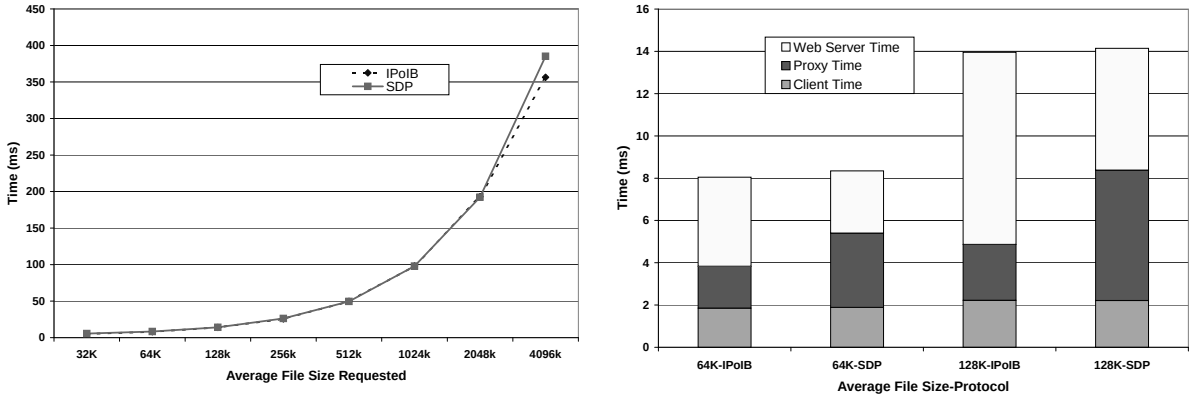


Figure 4. Client over Fast Ethernet: (a) Response Time and (b) Response Time Split-up

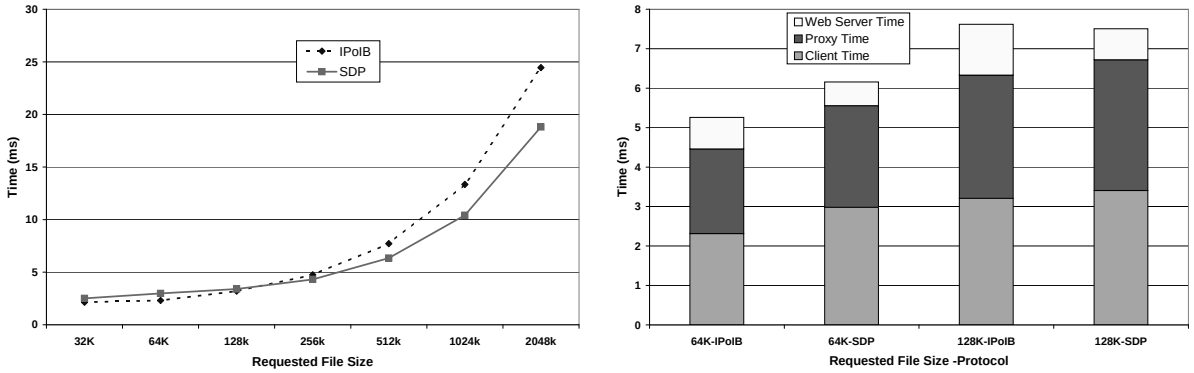


Figure 5. Client over IPoIB: (a) Response Time and (b) Response Time Split-up

request to the moment 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.

From the break-up graph (Figure 4b), we can easily 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. In spite of this improvement in the performance in the web-server time, there's no apparent improvement in the overall response time.

A possible reason for this lack of improvement is the slow interconnect used by the clients to contact the proxy server. Since the client connects to the data-center over fast ethernet, it is possible that 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 InfiniBand or may become available due to Internet proxies, ISPs etc.

Figure 5a 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; we used nodes from cluster 1 to act as clients in this case). This figure clearly shows a better performance for SDP, as compared to IPoIB for large file transfers above 128K. However, for small file sizes, there's no significant improvement. In fact, IPoIB outperforms SDP in this case. To understand the lack of performance benefits for small file sizes, we took a similar split up of the response time perceived by the client.

Figure 5b 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. 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 breaking down the time taken by the proxy in servicing the request.

Figures 7a and 7b 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,

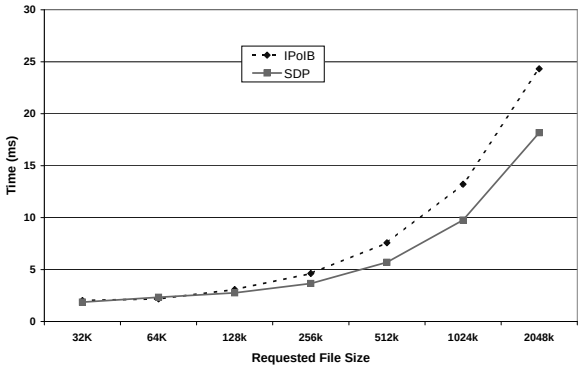


Figure 6. Fast Client Response Time without Connection Time

since this issue of connection setup time is completely implementation specific, we tried to estimate the (projected) performance SDP can provide if the connection time bottleneck was resolved.

Figure 6 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. A similar analysis for dynamic requests can be found in [3].

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 [8]. We also compare the performance of PVFS on the above two protocols with the performance of our previous implementation of PVFS over InfiniBand [16]. All experiments in this section have been performed on Cluster 1 (mentioned in Section 4).

6.1 Evaluation Methodology

There is a large difference between the bandwidth realized by the InfiniBand network (Figure 1b) 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 frequently resides 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.

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 [8], 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. Each compute node accesses 2 Mbytes data from each I/O node.

6.2 PVFS Concurrent Read and Write on ramfs

Figure 8 shows the read performance with the original implementation of PVFS over IPoIB and SDP and an implementation of PVFS over VAPI [16], 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. When there are sufficient compute nodes to carry the load, the bandwidth increases at a rate of approximately 140 Mbytes/s, 310 Mbytes/s and 380 Mbytes/s with each additional I/O node for IPoIB, SDP and VAPI respectively. 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 processes on the same physical node. So, we have one compute node lesser in all experiments with SDP.

Figure 9 shows the write performance of PVFS over IPoIB, SDP and VAPI. Again, when there are sufficient compute nodes to carry the load, the bandwidth increases at a rate of approximately 130 Mbytes/s, 210 Mbytes/s and 310 Mbytes/s with each additional I/O node for IPoIB, SDP and VAPI respectively.

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.

6.3 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*. Figure 10 shows the performance of PVFS write with sync with IPoIB, SDP and VAPI. 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 1b leaves more CPU cycles free for I/O servers to process concurrent requests. Due to the same reason, SDP achieves about 5%

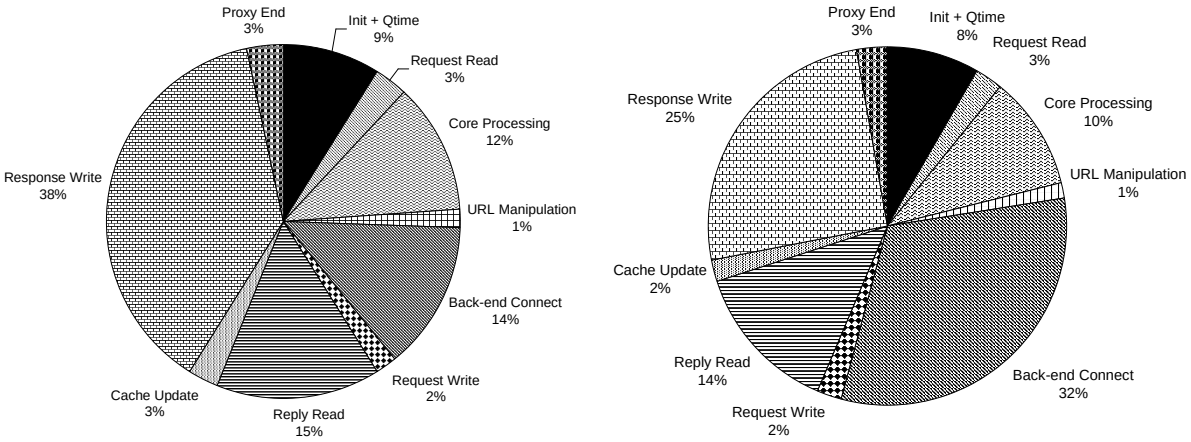
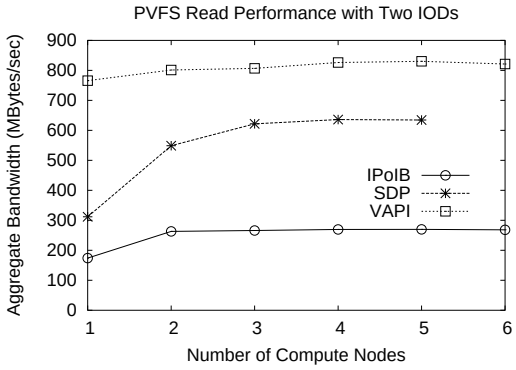
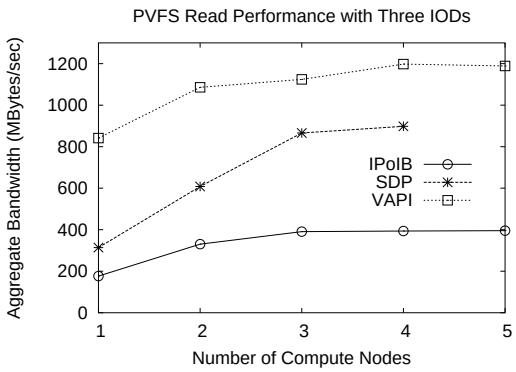


Figure 7. Proxy Split-up times: (a) IPoIB, (b) SDP





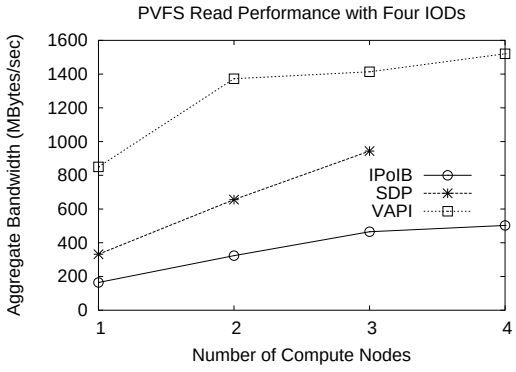
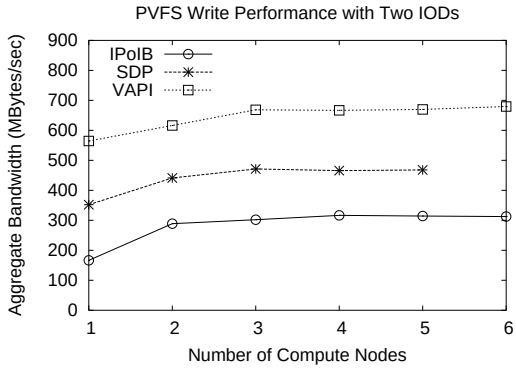
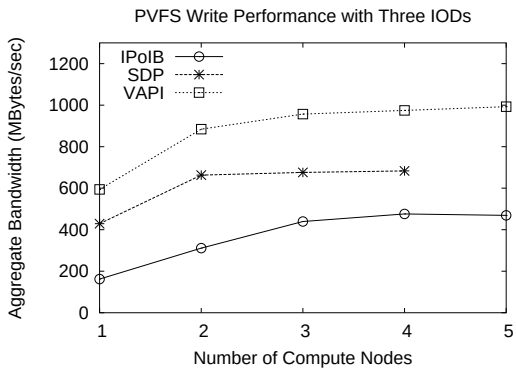


Figure 8. PVFS Read Performance Comparison





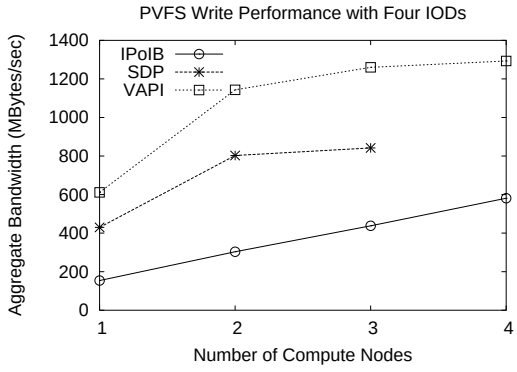


Figure 9. PVFS Write Performance Comparison

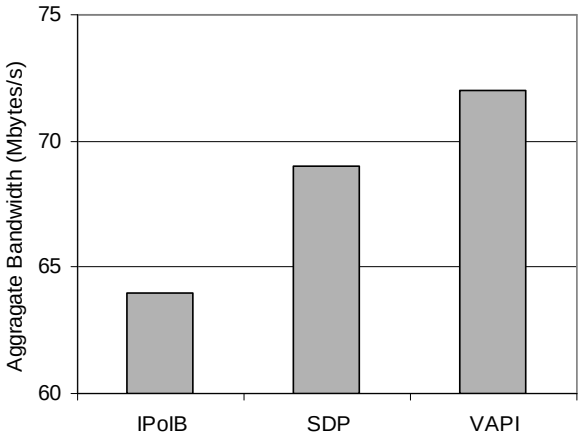


Figure 10. Performance of PVFS Write with Sync on ext3fs

lesser performance as compared to the native VAPI implementation.

7 Concluding Remarks and Future Work

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. 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 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 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 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 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. First, 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 working on is evaluating the performance of SDP in more specific areas of the data-center environment such as active caches, etc. Some initial results in this area can be found in [12].

Due to space constraints, several interesting results in the paper had to be dropped. We recommend interested readers to have a look at the technical report available at ftp://ftp.cis.ohio-state.edu/pub/tech-report/2003/TR54.pdf for these results.

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

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

[3] P. Balaji, S. Narravula, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Sockets Direct Protocol over InfiniBand: Is it Beneficial? Technical Report OSU-CISRC-10/03-TR54, The Ohio State University, 2003.

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

[5] P. Balaji, J. Wu, T. Kurc, U. Catalyurek, D. K. Panda, and J. Saltz. Impact of High Performance Sockets on Data Intensive Applications. In *HPDC*, 2003.

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

[7] P. Buonadonna, A. Geweke, and D. E. Culler. BVIA: An Implementation and Analysis of Virtual Interface Architecture. In *SC*, 98.

[8] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A Parallel File System for Linux Clusters. In *4th Annual Linux Showcase and Conference*. USENIX Association, 2000.

[9] A. Ching, A. Choudhary, W. Liao, R. Ross, and W. Gropp. Noncontiguous I/O through PVFS. In *Cluster Computing*, 02.

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

[11] J. S. Kim, K. Kim, and S. I. Jung. SOVIA: A User-level Sockets Layer over Virtual Interface Architecture. In *Cluster Computing*, 01.

[12] S. Narravula, P. Balaji, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda. Supporting Strong Coherency for Active Caches in Multi-Tier Data-Centers over InfiniBand. In *SAN*, 2004.

[13] H. V. Shah, C. Pu, and R. S. Madukkarumukumana. High Performance Sockets and RPC over Virtual Interface (VI) Architecture. In *CANPC workshop*, 99.

[14] R. Thakur, W. Gropp, and E. Lusk. On Implementing MPI-IO Portably and with High Performance. In *the 6th Workshop on I/O in Parallel and Distributed Systems*, 1999.

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

[16] J. Wu, P. Wyckoff, and D. K. Panda. PVFS over InfiniBand: Design and Performance Evaluation. In *ICPP*, 2003.