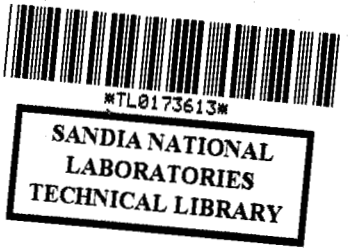
SANDIA REPORT

SAND2005-0698
Unlimited Release
Printed February 2005

IP Storage: A Performance and Security Study LDRD 04-1021

Helen Y. Chen
Jamie Van Randwyk
Neal Bierbaum
Frank Bielecki



Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SANDIA REPORT

SAND2005-0698
Unlimited Release
Printed February 2005

IP STORAGE: A PERFORMANCE AND SECURITY STUDY, LDRD 04-1021

Helen Y. Chen
Neal Bierbaum
Scalable Computing R & D

Jamie Van Randwyk
Frank Bielecki
Cyber Security Department

ABSTRACT

Effective, high-performance, networked filesystems and storage is needed to solve I/O bottlenecks between large compute platforms. Frequently, parallel techniques such as PFTP [1] are employed to overcome the adverse effect of TCP's congestion avoidance algorithm in order to achieve reasonable aggregate throughput. These techniques can suffer from end-system bottlenecks due to the protocol processing overhead and memory copies involved in moving large amounts of data during I/O. Moreover, transferring data using PFTP requires manual operation, lacking the transparency to allow for interactive visualization and computational steering of large-scale simulations from distributed locations.

This paper evaluates the emerging Internet SCSI (iSCSI) protocol [2] as the file/data transport in order that remote clients can transparently access data through a distributed global filesystem available to local clients. We started our work characterizing the performance behavior of iSCSI in Local Area Networks (LANs). We then proceeded to study the effect of propagation delay on throughput using remote iSCSI storage and explored optimization techniques to mitigate the adverse effects of long delay in high-bandwidth Wide Area Networks (WANs).

Lastly, we evaluated iSCSI in a Storage Area Network (SAN) for a Global Parallel Filesystem. We conducted our benchmark based on typical usage model of large-scale scientific applications at Sandia. We demonstrated the benefit of high-performance parallel I/O to scientific applications at the IEEE 2004 Supercomputing Conference, using experiences and knowledge gained from this study.

CONTENTS

FIGURES

1 Project Description

Effective, high-performance, networked filesystems and storage is needed to solve I/O bottlenecks between large compute platforms. Frequently, parallel techniques such as PFTP [1] are employed to overcome the adverse effect of TCP's congestion avoidance algorithm in order to achieve reasonable aggregate throughput. These techniques can suffer from end-system bottlenecks due to the protocol processing overhead and memory copies involved in moving large amounts of data during I/O. Moreover, transferring data using PFTP requires manual operation, lacking the transparency to allow for interactive visualization and computational steering of large-scale simulations from distributed locations.

This paper evaluates the emerging Internet SCSI (iSCSI) protocol [2] as the file/data transport in order that remote clients can transparently access data through a distributed global filesystem available to local clients. We started our work characterizing the performance behavior of iSCSI in Local Area Networks (LANs). We then proceeded to study the effect of propagation delay on throughput using remote iSCSI storage and explored optimization techniques to mitigate the adverse effects of long delay in high-bandwidth Wide Area Networks (WANs).

Lastly, we evaluated iSCSI in a Storage Area Network (SAN) for a Global Parallel Filesystem. We conducted our benchmarks based on typical usage model of large-scale scientific applications at Sandia. We demonstrated the benefit of high-performance parallel I/O to scientific applications at the IEEE 2004 Supercomputing Conference, using experiences and knowledge gained from this study.

2 iSCSI

2.1 Background

Traditionally, storage systems are directly attached to computer servers via a local bus. This architecture has inherent restrictions in its ability to scale in number as well as the ability to share its storage resources. A Storage Area Network [3] overcomes these limitations by using a dedicated, centrally managed infrastructure where storage I/O is performed via networking protocols. By decoupling the storage systems from the host computers, storage devices such as hard drives, redundant array of independent disks (RAID) controllers [4], and tertiary archival systems can be shared by multiple computer servers to allow more efficient utilization. However, SAN technologies have distance limitations and are typically deployed as data center solutions.

Emerging IP-based storage protocols place filesystem and data traffic onto an IP network, thereby enabling block-level I/O to SCSI clients across Wide Area Networks (WANs). iSCSI is the most promising IP storage technology that can be used to implement a global mass storage subsystem cost effectively. iSCSI achieves cost efficiency by leveraging the popularity of TCP/IP [5] and Ethernet [6]. Figure 1 depicts a conceptual layout of a hardware iSCSI solution that uses the IP network for file and block I/O. Note that software-based iSCSI simply uses the host resident TCP/IP stack in contrast to its hardware counterpart.

Through a literature search we found that previous studies focused mainly on iSCSI's performance within a data center [7]. For transparent data sharing over distance, our work further characterizes iSCSI's performance in high-bandwidth and long-delay networks. We investigated the impact of tunable parameters in iSCSI, TCP/IP, and the underlying Ethernet protocol to improve I/O performance over high-speed WANs.

At the iSCSI level, the parameters of interest from a performance perspective are: the SCSI command request size, the iSCSI command window credit amount, the number of simultaneous iSCSI connections in a session, and the option of sending solicited vs. unsolicited data. The command request size, the command window credit amount, and the number of simultaneous connections may impact both large read and write performance. The choice of solicited vs. unsolicited data may impact only write performance. At the TCP/IP level, the most important parameters are the send and receive window sizes, especially in networks with a large bandwidth-delay product such as the ASC DISCOM network [8]. In addition, the gigabit Ethernet NIC driver must maintain large buffers to match the large bandwidth and delay product to avoid "send stalling" [9]. Details on these parameters are presented in the following paragraphs.

The iSCSI command request size is the amount of data that is sent or received as part of a SCSI command encapsulated in an iSCSI packet. The iSCSI command window credit amount is manually set on the target and determines the maximum number of iSCSI commands that can be outstanding at a given time. The product of these two parameters will determine the maximum amount of data that can be pipelined in the network in order to deal with the network latency. Increasing this value will generally improve network throughput in high-speed WAN.

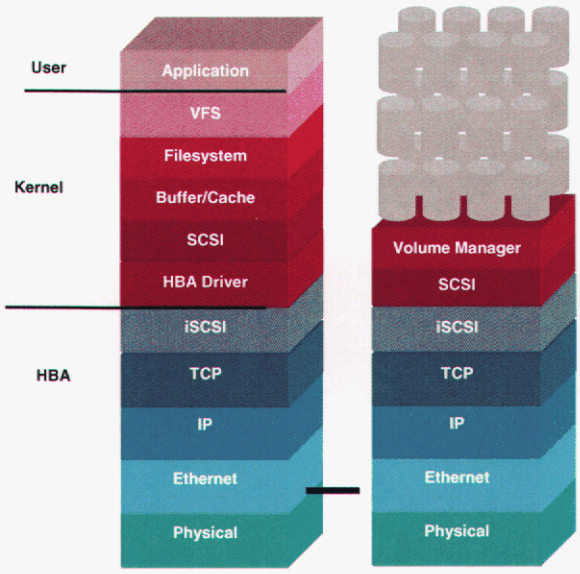


Figure 1, Protocol Layers of an iSCSI Storage System.

The primary reason for iSCSI to support multiple connections per session is to take advantage of trunking in gigabit LAN switches. Trunking allows each TCP connection to utilize a different physical link, thereby improving the overall throughput. In fact, the number of simultaneous connections in an iSCSI session will even impact its performance on a single link between the initiator and target. A packet loss in a TCP connection triggers TCP's slow-start and congestion avoidance algorithm, resulting in a drop in throughput that could take a long time to recover in long-delay and high-speed networks. By using multiple connections in a session, the overall impact of this temporary drop in throughput is reduced because each TCP connection can adjust its transfer rate; together they can achieve multiple times their fair share of bandwidth, effectively giving the iSCSI traffic priority over others during congestion. Unfortunately, the iSCSI protocol obeys the SCSI command ordering rules, which will reduce the parallelism among multiple connections, offsetting the benefit realized via multiple connections. An extension of the iSCSI protocol is being proposed to support out-of-order SCSI commands, leaving the innovation to iSCSI target vendors to handle out-of-order commands.

Three independent parameters determine the solicited vs. unsolicited types of data transfer: *FirstBurstLength*, *MaxBurstLength,* and *MaxRecvDataSegmentLength*. *FirstBurstLength* determines the maximum amount of unsolicited data that the initiator can send per command. *MaxBurstLength* determines the maximum amount of solicited data that the initiator can send per command. *MaxRecvDataSegmentLength* is the maximum data segment size that can be sent in each protocol data unit (PDU). There are many ways to set these parameters. In this study, we consider two cases that can produce results in two extremes: most-unsolicited and most-solicited data transfer allowed by the iSCSI protocol. Most-unsolicited data implies that the *FirstBurstLength* is greater than or equal to the maximum write command request size that the initiator generates. Most-solicited data implies that the unsolicited data mode is disabled during the login negotiation; it is effectively equivalent to setting the *FirstBurstLength* to zero. In this mode, the target will notify the initiator when it is ready to receive data for a given command.

This will give the target more control in the receive buffer allocation, but will introduce extra round trip delay compared to the fully unsolicited mode.

2.2 Benchmark Methodology

We evaluated and selected the following public domain benchmark tools to generate traffic and to monitor system and network-level statistics:

1. Xdd is a tool for measuring and characterizing storage subsystem I/O on single and cluster systems. It is designed to provide persistent and reproducible performance measurements of raw and buffered storage traffic.
2. Performance Co Pilot (PCP) is a framework that supports system-level performance monitoring and management. PCP monitors system-level performance to allow correlation of end-user quality of service with platform activity, tracking complex interactions between resource demands on a single system.
3. Ethereal is an open-source GUI-based network protocol analyzer. It allows an analyst to interactively browse packet data from a live network or from a previously saved packet capture file. In addition to TCP and IP, Ethereal can also parse the iSCSI protocol.

As part of this project a new, versatile test integration and control program was developed. This program, named *testrun,* provides a platform for integrating test definition, execution, and organization of results and information to ensure that accurate, meaningful analyses can be performed. It is written in a general manner to support a wide variety of testing projects. It will be released for general Sandia use at the end of this project.

An XML definition file for each unique type of test controls the *testrun* program. This description defines the test program (e.g., *xdd* or *IOzone*) to be run with the options defining the specific test. If multiple instances of an option are defined with different values, it will perform multiple tests with unique combinations of these options. It also defines the hosts and instances on each host of the test to be run. Each run is performed simultaneously across all hosts; each host is controlled via an SSH connection to the primary control host for complete security, even over a wide area network. In addition, *testrun* manages the sophisticated *Performance Copilot* monitor, starting it on each host performing the test immediately prior to the beginning of the test and shutting it down after the test is completed. At the completion of each *testrun* all results are collected into a four-level highly-structured result tree.

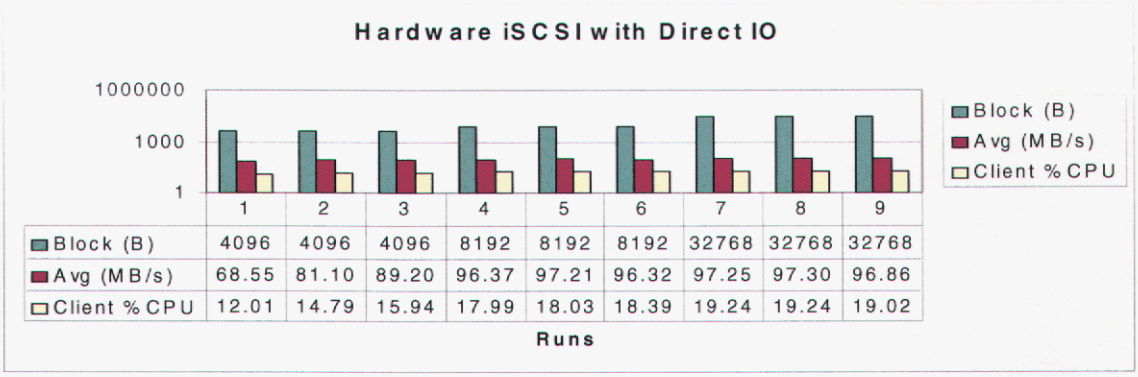
The results of each run are reported in a series of XML files to allow consistent, unambiguous searching and processing of hundreds or even thousands of runs by a variety of tools. The results are also reported in a multi-level tree of interlinked HTML files. This provides easy browsing from a top level of the entire test series run throughout the project down to a single instance of a test on a single host and then back again with a few clicks of a mouse button.

2.3 Results and Analysis

2.3.1 Local Area Network

For this set of experiments, we partnered with Adaptec and Network Appliance to prototype a small-scale iSCSI testbed. The Adaptec iSCSI Host Bus Adapter (HBA) implements hardware iSCSI with TCP/IP offloaded in an ASIC. Four Adaptec host bus adapters are installed one each in a Dell PowerEdge 2650, a dual 2.4GHz Xeon processor server. The HBAs act as iSCSI initiators and are connected to a Network Appliance FAS 960 Filer through four gigabit Ethernet ports back-to-back. The FAS 960 exports its volume manager to iSCSI clients through its software-based iSCSI interface and kernel resident TCP/IP stack. We measured a baseline of the performance of iSCSI using the *testrun* program described above.

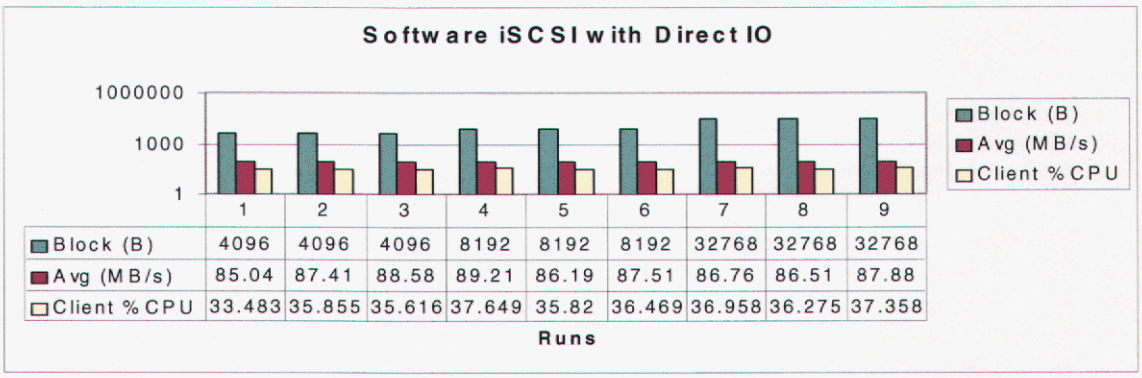
We measured the throughput of *xdd* sessions using 5 GB files. Nine tests were conducted in total, three each with I/O requests ranging 4K, 8K, and 32K Bytes respectively. The 5 GB file size was selected to offset the effect of the Linux virtual filesystem's buffer cache. As shown in Figure 2, a single *xdd* session can saturate the gigabit link with 8K or larger read requests, and the associated CPU overhead averaged a low 12 to 19%, leaving ample cycles for file level operations as well as scientific computation.



	1	2	3	4	5	6	7	8	9
Block (B)	4096	4096	4096	8192	8192	8192	32768	32768	32768
Avg (MB/s)	68.55	81.10	89.20	96.37	97.21	96.32	97.25	97.30	96.86
Client % CPU	12.01	14.79	15.94	17.99	18.03	18.39	19.24	19.24	19.02

Figure 2, Hardware iSCSI performance with Direct I/O

We repeated the above experiment with a software iSCSI implementation for comparison, using a public domain iSCSI initiator, developed principally by Cisco until recently, and a gigabit Ethernet NIC made by SysKonnect. Results are depicted in Figure 3. We observe that the software-level iSCSI's performance is comparable to its hardware counter part in the 4K and 8K tests, and is about 10% less in the 32K tests. The CPU overhead, however, has roughly doubled, although more than 60% of the CPU resources are still available to the filesystem as well as user applications. Additionally, we were able to achieve this performance only by turning on gigabit Ethernet's jumbo frame feature; we experienced CPU bottleneck using the 1500 byte frame due to the extra overhead necessary to process the TCP/IP protocol and more frequent interrupts from the increased number of arriving packets.



	1	2	3	4	5	6	7	8	9
Block (B)	4096	4096	4096	8192	8192	8192	32768	32768	32768
Avg (MB/s)	85.04	87.41	88.58	89.21	86.19	87.51	86.76	86.51	87.88
Client % CPU	33.483	35.855	35.616	37.649	35.82	36.469	36.958	36.275	37.358

Figure 3, Software iSCSI performance with Direct I/O

A multi-host test was conducted subsequently to evaluate the aggregate performance of iSCSI. Each host repeated a series of *xdd* runs identical to those described above. Results are presented in the following graph. As shown, the per-session throughput is much lower than the single-host single-session instances. We found that the degradation was not due to any shortage in local compute power. An examination at the FAS 960 filer revealed that 100 % of its CPU was consumed in processing iSCSI requests from the four concurrent sessions, indicating that hardware iSCSI is very much needed at the server side. The dual 2.2 GHz Xeon processors in the NetApp Filer cannot keep up with the concurrent demands from four iSCSI initiators. In addition to handling network I/O, the filer must also perform I/O operations for its own storage device.

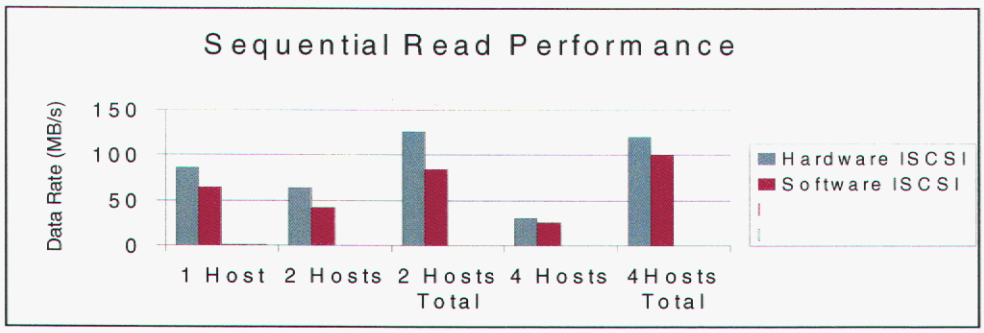


Figure 4, Multi Host Performance Tests

Because NepApp's loan agreement expired before we could conduct our WAN experiment, we repeated the same multi-host tests using two Intransa IP5000 iSCSI targets to establish a baseline. The new configuration consisted of four software iSCSI initiators on Dell 2650s accessing two Intransa targets through a 24-port Dell switch. We chose the software iSCSI implementation for the WAN experiment because hardware iSCSI lacked the flexibility and an interface to allow for necessary TCP and iSCSI tuning in order to fill the long, fat pipe. Results are plotted in Figure 5. As shown, one iSCSI session can fill a gigabit Ethernet pipe and two concurrent iSCSI sessions can scale linearly to fill two gigabit pipes. Four concurrent iSCSI sessions, however, did not achieve similar scaling due to network bottleneck -- a result of four initiators competing for two gigabit Ethernet ports. We also observed that more than 30% of the dual 2.4 GHz processors were consumed to achieve a typical gigabit Ethernet rate, which is in agreement with results from previous tests against the NetApp filer.

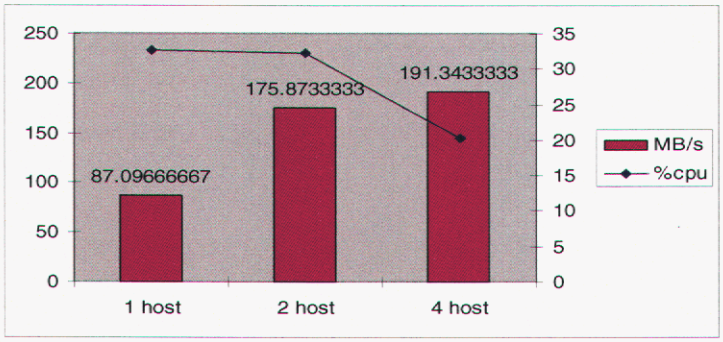


Figure 5, Software iSCSI tests with Intransa iSCSI targets

2.3.2 Wide Area Network

In order to isolate the effect of network latency from the idiosyncrasies of commercial network products, we used a WAN emulator in our initial experiment. This allowed us to easily vary the network latency while keeping all other parameters constant. Our experimental setup consisted of an open source software initiator by Cisco running on a Dell 2650 and an Intransa IP5000 iSCSI target, interconnected by an ADTECH 4000 WAN emulator from Spirent Communications. In our experiment, the network bandwidth is 1 Gbps, which is the maximum supported by the emulator. Since our focus is on network performance, we configured the IP5000 in write-back mode and set the traffic patterns such that all reads are served from the target's cache to eliminate possible disk I/O bottlenecks. We studied four values of round trip latency: 0 ms as a baseline, 2, 10 and 50 ms for different WAN scenarios. In addition, we validated our emulated WAN test over the Tri-Lab DISCOM network, allocating 1 Gbps of bandwidth from the 2.4 Gbps OC48 link. We present our tuning techniques and performance results in the following paragraphs.

TCP Tuning

TCP's primary function (as opposed to UDP) is to ensure reliable and in-order delivery of data. To achieve this, TCP uses an assigned sequence space to uniquely identify each byte in the data stream of a TCP session. It is the receiving host's responsibility to acknowledge cumulatively the data it has received by sending the sequence number of the last in-order byte. The receiving host queues out-of-order data in its reassembly queue until the holes in the sequence are filled. Meanwhile, the sending host must be prepared to retransmit unacknowledged data by keeping data in a retransmit queue.

TCP maintains a static amount of buffer between itself and the receiving application. The receiver implements a window-based flow control to prevent the sender from overrunning its buffer; the receiver enforces flow control by announcing its available buffer space in the TCP header. The amount of available space will change over time and is affected by the receiving application's ability to keep up with the rate of data-arrival from the network. The advertised receive window is the upper bound of the sender's window. Since only one window full of data can be sent within one RTT (round-trip time), TCP's theoretical throughput can be derived by dividing the effective window size by the network's RTT.

Over the past decades, Internet bandwidth has been increasing exponentially. It follows that a proportional amount of buffer space must also be maintained by end systems in order to fully utilize the much larger bandwidth, especially when they are separated by a long RTT. For our experimental 1 Gbps WAN test with a 25 ms RTT, we must maintain a TCP window that equals the product of 1 Gbps and the 25 ms RTT in order to fill the OC48 link.

$$\text{TCP window} = \text{BW} * \text{RTT} = 1\ \text{Gbps} * 0.25\text{s} = 25\ \text{Mb or } 3.125\ \text{MB} \quad (1)$$

In the scientific application usage model, often the amount of data going into the network is more than a router's output bandwidth. Since only a finite number of packets can be queued at the router, packets will be dropped during steady state. TCP uses "slow start" to probe the network for the bottleneck bandwidth in the path, and an "additive-increase and multiplicative-decrease" (AIMD) algorithm to further adjust its window size to accommodate the dynamically changing network conditions. Slow-start sets TCP's initial window to one segment and increases the size by one after each positive acknowledgement until a predefined threshold is reached; linear growth follows thereafter. A receiver sends duplicate acknowledgements of the same sequence number to notify the sender of missing packets. In response, the sender's TCP will reduce its transmission rate by halving its current window. The sender must record the reduced window and use it as its current upper bound if it is smaller than the receiver's advertised window. To recover, TCP grows its window linearly by adding one segment every round-trip-time (RTT).

Unfortunately AIMD can be very damaging to TCP performance in large Bandwidth-Delay-Product (BDP) networks because it will take many long RTTs to recover a very large original window. Many extensions to TCP have been proposed to tune the performance in large BDP networks. These include Fast TCP [10], High Speed TCP [11], TCP Vegas [12], Dynamic Right Sizing [13], etc. In general, these techniques use a larger TCP window, faster slow-start, and larger maximum-transfer-units to improve throughput. They try to avoid loss by reducing burst sizes, using Explicit Congestion Notifications (ECN), and increasing packet reorder threshold at the receiver. Faster recovery is attempted via an increased link maximum transfer unit (MTU or frame size), more aggressive AIMD, no delayed-ACK, and larger initial window and slow-start increments. A simplified macroscopic congestion model derived in [14] illustrates that TCP's maximum segment size (MSS) can directly impact TCP's throughput.

$$\text{Throughput} \sim= (\text{MSS} * .7) / (\text{RTT} * P^{1/2}) \quad (2)$$

Thus, we used Jumbo frames in our emulated WAN as well as in the experimental DISCOM WAN. The software iSCSI initiator relied on Linux's built-in TCP auto tuning capability to optimize performance, where the receiving TCP (on read I/O) will adjust its advertised TCP window to the smaller of the 2-RTT or the "ipv4.tcp_w(r) mem" amount of buffer. We set a 10 MB maximum for the send and receive buffers using the following commands:

```
sysctl -w net.core.rmem_max=10485760
sysctl -w net.core.wmem_max=10485760
sysctl -w net.core.rmem_default=10485760
sysctl -w net.core.wmem_default=10485760
```

```
sysctl -w net.ipv4.tcp_rmem="4096 10485760 10485760"
sysctl -w net.ipv4.tcp_wmem="4096 10485760 10485760"
```

In addition to the send and receive buffer tuning, it is also necessary to increase the sending NIC driver's queue size in order to prevent "send stalling":

```
ifconfig eth2 txqueuelen 1000
```

And it is necessary to increase the kernel network device backlog buffer (receive queue) in the receiver's kernel to prevent dropping packets at the receiving host:

```
sysctl -w net.core.netdev_max_backlog=2500
```

iSCSI Tuning

As mentioned earlier, a number of iSCSI parameters also need to be tuned for better WAN performance. This involves using the iSCSI 'most unsolicited' data mode as part of the iSCSI PDU to bypass the overhead for the R2T (ready to transfer) handshake. To achieve this, Intransa had to provide Sandia with development code to support the "most unsolicited" mode. The following commands are used to manually enable the "most unsolicited" data mode at the Intransa IP5000 target:

```
iscsi_ioctl_command -s 8 ImmediateData=Yes
iscsi_ioctl_command -s 8 InitialR2T=No
```

Lastly, the TCP window size for iSCSI sessions needs to be set on the IP5000 to the desired size via the setsocketopt() system call. Since the Intransa iSCSI target is Linux based, it should be able to take advantage of Linux's TCP auto tuning capability -- a request that we have also relayed to Intransa. To set the TCP window size at the IP5000, the following was used to provide a 10 MB window:

```
iscsi_ioctl_command -w 10240
```

Additionally, we had to modify the maximum number of outstanding commands allowed, also referred to as SCSI command window size, in order to utilize the large buffer allocated by the receiver's TCP. The following command sets the number of maximum outstanding SCSI commands to 64:

```
iscsi_ioctl_command -W 64
```

Similar tuning is performed on the iSCSI initiator. The Cisco Linux iSCSI initiator requires that we modify the configuration file to negotiate for the 'most unsolicited' mode with the iSCSI target. These changes are made in the /etc/iscsi.conf file:

```
ImmediateData=yes
InitialR2T=no
```

We also patched the Cisco Linux iSCSI initiator to support both a larger block size and a larger maximum number of outstanding SCSI commands. This helped to increase the throughput of tagged queuing devices such as the Intransa IP5000. This is achieved by increasing the value of the maximum number of outstanding commands in the iscsi-limits.h file. We found that setting ISCSI_CMDS_PER_LUN to 64 had delivered optimal throughput without oversubscribing the IP5000's memory subsystem. To better support larger block size and send more uniform data we increase the default max_sector size of the SCSI device driver from 256 (512 bytes) to 512 (1024 bytes). This worked well in our situation because 1024 is the default sector size for the Intransa iSCSI device. As a result, we achieved more uniform I/O processing and improved the throughput on the initiator. The results are presented below. To make this change, we edited the iscsi.c file and added 'sh->max_sectors=512;' to the structure declaration.

Measurements

Figure 6 summarizes the throughput of iSCSI over the emulated WAN. We ran the following xdd command for all tests, using a 256 KB I/O request size to write and read 1 GB files.

```
./xdd -op write -targets 1 /dev/raw/raw1 -reqsize 256 -blocksize 1024 -mbytes 1024 -queuedepth 32
```

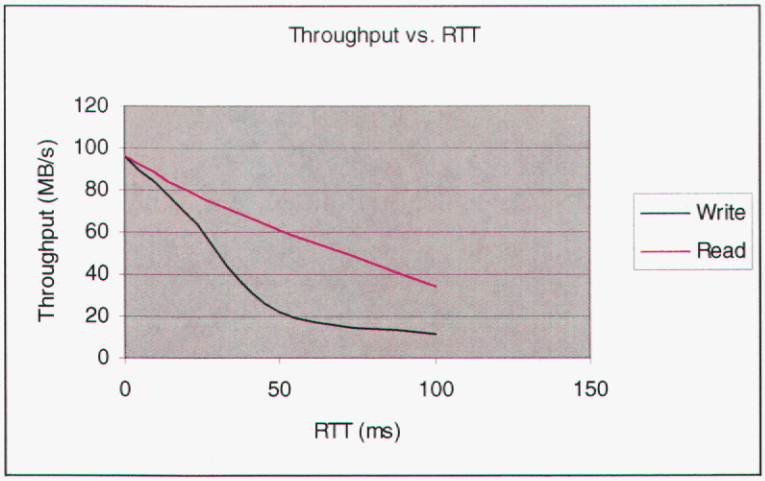


Figure 6, iSCSI WAN Performance

We observe similar throughput for READ and WRITE at latencies of 20 ms or less, but we observed better throughputs for READ than WRITE at latencies greater than 20 ms. We deduce that the asymmetries in performance were caused by differences in CPU speed between the initiator and target; the Dell 2650 uses dual 2.4 GHz Xeon processors and the IP5000 a single 800MHz MIPS processor. In this situation the receiver had to handle a larger burst of incoming traffic than in lower latency networks because TCP windows are larger in bigger RTT networks. As mentioned earlier, we were able to schedule 1 Gbps of bandwidth from the black DISCOM WAN to validate our results from our emulated WAN. We found about 10 MB/s of degradation in both READ and WRITE operations compared to our emulation. We believe this is caused by the presence of bit errors in DISCOM's SONET infrastructure, as well as occasional queuing delay experienced at DISCOM's IP routers and ATM switches. In conclusion, we found that iSCSI transport can deliver satisfactory performance to meet HPC requirements in both LAN and WAN environments. We selected an iSCSI-based parallel storage product to continue our study.

3 ISCSI-based Parallel Storage System

3.1 Background

High-performance computing (HPC) has driven the development of clustered computing architecture. This architecture uses commodity components and can therefore be implemented at 1/10 the cost of specialized Symmetric Multiprocessing systems (SMP). As such, the cost of computational science has dropped dramatically, which in turn promoted sophisticated parallel algorithms to be developed using the Message Passing Interface (MPI) [15].

Unfortunately, storage platforms failed to keep pace. Today labor intensive data movements are performed to stage computational tasks on local cluster nodes because data sharing through NFS [16] failed to scale in large clusters. The standardized NFS protocol presents data at a level of abstraction to provide a broad common view among all compute platforms: files and directories. But this level of abstraction forces metadata as well as block I/O requests to go through a single server, making it a roadblock to effectively scaling the level of shared access to its clients. Many cluster filesystems are being developed to address this issue [17]. Cluster filesystems combine a common namespace for data sharing and block I/O to storage for performance between distributed compute systems in order to alleviate the NFS bottleneck. In this study, we choose to evaluate an alternative approach that implements parallel storage architecture to address the same issue. We present background information on this approach and its performance characteristics in the following paragraphs.

TerraGRID

TerraGRID [18] uses a Shared Access Scheduling Scheme (SASS) to enable open-source "standalone" Linux filesystems to be deployed as a scalable, massively parallel, global filesystem. It takes advantage of an intelligent software implementation of the iSCSI protocol stack to deliver parallel, block-level I/O that works with whatever runs on Linux. In addition, it turns existing commodity fabrics such as Ethernet and

Infiniband into scalable I/O networks to achieve cost efficiency. TerraGRID fully harnesses Linux filesystems and utilities, with 99% of its components available as open source freeware. These components include: ext2, patches to Linux and ext2, and the Linux MD driver. A proprietary iSCSI driver provides the glue to bind all the open source components together to deliver a parallel storage system that allows every client to share an identical global name space and to execute SCSI requests in parallel. The protocol stack used by TerraGRID is depicted in Figure 7. On the server side, the iSCSI logic is implemented as a user level daemon; it presents a raw disk device, a file, or a set of files as block containers to the iSCSI initiator pool. On the client side, the TerraGRID iSCSI logic is implemented as a kernel module. This kernel module discovers all configured targets via the iSCSI session login protocol upon startup and presents these targets as SCSI devices to its operating system. Each initiator in turn runs software RAID in order to transmit I/O requests to all targets in parallel. By running MD across the same set of targets, all initiators can share the same global namespace and can schedule I/O requests in parallel to the common set of targets using the SASS algorithm.

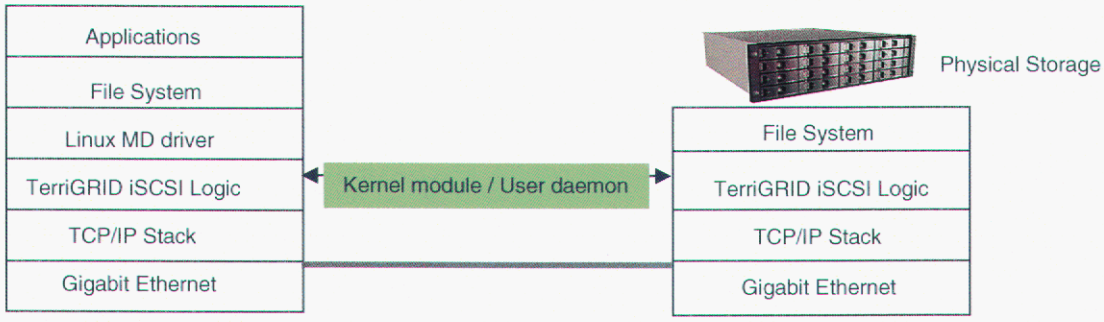


Figure 7, TerraGRID I/O Server Protocol Stack

3.2 Results

Because we had limited hardware resources, we were only able to conduct a "proof-of-concept" study in terms of scalability; our tests varied the number of clients from one to six against a software RAID device built across eight TerraGRID targets. Again, we configured "testrun" to run the xdd benchmark, reading and writing 5 GB files in 8 MB chunks, and scaling from one to six clients. We present the read performance in Figure 8 and the write performance in Figure 9.

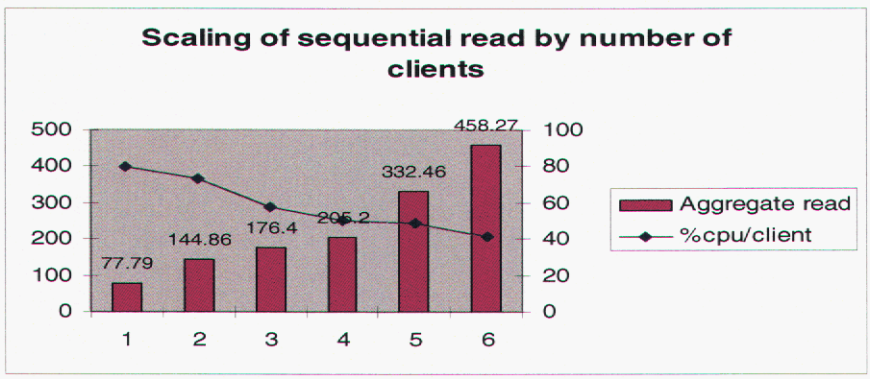


Figure 8, TerraGRID read scalability test.

Our read results (Figure 8) show that TerraGRID can achieve linear scale up. Because we have eight servers, each with one 15K rpm SCSI drive, their aggregate storage bandwidth maximized at 480 MB/s. Our results show that " xdd reads" have reached this upper bound and that the TerraGRID SASS was able to scale to handle I/O requests from six concurrent clients. From the same graph we observe that the CPU utilization on TerraGRID clients experienced a steady decline, from 80% to 40%, as the number of application nodes increases. This information helped us to eliminate CPU power from being the performance bottleneck. Furthermore, because only 60% of the total network bandwidth was consumed during the benchmark, it also eliminated the network from being the limiting factor.

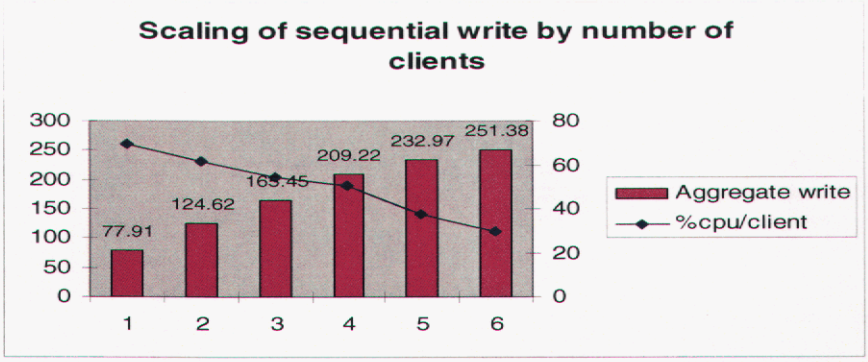


Figure 9, TerraGRID write scalability test.

Figure 9 depicts the results of our write benchmark. As shown, TerraGRID's write performance is poorer than the read performance. We attribute the poorer throughput to extra scheduling overhead necessary to carry out write operations; extra overhead to synchronize the leases of the file's metadata to different nodes was necessary in order to preserve data consistency during parallel writes.

We repeated the same benchmark over InfiniBand [19] to evaluate TerraGrid's performance in higher-speed networks (10 Gbps). Again, due to resource limitations, we used only four TerraGRID servers with a maximum aggregate disk bandwidth of 240 MB/s. As expected, we were disk-bound (Figure 10) on reads, achieving an aggregate throughput slightly over 211 MB/s. Also shown is the poorer write performance, with the aggregate throughput leveled at around 155 MB/s, consistent with the result obtained in the gigabit Ethernet experiments.

As mentioned earlier, so far only a proof-of-concept study was conducted. We plan to repeat our test on a 128-node cluster for a more meaningful evaluation of TerraGRID's scalability in terms of the number of clients it can support: can TerraGRID still maintain the same aggregate throughput bounded by storage bandwidth? Subsequent to the scalability validation, we also plan to explore the potential of extending TerraGRID's data sharing ability over the DISCOM WAN.

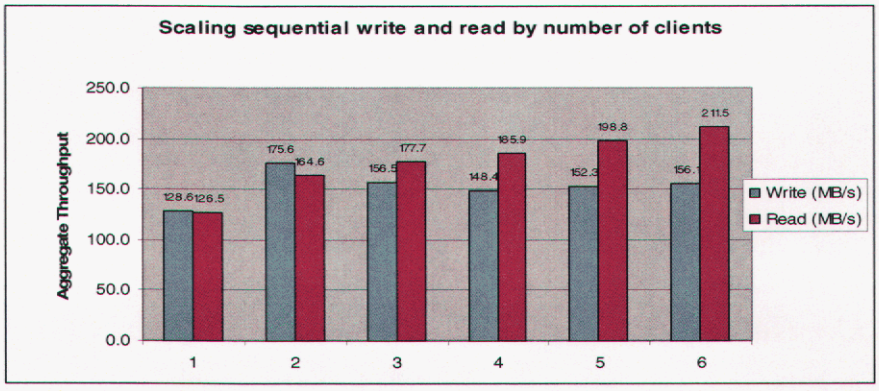


Figure 10, TerraGRID read and write scalability test over InfiniBand

4 The SC04 StorCloud Demonstration

StorCloud [20] was a new Supercomputing Conference initiative in 2004 that offered a high-performance storage capability in order to showcase state-of-the-art HPC technologies such as parallel filesystems, parallel algorithms, and interconnects. We decided to participate in this endeavor because it offered us the opportunity to partner with leading hardware and software industry partners to test out a much larger number of parallel file and storage technologies than possible otherwise. Our goal was to demonstrate the benefit of parallel I/O techniques and high-speed storage arrays to computational science. We invited three outstanding scientists from Sandia and Lawrence Livermore National Laboratories to demonstrate their innovations on the use of high-performance parallel I/O to enhance their scientific computation. These include: Blockbuster [21], a scalable animation display, from LLNL;

Massively Parallel Quantum Chemistry (MPQC) [22], and Direct Numerical Simulation for Homogeneous Charge Compression Auto-Ignition [23], from SNL. We are proud to announce that the LLNL Blockbuster won the "Fastest Random I/O" and MPQC the "Most Innovative Use of Storage" awards sponsored by the SC04 StorCloud Initiative. Details of the StorCloud testbed and the demonstration applications are presented below.

The StorCloud Testbed Infrastructure

Figure 11 depicts the logical topology of our StorCloud testbed. At the ASC booth, forty-eight Opteron-based compute servers from Appro and sixteen 64-bit Xeon-based I/O servers from Dell were interconnected using 4X InfiniBand. This cluster was booted using oneSIS [24] (cluster management software developed at Sandia) that supported the Terrascale TerraGRID, the Panasas ActiveScale [25], and the IBM GPFS [26] filesystems. TerraGRID accesses StorageTek disk arrays through sixteen 2-gigabit Fibre Channel links through the sixteen I/O servers. Panasas DirectFlow on the compute nodes reads from and writes to the Panasas ActiveScale storage via two 10-gigabit Ethernet trunks. IBM GPFS on the sixteen I/O servers are configured to access both StorageTek disks via Fibre Channel and IP5000 iSCSI targets via gigabit Ethernet. Please note that the StorageTek FC, the Intransa iSCSI, and the Panasas ActiveScale storage arrays were located at the StorCloud booth across the exhibitor show floor from the ASC booth.

GPFS is a high-performance shared-disk filesystem from IBM that can provide fast data access from all nodes in a homogenous or heterogeneous cluster of servers running either the AIX 5L or the Linux operating system. GPFS allows parallel applications simultaneous access to the same or different files from any node which has the GPFS filesystem mounted while also managing the metadata of GPFS. "Its I/O performance", IBM claims, "can meet the objectives of a wide range of applications including seismic data processing, digital library file serving, and data mining in business intelligence."

The Panasas ActiveScale File System turns files into smart data objects and dynamically distributes data activity across its StorageBlades cluster. The clustering architecture employed by Panasas enables parallel data paths between StorageBladesand Linux clients, thereby eliminating the inherent performance and capacity bottlenecks of traditional networked filesystems. "The core principle of the object-based architecture is managing data as large virtual objects in contrast with traditional storage systems that manage data in small blocks. Data objects can be resized without limitation and independently from other storage system activity, allowing the management of all data within a single namespace and providing independent and parallel growth properties."

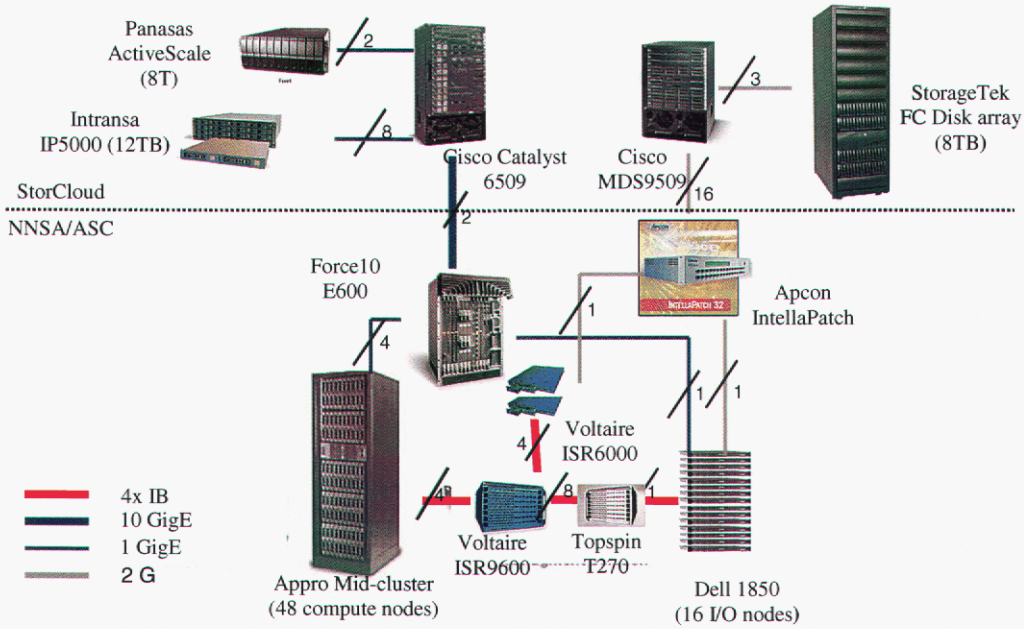
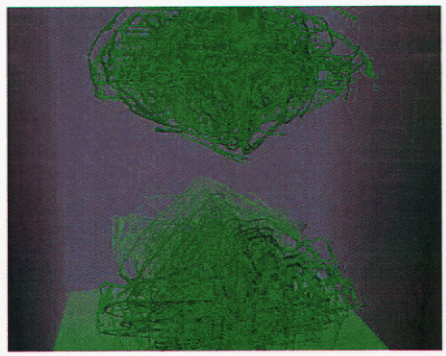


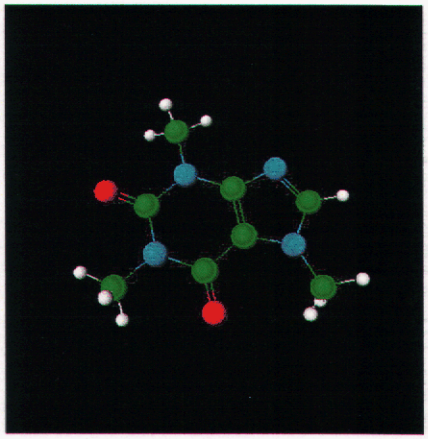
Figure 11, the StorCloud Testbed

Scalable Animation Display



Blockbuster is a scalable animation display application developed at Lawrence Livermore National Laboratory. Blockbuster relies on the combined power of a cluster of visualization nodes to perform rendering in parallel. During a Blockbuster run, each node in the cluster reads large chunks of data in parallel from different sections of a single scientific data-set. As such, this application will need to use a high-speed parallel filesystem in order to keep up with the rendering speed for the high resolution animation on large panel displays. The Blockbuster application will be run on the 16-node Dell cluster using the IBM GPFS to read large chunks of data in parallel from different sections of a single scientific data-set stored on the StorageTek storage at the StorCloud booth. We also demonstrate a more cost efficient capability that uses GPFS to access data on the iSCSI-based Intransa IP5000 storage also located at the StorCloud booth.

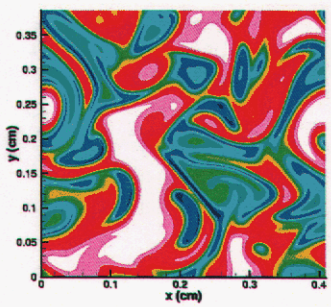
Massively Parallel Quantum Chemistry (MPQC)



MPQC is a quantum chemistry code developed at Sandia. It can be used to compute molecular energies, geometries, and other properties. It is designed to keep most of the intermediates required in main memory or re-computing relatively inexpensive quantities as needed. However, SNL scientists recently implemented second-order Moeller-Plesset perturbation theory explicitly including the inter-electron coordinate (MP2-R12) in MPQC. The code, therefore, has larger storage requirements than standard Moeller-Plesset perturbation theory (MP2). As such, the availability of high performance parallel storage will make possible computations that cannot currently be done, allowing larger systems to be treated or higher accuracy results to be obtained. We demonstrate this capability by running MPQC MP2-R12 on the 48-node Appro Mid-Cluster, accessing the StorageTek storage at the StorCloud booth via the 16-node Dell Cluster, which implements the Terrascale TerraGRID parallel I/O subsystem.

Direct Numerical Simulation for Homogeneous Charge Compression Auto-Ignition Using Scalable Parallel I/O

S3D is an existing Sandia Direct Numerical Simulation (DNS) compressible Navier-Stokes solver coupled with an integrator for detailed chemistry. The development of S3D has been supported by DOE's Office of Science and Basic Energy program. DNS databases generated by S3D are also used by the ASC Alliance program to test and validate turbulent combustion sub models for large-eddy simulations. A typical S3D time-step in a 2D run generates 600 MB of restart and visualization data. As such, production runs are constrained to save data at predetermined intervals, creating opportunities for potential loss of critical information, more complicated analysis, and difficulties in real-time code steering. We will demonstrate the acceleration of a high fidelity S3D simulation using the same 48-node Appro Mid-Cluster to access Panasas's ActiveScale filesystem located at the StorCloud booth.



5 Conclusion and Future Work

Our initial study demonstrated that iSCSI transport can deliver throughput at the gigabit Ethernet rate, even when iSCSI is implemented in software. The fact that iSCSI uses TCP/IP as its data transport allowed us to tap the wealth of knowledge from years of research to easily master tuning techniques for iSCSI to

perform over distance. We are confident that iSCSI is a suitable choice for implementing a high-performance, distributed, enterprise filesystem for transparent sharing of data over WAN.

Confident that iSCSI can be tuned to perform in LAN and WAN, we then evaluated an iSCSI-based parallel storage subsystem called TerraGRID. TerraGRID uses a proprietary, Shared Access Scheduling Scheme to turn multiple standalone Linux ext2 filesystems into a massively parallel global filesystem. Our test demonstrated that TerraGRID can scale to deliver optimal sequential reads and writes to parallel applications running on a 16-node testbed. We observed that TerraGRID was able to push enough I/O to saturate the aggregate disk bandwidth across all servers. In order to make a more meaningful assessment on TerraGRID's ability to synchronize its metadata and to manage distributed locks while servicing concurrent I/O requests, however, we plan to continue our study on clusters with 128 or more nodes.

Low latency is critical to a parallel filesystem's ability to scale its metadata management. As such, extending data-sharing to remote clients can be challenging because of the inherent propagation delay. However, we believe remote visualization only performs large sequential "reads", and since read operations do not modify file content, they will not incur much metadata update if any, and therefore, its performance should not be impacted by the presence of propagation delay. In the case of interactive compute steering, we envision that exclusive leases for the state of a file's metadata can be granted exclusively to each remote client, which in effect suspends the execution of the parallel applications that are waiting for steering in any case. This again will render WAN latency irrelevant with respect to the application's overall performance. We will continue our study to validate this hypothesis and at the same time investigate whether parallel NFS [27] can provide a standardized interface to export data sharing capabilities of many global filesystems.

Due to delayed deployment, we were not able to fully evaluate iSCSI security. But because iSCSI sits on top of TCP/IP, it has optional provisions for full IPsec [28], which includes user authentication, message authentication, encryption, and key exchange functionality. Because iSCSI is an application that runs with TCP/IP, its security implementation can leverage proven technology based on existing IPsec standards. Together with iSCSI's ability to use Access Control Lists (ACLs) on its target, we believe basic components exist to implement proper protections for an iSCSI-based distributed filesystem.

10 gigabit Ethernet is appearing in backbone networks today and will be adopted in server platforms within a year. The order of magnitude jump in speed, however, has generated concerns in a server's ability to sustain maximum throughput. A typical software implementation of TCP/IP consists of 70,000 instructions, requiring 1 GHz of processing power to drive at a 1 gigabit Ethernet rate. In addition to processing the TCP/IP protocol, more compute cycles are needed to service interrupts for moving data from the network to the kernel and then to a user buffer. Moreover, with each move, limited memory-bus bandwidth is consumed, keeping the server CPU from performing useful tasks. At 10 Gbps, this burden will be prohibitive. TCP offload engines (TOE) and Remote Direct Memory Access (RDMA) technologies are being developed to implement intelligent network interface cards (RNIC) [29] targeted to resolve this issue. How to integrate the emerging RNIC into our I/O infrastructure is critical to the future of HPC and thus is an area of interest for future study.

6 References

[1] Jason King, "Parallel FTP Performance in a High-Bandwidth, High-Latency WAN", http://www.llnl.gov/asci/discom/sc00paper_final.pdf, November 2000

[2] Hufferd, John L., iSCSI – The Universal Storage Connection, Addison-Wesley, 2003

[3] Simitci, Huseyin, "Storage Network Performance Analysis", Wiley Publishing, 2003, pg. 116-117, 290-295

[4] Schmidt, Friedhelm, "The SCSI Bus & IDE Interface – Protocols, Applications & Programming", Addison-Wesley, 1997

[5] Stevens, W. Richard, "TCP/IP Illustrated Volume 1 – The Protocol", Addison-Wesley Publishing Company, February 1994, pg. 297-321

[6] Stallings, William, "Networking Standards – A Guide to OSI, ISDN, LAN, and MAN", Addison-Wesley Publishing Company, July 1993, pg. 270-312

[7] http://www.stalker.com/notes/SFS.html

[8] http://www.sandia.gov/ASC/discom.html

[9] J. Mahdavi, "Enabling High Performance Data Transfers on Hosts: (Notes for Users and System Administrators)", Technical note, Revised: December 1997.

[10] Jin, D. Wei, SH Low G. Buhrmaster, J. Bunn, DH Choe, RLA Cottrell, JC Doyle W. Feng, O. Martin, H. Newman, "FAST TCP: From Theory to Experiments", http://netlab.caltech.edu/FAST

[11] Floyd, Sally, "High-speed TCP for Large Congestion Windows", RFC 3649, December 2003, ftp://ftp.rfc-editor.org/in-notes/rfc3649.txt

[12] Brakmo, L., O'Malley, S. and Peterson, L., "TCP Vegas: New Techniques for Congestion Detection and Avoidance", Proceedings of the SIGCOMM '94 Symposium, August 1994, pg. 24-35

[13] W. Feng, M. Fisk, M. Gardner, and E. Weigle, "Dynamic Right-Sizing: An Automated, Lightweight, and Scalable Technique for Enhancing Grid Performance," Lecture Notes in Computer Science, 2334: 69-83, 2002

[14] Mathis, Heffner, Reddy, "Web100: Extended TCP Instrumentation for Research, Education and Diagnosis," ACM Computer Communications Review, Vol. **33**, (3), July 2003.

[15] MPI http://www.beedub.com/clusterfs.html

[16] http://www.faqs.org/rfcs/rfc1094.html

[17] http://www.stalker.com/notes/SFS.html

[18] http://terrascale.com/prod_over_e.html

[19] Futral, William T., "InfiniBand Architecture – Development and Deployment, a Strategic Guide to Server I/O Solutions", Intel Press, 2001

[20] www.sc-conference.org/sc2004/storcloud.html, November, 2004

[21] http://blockbuster.sourceforge.net/

[22] http://www.mpqc.org/

[23] JHCHEN@sandia.gov and PPPEBAY@sandia.gov

[24] http://onesis.sourceforge.net/

[25] http://www.panasas.com/library.html

[26] http://www-1.ibm.com/servers/eserver/clusters/software/gpfs.pdf

[27] Talpey, Tom, Shepler, Spencer, Bauman, Jon, http://www.ietf.org/internet-drafts/draft-ietf-nfsv4-sess-00.txt, July, 2004

[28] S. Kent,"IPsec", http://ietf.org/rfc/rfc2401.txt, November 1998

[29] http://h200001.www2.hp.com/bc/docs/support/SupportManual/c00257031/c00257031.pdf

7 Acknowledgement

The authors would like to express our appreciation to Mitch Sukalski, Scott Cranford, and Jeff Decker for their technical input and editorial comments. In addition, we would like to thank the Tri-lab StorCloud team for their outstanding contributions, which made the demonstration a success at the Supercomputing 2004 conference. The infrastructure team members are Josh England, Michael lee, Matt Leininger, Mitch Sukalski, Mitch Williams, and Parks Fields. The Scientific Application team included Joseph Kenny, Curtis Janssen, Philippe Pebay, Jacqueline Chen, David Bremer, and Holger Jones.

Distribution

1 MS 9151 K. Washington.
1 MS 9151 J. Handrock
1 MS 9158 M. W. Sukalski
1 MS 9158 J. P. Kenny
1 MS 9158 C. Jannsen
1 MS 9158 N. R. Bierbaum
1 MS 9158 S. Cranford
1 MS 9158 H. Y. Chen
1 MS 9159 S. Thomas
1 MS 9152 J. Friesen
1 MS 9158 D. Evensky
1 MS 9152 D. Cohen
1 MS 9158 J. England
1 MS 9159 M. Hardwick
1 MS 9151 C. Oien
1 MS 9004 B. Hess
1 MS 9011 J. Howard
1 MS 9011 J. Van Randwyk
1 MS 9011 F. Bielecki
1 MS 9011 T. J. Toole
1 MS 9012 B. Maxwell
1 MS 9012 R. Gay
1 MS 9916 J. Berry
1 MS 9019 S. Carpenter
1 MS 0139 A. Hale
1 MS 0806 L. Stans
1 MS 0806 T. Pratt
1 MS 0806 L. Tolendino
1 MS 0806 S. Gossage
3 MS 9018 Central Technical Files, 8945-1
1 MS 0899 Technical Library, 9616
1 MS 9021 Classification Office, 8511 for Technical Library, MS 0899, 9616
1 MS 9021 Classification Office, 8511 for DOE/OSTI
1 MS 0188 D. Chavez, LDRD Office, 1030