iSER Storage Target for Object-based Storage Devices

To appear in the Proceedings of MSST'07, SNAPI Workshop, San Diego, CA, September 2007.

Dennis Dalessandro
Ohio Supercomputer Center
dennis@osc.edu

Ananth Devulapalli
Ohio Supercomputer Center
ananth@osc.edu

Pete Wyckoff
Ohio Supercomputer Center
pw@osc.edu

Abstract

In order to increase client capacity and performance, storage systems have begun to utilize the advantages offered by modern interconnects. Previously storage has been transported over costly fibre channel networks or ubiquitous but low performance Ethernet networks. However, with the adoption of the iSCSI extensions for RDMA (iSER) it is now possible to use RDMA based interconnects for storage while leveraging existing iSCSI tools and deployments.

Building on previous work with an object-based storage device emulator using the iSCSI transport, we extend its functionality to include iSER. Using an RDMA transport brings with it many design issues including the need register memory to be used by the network, and how to bridge the quite different RDMA completion semantics with existing event management based on file descriptors.

Experiments demonstrate reduced latency and greatly increased throughput compared to iSCSI implementations both on Gigabit Ethernet and on IP over InfiniBand.

1. Introduction

The performance of modern interconnect networks has grown by leaps and bounds. One of the biggest impacts on performance has been the introduction of remote direct memory access (RDMA). RDMA brings two main benefits. One is to provide protocol offload, the other is a zero-copy approach to sending and receiving data. However, the programming interfaces for using RDMA are a significant departure from existing sockets-based interfaces The semantics of flow control, event handling, and buffer management are also quite different.

Storage systems have been slow to adapt to changes in modern networking technology. Many deployments still use specialized networks like fibre channel to connect storage systems to compute resources. The iSCSI specification [6] makes the important contribution of allowing existing storage protocols (SCSI) to be transported over general purpose IP networks, including Ethernet. Recent drafts propose iSCSI extensions for RDMA (iSER) [4] which further permit the transport of SCSI commands on RDMA-capable interconnects such as InfiniBand and iWARP. Altogether, these changes permit the use of commodity high-performance networks for storage applications while continuing to use existing SCSI commands and management tools.

In addition to advancement by utilizing networks, storage devices themselves are undergoing changes. The notion of an object-based storage device (OSD) [7] puts the onus of data layout management on the device itself, rather than in the host system. This new semantic provides the disk the opportunity to be more intelligent in how data is stored and retrieved, offering improvements in performance and manageability. These new devices exploit features of the SCSI protocol rarely used by existing block-based devices, such as bidirectional commands that allow both reading and writing in the same operation, and large command descriptor blocks to transport security and other parameters along with the command. OSDs store attributes along with the objects, and the attributes can be retrieved and modified at the same time that object data is accessed.

Previously we have made use of iSCSI in order to implement an OSD emulator [1]. However, limitations of existing IP networks necessitate switching to RDMA to achieve the network performance required to drive the devices. This paper describes our design process in implementing iSER and presents performance results over InfiniBand. To our knowledge, this is the first open source iSER target, although there do exist proprietary iSER targets in industry. Our iSER implementation will be made freely available to the community.

2. Design

In general, a SCSI system includes two components, an initiator and a target. The initiator submits commands and awaits responses. The target services commands from initiators and returns responses. Data may flow from the initiator, from the client, or both (bidirectional). The iSER specification [4] requires all data transfers to be started by the target, regardless of direction. In a read operation, the target uses RDMA Write to move data to the initiator, while a write operation uses RDMA Read to fetch data from the initiator.

Using the Linux kernel, version 2.6.20 as a base, we modified the existing iSER initiator. Extensions to support extended CDBs and bidirectional data flow are required by many of the OSD commands we are interested in using. However, the bulk of our work is centered around adding iSER support to an existing user-space iSCSI target [2].

In previous work [1] we made target side modifications to support extended CDBs and bidirectional commands. Next we discuss some of the design issues in implementation of iSER target.

Memory registration One of the most severe stumbling blocks in moving any application to take advantage of RDMA features is memory registration. Before using RDMA, both the sending and receiving buffers must be registered with the operating system. This operation ensures that the underlying hardware pages will not be modified during the transfer, and provides the physical addresses of the buffers to the network card. However, the process itself is time consuming, and CPU intensive. Previous investigations [8] have shown that for InfiniBand, with a nominal transfer rate of 900 MB/s, the throughput drops to around 500 MB/s when memory registration and deregistration are included in the critical path.

Our target implementation uses pre-registered buffers for RDMA operations. In general such a scheme is difficult to justify due to the large per-connection resource requirements. However, in this application it may be appropriate. Since the target always initiates RDMA operations and never advertises RDMA buffers, it can securely use one pool of buffers for multiple clients and can manage its memory resources explicitly. Also, the architecture of the code is such that the iSCSI layer dictates incoming and outgoing buffer locations to the storage device layer, so supplying a registered buffer is relatively easy.

Event management There is a mismatch between what the iSCSI target [2] event framework assumes and what the RDMA notification interface that we use can provide. The existing TCP-based iSCSI target code has one file descriptor per connection and it is driven by readability or writeability of the socket. A single poll system call returns which sockets can be serviced, driving the TCP code to read or write as appropriate. The RDMA interface can be used in accordance with this design by requesting interrupts from the network card on work request completions. Notifications appear on the file descriptor that represents a completion queue to which all RDMA events are delivered.

However, the existing sockets-based code goes beyond this and changes the bitmask of requested events to control its code flow. For instance, after it finishes sending a response, it will modify the bitmask to only look for readability. Even if the socket is writeable, there is no data to write, hence polling for that status is not useful. The code also disables new message arrival during command execution as a sort of exclusion facility, again by modifying the bitmask. We cannot do this with the RDMA interface. Hence we must maintain an active list of tasks that have data to write and drive a progress engine to service them. This progress routine is woken up by small writes to a pipe used internally by the single-threaded code to keep track of its state. It blocks in the poll call only when there is nothing left to do.

Data completion semantics InfiniBand [3] does not guarantee that all data will be placed into application memory in the order it was received by the adapter. While work requests are processed in order, there are few restrictions on how the adapter must move incoming data to host memory. The particular case that arose in this work was in the case of SCSI read operations. iSER [4] specification requires all data transfers to be done using RDMA Write and RDMA Read, followed by a SCSI command response packet sent using Send. For SCSI read, the target uses RDMA Write one or more times to move data to the memory of the initiator, then sends a SCSI response packet using Send indicating the status of the command. The RDMA operations do not consume receive work requests or cause completion events on the initiator, and thus are not required to be ordered with respect to the incoming Send operation. To avoid the problem of the response message arriving before the RDMA message is flushed to the user buffer, we force the target to wait for the local completions of the RDMAs it started before sending the response. As iSCSI is based on TCP with its streaming semantics, some aspects such as this require more care when moved to message-based weakly-ordered networks.

Padding The iSCSI specification [6] clearly states that all segments in the protocol data unit (PDU) must be individually padded to four-byte boundaries. However, the iSER specification [4] remains mute on the subject of padding. Without clear guidance from the specifications, and since padding is really only needed with TCP to find PDU boundaries, we chose to use padding only between the segments in a control PDU, including SCSI command PDUs. Thus, any additional header segments (AHSs) are padded with zeroes to the next four bytes so that any following immediate data section will be word-aligned. Padding the data in RDMA transfers has no discernible benefit and adds significant complexity, in particular, the need to register a word of zeroes and add another entry to the gather list on the initiator.

3. Experiments

In order to evaluate iSCSI and iSER performance we test using three network configurations: iSCSI over TCP on Gigabit Ethernet, iSCSI over TCP layered on IP-over-IB, and our iSER implementation on InfiniBand. Mellanox 4X SDR adapters are used for InfiniBand, and Tigon 3 NICs are used for Ethernet. The machines used are equipped with Tyan S2891 mainboards and dual 2.4 GHz Opteron processors with 2 GB memory. Data is read and written to an ext3 file system on an 80 GB SATA drive. The machines run Linux, kernel version 2.6.20, with OpenFabrics RDMA

drivers and libraries [5].

3.1. Latency

To understand the impact of the network on end-to-end application performance, we evaluate the performance of representative OSD commands on the three networks, as shown in Table 1. The ping command is used to verify readiness of the target and is implemented as the SCSI test-unit-ready command. The create and remove commands, respectively, create and remove an OSD object at the target. The getattr and setattr commands modify attributes on an existing object.

Command	GigE	IPoIB	IB
Ping	87 ± 4	36 ± 4	33 ± 4
Create	265 ± 10	220 ± 4	207 ± 3
Remove	257 ± 18	215 ± 11	201 ± 15
Getattr	144 ± 3	86 ± 2	65 ± 1
Setattr	239 ± 54	201 ± 3	175 ± 3

Table 1. Latency for OSD commands in microseconds.

We observe that for all the networks, the ping command is fastest since it is executed at the SCSI layer at the target. The rest of the commands require OSD command processing which involves database operations. Getattr is the second fastest for all networks, since it just reads a record in a table. Setattr is slower since it modifies a table which requires a write lock increasing its latency. Finally create and remove are the slowest since they modify two tables. When we look at the latencies of these operations on different networks, there is progressive improvement in performance as we move from TCP to IPoIB and finally to InfiniBand. This is expected due to the lower latency of InfiniBand $\approx 7\ \mu s$ compared to IPoIB $\approx 16\ \mu s$, and TCP $\approx 40\ \mu s$. But this advantage of InfiniBand is not reflected at the application level due to the protocol and command execution overheads. Except for the getattr command, the commands do not involve data transfer from the target to the initiator. Furthermore, the data sent from the initiator to the target is small and fits in the initial request message, hence no RDMA Read operations are performed by the target. However, according to the iSER specification, the results of the getattr must be transferred using an RDMA Write. There is no iSCSI phase collapse optimization for iSER as there is for TCP and IPoIB. But in spite of the extra round trip, the reduced latency on InfiniBand makes it the fastest of the three networks.

3.2. Single-client Throughput

Figure 1 shows the delivered bandwidth as a function of message size, for a single SCSI read operation. The test was performed many times to calculate the standard deviations shown in the plot. The read throughput for InfiniBand increases with message size up to about 200 KB, then begins to slowly drop off as message size reaches the maximum of 512 KB. (This limitation is due to Linux block I/O buffer management used by the iSCSI initiator.) A similar trend is visible for IPoIB, but this drop is hardly visible in the Gigabit Ethernet case as the network is the bottleneck. The drop in throughput comes from the increasing time in the OSD emulator to process the data, using read system calls to copy the data from the kernel page cache.

As stated before, in iSER, reading of data is accomplished via RDMA writes from the target. Write operations are broken into 256 KB chunks (the default DataSegmentLength in Linux iSCSI), and only the last RDMA write generates a completion event. Once the last RDMA write completes, the command response is sent to the initiator.

The write throughput, is shown in Figure 2. Writes are performed in iSER using RDMA Read operations initiated by the target, again in 256 KB chunks. Here we see similar trends between IPoIB and InfiniBand, except that the IPoIB results are lower than in the read case. As in the iSER case, the data to be written is solicited by the target using "ready to transfer" packets that it sends back to the initiator. This adds another round trip for each data transfer, just like in the RDMA Read case for iSER.

3.3. Multi-client throughput

In Figure 3 we show the overall read performance as the number of concurrent clients accessing the single server increases, with write performance in Figure 4. In these two graphs the message size is fixed at 200 KB. The benchmark used to collect this data uses MPI to synchronize the clients at the beginning and end of loops over read or write operations, and calculates the throughput based on the slowest completion time.

Looking at Figure 3, we see that only two clients are sufficient to reach near the maximum InfiniBand wire speed. Effectively the second client allows pipelining of data processing in the OSD device with network processing by the iSCSI and iSER layers. We also observe that as we add more clients, the throughput does not degrade, as expected due to the non-blocking single-threaded event-driven design used in the target code. The IPoIB and GigE cases show no performance improvement with more clients. The latter is certainly limited by the network bit rate, while the former is likely limited by overheads of TCP processing in software.

The write case in Figure 4 requires more clients to reach the full bandwidth, but after 11 clients, the throughput is maxed out and does not degrade just as with the read case. The reason that read requires more clients to saturate the network is likely related to the differences in protocol processing for iSCSI Write vs iSCSI Read. The extra round trip messages to request data from the clients limit network

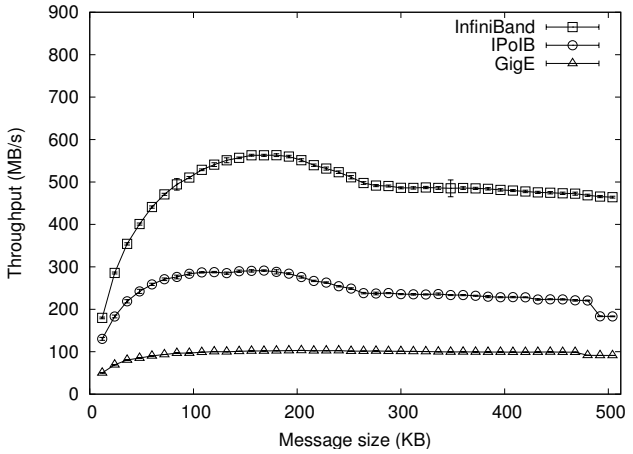

Figure 1. Single client read throughput.

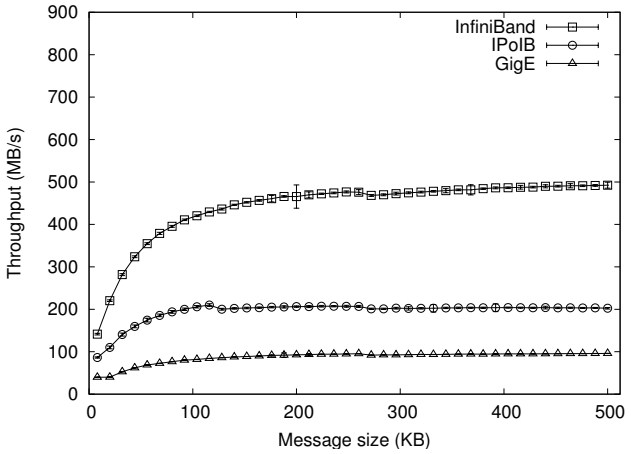

Figure 2. Single client write throughput.

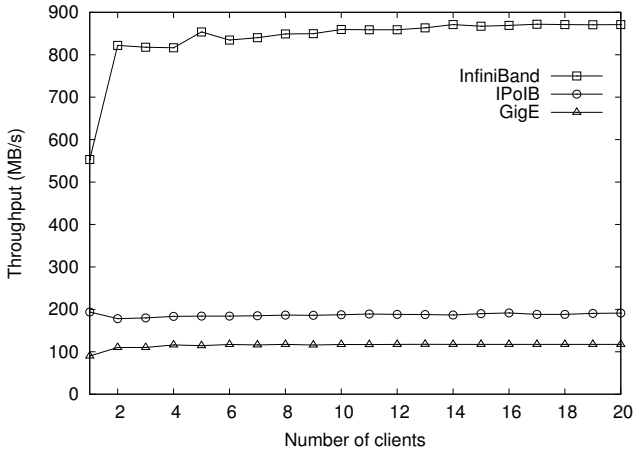

Figure 3. Multi-client read throughput, 200 KB message.

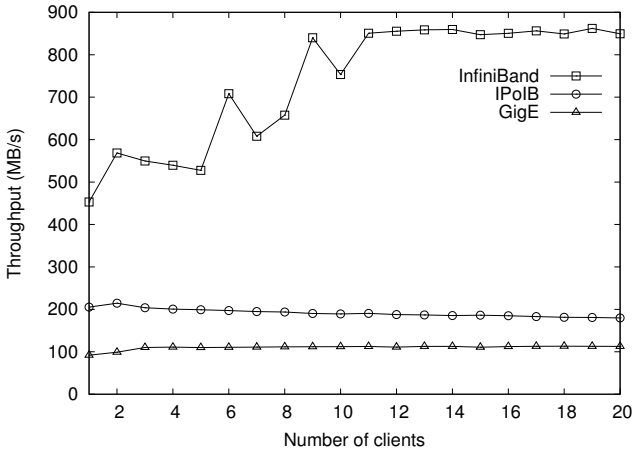

Figure 4. Multi-client write throughput, 200 KB message.

utilization. Adjusting iSCSI parameters to use larger immediate data and to use unsolicited data messages would improve performance at the cost of higher resource utilization per connection on the target. The non-monotonic jumps in the graph come from the difficulty in achieving consistent timings across multiple clients—an MPI barrier releases each initiator at a slightly different time.

4. Conclusion

This paper discusses the design and implementation of an iSER transport layer for a software SCSI target. It permits the use of RDMA networks for SCSI commands and delivers much higher performance than existing non-RDMA alternatives. While designed for the needs of object-based storage devices, it is applicable to all SCSI devices. The code will be merged into the existing open source stgt project soon.

References

[1] A. Devulapalli, D. Dalessandro, N. Ali, and P. Wyckoff. Integrating parallel file systems with object-based storage devices. In *Proceedings of SC'07, to appear*, Reno, NV, Nov. 2007.

[2] T. Fujita and M. Christie. tgt: framework for storage target drivers. In *Proceedings of the Ottawa Linux Symposium*, Ottawa, Canada, July 2006.

[3] InfiniBand Trade Association. *InfiniBand Architecture Specification*, Oct. 2004.

[4] M. Ko, M. Chadalapaka, et al. iSCSI extensions for RDMA. IETF Draft Specification, May 2007.

[5] OpenFabrics Alliance. OpenFabrics libraries. http://www.openfabrics.org/downloads.htm, 2007.

[6] J. Satran, K. Meth, et al. Internet small computer systems interface (iSCSI). IETF RFC 3720, Apr. 2004.

[7] R. O. Weber. SCSI object-based storage device commands (OSD-2). Technical report, INCITS Technical Committee T10/1729-D, Jan. 2007.

[8] P. Wyckoff and J. Wu. Memory registration caching correctness. In *Proceedings of CCGrid'05*, Cardiff, UK, May 2005.