Distributed-and-Split Data-Control Extension to SCSI for Scalable Storage Area Networks

Yitzhak Birk
Technion
birk@ee.technion.ac.il

Nafea Bishara
Technion
nafea@galileo.co.il

Abstract

A "Storage-Area Network" (SAN) comprises computers ("Initiators"), storage "block devices" ("Targets"), and a Controller(s). Most SANs use the SCSI protocol over various communication infrastructures. Presently, all Initiator - Target traffic must pass through the Controller, severely limiting scalability. We extend the SCSI-3 Transport layer to support distribution, and combine this with SCSI's support for data - control split to create DSDC, a novel architecture that can be used over any networking infrastructure: data may be sent directly between Initiators and Targets, relieving the Controller communication bottleneck; the use of multiple paths for data moreover relieves traffic bottlenecks on network links; finally, passing all commands through the Controller retains simplicity. DSDC thus enables the construction of much larger SANs while retaining the simplicity of a single Controller. A prototype SAN using Ethernet and Linux nodes, with DSDC implemented in the iSCSI transport layer protocol and in the Controller's SCSI application layer, has been constructed.

1. Introduction

1.1 Storage-area networks (SANs)

The 1990s were the decade of the stand-alone storage "block server", best exemplified by EMC's Symmetrix. The late 1990s saw the gradual unbundling and increasing modularity of storage subsystem architectures. For example, SCSI-3 is a distributed protocol with packet exchanges, and can be transported over FibreChannel, Infiniband and TCP/IP. (SCSI-1 and SCSI-2 [1], in contrast, are shared-bus, block-based I/O protocols, bundling together the various communication layers and the command set.)

Recently, further unbundling has been taking place with the emergence of Storage-Area Network (SAN) architectures, comprising storage block devices (Targets), clients (Initiators) such as application servers and file-servers, and stand-alone Controllers, all interconnected by a Packet-switched network (Fig. 1). SCSI-3 has become the de-facto SAN I/O protocol.

The storage Controller is the heart of a SAN, providing the Initiators with a simple view of the Targets, and providing functions such as RAID, disk virtualization, mirroring, remote backup and snapshots. Stand-alone Controllers facilitate interoperability and competition among Target vendors. The storage Controller normally manages one or more Targets. A Target can be a JBOD (Just a bunch of disks), an advanced disk array, or even

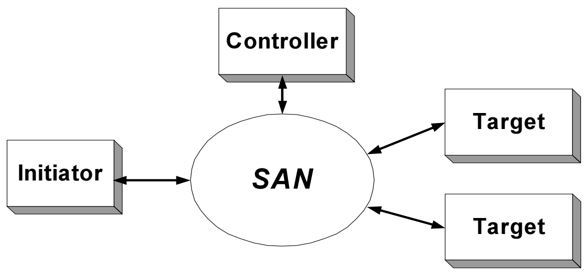

Figure 1. A SAN with an initiator, controller, and two targets

another storage Controller in a hierarchical architecture. A SCSI-3 Controller is a special SCSI device that relays SCSI commands from an Initiator to one or more Targets, often while translating the SCSI command. It impersonates a Target to the Initiator, and an Initiator to the Target. (Certain lower-level functions continue to be carried out within the Targets.)

1.2 SAN scalability limitations

Presently, all communication between Initiators and Targets passes through the SAN Controller. With 10Gb/s FibreChannel, 10Gb/s Ethernet/iSCSI and Infiniband, and with single-disk transfer rates exceeding 1Gb/s, Controller bandwidth is becoming the bottleneck and is limiting SAN scalability. This is most pronounced in communication-intensive applications such as content servers. The problem can be overcome with distributed control, but at the cost of much greater complexity. Much effort has therefore been aimed at relieving this bottleneck, and this is also our goal in this paper.

1.3 Related work

Several studies focused on network-attached storage (NAS) and distributed file servers or network-attached secure/storage devices (NASD) [2][3]. In all these, relatively high-level semantics, such as files and objects, are used by the storage-containing entities, and some even execute parts of applications.

Other studies focused on block-based storage area networks (SANs). Parallel Transport Protocol (PTP) [4] was proposed as a generic protocol for distributed and parallel data transfers. It supports data transfer over several connections between a source and a sink device, optionally controlled by a 3rd party Controller. PTP is a generic data movement protocol (Like a "network DMA") that could be applied to SCSI, Network File System (NFS), etc. However, the use of PTP in SANs would require a major software modification to all SCSI entities, and is totally different from the modern SCSI transport protocols.

Various derivatives of "network DMA", entitled "Remote DMA" (RDMA), focus on efficient data placement among peers in order to save memory copies, and are thus orthogonal to our work.

iSCSI [5] defines a multiple-connection session between a SCSI Initiator and a SCSI Target in order to boost performance, and provides a fast fail-over/recovery mechanism. However, the connections must all have the same end points. Therefore, all data and control still go through the Controller. The IETF iSCSI working group rejected the "multi-connection with split control-data protocol" (also referred to as Asymmetric Connection) due to "complexity" and "lack of consensus", fearing that it would delay agreement and implementation of iSCSI. In this work, we show that things can be manageable.

To our knowledge, none of the previous studies addressed the Controller communication bottleneck problem in conjunction with the SCSI-3 delivery subsystem, which is the de-facto most common storage subsystem protocol.

1.4 Outline of this paper

In this paper, we focus on block-based SANs. Our goal is to relieve the communication bottleneck in the Controller, while retaining the simplicity of centralized control. While so doing does not permit infinite scalability, it enables the construction of very large, yet simple, SANs. An additional goal is to achieve the above in conjunction with SCSI-3, while minimizing changes: no changes to the SCSI application layer (and, of course, to applications), no changes to the network layer, no hardware changes, and only minimal changes to the SCSI transport layer. (The SCSI transport layer is above the networking layer, and should not be confused with the network's transport layer.) Finally, interoperability with unmodified entities is important as well.

We introduce Distributed-and-Split Data-Control (DSDC), a generic extension to any SCSI transport layer protocol. DSDC boosts SAN performance and enhances scalability. SCSI-DSDC can co-exist with non DSDC-enabled SCSI devices.

The remainder of the paper is organized as follows. Section 2 reviews SCSI-3. Section 3 presents DSDC. Section 4 discusses correctness, completeness, and side effects of DSDC. Section 5 describes the prototype and presents some performance results, and Section 6 offers concluding remarks.

2. The SCSI-3 protocol suite

2.1 The SCSI-3 layered model

SCSI-3 splits the block-based protocol into three layers: SCSI Application layer, SCSI Transport layer and SCSI Physical layer (Fig. 2). The latter is actually the entire network layer, and we refer to it as such.

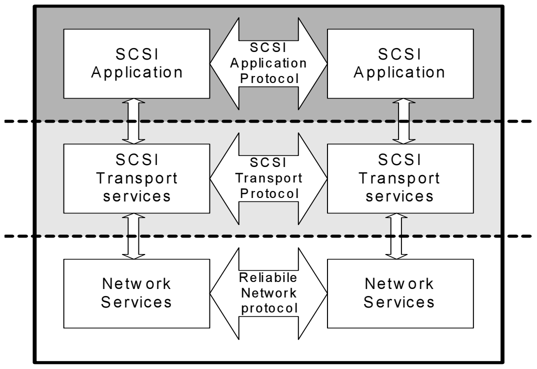

Figure 2. The SCSI-3 layered model

The application layer generates SCSI commands, or Command Descriptor Blocks (CDB), and receives a status on every command. A command is passed (along with a read or write buffer pointer when relevant) to the SCSI Transport layer. The SCSI transport protocol is a wrapper protocol responsible for exchanging SCSI commands, data and status among different SCSI devices. The transport protocol provides reliable message forwarding among SCSI nodes, with every message delivered once (no lost messages or duplicates), intact and in order. Many SCSI transport protocols have been developed, including "SCSI-3 Fiber Channel Protocol" (FCP), "SCSI-3 Bus Protocol" (SBP, or SCSI-2) and "SCSI over TCP/IP". The SCSI Network layer can be implemented over general-purpose networks or dedicated physical layers.

2.2 SCSI-3 commands

SCSI-3 includes hundreds of commands, but our focus is on the READ and WRITE commands, as these move data. Handling of other commands is not changed by DSDC. The data flows in READs and WRITEs are controlled by the Target. It sends READ data to the Initiator unconditionally, and schedules WRITE data transfers by sending to the Initiator a special SCSI transport-layer message, "Ready-To-Transfer" (R2T).

SCSI WRITE Commands

There are two types of SCSI WRITE transactions from the transport layer's point of view: SOLICITED WRITE, wherein the Target paces the transfer using R2T, and UNSOLICITED WRITE, wherein the Initiator sends the data and the Target has no control over the data flow. All SCSI transport protocols support SOLICITED WRITE, and recommend its use in SANs. We only modify Solicited WRITE.

SOLICITED SCSI WRITE:

Step 1: A command is sent by an Initiator. It contains an Initiator Task Tag (ITT), a Target Logical Unit Number (LUN), a Logical Block Address (LBA) and TotalLength.

Step 2: Data is sent to the Target, paced by it using R2Ts. An R2T message includes ITT, Target task tag (TTT), Buffer offset and TotalLength.

Step 3: The Initiator responds to each R2T by sending the data to the Target along with TTT and TotalLength.

Step 4: Status is returned from the Target to the Initiator after all data has been received.

ITT is an Initiator-wide tag uniquely identifying each of multiple outstanding SCSI commands. The Target task tag (TTT) is a target-wide unique tag to identify the data transfer. It is used to distinguish among multiple outstanding R2Ts.

SCSI READ Commands

SCSI READ:

Step 1: The command is sent by an Initiator to a Target. It contains an ITT, a Target LUN, an LBA and Data length (TotalLength).

Step 2: The read data is returned by the Target in one or more data transfers. Each transfer includes: ITT, Read buffer offset, and TotalLength.

Step 3: Status is returned from the Target to the Initiator after all the data has been sent.

Whenever a Controller is involved, there is no direct connection between the Initiator and the Target. Instead, two transactions take place: 1) Initiator – Controller, and 2) Controller – Target. The Controller behaves as a SCSI Target and Initiator, respectively. The Controller receives the SCSI command from an Initiator with ITT(I), TotalLength(I) and LBA(I) as known to the Initiator. It maps this command to one or more Targets, sets the mapped LBA for each Target, and marks each command with its own task tag ITT(C). With multiple Targets, the Controller must also serve as a rendezvous point that aggregates the status returned from all Targets, and must send one status reply back to the Initiator.

SCSI also has extended versions of WRITE and READ commands in support of efficient parity computation [1]. These instruct the Target to compute the bit-by-bit XOR of a block that is sent by the Initiator and one that is stored in the Target. XDWRITE: the new block is written to disk; XPWRITE: the result of the XOR is written to disk; XDREAD: the result of an earlier XOR is returned to the Initiator; XDWRITEREAD: XDWRITE followed by XDREAD. The execution steps are the same as for the basic commands.

3. DSDC

DSDC is a novel, yet simple, extension that can be applied to any SCSI transport protocol. It is confined to software/firmware changes in the SCSI transport layer. The SCSI Application layer, SCSI command set and the disk drive SCSI firmware are not changed at all. Also, SCSI network layer protocols need not change. (In the Controller, changes are also made to the Application layer in order to implement the higher-level functions.)

For READ and SOLICITED WRITE commands, DSDC supports direct data transfers between Initiators and Targets, bypassing the SAN Controller. The paths for control messages (Command and Status) of READ and SOLICITED WRITE commands, as well as the data and control of all other commands, are the same as in the regular SCSI transport. This relieves the communication bottleneck in the Controller while minimizing changes. DSDC assumes the existence of a network layer connection between the Initiator(s) and the Target(s).

3.1 Detailed description of DSDC

For brevity and facility of exposition, we present DSDC using space-time diagrams in the context of a SAN with a single Initiator, a single Controller and *k* Targets. Also, we only depict the SCSI transport layer PDUs (Protocol Data Units). Each arrow in the diagrams represents a message. Arguments added to a message by DSDC are underlined. Arguments created by or associated with an Initiator, Controller and Target T_k are denoted, respectively, by "(I)", "(C)", and "(T_k)". Dashed arrows denote new network-layer Initiator – Target connections that do not presently exist in SANs. We next describe the

SOLICITED WRITE and READ with DSDC, using a generic transport layer protocol.

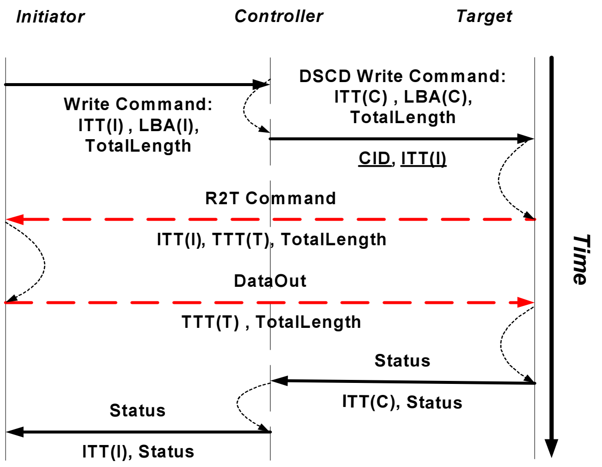

Figure 3. SOLICITED WRITE with DSDC

SOLICITED WRITE Command with DSDC

As depicted in Fig. 3 for a single Target, there are three changes relative to traditional SCSI-3:

1) Direct network connections are opened between the Initiator and the Target(s).
2) When the Controller issues the command to the Target, it adds two arguments: The Connection ID (CID) that instructs the Target which connection to use for data transfer, and the ITT that the Target should plug into the R2T message.

The R2T is sent from the Target to the Initiator directly, with the Initiator's own ITT(I). The Initiator must be able to receive multiple R2Ts concurrently from different Targets, with arbitrary read offsets.

READ Command in DSDC with multiple Targets

Fig. 4 depicts a READ command with a single Initiator, one Controller and two Targets. Here, as with the WRITE command, direct Initiator-Target connections are opened, and the Controller provides the required information to the Targets. Each Controller-to-Target command contains the regular non-DSDC arguments:

- ITT(Ci) is used by the Controller to identify its commands to Target *i*.
- LBA(Ci) is the logical block address in Target *i* after the mapping.
- TotalLength(Ci) is the length of the required data from Target *i*.

DSDC adds three arguments to the SCSI transport layer PDU:

- ITT(I) identifies the Initiator's Tag to the Target, enabling it to send data directly to the Initiator.
- CID(Ci) specifies the connection that Target *i* should use for the transfer.
- FixedOffset(Ci) is the relative offset of the READ data from Target *i* in the Initiator buffer. This permits out-of-order data delivery from the different Targets.

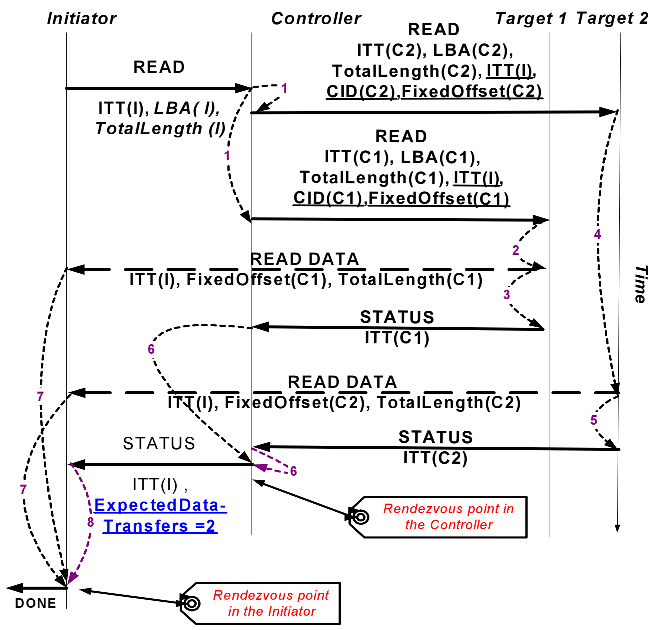

Figure 4. Multi-Target READ with DSDC

Acting on the commands received from the Controller, each Target sends data directly to the Initiator over a network layer connection added by DSDC.

Following the data (arrows 3 and 5), a Target sends its status to the Controller, which in turn aggregates the returned status and sends it to the Initiator (as usual). It also appends "ExpectedDataTransfers", the number of successful status messages returned from the various Targets. Each such message corresponds to a data transmission on the direct Target-Initiator connection.

Data and status may arrive out of order. In order to detect the end of the READ transaction and its final status, the Initiator performs a simple algorithm [6]. The concept is that the status packet tells the Initiator how many data transfers to expect, and the Initiator compares this with the actual number of received data transfers in order to detect the end of the entire transfer. The Initiator may receive data blocks before it knows the number of expected transfers, but begins testing for completion only once this information becomes available.

3.2 Summary of changes to SCSI-3 transport

The changes, mostly confined to the SCSI transport-layer software and absolutely none in hardware [6], are:

1. Opening additional network layer Initiator-Target connections, and optionally using them concurrently. (This is simple in a switched SAN.)
2. Adding several arguments to SCSI transport layer protocol data units.
3. Minor modifications to the main data path software in the Initiator, Controller and Target.
4. Changes in the Controller's Application layer in support of the new functions.

3.3 Load implications

The Initiator in a DSDC system must establish persistent connections with the multiple Targets controlled by the Controller. This adds constant overhead to the setup/login phase, and increases the number of transport layer connections (E.g., TCP). In addition, the Initiator must execute a short algorithm to overcome out-of-order reception in READ transactions. This algorithm has a time complexity of O(1) for every data transfer, so we do not expect it to have a major effect on the Initiator. The only additional overhead to the Target is the setup/login phase overhead of opening the TCP connection to the Initiator. We next turn our attention to the data-transfer load for various commands and configurations.

WRITE to a single Target. With DSDC, this does not involve the Controller in any data transfers. Moreover, the Initiator is not affected and the number of block transfers over the network drops from two to one.
WRITE in mirrored systems. DSDC obviates the need for an Initiator-to-Controller transfer, and shifts the rest of the Controller's transfer load to the Initiator.
WRITE in RAID 5 systems. The Initiator is not affected by the use of a RAID [7], since it simply sends the data once, so we focus on the Controller. We consider a K+1 parity group, with the RAID managed by the stand-alone Controller, and assume the use of the extended SCSI commands:

Single-block WRITE. Without DSDC, the Controller must transfer four blocks: receive the "new" block from the Initiator, send it to the Target, receive the XOR of the new and old blocks from the Target, and send this block to the parity Target. With DSDC, the Controller is not involved in the first two transfers, so its data traffic is cut in half. The total number of transferred blocks drops from four to three, thereby also reducing network load.

K-block WRITE. DSDC currently offers no advantage (2K+1 blocks transferred, all through the Controller). However, once support for "third party transfers" is added, permitting Target-to-Target transfers, things change: there is no data traffic through the DSDC Controller, and 2K blocks are transferred in total (K from Initiator to Targets and K from Target to Target for chained parity computation.). Without DSDC, one can either have 2K+1 transfers that involve the Controller, or 2K that involve the Controller plus K additional Target-to-Target transfers.

READ. The Controller is not involved in any data transfers. Initiator and Target are unaffected, so network data traffic is also halved.

4. Additional issues

Completeness follows from the fact that all commands are handled.
Correctness:
- ♦ DSDC uses the same control PDUs in the same sequence.
- ♦ The written data is received by the Targets before the Initiator gets the status message and releases buffers.
- ♦ DSDC assures correct Initiator handling of out-of-order arrivals of READ data and status messages.
- ♦ PDU loss, link failure and recovery are covered [6].
- ♦ DSDC introduces no new security problems [6].

Performance side-effects. Controllers feature large caches in order to decrease the load on the Targets and support the Initiators with faster responses. With DSDC, caching can be provided in two ways:
- ♦ Move the caching to the Targets. (Caches already exist in medium- and high-end Targets.)
- ♦ Let the Controller decide, per transaction, whether to use its cache or direct Initiator-Target data transfers.

Interoperability with non-DSDC devices. A Controller discovers the capabilities of Initiators and Targets in the "login" negotiation phase, and can work as a conventional non-DSDC Controller. This does not require any change in the non-DSDC Initiator or Target. A DSDC Controller can moreover interact concurrently with Initiators and Targets that support DSDC and with ones that do not.
See [6] for details of topics discussed in this section.

5. Prototype and performance measurements

The prototype comprises four PCs running Linux: one Target, including a SCSI disk, one Controller and two Initiators, all interconnected by Ethernet. iSCSI is the SCSI transport layer. All modules are loadable drivers, with no need to change or recompile the kernel. The Controller comprises "glue" between an Initiator driver, a Target driver, and SCSI Controller application drivers that carry out all Controller functions. The changes in the Initiator and Target each required fewer than 50 lines of code. The Controller required more extensive changes, but these were in its application layer.

Fig. 5 depicts a sample single-Target WRITE performance comparison between a conventional SAN and one with DSDC. A single Initiator, Target and Controller were used. The Controller had a 10Mb/s Ethernet connection, whereas the others had 100Mb/s. This simulates the Controller connection bottleneck when multiple Initiators and Targets are used. For data size of 4Kbyte, the I/O per second performance of DSDC and the traditional Controller are identical, clearly indicating the bottleneck is the control processing. When the data transaction size increases to 16KB and 64KB, the system throughput increases in both cases, but DSDC has a four-fold performance advantage.

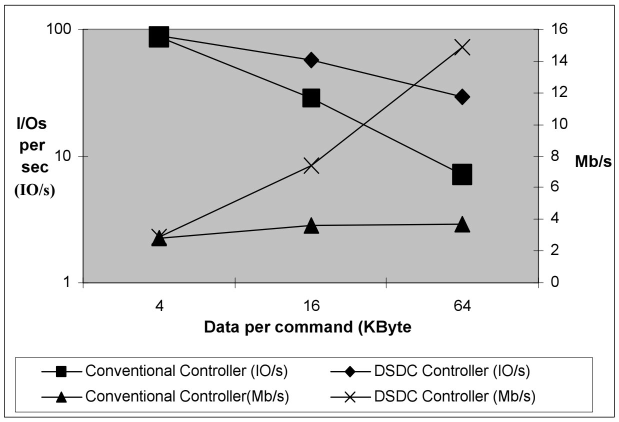

Figure 5. DSDC prototype: WRITE performance

Fig. 6 depicts a sample READ performance comparison using the same system. Without DSDC, the total throughput is limited to 4Mbps regardless of the transaction size. This and the reduction in I/Os per second as transfer size is increased clearly indicate that the system is bandwidth limited. With DSDC, we achieved 25Mbps despite the Controller's 10Mbps connection.

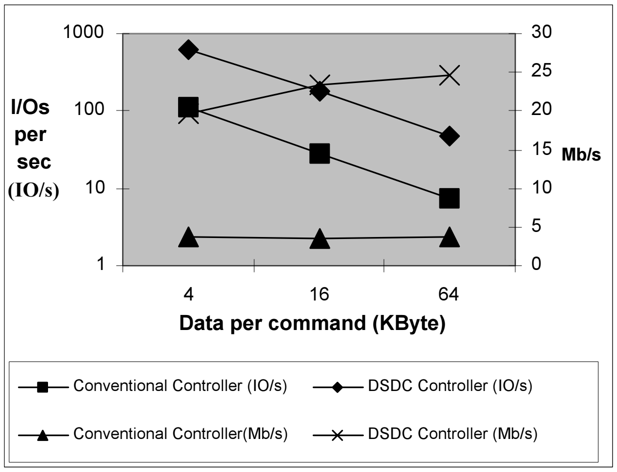

Figure 6. DSDC prototype: READ performance

Changing the Controller interface from 10Mbps to 100Mbps, the traditional Controller gave results close to DSDC, proving that the 4Mbps limit is really a bandwidth limitation. We also found that the maximum throughput of this SAN was close to 25Mbps due to Target disk bandwidth.
Additional results will be reported in [6].

6. Conclusions

DSDC is an extension to the SCSI-3 transport layer that can be used to permit data transfers in a SAN to take place directly between Initiators and Targets. With small, confined changes to existing protocols, we have been able to dramatically alleviate the communication bottleneck in the SAN Controller, except when writing entire parity groups to a Controller-maintained RAID-5. Support of Target-to-Target transfers will extend the benefits to this case as well. DSDC thus permits the construction of much larger SANs before having to resort to complex, distributed control. Moreover, backward compatibility with non-DSDC entities in the SAN is maintained, so migration is simple. The prototype demonstrates the correctness, completeness and ease of implementation. In view of these findings, we believe that DSDC should be incorporated into the SCSI transport-layer standards.

Topics for further research include the application of DSDC to RemoteDMA, as well as cache-location optimization for DSDC.

Acknowledgments and credits go to Evgeny Rivkin, Oran Richman, Saeed Bishara, Liran Liss, Ben-Ami Yasour and Hrvoje Bilic for their assistance via projects in the Parallel Systems Lab, EE Department, Technion.

7. References

[1] "SCSI Standards", ANSI T10 Technical working group, http://www.t10.org
[2] G.A. Gibson and R. Van Meter "Network Attached Storage Architecture", Communications of the ACM, vol. 43, no. 11, Nov. 2000.
[3] G.A Gibson et all, "A Case for Network Attached Secure Disks", CMU SCS Tech. Rep. CMU-CS-96-142, 12/1996.
[4] L. Berdahl, "Parallel transport protocol proposal", draft, Lawrence Livermore Nat. Labs, Jan. 1995.
[5] J. Satran et al, "iSCSI Internet standard draft", v. 0.12. IETF IPS Working group, www.ietf.org/internet-drafts/
[6] Y. Birk and N. Bishara, "SCSI-DSDC", Technion EE Tech. Report, in preparation (Sep. 2002).
[7] D. Patterson, G. Gibson, and R. Katz, "A Case for Redundant Arrays of Inexpensive Disks," Proc. ACM SIGMOD, June 1988.