Architecture and Performance Potential of *STICS* — *SCSI-To-IP Cache Storage*

Xubin He[1] and Qing Yang
Department of Electrical and Computer Engineering
University of Rhode Island, Kingston, RI 02881
{hexb, qyang}@ele.uri.edu

Abstract

Data storage plays an essential role in today's fast-growing data-intensive network services. New standards and products emerge very rapidly for networked data storages. Given the mature Internet infrastructure, overwhelming preference among IT community recently is using IP for storage networking because of economy and convenience. iSCSI is one of the most recent standards that allows SCSI protocols to be carried out over IP networks. However, there are many disparities between SCSI and IP in terms of speeds, bandwidths, data unit size, and design considerations that prevent fast and efficient deployment of SAN (Storage Area Network) over IP. This paper introduces *STICS* (*SCSI-To-IP Cache Storage*), a novel storage architecture that couples reliable and high-speed data caching with low-overhead conversion between SCSI and IP protocols. A STICS block consists of one or several storage devices such as disks or RAID, and an intelligent processing unit with CPU and RAM. The storage devices are used to cache and store data while the intelligent processing unit carries out caching algorithm, protocol conversion, and self-management functions. Through efficient caching algorithm and localization of certain unnecessary protocol overheads, STICS can significantly improve performance, reliability, manageability, and scalability over current iSCSI systems. Furthermore, STICS can be used as a basic plug-and-play building block for data storage over IP. Analogous to "*cache memory*" invented several decades ago for bridging the speed gap between CPU and memory, STICS is the first-ever "*cache storage*" for bridging the gap between SCSI and IP making it possible to build efficient SAN over IP. We have carried out a partial implementation and simulation experiments to study the performance potential of STICS. Numerical results using popular PostMark benchmark program and EMC's trace have shown dramatic performance gain over the iSCSI implementation.

Keywords: Cache, Disk I/O, Networked Storage, NAS, SAN, iSCSI

[1] We would like the paper to be considered for best student paper nomination.

1. Introduction

As we enter a new era of computing, data storage has changed its role from "secondary" with respect to CPU and RAM to primary importance in today's information world. Online data storage doubles every 9 months [2] due to ever growing demand for networked information services [12, 24]. While the importance of data storage is a well-known fact, published literature is limited in the computer architecture research community reporting networked storage architectures. We believe this situation should and will change very quickly as information has surpassed raw computational power as the important commodity. As stated in [4], "In our increasingly internet-dependent business and computing environment, network storage is the computer".

In general, networked storage architectures have evolved from network-attached storage (NAS) [4, 6, 16], storage area network (SAN) [11, 17, 19], to most recent storage over IP (iSCSI) [6, 9, 21]. NAS architecture allows a storage system/device to be directly connected to a standard network, typically via Ethernet. Clients in the network can access the NAS directly. A NAS based storage subsystem has built-in file system to provide clients with file system functionality. SAN technology, on the other hand, provides a simple block level interface for manipulating nonvolatile magnetic media. Typically, a SAN consists of networked storage devices interconnected through a dedicated Fibre Channel network. The basic premise of a SAN is to replace the current "point-to-point" infrastructure of server to storage communications with one that allows "any-to-any" communications. A SAN provides high connectivity, scalability, and availability using a specialized network interface: Fibre Channel network. Deploying such a specialized network usually introduces additional cost for implementation, maintenance, and management. iSCSI is the most recent emerging technology with the goal of implementing the SAN technology over the better-understood and mature network infrastructure: the Internet (TCP/IP).

Implementing SAN over IP brings economy and convenience whereas it also raises issues such as performance and reliability. Currently, there are basically two existing approaches: one encapsulates SCSI protocol in TCP/IP at host bus adapter (HBA) level [21] and the other carries out SCSI and IP protocol conversion at a specialized switch [17]. Both approach have severe performance limitations. To encapsulate SCSI protocol over IP requires significant amount of overhead traffic for SCSI commands transfers and handshaking over the Internet. Converting protocols at a switch places special burden to an already-overloaded switch and creates another specialized networking equipment in a SAN. Furthermore, the Internet was not designed for transferring storage data blocks. Many features such as MTU (Maximum Transfer Unit), data gram fragmentation, routing, and congestion control may become obstacle for providing enough instant bandwidth for large block transfers of storage data.

This paper introduces a new storage architecture called STICS (SCSI-To-IP Cache Storage) aimed to solve the above-mentioned issues facing storage designers to implement SAN over the Internet. A typical STICS bock consists of one or several storage devices such as disks or RAID and an intelligent processing unit with an

embedded processor and sufficient RAM. It has two standard interfaces: one is SCSI interface and the other is standard Ethernet interface. Besides the regular data storage in STICS, one storage device is used as a nonvolatile cache that caches data coming from possibly two directions: block data from SCSI interface and network data from Ethernet interface. In addition to standard SCSI and IP protocols running on the intelligent processing unit, a local file system also resides in the processing unit. The file system is not a standard file system but a simplified Log-structured file system [22] that writes data very quickly and provides special advantages to cache data both ways. Besides caching storage data in both directions, STICS also localizes SCSI commands and handshaking operations to reduce unnecessary traffic over the Internet. In this way, it acts as a storage filter to discards a fraction of the data that would otherwise move across the Internet, reducing the bottleneck imposed by limited Internet bandwidth and increasing storage data rate. Apparent advantages of the STICS are:

- It provides an iSCSI network cache to smooth out the traffic and improve overall performance. Such a cache or bridge is not only helpful but also necessary to certain degree because of the different nature of SCSI and IP such as speed, data unit size, protocols, and requirements. Wherever there is a speed disparity, cache helps. Analogous to "*cache memory*" used to cache memory data for CPU, STICS is a "*cache storage*" used to cache networked storage data for server host.
- It utilizes the Log-structured file system to quickly write data into magnetic media for caching data coming from both directions.
- Because STICS uses log disk to cache data, it is a nonvolatile cache, which is extremely important for caching storage data reliably since once data is written to a storage, it is considered to be safe.
- Although both NAS and STICS have storage devices and Ethernet interfaces, STICS is a perfect complement to NAS. NAS allows direct connection to an Ethernet to be accessed by networked computers, while STICS allows direct connection to a SCSI interface of a computer that in turn can access a SAN implemented over the Internet.
- By localizing part of SCSI protocol and filtering out some unnecessary traffic, STICS can reduce the bandwidth requirement of the Internet to implement SAN.
- Active disks [1,14,20] are becoming feasible and popular. STICS represents another specific and practical implementation of active disks.
- It is a standard plug-and-play building block for SAN over the Internet. If ISTORE [2] is standard "brick" for building, then STICS can be considered as a standard "beam" or "post" that provides interconnect and support of a construction (provided that the STICS is "big" and "strong" enough).

Overall, STICS adds a new dimension to the networked storage architectures. To quantitatively evaluate the performance potential of STICS in real world network environment, we have partially implemented the STICS under the Linux OS. While all network operations of STICS are implemented over an Ethernet switch, cache algorithms and file system functions are simulated because of time limit. We have used PostMark [10] benchmark and EMC's trace to measure system performance. PostMark results show that STICS provides 53% to 78% performance improvement over iSCSI implementation

in terms of average system throughput. An order of magnitude performance gain is observed for 90% of I/O requests under the EMC's trace in terms of response time.

The paper is organized as follows. Next section presents the architecture and overall design of STICS. Section 3 presents our initial experiments and performance evaluations. We discuss related research work in Section 4 and conclude our paper in Section 5.

2. Architecture and Design of STICS

Figure 1 shows a typical SAN implementation over IP using STICS. Any number of storage devices or server computers can be connected to the standard Internet through STICS to form the SAN. Instead of using a specialized network or specialized switch, STICS connects a regular host server or a storage device to the standard IP network. Consider STICS 1 in the diagram. It is directly connected to the SCSI HBA of Host 1 as a local storage device. It also acts as a cache and bridge to allow Host 1 to access, at block level, any storage device connected to the SAN such as NAS, STICS 2, and STICS 3 etc. In order to allow a smooth data access from Host 1 to the SAN, STICS 1 provides SCSI protocol service, caching service, naming service, and IP protocol service.

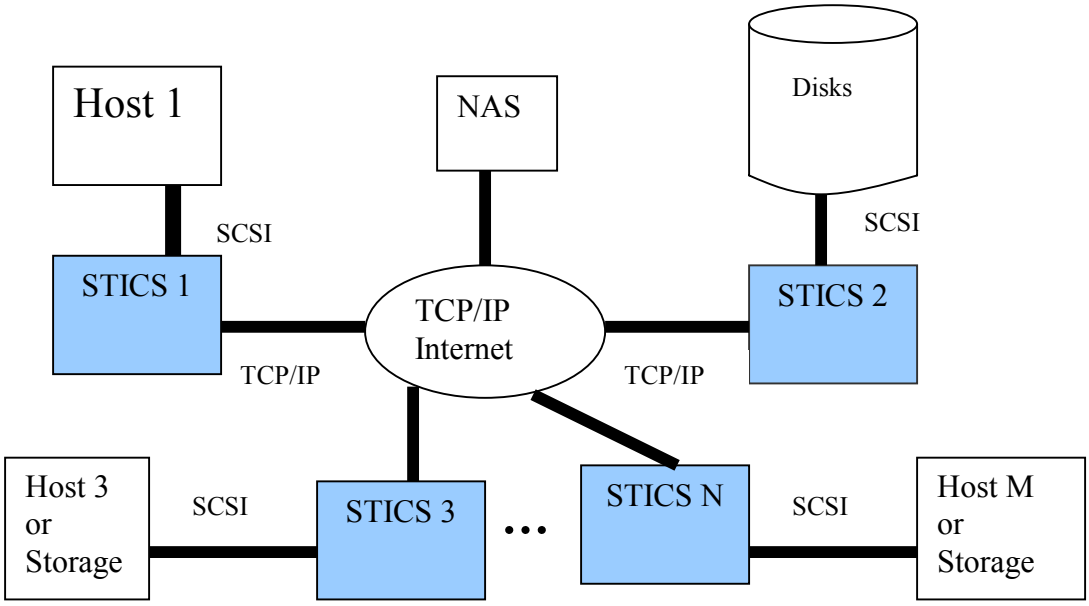


Figure 1: System overview. A STICS connects to the host via SCSI interface and connects to other STICS' or NAS via Internet.

The basic structure of STICS is shown in Figure 2. It consists of five main components:

1) A SCSI interface: STICS supports SCSI communications with hosts and other extended storage devices. Via the SCSI interface, STICS may run under two different modes: initiator mode or target mode [25]. When a STICS is used to connect to a host, it runs in target mode receiving requests from the host, carrying out the IO processing possibly through network, and sending back results to the host. In this case, the

STICS acts as a directly attached storage device to the host. When a STICS is used to connect to a storage device such as a disk or RAID to extend storage, it runs in initiator mode, and it sends or forwards SCSI requests to the extended storage device. For example, in Figure 1, STICS 1 runs in target mode while STICS 2 runs in initiator mode.

2) An Ethernet interface: Via the network interface, a STICS can be connected to the Internet and share storage with other STICS's or network attached storages (NAS).
3) An intelligent processing unit: This processing unit has an embedded processor and a RAM. A specialized Log-structured file system, standard SCSI protocols, and IP protocols run on the processing unit. The RAM is mainly used as buffer cache. A small NVRAM (1-4MB) is also used to maintain the meta data such as hash table, LRU list, and the mapping information (STICS_MAP). These meta data are stored in this NVRAM before writing to disks. The use of the NVRAM avoids frequently writing and reading meta data to/from disks. Alternatively, we can also use Soft Updates [3] technique to keep meta data consistency without using NVRAM.
4) A log disk: The log disk is a sequential accessed device. It is used to cache data along with the RAM above in the processing unit. The log disk and the RAM form a two-level hierarchical cache similar to DCD [7,8].
5) Storage device: The regular storage device can be a disk, a RAID, or *JBOD* (Just-Bunch-Of-Disks). This storage device forms the basic storage component in a networked storage system. From point of view of a server host to which the STICS is connected through the SCSI interface, this storage device can be considered as a local disk. From the point of view of the IP network through the network interface, this storage can be considered as a component of a networked storage system such as a SAN with an IP address as its ID.

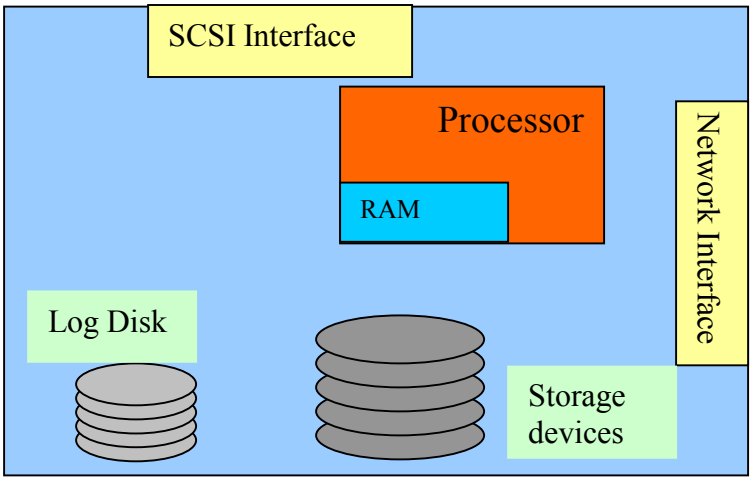


Figure 2: STICS architecture. A STICS block consists of a log disk, storage device, an intelligent processing unit with an embedded processor and sufficient RAM. Each STICS block has one SCSI interface and one network interface.

2.1 STICS naming service

To allow a true "any-to-any" communication between servers and storage devices, a global naming is necessary. In our design, each STICS is named by a global location number (GLN) which is unique for each STICS. Currently we assign an IP address to each STICS and use this IP as the GLN.

2.2 Cache Structure of STICS

The cache organization in STICS consists of two level hierarchies: a RAM cache and a log disk. Frequently accessed data reside in the RAM that is organized as LRU cache as shown in Figure 3. Whenever the newly written data in the RAM are sufficiently large or whenever the log disk is free, data are written into the log disk. There are also less frequently accessed data kept in the log disk. Data in the log disk are organized in the format of *segments* similar to that in a Log-structured File System [22]. A segment contains a number of *slots* each of which can hold one data block. Data blocks in segments are addressed by their *Segment IDs* and *Slot IDs*.

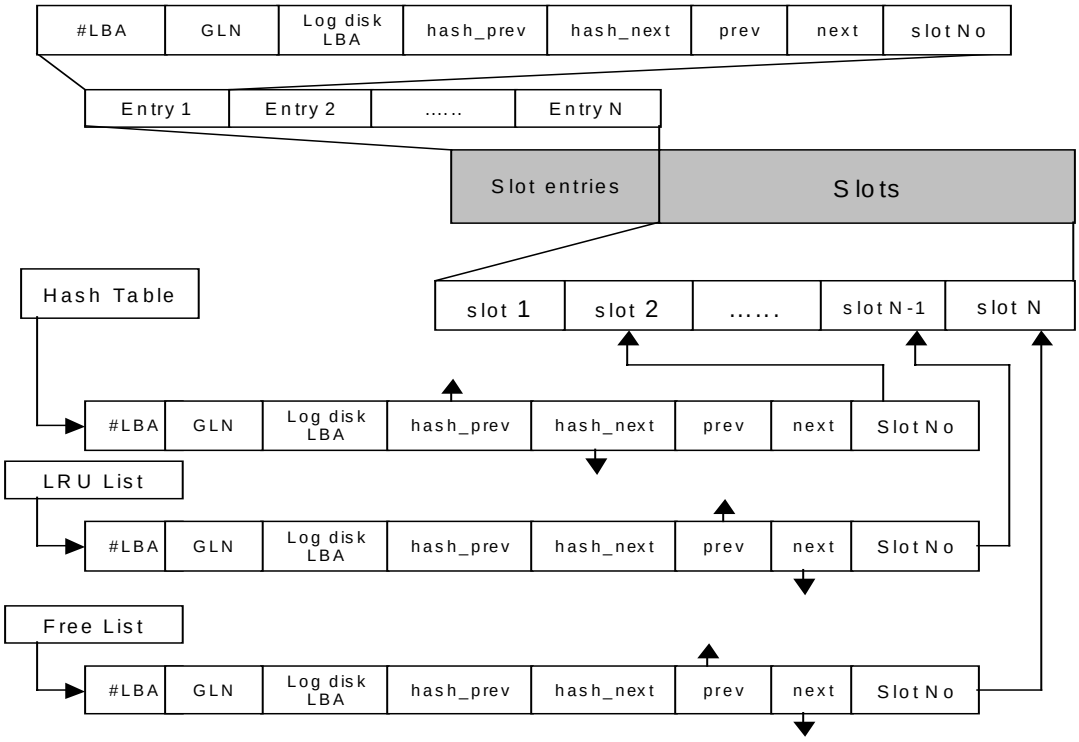


***Figure 3**: RAM buffer layout. RAM buffer consists of slot entries and slots. The hash table, LRU list and Free list are used to organize the slot entries.*

One of the challenging tasks in this research is to design an efficient data structure and a search algorithm for RAM cache. As shown in Figure 3, the RAM cache consists of a Hash table which is used to locate data in the cache, a data buffer which contains several data slots, and a few In-memory headers. Data blocks stored in the RAM cache are addressed by their *Logical Block Addresses (LBAs).* The Hash Table contains location

information for each of the valid data blocks in the cache and uses LBAs of incoming requests as search keys. The slot size is set to be the size of a block. A slot entry consists of the following fields:

- An LBA entry that is the LBA of the cache line and serves as the search key of hash table;
- Global Location Number (GLN) if the slot contains data from or to other STICS.
- A log disk LBA is divided into 2 parts:
  1) A state tag (2 bits), used to specify where the slot data is: IN_RAM_BUFFER, IN_LOG_DISK, IN_DATA_DISK or IN_OTHER_STICS;
  2) A log disk block index (30 bits), used to specify the log disk block number if the state tag indicates IN_LOG_DISK. The size of each log disk can be up to 2^{30} blocks.
- Two pointers (hash_prev and hash_next) are used to link the hash table;
- Two pointers (prev and next) are used to link the LRU list and FREE list;
- A Slot-No is used to describe the in-memory location of the cached data.

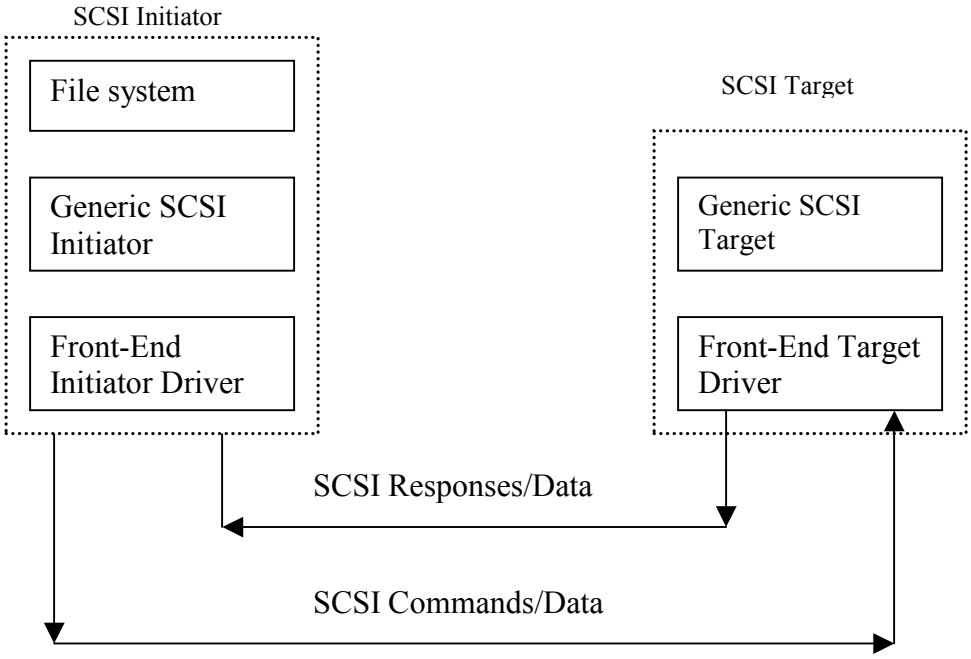


Figure 4: SCSI initiator and target sub-systems.

2.3 STICS modes

As we mentioned above, A STICS may run under two modes: initiator mode or target mode. SCSI initiator and target modes are outlined in Figure 4. When running in target mode, a STICS is connected to a host and the host is running in initiator mode. Otherwise a STICS runs in initiator mode. Initiator mode is the default mode of SCSI. All server host platforms including Linux support SCSI initiator mode. We use the standard SCSI

initiator mode in our STICS. The SCSI target runs in parallel to the initiator and is concerned only with the processing of SCSI commands. We define a set of target APIs for STICS. These APIs include SCSI functions such as SCSI_DETECT, SCSI_RELEASE, SCSI_READ, SCSI_WRITE and etc. When running under target mode, a STICS looks like a standard SCSI device to a connected host.

2.4 Basic operations

For each STICS, we define a variable STICS_LOAD to represent its current load. The higher the STICS_LOAD, the busier the STICS is. When a STICS system starts, its STICS_LOAD is set to zero. When the STICS accepts a request, STICS_LOAD is incremented and when a request finishes, STICS_LOAD is decremented. Besides STICS_LOAD, we define a STICS_MAP to map all STICS loads within the network. STICS_MAP is a set of <GLN, STICS_LOAD> pairs. The STICS_MAP is also updated dynamically.

2.4.1. Write

Write requests may come from one of two sources: the host via SCSI interface and another STICS via the Ethernet interface. The operations of these two types of writes are as follows.

Write requests from the host via SCSI interface: After receiving a write request, the *STICS* first searches the Hash Table by the LBA address. If an entry is found, the entry is overwritten by the incoming write. Otherwise, a free slot entry is allocated from the Free List, the data are copied into the corresponding slot, and its address is recorded in the Hash table. The LRU list and Free List are then updated. When enough data slots (16 in our preliminary implementation) are accumulated or when the log disk is idle, the data slots are written into log disk sequentially in one large write. After the log write completes successfully, STICS signals the host that the request is complete.

Write requests from another STICS via Ethernet interface: A packet coming from the network interface may turns out to be a write operation from a remote STICS on the network. After receiving such a write request upon unpacking the network packet, STICS gets a data block with GLN and LBA. It then searches the Hash Table by the LBA and GLN. If an entry is found, the entry is overwritten by the incoming write. Otherwise, a free slot entry is allocated from the Free List, and the data are then copied into the corresponding slot. Its address is recorded in the Hash table. The LRU list and Free List are updated accordingly.

2.4.2 Read

Similar to write operations, read operations may also come either from the host via SCSI interface or from another STICS via the Ethernet interface.

Read requests from the host via SCSI interface: After receiving a read request, the *STICS* searches the Hash Table by the LBA to determine the location of the data. Data requested may be in one of four different places: the RAM buffer, the log disk(s), the storage device in the local STICS, or a storage device in another STICS on the network. If the data is found in the RAM buffer, the data are copied from the RAM buffer to the requesting buffer. The STICS then signals the host that the request is complete. If the data is found in the log disk or the local storage device, the data are read from the log disk or storage device into the requesting buffer. Otherwise, the *STICS* encapsulates the request including LBA, current GLN, and destination GLN into an IP packet and forwards it to the corresponding STICS.

Read requests from another STICS via Ethernet interface: When a read request is found after unpacking an incoming IP packet, the STICS obtains the GLN and LBA from the packet. It then searches the Hash Table by the LBA and the source GLN to determine the location of the data. It locates and reads data from that location. Finally, it sends the data back to the source STICS through the network.

2.4.3 Destages

The operation of moving data from a higher-level storage device to a lower level storage device is defined as *destage* operation. There are two levels of destage operations in STICS: destaging data from the RAM buffer to the log disk (*Level 1 destage*) and destaging data from log disk to a storage device (*Level 2 destage*). We implement a separate kernel thread, *LogDestage,* to perform the destaging tasks. The *LogDestage* thread is registered during system initialization and monitors the *STICS* states. The thread keeps sleep at most of the time, and is activated when one of the following events occurs: 1) the number of slots in the RAM buffer exceeds a threshold value, 2) the log disk is idle, 3) the *STICS* detects an idle period, 4) the *STICS* RAM buffer and/or the log disk becomes full. *Level 1 destage* has higher priority than *Level 2 destage*. Once the *Level 1 destage* starts, it continues until a log of data in the RAM buffer is written to the log disk. *Level 2 destage* may be interrupted if a new request comes in or until the log disk becomes empty. If the destage process is interrupted, the destage thread would be suspended until the STICS detects another idle period.

As for *Level 1 destage*, the data in the RAM buffer are written to the log disk sequentially in large size (64KB). The log disk header and the corresponding in-memory slot entries are updated. All data are written to the log disk in "append" mode, which ensures that every time the data are written to consecutive log disk blocks.

For *Level 2 destage*, we use a "last-write-first-destage" algorithm according to the LRU List. At this point, we chose a STICS with lowest STICS_LOAD to accept data. Each time 64KB data are read from the consecutive blocks of the log disk and written to the chosen STICS storage disks. The LRU list and free list are updated subsequently.

3. Performance Evaluations

3.1 Methodology

Our experimental settings for the purpose of evaluating the performance of iSCSI and STICS are shown in Figures 5 and 6. Three PCs are involved in our experiments, namely *Trout*, *Cod* and *Squid*. For iSCSI, the Trout serves as the host and the Squid as the iSCSI target as shown in Figure 5. For STICS experiment, we add our STICS simulator between the host and the target as shown in Figure 6. STICS simulator runs in *Cod* that caches data and iSCSI communications. All these machines are interconnected through a 100Mbps switch to form an isolated LAN. Each machine is running Linux kernel 2.4.2 with a 3c905 TX 100Mbps network interface card (NIC) and an Adaptec 39160 high performance SCSI adaptor. The configurations of these machines are described in Table 1 and the characteristics of individual disks are summarized in Table 2.

Table 1: Machines configurations

	Processor	RAM	IDE disk	SCSI disk
Trout	PII-450	192MB	Maxtor 91366u4	N/A
Cod	PII-400	128MB	WDC AC38400	IBM DNES-309170
Squid	K6-500	128MB	Maxtor 91020D6	IBM DNES-318350

Table 2: Disk parameters

Disk Model	Interface	Capacity	Data buffer	RPM	Latency (ms)	Transfer rate(MB/s)	Seek time (ms)	Manufacturer
DNES-318350	Ultra SCSI	18.2G	2MB	7200	4.17	12.7-20.2	7.0	IBM
DNES-309170	Ultra SCSI	9.1G	2MB	7200	4.17	12.7-20.2	7.0	IBM
91366U4	ATA-5	13.6G	2MB	7200	4.18	Up to 33.7	9.0	Maxtor
91020D6	ATA-4	10.2G	256KB	5400	5.56	18.6	9.0	Maxtor
AC38400	UDMA	8.4G	256KB	5400	5.5	16.6	10.5	WDC

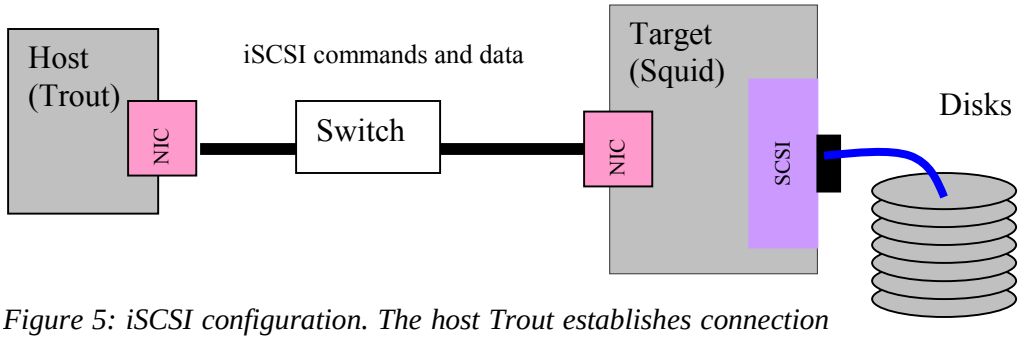


Figure 5: iSCSI configuration. The host Trout establishes connection to target, and the target Squid responds and connects. Then the Squid exports hard drive and Trout sees the disks as local.

For iSCSI implementation, we compiled and run the Linux iSCSI developed by Intel Corporation [9]. The iSCSI is compiled under Linux kernel 2.4.2 and configured as shown in Figure 5. There are 4 steps for the two machines to establish communications via iSCSI. First, the host establishes connection to target; second, the target responds and connects; third, the target machine exports its disks and finally the host sees these disks as local. All these steps are finished through socket communications. After these steps, the iSCSI is in "*full feature phase*" mode where SCSI commands and data can be

exchanged between the host and the target. For each SCSI operation, there will be at least 4 socket communications as follows: 1) The host encapsulates the SCSI command into packet data unit (*PDU*) and sends this PDU to the target; 2) The target receives and decapsulates the PDU. It then encapsulates a response into a PDU and sends it back to the host; 3) The host receives and decapsulates the response PDU. It then encapsulates the data into a PDU and sends it to the target if the target is ready to transfer; 4) the target receives the data PDU and sends another response to the host to acknowledge the finish of the SCSI operation.

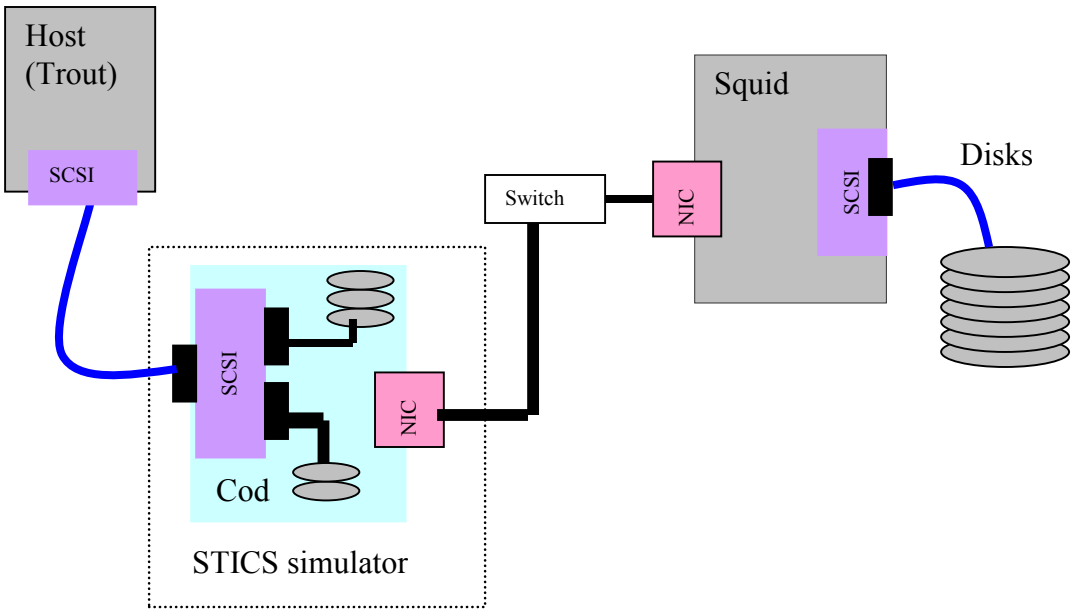


Figure 6: STICS setup. The STICS simulator caches data from both SCSI and network.

We simulated our STICS using a PC running Linux kernel 2.4.2 with target mode support. The STICS simulator is built on top of DCD driver developed by us [7]. The SCSI target mode function is borrowed from the University of New Hampshire Interoperability Lab's SCSI target package [18]. We use 4 MB of the system RAM to simulate STICS NVRAM buffer, and the log disk is a standalone hard drive. A hash table, a LRU list, and mapping information (STICS_MAP) are maintained in the NVRAM. The STICS simulator can run under two modes. To the host (*Trout*), it runs as target mode and to the target storage (*Squid*), it runs as initiator mode. When SCSI requests come from the host, the simulator first processes the requests locally. For write requests, the simulator writes the data to its RAM buffer. When the log disk is idle or the NVRAM is full, the data will be destaged to the log disk through level 1 destage. After data is written to the log disk, STICS signals host write complete. When the log disk is full or the system is idle, the data in log disk will be destaged to the lower level storage. At this point the STICS will decide to store data locally or to the remote disks according to STICS_MAP. In our simulation we store the data to local storage and remote disks equally likely. The hash table and LRU list which reside in the NVRAM are updated. When a read request comes

in, the STICS searches the hash table, locates where the data are, and accesses the data from RAM, log disk, local storage, or remote disks via network.

3.2 Benchmark program and workload characteristics

It is important to use realistic workloads to drive our simulator for a fair performance evaluation and comparison. For this reason, we chose to use real world trace and benchmark program.

The benchmark we used to measure system throughput is PostMark [10] which is a popular file system benchmark developed by Network Appliance. It measures performance in terms of transaction rates in an ephemeral small-file environment by creating a large pool of continually changing files. "PostMark was created to simulate heavy small-file system loads with a minimal amount of software and configuration effort and to provide complete reproducibility [10]." PostMark generates an initial pool of random text files ranging in size from a configurable low bound to a configurable high bound. This file pool is of configurable size and can be located on any accessible file system. Once the pool has been created, a specified number of transactions occur. Each transaction consists of a pair of smaller transactions, i.e. *Create file or Delete file* and *Read file or Append file.* Each transaction type and its affected files are chosen randomly. The read and write block size can be tuned. On completion of each run, a report is generated showing some metrics such as elapsed time, transaction rate, total number of files created and so on.

In addition to PostMark, we also used a real-world trace obtained from EMC Corporation. The trace, referred to as *EMC-tel* trace hereafter, was collected by an EMC Symmetrix disk array system installed at a telecommunication consumer site. The trace file contains 230370 requests, with a fixed request size of 4 blocks. The trace is write-dominated with a write ratio of 94.7%. In order for the trace to be read by our STICS and the iSCSI implementation, we developed a program called *ReqGenerator* to convert the traces to high-level I/O requests. These requests are then fed to our simulator and iSCSI system to measure performance.

3.3 Results and discussions

3.3.1 Throughput

Our first experiment is to use PostMark to measure the I/O throughput in terms of transactions per second. In our tests, PostMark was configured in two different ways as in [10]. First, a small pool of 1,000 initial files and 50,000 transactions; and second a large pool of 20,000 initial files and 100,000 transactions. The total sizes of accessed data are 330MB (161.35MB read and 168.38MB write) and 740MB (303.46 MB read and 436.18MB write) respectively. They are much larger than the system RAM (128MB). The block sizes change from 512 bytes to 4KB. The IO operations are set to synchronous mode. We left all other PostMark parameters at their default settings.

In Figure 7, we plotted two separate bar graphs corresponding to the small file pool case and the large one, respectively. Each pair of bars represents the system throughputs of STICS (light blue bars) and iSCSI (dark red bars) for a specific data block size. It is clear from this figure that STICS shows obvious better system throughput than the iSCSI. The performance improvement of STICS over iSCSI is consistent across different block sizes and for both small pool and large pool cases. The performance gain of STICS over iSCSI ranges from 53% to 78%.

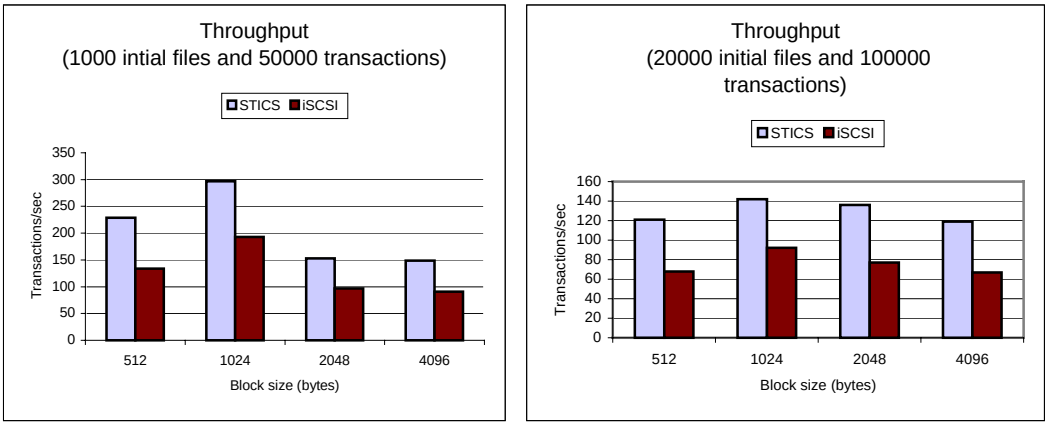


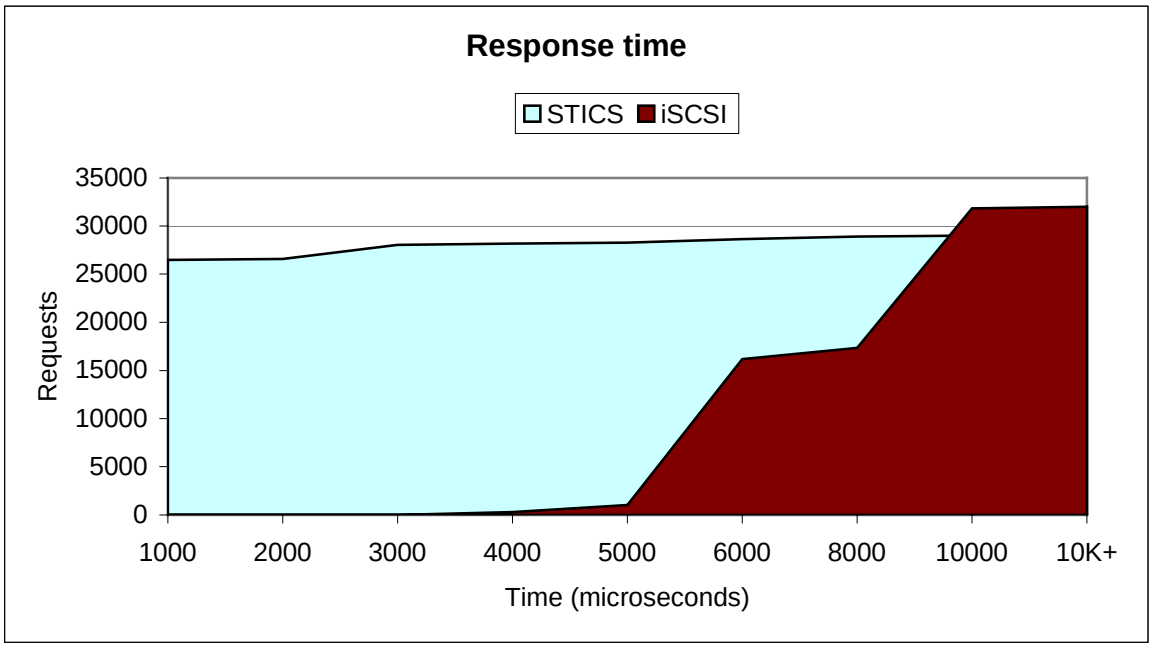
Figure 7: Postmark measurements.

3.3.2 Response times

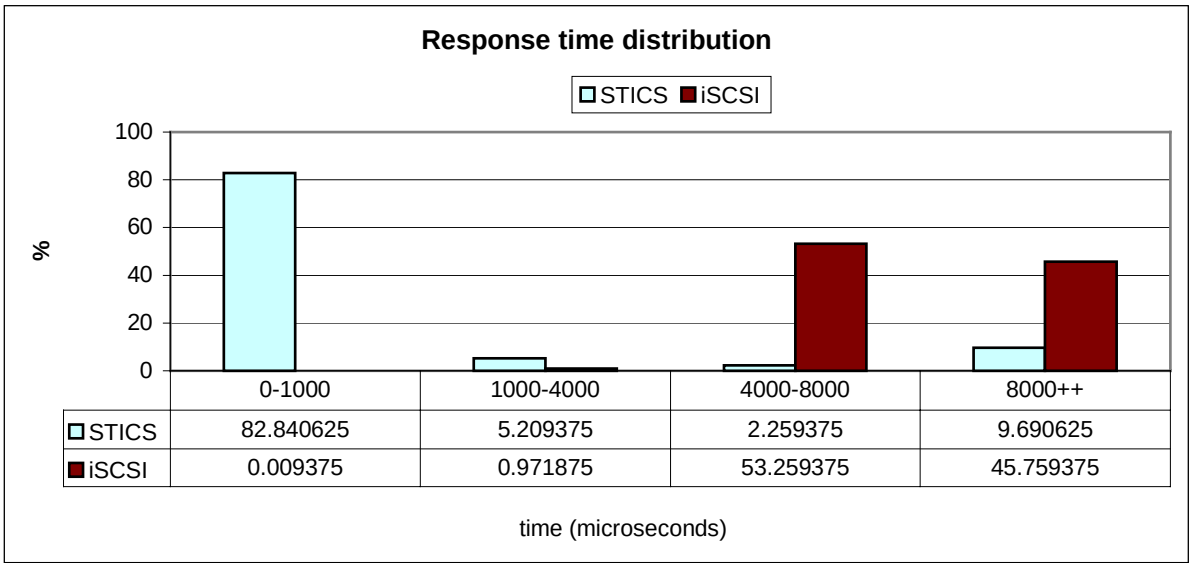
Our next experiment is to measure and compare the response times of STICS and iSCSI under EMC trace. In Figure 8a, we plotted histogram of request numbers against response times, i.e. X-axis represents response time and Y-axis represents the number of storage requests finished within a particular response time. For example, a point (X, Y)=(1000, 25000) means that there are 25,000 requests finished within 1000 microseconds. The lighter (blue) part of the figure is for STICS whereas the darker (red) part for iSCSI. To make it clearer, we also draw a bar graph representing percentage of requests finished within a given time as shown in Figure 8b. It is interesting to note in this figure that STICS does an excellent job in smoothing out the speed disparity between SCSI and IP. With STICS, over 80% of requests are finished within 1000 microseconds and most them are finished within 500 microseconds. For iSCSI with no STICS, about 53% of requests take over 4000 microseconds, about 46% take over 8000 microseconds, and almost no request finishes within 1000 microseconds. These measured data are very significant and represent dramatic performance advantages of STICS.

While STICS improves the iSCSI performance by an order of magnitude for 90% of storage requests, the average speedup of STICS is only about 85%. The average response time of STICS is 3652 whereas the average response time of iSCSI is about 6757. Figure 9 shows average response times of groups of 1000 requests each, i.e. we average the response times of every 1000 requests as a group and show the average response times for all groups. In our experiments, we noticed that 5% of requests take over 8000 microseconds for STICS. Some requests even take up to 120,000 microseconds. These

few peak points drag down the average performance of STICS. These excessive large response times can be attributed to the *destaging* process. In our current simulation, we allow the *level 1 destaging* process to continue until the entire RAM buffer is empty before serving a new storage request. It takes a long time to move data in a full RAM to the log disk. We are still working on the optimization of the *destage* algorithm. We believe there is sufficient room to improve the *destaging* process to avoid the few peak response times of STICS.



(a)



(b)

Figure 8: Response time distributions.

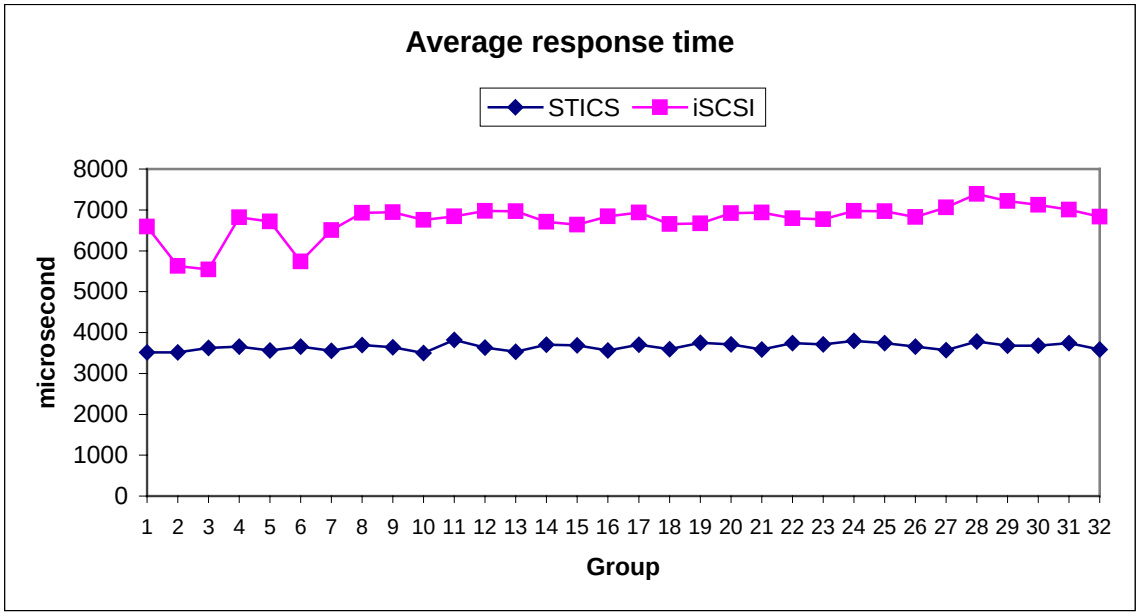


Figure 9: Average response time of STICS and iSCSI.

4. Related Work

Existing research that is most closely related to STICS is Network Attached Storage (NAS) [4,20] and Network-Attached Secure Disk (NASD) [5], a research project at Carnegie Mellon University. Both technologies provide direct network connection for clients to access through network interfaces and file system functionality. NAS-based storage appliances range from terabyte servers to a simple disk with Ethernet plug. These storages are generally simple to manage. NASD provides secure interfaces via cryptographic support and divides NFS-like functionality between a file manager and the disk drives. The file manager is responsible primarily for verifying credentials and establishing access tokens. Clients can access all devices directly and in parallel once approved by the file manager. As discussed in the introduction of this paper, STICS presents a perfect complement to NAS. STICS provides a direct SCSI connection to a server host to allow the server to access at block level a SAN implemented over the Internet. In addition to being a storage component of the SAN, a STICS performs network cache functions for a smooth and efficient SAN implementation over IP network. In a local area network (LAN), the LAN used to interconnect STICS and various storage devices can be the same LAN as the one connecting servers accessing storages, or it can be a separate LAN dedicated for SAN without competing for network bandwidth with the servers.

Another important work related to our research is *Petal* [13, 23], a research project of Compaq's Systems Research Center. Petal uses a collection of NAS-like storage servers interconnected using custom LAN to form a unified virtual disk space to clients at block level. A virtual disk is globally accessible to all Petal clients within the network. Petal was built using a LAN protocol but logically designed a SAN interface. Each storage

server keeps a global state, which makes management in a Petal system especially easy. *iSCSI* (Internet SCSI) [6,9,21] emerged very recently provides an ideal alternative to Petal's custom LAN-based SAN protocol. Taking advantage of existing Internet protocols and media, it is a nature way for storage to make use of TCP/IP as demonstrated by earlier research work of Meter et al of USC, *VISA* [15] to transfer SCSI commands and data using IP protocol.

iSCSI protocol is a mapping of the SCSI remote procedure invocation model over the TCP/IP protocol [21]. The connection between the initiator and the target is established when a session starts. The SCSI commands and data are transferred between the initiator and target that need to be synchronized remotely. STICS architecture attempts to localize some of SCSI protocol traffic by accepting SCSI commands and data from the host. Filtered data block is sent to the storage target using Internet. This SCSI-in-Block-out mechanism provides an immediate and transparent solution both to the host and the storage eliminating some unnecessary remote synchronization. Furthermore, STICS provides a nonvolatile cache exclusively for SCSI commands and data that are supposed to be transferred through the network. This cache will reduce latency from the host point of view as well as avoid many unnecessary data transfer over the network, because many data are frequently overwritten.

5. Conclusions and Future Work

In this paper, we have introduced a new concept "*SCSI-To-IP cache storage*" (STICS) to bridge the disparities between SCSI and IP in order to facilitate implementation of SAN over the Internet. STICS adds a new dimension to networked storage architectures allowing any server host to efficiently access a SAN on Internet through a standard SCSI interface. Using a nonvolatile "*cache storage*", STICS smoothes out the storage data traffic between SCSI and IP very much like the way "*cache memory*" smoothes out CPU-memory traffic. We have carried out a partial implementation and simulation of STICS under the Linux operating system. While the caching algorithms and file system operations inside a STICS are simulated using simulators, SCSI protocols, data transfers, and iSCSI protocols are actually implemented using standard SCSI HBA, Ethernet controller cards, and an Ethernet switch. We measured the performance of STICS as compared to a typical iSCSI implementation using a popular benchmark (PostMark) and a real world I/O workload (EMC's trace). PostMark results have shown that STICS outperforms iSCSI by 53%-78% in terms of average system throughput. Numerical results under EMC's trace show an order of magnitude performance gain for 90% of storage requests in terms of response time. Furthermore, STICS is a plug-and-play building block for storage networks.

We are currently in the process of completely building the STICS box using the Linux operating system. Besides performance and reliability, manageability, adaptivity, and scalability are under consideration [26].

Acknowledgements

This research is sponsored in part by National Science Foundation under Grants CCR-0073377 and MIP9714370. We would like to thank EMC Corporation for providing trace files to us. We would like to thank Jian Li for discussing about the SCSI target mode and his invaluable assistance in our experiments. We would also like to thank Yinan Liu for his suggestions on graph editing. Finally, we would like to thank department system administrator Tim Toolan for graciously allowing us to borrow the SCSI external cables and a high-speed switch.

References

[1] A. Acharya, M. Uysal, and J. Saltz, "Active Disks: Programming Model, Algorithms and Evaluation," *Proceedings of the 8th international conference on Architectural support for programming languages and operating systems (ASPLOS'98)*, October 2 - 7, 1998, San Jose, CA, pp.81-91.

[2] A. Brown, D. Oppenheimer, K. Keeton, R. Thomas, J. Kubiatowicz, and D.A. Patterson, "ISTORE: Introspective Storage for Data-Intensive Network Services," *Proceedings of the 7th Workshop on Hot Topics in Operating Systems (HotOS-VII),* Rio Rico, Arizona, March 1999.

[3] G. Ganger, M. McKusick, C. Soules, and Y. Patt, "Soft updates: a solution to the metadata update problem in file systems," *ACM Transactions on Computer Systems*, Vol. 18, No. 2, 2000, pp.127-153.

[4] G. Gibson, R. Meter, "Network Attached Storage Architecture," *Communications of the ACM*, Vol. 43, No 11, November 2000, pp.37-45.

[5] G. Gibson, D. Nagle, W. Courtright II, N. Lanza, P. Mazaitis, M. Unangst, and J. Zelenka, "NASD Scalable Storage Systems," *USENIX99, Extreme Linux Workshop*, Monterey, CA, June 1999.

[6] R. Hernandez, C. Kion, and G. Cole, "IP Storage Networking: IBM NAS and iSCSI Solutions," *Redbooks Publications (IBM), SG24-6240-00*, June 2001.

[7] Y. Hu and Q. Yang, "DCD-disk caching disk: A New Approach for Boosting I/O Performance," *23rd Annual Intl. Symposium on Computer Architecture*, Philadelphia PA, May, 1996, pp.169-178.

[8] Y. Hu, Q. Yang, and T. Nightingale, "RAPID-Cache --- A Reliable and Inexpensive Write Cache for Disk I/O Systems", *In the 5th International Symposium on High Performance Computer Architecture (HPCA-5)*, Orlando, Florida, Jan. 1999.

[9] Intel iSCSI project, URL: *http://sourceforge.net/projects/intel-iscsi*, April 2001.

[10] J. Katcher, "PostMark: A New File System Benchmark," Technical Report TR3022, Network Appliance, *URL: http://www.netapp.com/tech_library/3022.html*.

[11] R. Khattar, M. Murphy, G. Tarella and K. Nystrom, "Introduction to Storage Area Network," *Redbooks Publications (IBM), SG24-5470-00*, September 1999.

[12] J. Kubiatowicz, et al. "OceanStore: An Architecture for Global-Scale Persistent Storage," *Proceedings of the international conference on Architectural support for programming languages and operating systems (ASPLOS'2000),* December 2000.

[13] E. Lee and C. Thekkath, "Petal: Distributed Virtual Disks," *Proceedings of the international conference on Architectural support for programming languages and operating systems (ASPLOS 1996)*, pp.84-92.
[14] H. Lim, V. Kapoor, C. Wighe, and D. Du, "Active Disk File System: A Distributed, Scalable File System," *Proceedings of the 18th IEEE Symposium on Mass Storage Systems and Technologies*, San Diego, April 2001, pp. 101-115.
[15] R. Meter, G. Finn, S. Hotz, "VISA: Netstation's Virtual Internet SCSI Adapter," *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, California, October 3-7, 1998, pp.71-80.
[16] D. Nagle, G. Ganger, J. Butler, G. Goodson, and C. Sabol, "Network Support for Network-Attached Storage," *Hot Interconnects'1999*, August 1999.
[17] Nishan System white paper, "Storage over IP (SoIP) Framework – The Next Generation SAN," *URL: http://www.nishansystems.com/techlib/techlib_papers.html*, June 2001.
[18] A. Palekar and R. Russell, "Design and Implementation of a SCSI Target for Storage Area Networks," Technical Report TR 01-01, University of New Hampshire, *URL: ftp://ftp.iol.unh.edu/pub/iscsi/tr0101.pdf*, May 2001.
[19] B. Phillips, "Have Storage Area Networks Come of Age?" *IEEE Computer*, Vol. 31, No. 7, 1998.
[20] E. Riedel, G. Faloutsos, G. Gibson and D. Nagle, "Active Disks for Large-Scale Data Processing," *IEEE Computer*, Vol. 34, No. 6, June 2001.
[21] J. Satran, et al. "iSCSI draft standard," *URL: http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-06.txt*, April 2001.
[22] M. Seltzer, K. Bostic, M. McKusick, C. Staelin, "An Implementation of a Log-Structured File System for UNIX," *Winter USENIX Proceedings*, Jan. 1993, pp. 201-220.
[23] C. Thekkath, T. Mann, and E. Lee, "Frangipani: A scalable distributed file system," *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, October 1997, pp. 224-237.
[24] A. Veitch, E. Riedel, S. Towers, and J. Wilkes, "Towards Global Storage Management and Data Placement," *Technical Memo HPL-SSP-2001-1, HP Labs,* March 2001.
[25] Working Draft, "Information Technology: The SCSI Architecture Model-2 (SAM-2)," Revision 14, T10-1157-D, *URL: ftp://ftp.t10.org/t10/drafts/sam2/sam2r14.pdf*, September 2000.
[26] Q. Yang and X. He, "Interface and File System Designs for Implementing STICS — SCSI-To-IP Cache Storage," *One page work-in-progress report submitted to Conference on File And Storage Technologies (FAST'2002).*