Introducing SCSI-to-IP Cache for Storage Area Networks

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

Abstract

Data storage plays an essential role in today's fast-growing data-intensive network services. iSCSI is one of the most recent standards that allow SCSI protocols to be carried out over IP networks. However, the disparities between SCSI and IP 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. Through the efficient caching algorithm and localization of certain unnecessary protocol overheads, STICS significantly improves performance over current iSCSI systems. Furthermore, STICS can be used as a basic plug-and-play building block for data storage over IP. We have implemented software STICS prototype on Linux operating system. Numerical results using popular PostMark benchmark program and EMC's trace have shown dramatic performance gain over the current iSCSI implementation.

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 due to ever-growing demand for networked information services [8]. In general, networked storage architectures have evolved from network-attached storage (NAS) [2, 13], storage area network (SAN) [7], to most recent storage over IP (iSCSI) [6, 14]. 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. The basic premise of a SAN is to replace the "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 protocol: FC-4 protocol. Deploying such a specialized network usually introduces additional cost for implementation, maintenance, and management. iSCSI is the most recently 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 performance issues. Currently, there are basically two existing approaches: one carries out SCSI and IP protocol conversion at a specialized switch and the other encapsulates SCSI protocol in TCP/IP at host bus adapter (HBA) level [14]. Both approaches have severe performance limitations. Converting protocols at a switch places special burden to an already-overloaded switch and creates another specialized networking equipment in a SAN. Such a specialized switch not only is costly as compared to off-the-shelf Ethernet switches but also complicates installation, management, and maintenance. To encapsulate SCSI protocol over IP requires significant amount of overhead traffic for SCSI commands transfers and handshaking over the Internet. In addition, packet transfer latency exists over the network, particularly over long distances. Such latency does not reduce linearly with the increase of network bandwidth.

Protocol disparities and network latencies motivate us to introduce a new storage architecture: SCSI-To-IP Cache Storage, or STICS for short. The purpose of STICS is to bridge the disparities between SCSI and IP so that efficient SAN can be built over the Internet. 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. By localizing part of SCSI protocol and filtering out some unnecessary traffic, STICS can reduce the bandwidth requirement of the Internet to implement SAN.

To quantitatively evaluate the performance potential of STICS in real world network environment, we have implemented the STICS under Linux. We have used PostMark benchmark and EMC's trace to measure system performance. PostMark results show that STICS provides up to 4 times performance improvement over iSCSI implementation in terms of average system throughput. For EMC's trace, our STICS shows up to 6 times as fast as the iSCSI in terms of average response time.

2. Architecture

The idea of STICS is very simple. It is just a cache that bridges the protocol and speed disparities between SCSI and IP. 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 a 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.

2.1. Cache structure of STICS

Each STICS has a read cache consisting of a large DRAM and a write cache consisting of a 2 levels hierarchy with a small NVRAM on top of a log disk. Frequently accessed data reside in the DRAM that is organized as LRU cache for read operations. Write data are first stored in the small NVRAM. Whenever the newly written data in the NVRAM are sufficiently large or whenever the log disk is free, a log of data is written into the log disk sequentially. After the log write, the NVRAM becomes available to absorb additional write data. At the same time, a copy of the log is placed in the DRAM to speed up possible read operations of the data that have just been written to the log disk. Data in the log disk are organized in the format of *segments* similar to that in a Log-structured File System [15]. A segment contains a number of *slots* each of which can hold one data block. Data blocks in a segment are addressed by their *Segment IDs* and *Slot IDs*.

A Hash table is used to locate data in the RAM buffer including DRAM and NVRAM. DRAM and NVRAM can be differentiated through their addresses. A LRU list and a Free List are used to keep tracks of the most recently used data and the free slots respectively. Data blocks stored in the RAM buffer are addressed by their *Logical Block Addresses (LBAs).* The Hash Table contains location information for each of the valid data blocks in the buffer and uses LBAs of incoming requests as search keys. The slot size is set to be the size of a block.

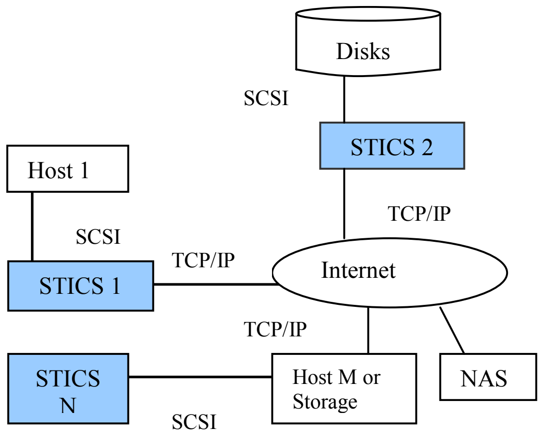


Figure 1: System overview. A STICS connects to the host via SCSI interface and connects to other STICS' or NAS via Internet.

2.2. Basic operations

Write. Write requests may come from one of two sources: the host via SCSI interface and another STICS via the Ethernet interface.

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, and is moved to the NVRAM if it is in DRAM. If no entry is found, a free slot entry in the NVRAM 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 (128 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 and the log is moved from the NVRAM to DRAM.

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, STICS gets a data block with STICS IP and LBA. It then searches the Hash Table by the LBA and IP. The same writing process as above is then performed.

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 IP, and destination IP address 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 IP and LBA from the packet. It then searches the Hash Table by the LBA and the source IP to determine the location of the data and sends the data back to the source STICS through the network.

Destages. The operation of moving data from a higher-level storage device to a lower level storage device is defined as *destage* operation [16]. There are two levels of destage operations in STICS: destaging data from the NVRAM buffer to the log disk (*Level 1 destage*) and destaging data from log disk to a storage device (*Level 2 destage*). *Level 1 destage* activates whenever the log disk is idle and there are data to be destaged in the NVRAM. *Level 2 destage* activates whenever one of the following events occurs: 1) the STICS detects a CPU idle period; 2) the size of data in the log disk exceeds a threshold value. *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 NVRAM 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. For extreme burst writes, where the log disk is full, *Level 1 destage* forces subsequent writes to the addressed network storage to bypass the log disk to avoid cache overflow [16].

As for *Level 1 destage*, the data in the NVRAM buffer are written to the log disk sequentially in large size (64KB). At the same time, the data are moved from NVRAM to DRAM. 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 "first-write-first-destage" algorithm according to the LRU List. Each time 64KB data are read from the consecutive blocks of the log disk and written to the addressed network storage. The LRU list and free list are updated subsequently.

2.3 Cache Coherence

There are three ways to configure a distributed storage system using STICS, placing STICS near the host, target storage, or both. If we place a STICS near the host, the corresponding STICS building block is a private cache. If we place a STICS near the storage, we have a shared cache system. There are tradeoffs between shared cache and private cache configurations. From the point of view of cache efficiency, we would like to place cache as close to a host as possible to minimize latency. Such a private cache system allows multiple copies of a shared storage data to reside in different caches giving rise to the well-known cache coherence problem. Shared caches, on the other hand, do not have such cache coherence problem because each cache is associated with target storage. However, each request has to go through the network to obtain data at the target storage side. We have considered both private and shared cache configurations. Shared cache configuration is relatively simple. For private cache configuration, a coherence protocol is necessary. One possible way to implement a cache coherence protocol in private cache system is using the local consistency (LC) model [1], which helps to minimize meta-data network traffic pertaining to coherence protocol. The details of the cache coherence protocol are out of scope of this paper. Interested readers are referred to [3].

2.4 Implementation

There are several ways to implement STICS. A software STICS is a device driver or kernel module that controls and coordinates SCSI host bus adaptor (HBA) and network interface card (NIC). It uses a part of host's system RAM and part of disk to form the cache. STICS can also be implemented at HBA controller level as a STICS card. Such a card has sufficient intelligence with RAM, IDE or SCSI interface, and Ethernet interface. The IDE or SCSI interface is used to connect to a log disk for caching. Finally, STICS can be implemented as a complete cache box with built-in controller, log disks, and local storage.

Currently we have implemented a software prototype of STICS on Linux kernel 2.4.2, and it is implemented as kernel module which can be loaded and unloaded dynamically. Our implementation uses a part of system RAM and an additional hard disk for caching function.

Table 1: Disk parameters

Disk Model	Manufacture	Interface	Capacity	Data buffer	RPM	Latency (ms)	Transfer rate (MB/s)	Seek time (ms)
O7N3200	IBM	Ultra SCSI	36.7G	N/A	10000	3.0	29.8	4.9
AS010a1	Maxtor	Ultra ATA/100	10.2G	2MB	7200	4.17	16.6	8.5

There is no local storage and all I/O operations are remote operations going through the network.

3. Performance Evaluations

3.1. Methodology

For the purpose of performance evaluation, we have implemented STICS prototype and deployed a software iSCSI. For a fair performance comparison, both iSCSI and STICS have exactly the same CPU and RAM size. This RAM includes read cache and write buffer used in STICS. All I/O operations in both iSCSI and STICS are forced to be remote operations to target disks through a switch.

Table 2: Machines configurations

	CPU	RAM	IDE disk	SCSI disk
Trout	PII-450	128MB	2 AS010a1	N/A
Cod	PII-400	128MB	AS010a1	N/A
Squid	PII-400	128MB	2 AS010a1	O7N3200

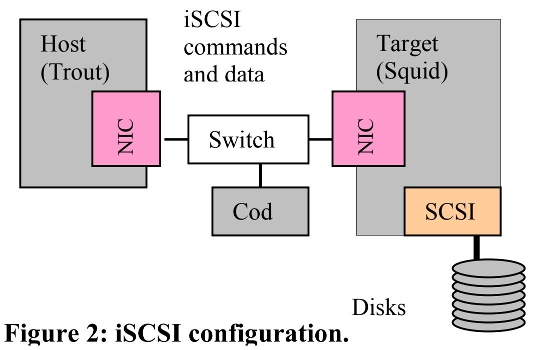


Figure 2: iSCSI configuration.

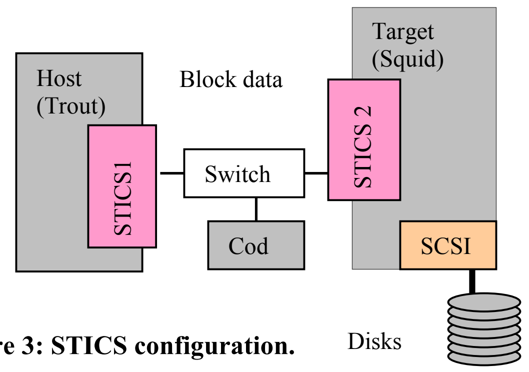


Figure 3: STICS configuration.

Our experimental settings are shown in Figures 2 and 3. Three PCs are involved in our experiments, namely *Trout*, *Cod* and *Squid*. *Trout* serves as the host and *Squid* as the storage target. *Cod* serves as a switch console to monitor the network traffic. For STICS experiment, a software STICS is loaded as kernel module. All these machines are interconnected through an 8-port Gigabit switch (Intel NetStructure 470T) to form an isolated LAN. Each machine is running Linux kernel 2.4.2 with a Netgear GA622T Gigabit network interface card (NIC) and an Adaptec 39160 high performance SCSI adaptor. The network cards and switch can be tuned to Gigabit and 100Mbit dynamically. The configurations of these machines are described in Table 2 and the characteristics of individual disks are summarized in Table 1.

For iSCSI implementation, we compiled and run the Linux iSCSI developed by Intel Corporation [6]. The iSCSI is compiled under Linux kernel 2.4.2 and configured as shown in Figure 2.

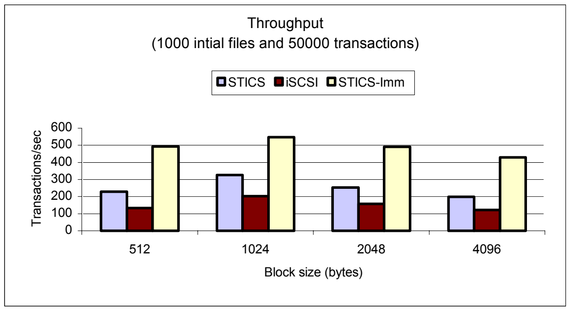
Our STICS is running on Linux kernel 2.4.2 with target mode support and is loaded as a kernel module as shown in Figure 3. Four MB of the system RAM is used to simulate STICS NVRAM buffer, another 16MB of the system RAM is used as the DRAM read cache in our STICS, and the log disk is a standalone hard drive. When requests come from the host, the STICS first processes the requests locally. For write requests, the STICS writes the data to its write buffer. Whenever the log disk is idle, 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 and moves the data to DRAM cache. When data in the log disk exceeds a threshold or the system is idle, the data in log disk will be destaged to the remote target storage through the network. The hash table and LRU list 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 buffer, log disk, or remote disks via network.

3.2. Benchmark and workload characteristics

The benchmark we used to measure system throughput is PostMark 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 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 system installed at a telecommunication consumer site. The trace file contains 230370 requests, with a fixed request size of 4 blocks. The whole dataset size is 900M bytes. The trace is write-dominated with a write ratio of 89%. The average request rate is about 333 requests/second. 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 STICS and iSCSI system to measure performance.

3.3. Measured results and discussions



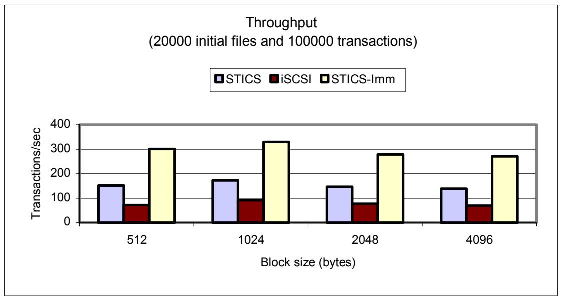


Figure 4: PostMark measurements (100Mbit network).

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. 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 436MB (151.05MB read and 285.08MB write) and 740MB (303.46 MB read and 436.18MB write) respectively. They are much larger than host system RAM (128MB). We left all other PostMark parameters at their default settings. The network is configured as a 100Mbit network.

In Figure 4, we plotted two separate bar graphs corresponding to the small file pool case and the large one, respectively. Each group of bars represents the system throughputs of STICS with report after complete (STICS), iSCSI (iSCSI) and STICS with immediate report (STICS-Imm: host is acknowledged immediately after a write data is in RAM) 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 gains of STICS with report after complete over iSCSI range from 60% to 110%. STICS with immediate report outperforms iSCSI by a factor of 2.69 to 4.18.

Table 3: packet distribution

	# Of packets with different sizes					
	<64 Bytes	65-127	128-255	256-511	512-1023	>1024
iSCSI	7	1,937,724	91	60	27	1,415,912
STICS	4	431,216	16	30	7	607,827

Table 4: Network traffic

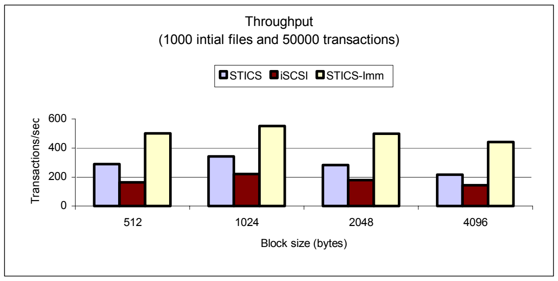
	iSCSI	STICS
Total Packets	3,353,821	839,100
Full/Partial Packet Ratio	0.73	1.41
Bytes Transferred	1,914,566,504	980,963,821
Average Bytes/Packet	571	944

To understand why STICS provides such impressive performance gains over the iSCSI, we monitored the network activities at the Ethernet Switch through the console machine *cod* for both STICS and iSCSI implementations. While both implementations write all data from the host to the remote storage, STICS transfers dramatically less packets over the network than iSCSI does. Tables 3 and 4 show the measured network activities for both STICS and iSCSI. Based on our analysis of the numerical results, we believe that the performance gain of STICS over iSCSI can be attributed to the following facts. First, the log disk along with the RAM buffer forms a large cache for the host and absorbs small writes quickly, which reduces the network traffic because many data are overwritten in the local log disk. As shown in Table 4, the number of total bytes transferred over the network is reduced from 1,914,566,504 to 980,963,821 although the total data stored in the target storage is the same. Secondly, STICS eliminates many remote handshaking caused by iSCSI, which in turn reduce the network traffic. We noticed in Table 3 that the small size packets which are mainly used to transfer iSCSI handshaking messages are dramatically reduced from 1,937,724 to 431,216. Thirdly, by combining small writes into large ones, STICS increases the network bandwidth utilization. If we define full packet as the packet with size larger than 1024 bytes of payload data, and other packets are defined as partial packets. As shown in Table 4, STICS improves the ratio of full packets to partial packets from 0.73 to 1.41, and average bytes per packet is increased from 571 in iSCSI to 944 in STICS.

Above results are measured under 100Mbit network, when we configured the switch and network cards as Gigabit network, we observed similar results as shown in figure 5. The performance gains of STICS with report after complete over iSCSI range from 51% to 80%. STICS with immediate report outperforms iSCSI by a factor of 2.49 to 3.07. The reason is as follows. When the network is improved from 100Mbit to 1 Gigabit, the network latency is not decreased linearly. In our test, we found the average latencies for 100Mbit and 1Gigabit network are 128.99 and 106.78 microseconds. The network performance is improved less than 20% in terms of latency from 100Mbit to Gigabit network.

Response times. Our next experiment is to measure and compare the response times of STICS and iSCSI under EMC trace. The network is configured as a Gigabit network. Response times of all individual I/O requests are plotted in Figure 6 for STICS with immediate report (Figure 6a), STICS with report after complete (Figure 6b) and iSCSI (Figure 6c). Each dot in a figure represents the response time of an individual I/O request. It can be seen from the figures that overall response times of STICS are much smaller than that of iSCSI. In Figure 6b, we noticed 4 requests take up to 300ms. 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 implementation, we allow the *level 2 destaging* process to continue until the entire log segment is empty before serving a new storage request. It takes a long time to move data in a full log segment to the remote data 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.

We also plotted histogram of request numbers against response times in Figure 7. In this figure, X-axis represents response time and Y-axis represents the number of storage requests finished at a particular response time. For example, a point (X, Y)=(10, 2500) means that there are 2,500 requests finished within 10 ms. As shown in Figure 7a, for the STICS with immediate report, most requests are finished within 2 ms, because STICS signals the complete of requests when the data are transferred to NVRAM buffer for write requests. The average response time is 2.7 ms. For the STICS with report after complete as shown in Figure 7b, the response times of the majority of requests fall within the range of 2-5 ms. The rest of requests take longer time to finish but very few of them take longer than 40ms. The average response time is 5.71 ms.



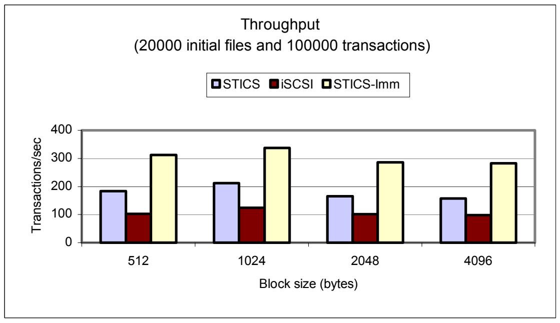
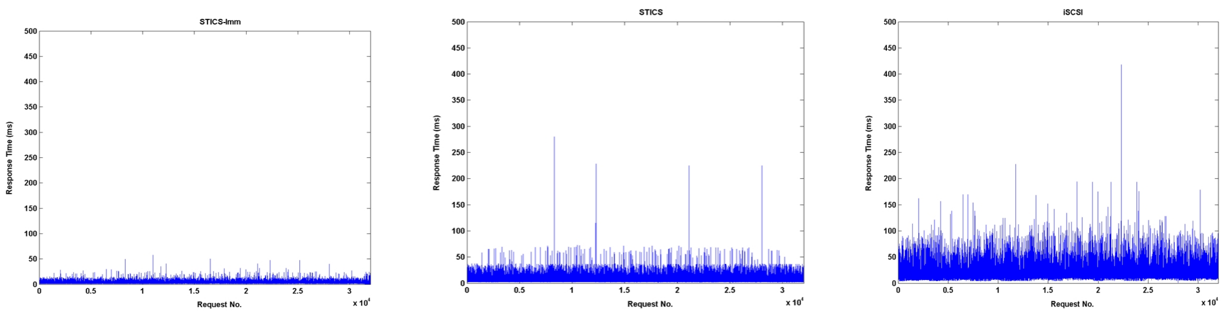


Figure 5: PostMark measurements (Gigabit network).

The iSCSI, on the other hand, has obvious larger response time. The response times of the majority of requests fall within the range of 6-28 milliseconds as shown in Figure 7c. No requests are finished within 5 milliseconds. Some of them even take up to 400ms. The average response time is 16.73ms, which is 2.9 times as much as STICS with report after complete and 6.2 times as much as STICS with immediate report. Such a long responses time can be mainly attributed to the excessive network traffic of iSCSI.

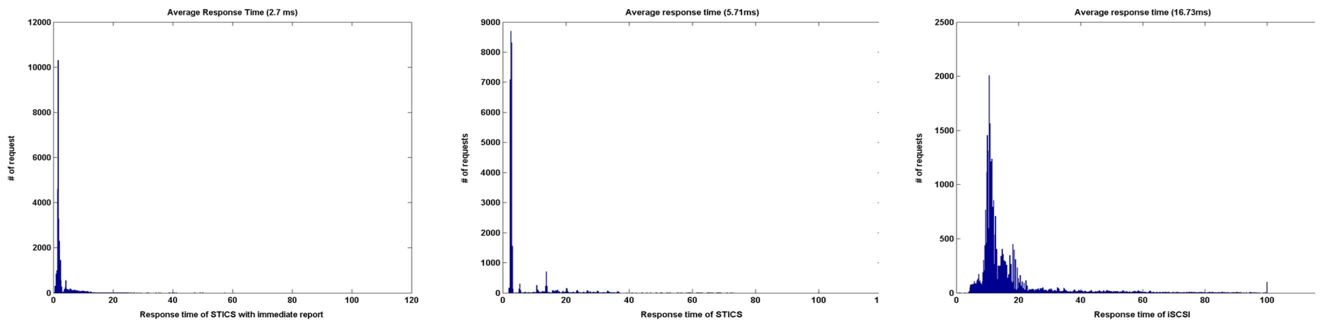
3.4. Costs Analysis

As shown in the last subsection, STICS presents significant performance gains over the standard iSCSI implementation. One obvious question to ask is whether such performance improvement comes at extra hardware cost. To answer this question, we have carried out cost analysis as compared to iSCSI. In our experimental implementations, all hardware components such as CPU, RAM, cabling and switches are exactly the same for both iSCSI and STICS except for an additional disk in STICS for caching. With rapid dropping of disk prices, such an additional disk is easily justifiable. Typical cost of a 10 GB disk is well under $100 while a typical SAN costs over tens of thousands dollars, implying a very small



a) STICS with immediate report b) STICS with report after complete. c) iSCSI.

Figure 6: Response times for EMC-tel trace. Each dot shows the response time of an individual I/O request.



a) STICS with immediate report b) STICS with report after complete. c) iSCSI.

Figure 7: Histograms of I/O response times for trace EMC-tel.

fraction of additional cost of STICS. Table 5 lists the practical cost of building a minimum SAN configuration with 6 servers and 200 GB using iSCSI and STICS, respectively (all the list prices are as of January 2001). As shown in this table, the cost difference between the two is well under 7%. Considering software cost (***$22,059***) and maintenance cost (***$8,676***) for the same SAN system [10], the cost difference between the two is much less than 3%. We believe trading 3% of additional cost for 6 folds performance gain is certainly worthwhile.

Table 5: Hardware costs comparison

	iSCSI			STICS		
	Qty	Cost	Total	Qty	Cost	Total
HBA	12	$339	$4,068	12	$339	$4,068
Switch	1	$1,229	$1,229	1	$1,229	$1,229
GB NIC	12	$319	$3,828	12	$319	$3,828
OS HDD	12	$85	$1,020	12	$85	$1,020
SCSI HDD	6	$799	$4,794	6	$799	$4,794
Log Disks				***12***	***$85***	***$1,020***
Total	***$14,939***			***$15,959***		

We have also considered the cost of implementing iSCSI and STICS in hardware. For the same SAN configuration with 6 servers, iSCSI would need an iSCSI to SCSI converter costing $5,083 [10] or iSCSI cards. The additional hardware for each STICS would include an I/O processor with 4-MB NVRAM. We can conservatively estimate the total cost in addition to Table 5 for 12 STICS to be under $5,000.

4. Related Work

Existing research that is most closely related to STICS is network attached storage (NAS)[2]. The NAS technology provides direct network connection for hosts to access through network interfaces at file system level. 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.

Another important work related to our research is *Petal* [9], a research project of Compaq's Systems Research Center. Petal uses a collection of NAS-like storage servers interconnected using specially customized LAN to form a unified virtual disk space to clients at block level. *iSCSI* (Internet SCSI) [6,14] emerged very recently provides an ideal alternative to Petal's customized 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* [12] 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 [14]. 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 reduces latency from the host point of view as well as avoids many unnecessary data transfer over the network, because many data are frequently overwritten.

The idea of using a disk-based log to improve system performance or to improve the reliability of RAM has been used in both file system and database systems for a long time. For example, the Log-structured File System (LFS [15]), Disk Caching Disk (DCD [4]), and other similar systems all use disk-based data/metadata logging to improve file system performance and speed-up crash recovery. Several RAID systems have implemented the LFS algorithm at the RAID controller level [5,11,17]. LFS collects writes in a RAM buffer to form large logs and writes large logs to data disks. While many implementation techniques are borrowed from existing work, the novelty of STICS is the new concept of caching between SCSI and IP.

5. Conclusions

In this paper, we have introduced a new concept 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 implemented a prototype STICS under Linux operating system. 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 up to 4 times in terms of average system throughput. Numerical results under EMC's trace show a factor of 3 to 6 performance gain in terms of average response time. Furthermore, STICS is a plug-and-play building block for storage networks.

Acknowledgements

This research is sponsored in part by National Science Foundation under grants MIP-9714370 and CCR-0073377. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The authors would like to thank the anonymous reviewers for their many helpful comments and suggestions. We would like to thank EMC Corporation for providing trace files to us. We would like to thank Jian Li for his assistance in the experiments and Yinan Liu for his suggestions on graph editing.

References

[1] M. Ahamad and R. Kordale, "Scalable Consistency Protocols for Distributed Services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, pp. 888, 1999.

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

[3] X. He, Q. Yang, and M Zhang, "Introducing SCSI-To-IP Cache for Storage Area Networks", *Technical Report*, URL: http://ele.uri.edu/~hexb/publications/STICS-Tech-200112.pdf.

[4] Y. Hu and Q. Yang, "DCD-disk caching disk: A New Approach for Boosting I/O Performance," *ISCA'96*, pp.169-178, 1996.

[5] Y. Hu, Q. Yang, and T. Nightingale, "RAPID-Cache --- A Reliable and Inexpensive Write Cache for Disk I/O Systems," *HPCA5,* Jan. 1999.

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

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

[8] 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),* 2000.

[9] E. Lee and C. Thekkath, "Petal: Distributed Virtual Disks," *Proceedings of the international conference on Architectural support for programming languages and operating systems*, pp.84-92, 1996.

[10] B. Mackin, "A Study of iSCSI Total Cost of Ownership (TCO) vs. Fibre Channel and SCSI," *URL: http://www.adaptec.com,* Oct. 2001.

[11] J. Menon, "A Performance Comparison of RAID-5 and Log-Structured Arrays," *Proc. Of 4th IEEE Int'l Symp. High Performance Distributed Computing*, pp. 167-178, 1995.

[12] 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 (ASPLOS)*, pp.71-80, 1998.

[13] E. Miller, D. Long, W. Freeman, and B. Reed, "Strong Security for Network-Attached Storage," Proc. of the *Conference on Fast and Storage Technologies (FAST'2002)*, 2002.

[14] J. Satran, et al. "iSCSI draft standard," *URL: http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-12.txt*, 2002.

[15] M. Seltzer, K. Bostic, M. McKusick, C. Staelin, "An Implementation of a Log-Structured File System for UNIX," *Winter USENIX Proceedings*, pp. 201-220, Jan. 1993.

[16] A. Varma and Q. Jacobson, "Destage algorithms for disk arrays with non-volatile caches," *Proceedings of the 22nd annual international symposium on Computer architecture (ISCA'95)*, pp. 83-95, 1995.

[17] J. Wilkes, R. Golding, C. Staelin, T. Sullivan, "The HP AutoRAID Hierarchical Storage System," *Proc. Of the Fifteenth ACM Symposium on Operating System Principles*, pp. 96-108, 1995.

[18] Y. Zhou, J.F. Philbin, and K. Li, "Multi-Queue Replacement Algorithm for Second Level Buffer Caches," *USENIX Annual Technical Conference,* 2001.