STICS: SCSI-To-IP Cache for Storage Area Networks

Xubin He[1], *Member, IEEE*
Electrical and Computer Engineering
Tennessee Technological University
Cookeville, TN 38505
hexb@tntech.edu

Ming Zhang, and Qing Yang, *Senior Member, IEEE*
Electrical and Computer Engineering
University of Rhode Island
Kingston, RI 02881
{mingz, 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 storage. Given the mature Internet infrastructure, the overwhelming preference among the IT community recently is using IP for storage networking because of economy and convenience. iSCSI is one of the most recent standards that allow SCSI protocols to be carried out over IP networks. However, there are many disparities between SCSI and IP in terms of protocols, speeds, bandwidths, data unit sizes, 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 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 the caching algorithm, protocol conversion, and self-management functions. Through the efficient caching algorithm and localization of certain unnecessary protocol overheads, STICS can significantly improve performance, reliability, 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 an efficient SAN over IP. We have implemented software STICS prototype on Linux operating system. Numerical results using popular benchmarks such as vxbench, IOzone, PostMark, and EMC's trace have shown a dramatic performance gain over the current iSCSI implementation.

Index Terms — Cache, networked storage, NAS, SAN, iSCSI, performance evaluation.

[1] Correspondence author, Tel: 931-3723462, Fax: 931-3723436

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 [13]. Online data storage doubles every 9 months [7] due to an ever-growing demand for networked information services [8, 25, 51]. In general, networked storage architectures have evolved from network-attached storage (NAS) [11, 17, 35, 37], storage area network (SAN) [23, 39, 42], to most recent storage over IP (IP SAN) [17,44]. 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 (FC-4 protocol) network. 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 [4,20,30,45,49] is the most recently emerging technology with the goal of implementing the IP SAN.

Compared to FC-4, implementing SAN over IP (IP SAN) has several advantages [34,52]:

- IP SAN can run over standard off-the-shelf network components, such as switched Ethernet, which reduces the cost. One can extend and expand the switched network easily and quickly while riding the cost/performance improvement trends of Ethernet.

- IP SAN can exploit existing IP-based protocols, and IP SANs using iSCSI can be managed using existing and familiar IP-based tools such as SNMP, while Fibre Channel SANs require specialized management infrastructure.
- A network that incorporates IP SANs need use only a single kind of network infrastructure (Ethernet) for both data and storage traffic, whereas use of Fibre Channel Protocol (FCP) requires a separate kind of infrastructure (Fibre Channel) for storage.

IP SAN brings economy and convenience whereas it also raises performance issues, which is the main downside of current IP SAN as compared to FC-SAN. Currently, there are basically two existing approaches to implement IP SAN using iSCSI: one carries out SCSI and IP protocol conversion at a specialized switch [39] and the other encapsulates SCSI protocol in TCP/IP at the host bus adapter (HBA) level [45]. Both approaches have severe performance limitations. Converting protocols at a switch places an additional special burden on an already-overloaded switch and requires specialized networking equipment in a SAN. Such a specialized switch is not only 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. On a typical iSCSI implementation, we have measured around 58% of TCP/IP packets being less than 127 bytes long, implying an overwhelming quantity of small packets transferring SCSI commands and status (most of them are only one byte). A majority of such small packet traffic over the net is not necessary because of the reliable and connection-oriented services provided by underlying TCP/IP. Our experiments using the PostMark benchmark [22] have shown that efficient caching can reduce the total number of packets transferred over the network from 3,353,821 to 839,100 for same amount of remote storage data, a 75 percent reduction!

In addition to the above-mentioned protocol disparities between SCSI and IP, packet transfer latency exists over the network, particularly over long distances. Such latency does not decrease linearly with an increase in network bandwidth. For example, we measured average network latencies over 100Mbit and 1Gigabit Ethernet switches to be 128.99 and 106.78 microseconds, respectively. These results indicate that even though the bandwidth of Ethernet switches has increased to gigabit or tens of gigabits, network latencies resulting from packet propagation delays are still there.

Protocol disparities and network latencies motivate us to introduce a new storage architecture: SCSI-To-IP Cache Storage (STICS). The purpose of STICS is to bridge the disparities between SCSI and IP so that an efficient SAN can be built over the Internet. A typical STICS block consists of a disk and an intelligent processing unit with an embedded processor and sufficient RAM. It has two standard interfaces: a SCSI interface and a standard Ethernet interface. The disk is used as a nonvolatile cache that caches data coming from possibly two directions: block data from the SCSI interface and network data from the Ethernet interface. In addition to standard SCSI and IP protocols running on the intelligent processing unit, it also implements a special caching algorithm controlling a two level cache hierarchy that writes data very quickly. Besides caching storage data, 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 discard a fraction of the data that would otherwise move across the Internet, reducing the bottleneck problem imposed by limited Internet bandwidth and increasing storage data transfer 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 a 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 a CPU [36], STICS is a "*cache storage*" used to cache networked storage data for a server host.
- It utilizes the techniques in a Log-structured file system [46,53] to quickly write data into magnetic media for caching data coming from both directions. A disk is used in caching, which is extremely important for caching data reliably since once data is written to a nonvolatile storage, it is considered to be safe.
- By localizing part of SCSI protocol and filtering out some unnecessary traffic, STICS can reduce the bandwidth required to implement a SAN.
- Active disks [1,29,43] 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 [7] is standard "brick" for building storage systems, then STICS can be considered as a standard "beam" or "post" that provides interconnect and support for the construction of SANs.

Overall, STICS adds a new dimension to the networked storage architectures. To quantitatively evaluate the performance potential of STICS in a real network environment, we have implemented the STICS under the Linux OS over an Ethernet switch. We have used popular benchmark programs, such as PostMark [22], IOzone [40], vxbench[2], and EMC's trace to measure system performance. Benchmark results show that STICS provides up to 4.3 times performance improvement over iSCSI implementation in terms of average system throughput. For EMC's trace measurement, STICS can be 6 times as fast as iSCSI in terms of average response time.

The paper is organized as follows. Next section summarizes related work. Section 3 presents the STICS architecture, followed by detailed descriptions of the design and implementation in Section 4. Section 5 presents our performance evaluation methodology and numerical results. Finally, Section 6 concludes the paper.

2 RELATED WORK

Existing research that is most closely related to STICS is Network Attached Storage (NAS) [11,12,43]. The NAS technology provides direct network connection for hosts to access through network interfaces. It also provides file system functionality. NAS-based storage appliances range from terabyte servers to a simple disk with an Ethernet plug. The main difference between NAS and SAN is that NAS provides storage at the file system level while SAN provides storage at the block device level. Another difference is that NAS is attached to the same LAN as the one connecting servers accessing storage, while SAN has a dedicated network connecting storage devices without competing for network bandwidth with the servers. STICS provides a direct SCSI connection to a server

[2] An I/O benchmark developed by VERITAS Corp.

host to allow the server to access a SAN implemented over the Internet at the block level. In addition to being a storage component of the SAN, STICS performs network cache functions for a smooth and efficient SAN implementation over IP network.

Another important related effort is *Petal* [27, 48], a research project of Compaq's Systems Research Center. Petal uses a collection of NAS-like storage servers interconnected using a specially customized LAN to form a unified virtual disk space to clients at the block level. *iSCSI* (Internet SCSI) [17,20,45], which 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 natural way for storage to make use of TCP/IP, as demonstrated by the earlier research work of Meter et al, *VISA* [33] 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 [45]. The STICS architecture attempts to localize some of SCSI protocol traffic by accepting SCSI commands and data from the host and filtering data block to be sent to the remote storage target. 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's point of view and avoids many unnecessary data transfers 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 [46,53]), Disk Caching Disk (DCD

[18]), 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 [19,32,54]. 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.

3 ARCHITECTURE

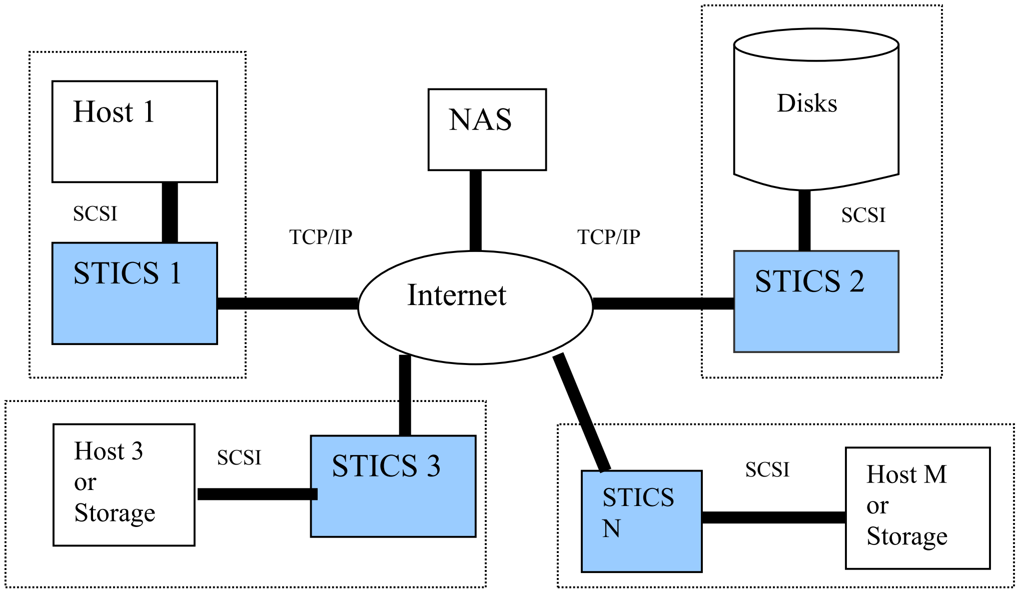


Figure 1: System overview. A STICS connects to the host via SCSI interface and connects to other STICS' or NAS via Internet.

Essentially STICS is 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 using off-the-shelf components. 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 the 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 service, caching service, naming service, and IP protocol service.

Figure 2 shows the basic structure of STICS, which 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 [55]. When a STICS is used to connect to a host, it runs in target mode receiving requests from the host, carrying out the I/O processing possibly through network, and sending back results to the host. In this case, the STICS acts as a directly attached storage device from the host's point of view. 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 storage (NAS).
3) An intelligent processing unit: This processing unit has an embedded processor and an amount of RAM. A specialized Log-structured file system, standard SCSI protocols, and IP protocols run on the processing unit. The RAM consists of a regular DRAM for read caching and a small (1-4MB) NVRAM (nonvolatile RAM) for write caching. The NVRAM is also used to maintain the meta data

such as hash table, LRU list, and the mapping information (STICS_MAP). Alternatively, we can also use Soft Updates [10] technique to keep meta data consistency without using NVRAM.

4) A log disk: The log disk is a sequentially accessed device. It is used to cache write data along with the NVRAM above in the processing unit. The log disk and the NVRAM form a two-level hierarchical cache.

5) Storage device: The regular storage device is an optional component. It can be a disk, a RAID, or *JBOD* (Just-Bunch-Of-Disks). It is used as an extra and backup storage capacity. For example, it can be used as temporary offline storage when the network fails. From the 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, it can be considered as a component of a SAN with an IP address as its ID.

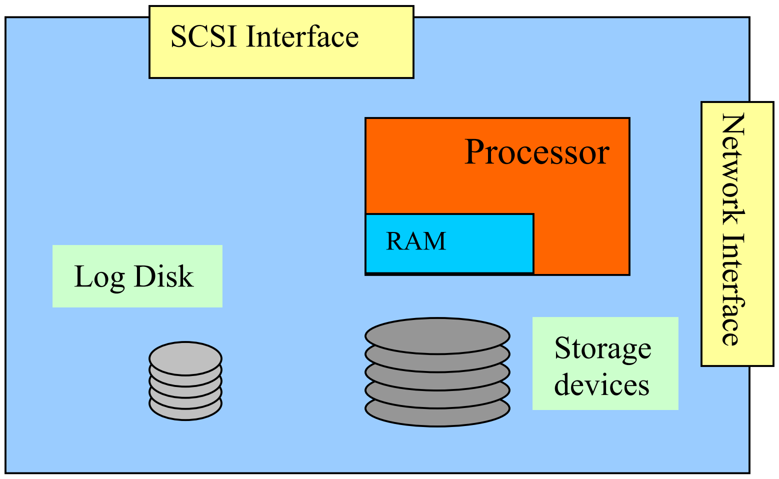


Figure 2: STICS architecture.

4 DESIGN AND IMPLEMENTATION

4.1 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.

4.2 Cache Structure of STICS

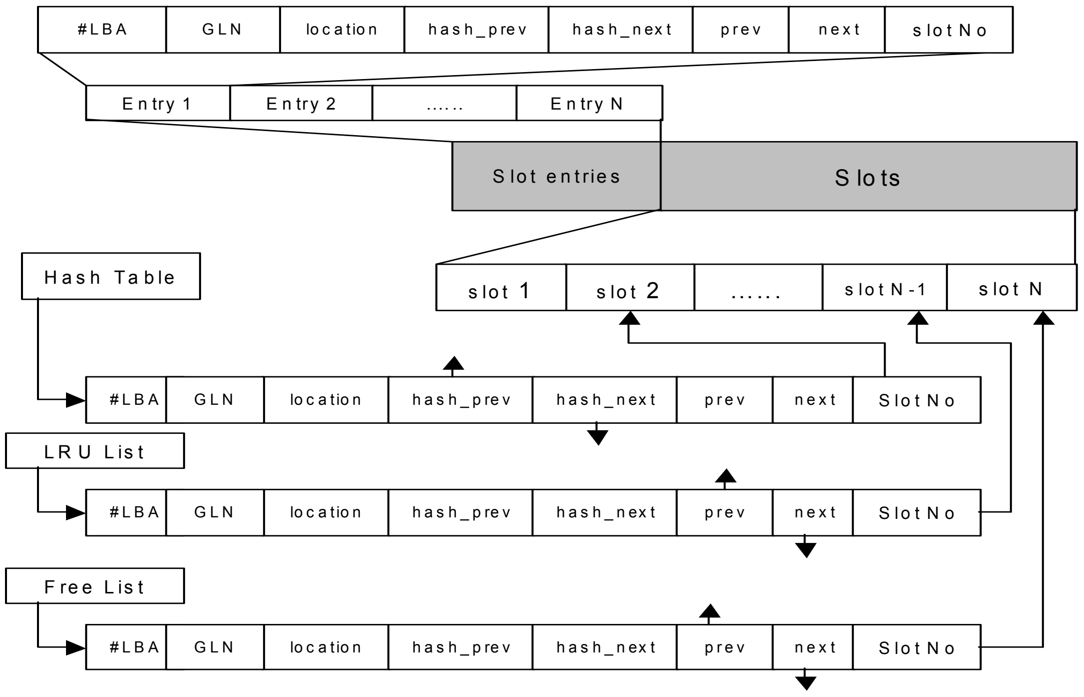


Figure 3: RAM buffer layout. RAM buffer consists of slot entries and slots. The hash table, LRU list and Free list are used to organize the slot entries.

Each STICS has a read cache consisting of a large DRAM and a write cache consisting of a 2-level hierarchy with a small NVRAM on top of a log disk. Frequently accessed data reside in the DRAM that is organized as LRU cache [21] 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 [46]. 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*.

Figure 3 shows the data structure in both DRAM and NVRAM. 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. A slot entry consists of the following fields:

- An *LBA* of a cache line. It serves as the search key of hash table;
- Global Location Number (*GLN*) if the slot contains data from or to other STICS.
- A *location* field containing 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*) used to link the hash table;
- Two pointers (*prev* and *next*) used to link the LRU list and FREE list;
- A *Slot-No* used to describe the in-memory location of the cached data.

4.3 Basic operations

4.3.1 Write

Write requests may come from one of two sources: the host via the 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 turn 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 GLN and LBA. It then searches the Hash Table by the LBA and GLN. The same writing process as above is then performed.

4.3.2 Read

Similar to write operations, read operations may also come either from the host via the 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.

4.3.3 Destages

The operation of moving data from a higher-level storage device to a lower level storage device is defined as *destage* operation [50]. Two levels of destage operations are defined in STICS: destaging data from the NVRAM buffer to the log disk (*Level 1 destage*) and destaging data from the 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. *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 [50].

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. Currently, we are using the LRU replacement algorithm, and other algorithms [56] are in consideration for the future implementation. 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.

4.4 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 [38]. 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 [2,3,6,9,16,24,26,28,41]. 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 [3], which helps to minimize meta-data network traffic pertaining to coherence protocol. A shared-read/exclusive-write lock (token) can be used to implement the necessary synchronization [5, 47]. The details of the cache coherence protocol are out of scope of this paper. Interested readers are referred to [14].

4.5 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 that can be loaded and unloaded dynamically. We simulate NVRAM using part of system RAM. This part of system RAM is reserved when the system is boot up, and it is not accessible to other applications. So this part of RAM is "immune" to application-level software crash and more reliable than regular RAM. The log disk is an additional IDE hard disk for caching function. There is no local storage and all I/O operations are remote operations going through the network.

5 PERFORMANCE EVALUATIONS

5.1 Methodology

For the purpose of performance evaluation, we have implemented a 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.

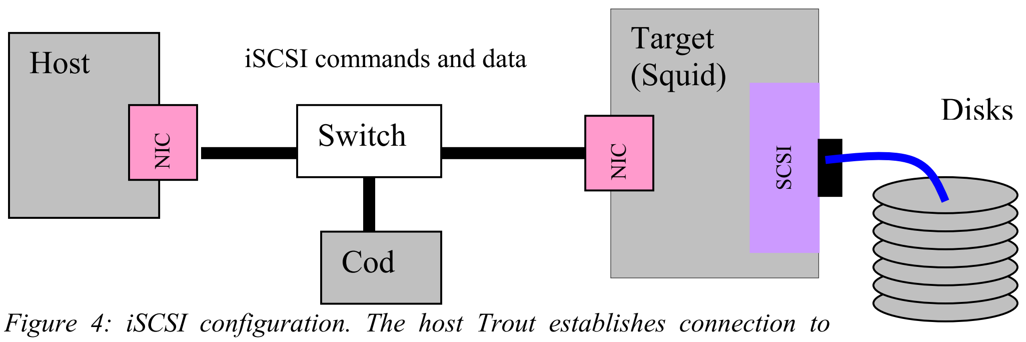


Figure 4: iSCSI configuration. The host Trout establishes connection to target, and the target Squid responds and connects. Then the Squid exports hard drive and Trout sees the disks as local.

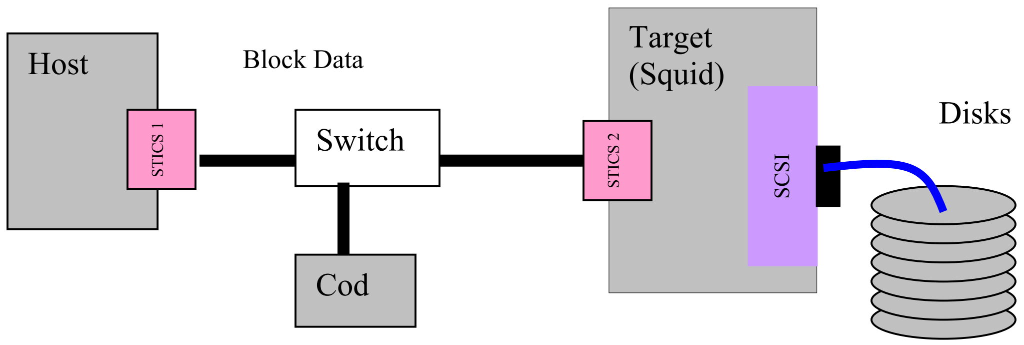


Figure 5: STICS configuration. The STICS cache data from both SCSI and network.

Our experimental settings are shown in Figures 4 and 5. Three PCs and a SUN workstation are involved in our experiments, namely *Trout*, *Cod, Squid, and Clam*. *Trout* and *Clam* 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. To focus on the STICS performance measurement, we disabled STICS2's cache function on the target machine. All cache function is performed by STICS1 on the initiator side. STICS2 only receives and forwards I/O requests in our experiments. 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 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	128MB	2 Maxtor AS010a1	N/A
Cod	PII-400	128MB	Maxtor AS010a1	N/A
Squid	PII-400	128MB	2 Maxtor AS010a1	IBM O7N3200
Clam	Ultra SPARC II 450	256MB	N/A	SUN18G

Table 2: Disk parameters

Disk Model	Interface	Capacity	Data buffer	RPM	Latency (ms)	Transfer rate (MB/s)	Seek time (ms)	Manufacturer
O7N3200	Ultra SCSI	36.7G	N/A	10000	3.0	29.8	4.9	IBM
AS010a1	Ultra ATA/100	10.2G	2MB	7200	4.17	16.6	8.5	Maxtor
SUN18G	Ultra SCSI	18.2G	512KB	10000	3.0	18	8.5	SUN

For iSCSI implementation, we compiled and run the Linux iSCSI developed by Intel Corporation [20]. The iSCSI is compiled under Linux kernel 2.4.2 and configured as shown in Figure 4. 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. iSCSI supports both solicited and unsolicited writes, but we found in current iSCSI implementation, the performance difference between solicited and unsolicited writes are less than 10%, which is very small compared to our STICS performance gain, so in our experiments, we configured iSCSI with solicited writes which is the default setting.

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 5. 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.

In our previous discussions, all STICS are configured in "report after complete" mode. This scheme has a good reliability because a write is guaranteed to be stored in a disk before the CPU is acknowledged. If the 4-MB RAM buffer is nonvolatile, "immediate report" mode can be used, where as soon as the data are transferred to the RAM buffer, STICS sends an acknowledgement of "write complete" to the host.

5.2 Benchmark Programs and Workload Characteristics

It is important to use realistic workloads to drive our STICS for a fair performance evaluation and comparison. For this reason, we chose to use 3 popular benchmark programs and a real world trace.

The benchmarks we used to measure system throughput are PostMark [22] which is a popular file system benchmark developed by Network Appliance, IOzone [40], and vxbench developed by VERITAS. PostMark 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 [22]." 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. IOzone and vxbench generate and measure a variety of file operations based on a big file. Vxbench can only run on Solaris operating system. We used these two benchmarks to measure the I/O throughput in terms of KB/Sec.

In addition to benchmark programs, we also used a real-world trace obtained from EMC Corporation. The trace, referred to as *EMC-tel* trace hereafter, was collected by an

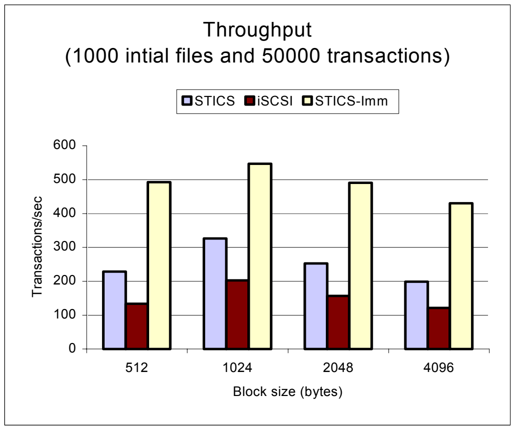
EMC Symmetrix system installed at a telecommunication consumer site. The trace file contains 230370 requests, with a fixed request size of 4 blocks. The whole dataset size is 900M bytes. The trace is write-dominated with a write ratio of 89%. The average request rate is about 333 requests/second. In 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.

5.3 Measured results and discussions

5.3.1 PostMark Results

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 6, 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: light blue bars), iSCSI (iSCSI: dark red bars) and STICS with immediate report (STICS-Imm: light yellow 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 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.



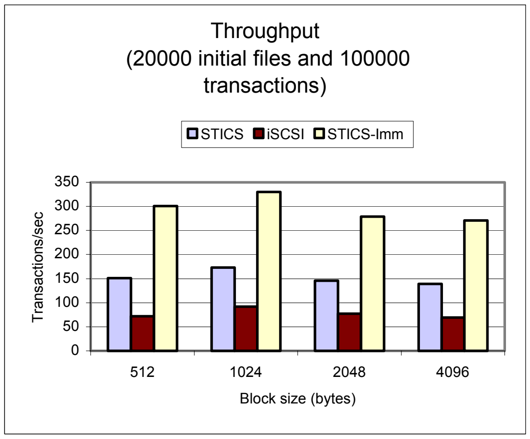


Figure 6: PostMark measurements (100Mbit network).

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 very 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.

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

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

At this point, readers may wonder how many I/O requests are satisfied by the host Linux file system cache and how many are satisfied by STICS cache. To answer this question, we profile the I/O requests by adding several counters as follows to record the number of requests received at each layer.

- *ReqVFSRcv*: This counter is used to record how many I/O requests received at Linux file system layer. This is done by modifying Linux kernel file system *read_write* function.
- *ReqToRaw*: This counter is used to record how many I/O requests are forwarded to low-level block layer. This is done by modifying Linux kernel block I/O *ll_rw_blk*

function. The difference between *ReqToRaw* and *ReqBufferRcv* roughly reflects the number of requests satisfied by Linux file system cache (*ReqFSCache*).

- *ReqSTICSCache*: These counter records the number of requests satisfied by local STICS1 cache.

Table 5 shows the detail breakdown of I/O requests. It's obvious that during the STICS test, 39.58% I/O requests are satisfied by the host file system cache, and additional 29.2% requests are satisfied by STICS cache. We also found that compared to original iSCSI, the reservation of 20MB system RAM for STICS did not dramatically reduce the hit ratio of file system cache (39.58% vs. 39.84%). The reason is that more requests are satisfied by STICS cache, the replacement possibility is reduced in the file system cache.

Table 5: Requests Breakdown

	ReqVFSRcv	*ReqToRAW*	*ReqFSCache (Percentage)*	*ReqSTICSCache (Percentage)*
iSCSI	3104257	1867529	1236728 (39.84%)	N/A
STICS	3069438	1854554	1214884 (39.58%)	896582 (29.21%)

Table 6: Major SCSI Commands Breakdown

	READ	*WRITE*	*INQUIRY*	*READ_CAPACITY*	*TEST_UNIT_READY*
iSCSI	237659	309182	2009	3	53
STICS	94502	200531	1278	3	47

Besides the above requests, we have also measured the number of major SCSI commands [55] received on the target side as shown below in Table 6. We have observed dramatic reduction of data transfer commands (*READ* and *WRITE*). STICS filters out most *READ* commands (from 237,659 to 94,502) because many reads are satisfied by local STICS cache. We have also observed a reduction in the number of *INQUIRY* command (from 2009 to 1278) because of less number of data transfer commands (the host does not have to inquiry the storage device so often as seen in original iSCSI implementation).

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 7. 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.

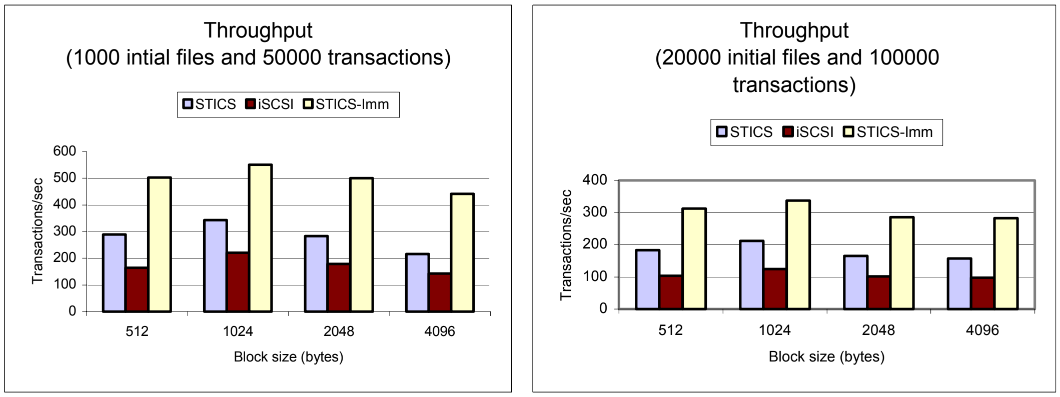


Figure 7: PostMark measurements (Gigabit network).

5.3.2 IOzone Results

Above experiment shows that STICS outperforms iSCSI for workloads consisting of a lot of small files. Our next experiment is to use IOzone to measure the behavior of STICS and iSCSI under a huge file. The network is configured to a 100Mbit network. All I/O operations are set to “Synchronous” mode, which means the host is not acknowledged until data is written to a disk. The data set is 512M bytes in our test.

In Figure 8, we plotted bar graphs for random read and random write operations against request size for STICS and iSCSI, seperately. The request sizes range from 1KB to 32 KB. In all scenarios, STICS outperforms iSCSI. The performance gain of STICS over iSCSI ranges from 51% to a factor of 4.3.

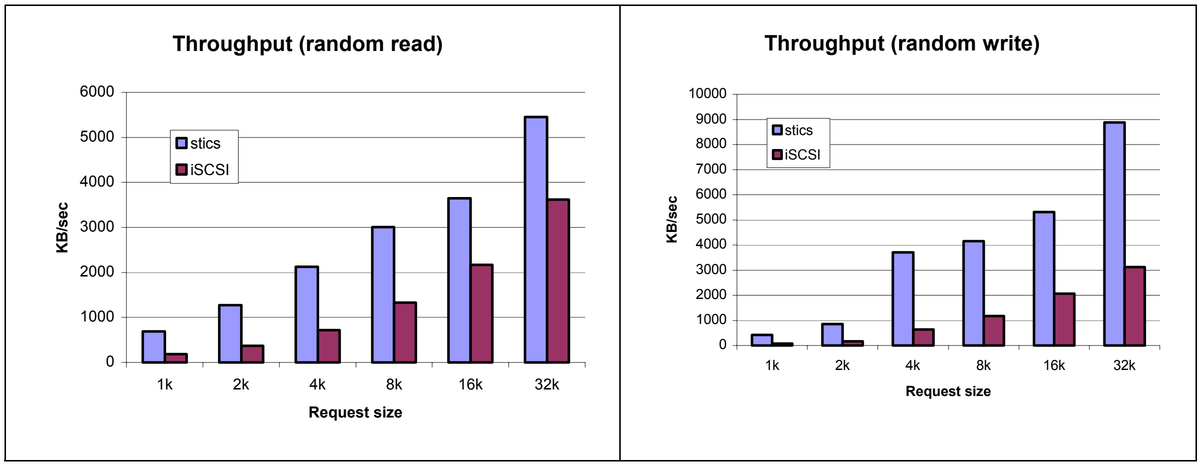


Figure 8: IOzone measurements

5.3.3 vxbench Results

In order to verify that STICS works well under different host platforms, our next experiment is to test STICS under Solaris operating systems. In this test, we use vxbench to measure the performance, which is a popular file system benchmark program developed by VERITAS Corp. The network is configured to a 100Mbit network. The data set is set to 512M bytes.

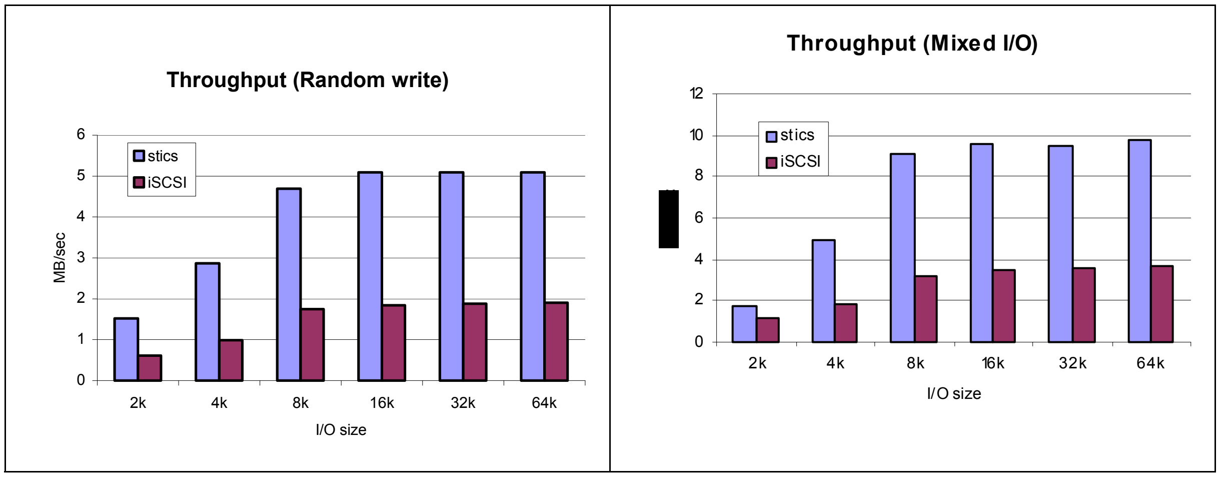


Figure 9: vxbench Results (Throughput)

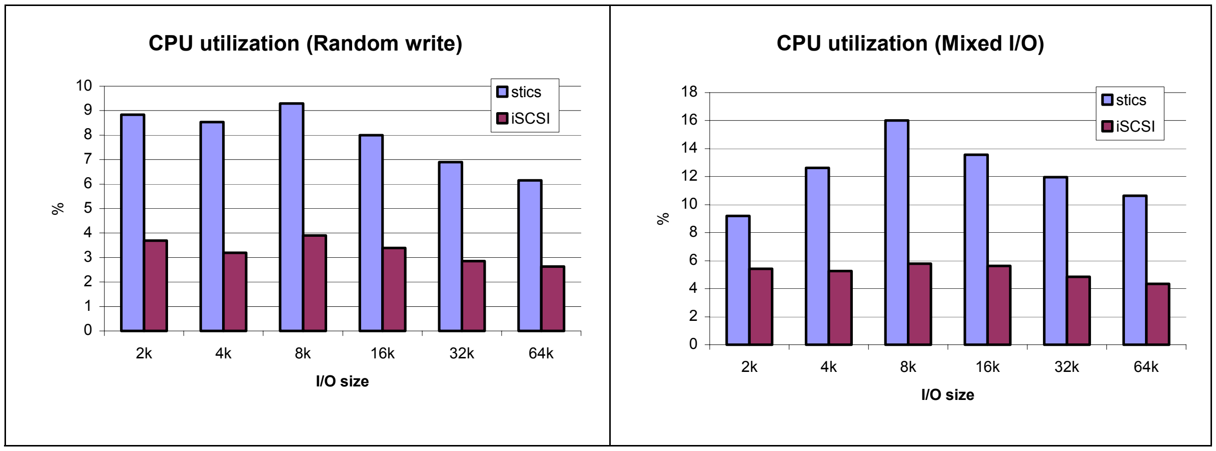


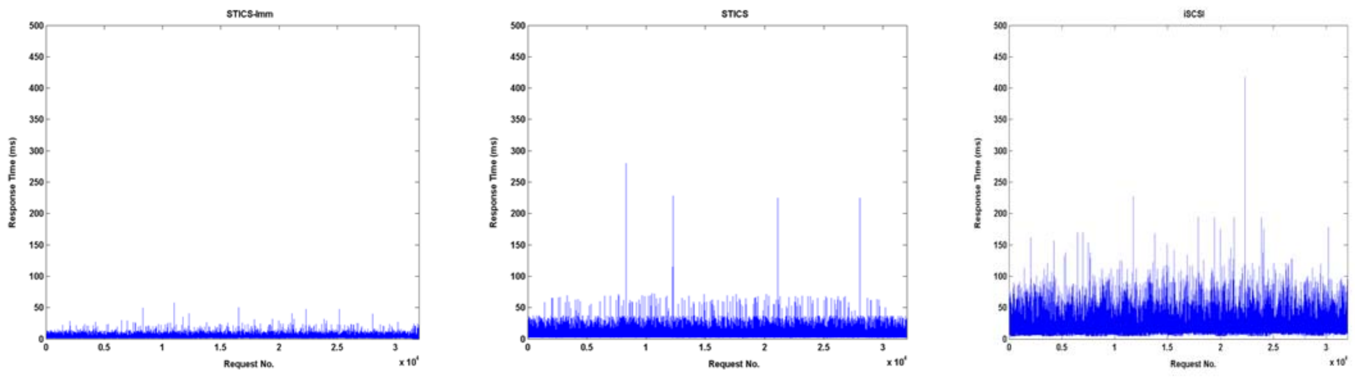
Figure 10: vxbench Results (CPU Utilization)

We measured the system throughput and host CPU utilization for two workload patterns: random write and mixed I/O as shown in Figure 9 and 10, respectively. Under STICS, the host CPU utilization increased dramatically, which improved the system throughput as in Figure 9. The performance gain of STICS over iSCSI ranges from 52% to a factor of 1.9.

5.3.4 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 11 for STICS with immediate report

(Figure 11a), STICS with report after complete (Figure 11b) and iSCSI (Figure 11c). 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 8b, 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.

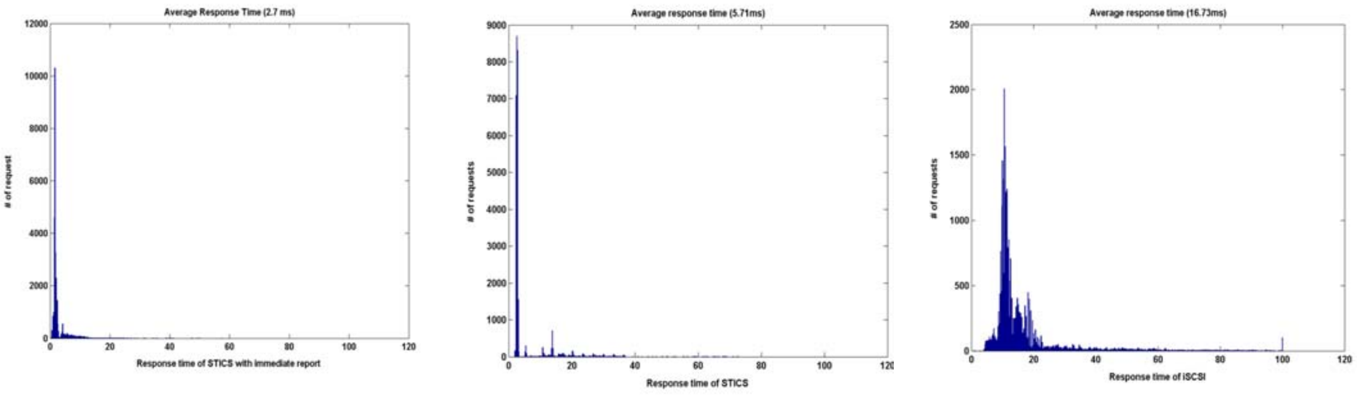


a) STICS with immediate report b) STICS with report after complete. c) iSCSI.

Figure 11: Response times for EMC-tel trace. Each dot in this figure shows the response time of an individual I/O request.

We also plotted histogram of request numbers against response times in Figure 12. 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 milliseconds. As shown in Figure

12a, for the STICS with immediate report, most requests are finished within 2 milliseconds, 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 milliseconds. For the STICS with report after complete as shown in Figure 12b, the response times of the majority of requests fall within the range of 2-5 milliseconds. 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.



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

Figure 12: Histograms of I/O response times for trace EMC-tel.

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 9c. 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.

5.4 Costs, Reliability, and Scalability 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 20 GB disk is just around $50 while a typical SAN costs over tens of thousands dollars, implying a very small fraction of additional cost of STICS. Table 7 lists the practical cost of building a small (480GB) SAN configuration with 6 servers using iSCSI and STICS, respectively[3]. As shown in this table, the cost difference between the two is just around 5%. Considering software cost (***$22,059***) and maintenance cost (***$8,676***) for the even smaller SAN system (200GB) [31], the cost difference between the two is much less than 2%. We believe trading 2% of additional cost for 6 folds performance gain is certainly worthwhile.

Table 7: Hardware costs comparison

	iSCSI			STICS		
	Qty	Cost	Total	Qty	Cost	Total
HBA card	12	$336.95	$4,043.40	12	$336.95	$4,043.40
Switch	1	$1,803.95	$1,803.95	1	$1,803.95	$1,803.95
GB NIC	12	$185.95	$2,231.40	12	$185.95	$2,231.40
OS HDD	12	$52.16	$625.92	12	$52.16	$625.92
SCSI Storage HDD	6	$573.26	$3,439.56	6	$573.26	$3,439.56
Log Disks				***12***	***$52.16***	***$625.92***
Total	***$12,144.23***			***$12770.15***		

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

[3] Prices are as of 12/12/2003 at www.dell.com

converter costing $5,083 [31] 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 6 for 12 STICS to be under $5,000. While at the same time, the cost of a Dell entry level SAN (CX200LC) with 360GB is $25,604.

High reliability of STICS is obvious as compared to traditional storage cache using large RAM because STICS uses disks for caching. The small NVRAM in our cache hierarchy is only up to 4MB. Transient data stay in this NVRAM less than a few hundreds milliseconds. Majority of cached data are in disks that are made extremely reliable today with the mean time to failure of millions of hours. RAM, on the other hand, has much higher failure rate with mean time to failure of a few thousands hours. In addition, RAM cache is also vulnerable to hardware failures such as board failure, CPU failure, and so forth. Disks can be unplugged from a failed system and plugged to another good system with data intact. In addition, log disks can be mirrored to further increase the reliability.

STICS-based SAN systems are also highly scalable. Off-the-shelf Ethernet Switches can be used to connect as many STICS as possible without obvious bottleneck. Furthermore, the LAN connecting STICS can be a completely separate network from the LAN interconnecting servers. This is in contrast to NAS that is attached to the same LAN where servers are connected, competing for the same network resources with servers that access the NAS.

To show the scalability of STICS, we built a larger system as shown in Figure 13, where STAR1 is the initiator, and STAR2 through 5 are targets. STAR6 is the network traffic monitor. STICS are installed in STAR1..5, where only STICS on the initiator is cache enabled. We also deployed an iSCSI system using the same 5 nodes, where STAR1 is the iSCSI initiator and STAR2 through 5 are iSCSI targets.

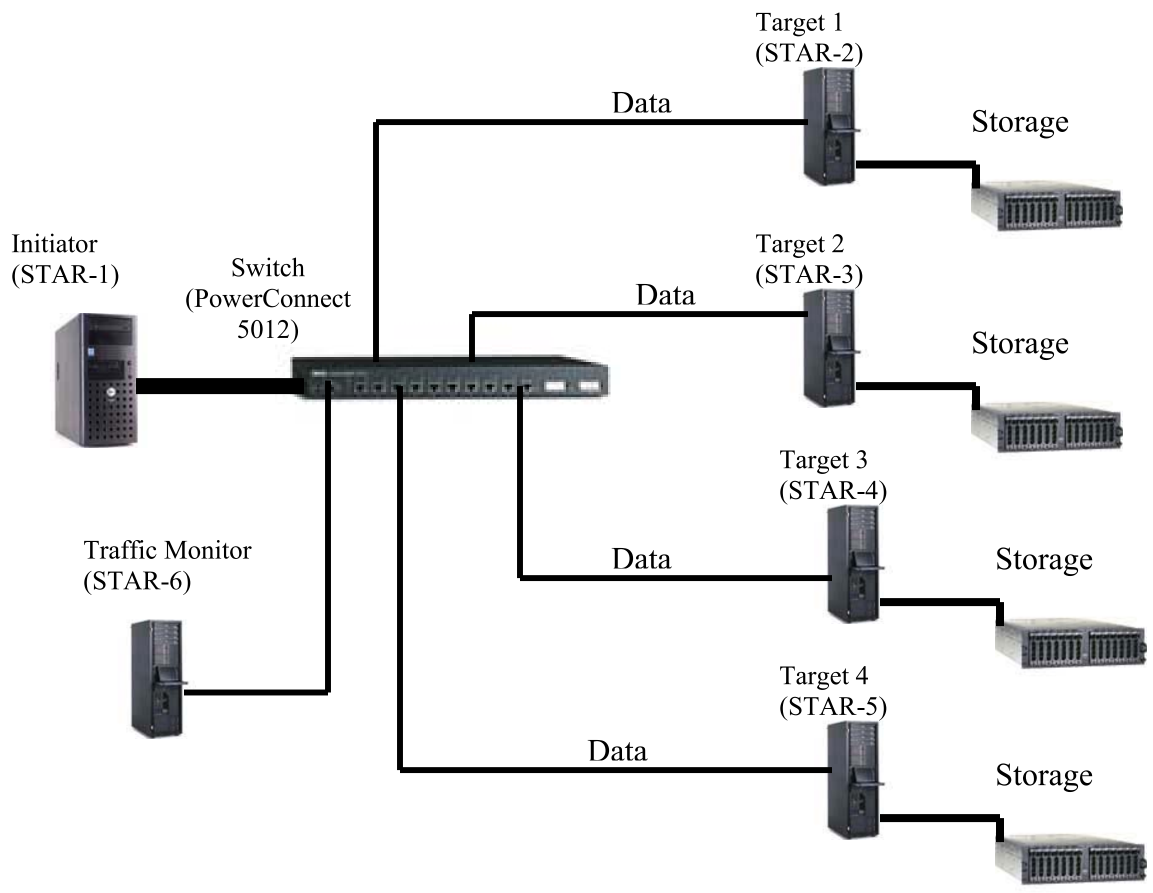


Figure 13: A larger system area network utilizing STICS.

Table 8: Performance Comparison between STICS and iSCSI with 5 nodes (20000 initial files and 100000 transactions)

	Block Size (bytes)			
	512	1024	2048	4096
iSCSI	318	367	354	283
STICS	552	589	537	432
STICS/iSCSI	1.74	1.61	1.52	1.53

We measured and compared the performance of both STICS with report after complete and iSCSI using PostMark benchmark under similar configuration as Section 5.3.1 under Gigabit network. The measurement results in terms of transactions per second are shown in Table 8. We observed performance gains ranging from 52% to 74% for different block sizes.

6 CONCLUSIONS

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 implemented a prototype STICS under the Linux operating system. We measured the performance of STICS as compared to a typical iSCSI implementation using popular benchmarks (PostMark, IOzone, and vxbench) and a real world I/O workload (EMC's trace). PostMark, Iozone, and vxbench results have shown that STICS enhances performance of iSCSI by a factor of 4.18, 4.3, and 1.9, respectively in terms of average system throughput. Numerical results under EMC's trace show a factor of 2.9 to 6.2 performance gain in terms of average response time. Furthermore, STICS is a plug-and-play building block for storage networks.

ACKNOWLEDGEMENTS

This research is supported in part by National Science Foundation under grants MIP-9714370 and CCR-0073377, and a Prototype Funding Award of University of Rhode Island Foundation. 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 Foundations. We would like to thank EMC Corporation for providing trace files to us. We would like to thank Jian Li for his invaluable assistance in our experiments. The first author's research is also supported in part by the Manufacturing Center at Tennessee Technological University. A short and preliminary version of this paper was presented at the *International Conference on Parallel Processing (ICPP'2002)* [15], Vancouver, Canada, August 18-21, 2002.

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)*, pp.81-91, October 2-7, 1998.

[2] S. Adve, V. Adve, M. Hill, and M. Vernon, "Comparison of Hardware and Software Cache Coherence Schemes," *Proceedings of the 18th Annual International Symposium on Computer Architecture (ISCA'91)*, pp. 298-308, May 1991.

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

[4] S. Aiken, D. Grunwald, A. Pleszkun, and J. Willeke, "A Performance Analysis of iSCSI Protocol," *IEEE Mass Storage Conference*, 2003.

[5] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang, "Serverless Network File Systems," *ACM Transactions on Computer Systems*, vol. 14, pp. 41-79, 1996.

[6] E. Bilir, R. Dickson, Y. Hu, M. Plakal, D. Sorin, M. Hill, and D. Wood, "Multicast Snooping: A New Coherence Method Using a Multicast Address Network," *Proceedings of the 26th annual international symposium on Computer architecture (ISCA'99)*, pp.294-304, 1999.

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

[8] Y. Chen, L. Ni, C.-Z. Xu, J. Kusler, and P. Zheng, "CoStore: A reliable and highly available storage systems," *Proc. of the 16th Annual Int'l Symposium on High Performance Computing Systems and Applications*, Canada, June 2002.

[9] F. Dahlgren and P. Stenstrom, "Using Write Caches to Improve Performance of Cache Coherence Protocols in Shared-Memory Multiprocessors," in *Journal of Parallel and Distributed Computing*, Vol. 26, No. 2, pp. 193-210, April 1995.

[10] 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, pp.127-153, 2000.

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

[12] 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.

[13] J. Gray, "Storage Bricks Have Arrived," Keynote in *Conference on Fast and Storage Technologies (FAST'2002)*, Monterey, CA, January 28-30, 2002.

[14] 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.*

[15] X. He, Q. Yang, and M. Zhang, "Introducing SCSI-To-IP Cache for Storage Area Networks," the *2002 International Conference on Parallel Processing (ICPP'2002)*, August 18-21, 2002.

[16] J. Hennessy, M. Heinrich, and A. Gupta, "Cache-Coherent Distributed Shared Memory: Perspective on Its Development and Future Challenges," *Proceedings of the IEEE*, vol. 87, pp. 418-429, 1998.

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

[18] Y. Hu and Q. Yang, "DCD-disk caching disk: A New Approach for Boosting I/O Performance," *23rd Annual Intl. Symposium on Computer Architecture (ISCA'96)*, pp.169-178, May 1996.

[19] 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.

[20] Intel iSCSI project, URL: *http://sourceforge.net/projects/intel-iscsi*, Jan. 2003.

[21] J. Jeong and M. Dubois, "Cost-sensitive Cache Replacement Algorithms," *Proc. of the 9th International Symposium on High Performance Computer Architecture (HPCA-9)*, Anaheim, CA, Feb. 2003.

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

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

[24] A. Klauser and R. Posch, "Distributed Caching in Networked File Systems," *Journal of Universal Computer Science*, Vol. 1, No. 6, pp. 399-408, June 1995.

[25] 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.

[26] A. Lebeck and D. Wood, "Dynamic Self-Invalidation: Reducing Coherence Overhead in Shared-Memory Multiprocessors," *Proceedings of the 22nd annual international symposium on Computer architecture (ISCA'95)*, pp. 48-59, 1995.

[27] 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, 1996.

[28] D. Lilja, "Cache Coherence in Large-Scale Shared Memory Multiprocessors: Issues and Comparisons," *ACM Computing Surveys,* vol. 25, pp. 303-338, 1993.

[29] 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*, pp. 101-115, April 2001.

[30] Y. Lu and D. Du, "Performance Study of iSCSI-based Storage Systems," *IEEE Communications*, Vol. 41, No. 8, 2003.

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

[32] 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, Aug. 1995.

[33] 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, Oct. 1998.

[34] K. Meth and J. Satran, "Features of the iSCSI Protocol," *IEEE Communications*, Vol. 41, No. 8, 2003.

[35] 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)*, Monterey, CA, January 28-30, 2002.

[36] V. Milutinovic and M. Valero, "The Evolution of Cache Memories," *Special Issue on Cache Memory IEEE Transactions on Computers*, pp. 97-99, February 1999.

[37] D. Nagle, G. Ganger, J. Butler, G. Goodson, and C. Sabol, "Network Support for Network-Attached Storage," *Hot Interconnects'1999*, August 1999.

[38] W. T. Ng, et al., "Obtaining High Performance for Storage Outsourcing," Proceedings of the FAST'02, January 2002.

[39] Nishan System white paper, "Storage over IP (SoIP) Framework – The Next Generation SAN," *URL: http://www.nishansystems.com/techlib/techlib_papers.html*, June 2001.

[40] W. Norcott, D. Capps, *IOzone file system benchmark*, URL: www.iozone.org

[41] A. Nowatzyk, G. Aybay, M. Browne, E. Kelly, M. Parkin, B. Radke, S. Vishin, "Exploiting Parallelism in Cache Coherency Protocol Engines," *Euro-Par'95*, pp.269-286, 1995.

[42] B. Phillips, "Have Storage Area Networks Come of Age?" *IEEE Computer*, Vol. 31, No. 7, 1998.

[43] E. Riedel, G. Faloutsos, G. Gibson and D. Nagle, "Active Disks for Large-Scale Data Processing," *IEEE Computer*, Vol. 34, No. 6, June 2001.

[44] P. Sarkar, S. Uttamchandani, and K. Voruganti, "Storage Over IP: When Does Hardware Support Help?" *Proc. of 2nd USENIX Conference on File And Storage Technologies (FAST'2003)*, San Francisco, CA, March 2003.

[45] J. Satran, et al. "iSCSI draft standard," *URL: http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-20.txt*, Jan. 2003.

[46] 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.

[47] S. Soltis, T. Ruwart, and M. O'Keefe, "The Global File System," In *Proceedings of the 5th NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies*, College Park, MD, 1996.

[48] C. Thekkath, T. Mann, and E. Lee, "Frangipani: A scalable distributed file system," *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 224-237, October 1997.

[49] University of New Hampshire Interoperability Lab iSCSI consortium, *URL: http://www.iol.unh.edu/consortiums/iscsi/*, Oct. 2001.

[50] 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.

[51] 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.

[52] P. Wang, et al.,"IP SAN-from iSCSI to IP-Addressable Ethernet Disks," *20th IEEE Conference on Mass storage Systems and Technologies*, 2003.

[53] R. Wang, T. Anderson, and D. Patterson, "Virtual Log Based File Systems for a Programmable Disk," *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 29-43, February 22-25, 1999.

[54] J. Wilkes, R. Golding, C. Staelin, T. Sullivan, "The HP AutoRAID Hierarchical Storage System," *Proc. Of the Fifteenth ACM Symposium on Operating System Principles*, Dec. 3-6, 1995, pp. 96-108.

[55] 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*, Sept. 2000.

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