Introducing SCSI-To-IP Cache for Storage Area Networks

Abstract

Data storage plays an essential role in today's fast-growing data-intensive network services. New standards and products emerge very rapidly for networked data storages. Given the mature Internet infrastructure, overwhelming preference among IT community recently is using IP for storage networking because of economy and convenience. iSCSI is one of the most recent standards that 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 such as disks or RAID, and an intelligent processing unit with CPU and RAM. The storage devices are used to cache and store data while the intelligent processing unit carries out caching algorithm, protocol conversion, and self-management functions. Through 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 efficient SAN 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.

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

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 [6] due to ever-growing demand for networked information services [19, 39]. In general, networked storage architectures have evolved from network-attached storage (NAS) [9, 12, 28], storage area network (SAN) [17, 29, 31], to most recent storage over IP (iSCSI) [12, 15, 33,37]. 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 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 [29] and the other encapsulates SCSI protocol in TCP/IP at host bus adapter (HBA) level [33]. 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. 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 size packets to transfer SCSI commands and status (most of them are only one byte). 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 PostMark benchmark [16] have shown that efficient caching can reduce total number of packets transferred over the net from 3,353,821 to 839,100 for same amount of remote storage data, a 4 times 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 reduce linearly with the increase of 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 Ethernet switches increase their bandwidth to gigabit or tens of gigabits due to technology advances, 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, 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. 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: one is SCSI interface and the other is standard Ethernet interface. The disk is used as a nonvolatile cache that caches data coming from possibly two directions: block data from SCSI interface and network data from Ethernet interface. In addition to standard SCSI and IP protocols running on the intelligent processing unit, it also implements a special caching algorithm controlling a two level cache hierarchy that writes data very quickly. Besides caching storage data in both directions, STICS also localizes SCSI commands and

handshaking operations to reduce unnecessary traffic over the Internet. In this way, it acts as a storage filter to discards a fraction of the data that would otherwise move across the Internet, reducing the bottleneck 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 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 [27], STICS is a "*cache storage*" used to cache networked storage data for server host.
- It utilizes the techniques in Log-structured file system [34,40] to quickly write data into magnetic media for caching data coming from both directions. Because a disk is used in caching, it is nonvolatile, which is extremely important for caching storage data reliably since once data is written to a storage, it is considered to be safe.
- By localizing part of SCSI protocol and filtering out some unnecessary traffic, STICS can reduce the bandwidth requirement of the Internet to implement SAN.
- Active disks [1,23,32] 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 [6] is standard "brick" for building storage systems, then STICS can be considered as a standard "beam" or "post" that provides interconnect and support for construction of SANs.

Overall, STICS adds a new dimension to the networked storage architectures. To quantitatively evaluate the performance potential of STICS in real world network environment, we have implemented the STICS under the Linux OS over an Ethernet switch. We have used PostMark [16] 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 measurement, our STICS shows up to 6 times as fast as the iSCSI in terms of average response time.

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

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.

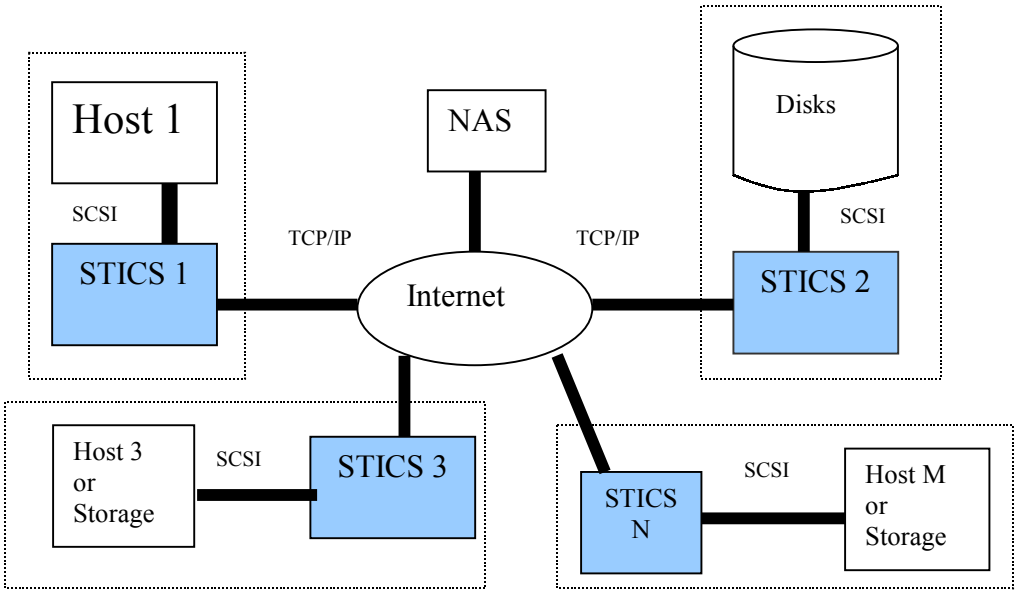


Figure 1: System overview. A STICS connects to the host via SCSI interface and connects to other STICS' or NAS via Internet.

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

1) A SCSI interface: STICS supports SCSI communications with hosts and other extended storage devices. Via the SCSI interface, STICS may run under two different modes: initiator mode or target mode [42]. When a STICS is used to connect to a host, it runs in target mode receiving requests from the host, carrying out the IO processing possibly through network, and sending back results to the host. In this case, the STICS acts as a directly attached storage device to the host. When a STICS is used to connect to a storage device such as a disk or RAID to extend storage, it runs in initiator mode, and it sends or forwards SCSI requests to the extended storage device. For example, in Figure 1, STICS 1 runs in target mode while STICS 2 runs in initiator mode.
2) An Ethernet interface: Via the network interface, a STICS can be connected to the Internet and share storage with other STICS's or network attached storages (NAS).
3) An intelligent processing unit: This processing unit has an embedded processor and a RAM. A specialized Log-structured file system, standard SCSI protocols, and IP protocols run on the processing unit. The RAM 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 [8] technique to keep meta data consistency without using NVRAM.

4) A log disk: The log disk is a sequential accessed device. It is used to cache 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 can be a disk, a RAID, or *JBOD* (Just-Bunch-Of-Disks). This storage device forms the basic storage component in a networked storage system. From point of view of a server host to which the STICS is connected through the SCSI interface, this storage device can be considered as a local disk. From the point of view of the IP network through the network interface, this storage can be considered as a component of a SAN with an IP address as its ID.

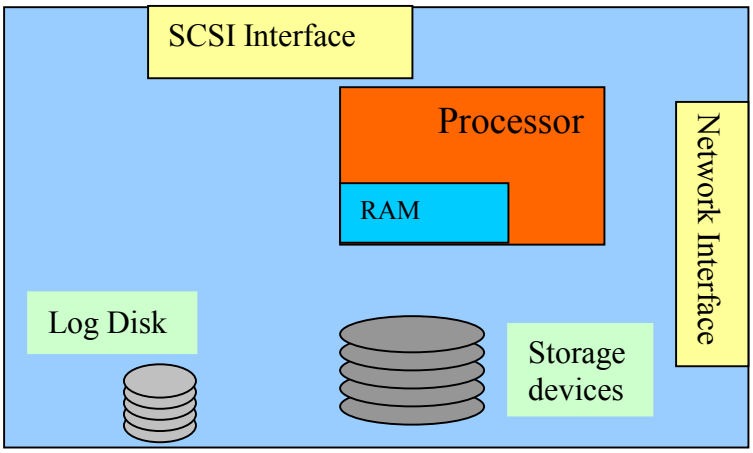


Figure 2: STICS architecture. A STICS block consists of a log disk, storage device, an intelligent processing unit with an embedded processor and sufficient RAM. Each STICS block has one SCSI interface and one network interface.

3. Design and Implementation

3.1 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.

3.2 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 [34]. 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 cache 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.

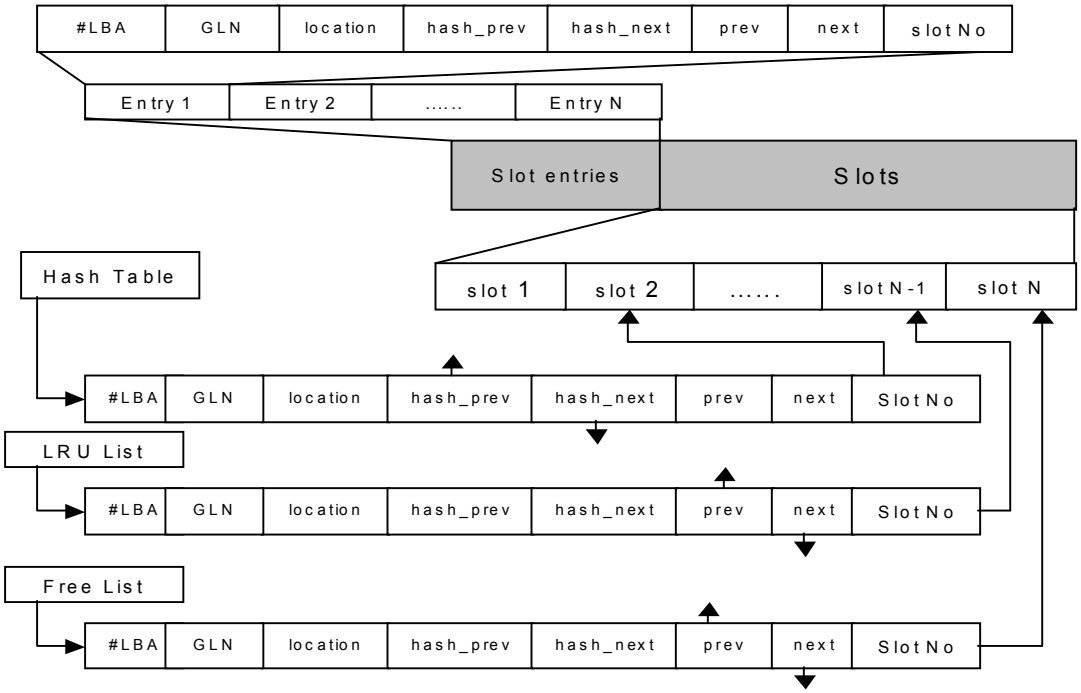


Figure 3: RAM buffer layout. RAM buffer consists of slot entries and slots. The hash table, LRU list and Free list are used to organize the slot entries.

Data blocks stored in the RAM cache are addressed by their *Logical Block Addresses (LBAs)*. The Hash Table contains location information for each of the valid data blocks in the cache and uses LBAs of incoming requests as search keys. The slot size is set to be the size of a block. A slot entry consists of the following fields:

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

3.3 Basic operations

3.3.1 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 GLN and LBA. It then searches the Hash Table by the LBA and GLN. The same writing process as above is then performed.

3.3.2 Read

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

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

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

3.3.3 Destages

The operation of moving data from a higher-level storage device to a lower level storage device is defined as *destage* operation [38]. 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*). 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 [38].

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.

3.4 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. 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 [2,3,5,7,11,18,20,22,30]. 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. In order to guide designers to make design decisions, we have considered both private and shared cache configurations. Shared cache configuration is relatively simple. For private cache configuration, we have designed the following coherence protocol.

Our cache coherence protocol in private cache system is based on the local consistency (LC) model [3], which helps to minimize meta-data network traffic pertaining to coherence protocol. The cache coherence granularity is configurable ranging from 32KB to 256KB to avoid false sharing [36]. We use shared-read/exclusive-write locks (tokens) to implement the necessary synchronization [4, 35]. Associated with each shared storage, there is a lock server that keeps track of lock status of shared data. The lock server is responsible for passing correct data to a requesting STICS cache together with the corresponding lock upon request. There are four types of messages for lock operations: request, grant, revoke, and release. Lock upgrade and downgrade operations are also handled with these four message types. The rule of lock service is as follows:

- A shared-read lock allows a STICS cache to read the shared data and cache it in the LRU Cache. If a STICS cache is asked to release its read lock, it must invalidate its cache entry before complying.
- An exclusive-write lock allows a STICS cache to read and write the single valid data copy and cache it in the NVRAM and the log disk. At this time, the cached data can be different from the original data in the shared storage since the STICS cache is the only one that holds the exclusive write lock. If the STICS cache is asked to release its write lock or downgrade the lock to read, it must write the dirty data to the shared data storage before complying. The STICS cache can retain its cache entry if it is downgrading the lock, but must invalidate the cache entry if releasing the lock.

To hold meta-data related to cache coherence protocol, each lock server and private cache (lock owner) maintains a lock status (LS) table. The LS table keeps track of lock information for each shared disk segment. The size of LS table is configurable. At a lock server, each entry in the LS table is 9 bytes long containing three fields: disk segment ID (4 bytes), lock owner's GLN (4 bytes), and lock information (1 byte). For 1 Terabyte data, the total size of the LS table is 36MB, about 0.004% overhead. The LS table at a private cache has the same structure as the one at a lock server, except that the GLN field records a lock server's GLN (4 bytes) instead of a lock owner's GLN.

In the protocol above, there is a possibility of deadlock or livelock when two operations need the same set of locks at the same time. A two-phase approach is designed to avoid such situation:

- Phase 1: A STICS tries to get all the locks it needs for an operation and is willing to release any lock it has acquired immediately.
- Phase 2: The STICS re-checks the locks and re-acquires locks if lost in Phase 1. If any data that was covered by a lock in Phase 1 has been modified since the lock was released, the STICS releases the lock, aborts the operation and rolls back to repeat Phase 1. Otherwise, it performs its serialized operation locally. This "abort and retry" mechanism is necessary to maintain serialization for each contending operation.

In a STICS-based SAN, a STICS cache or a lock server may be unreachable because of host/storage offline, STICS failure, or network disconnection. To deal with this situation,

we use a 30-second timeout [36] function with an unreachable location (UL) table with entries being GLN of unreachable lock owners or lock servers. When a lock server or a lock owner becomes unreachable, the corresponding UL table and LS table are updated accordingly. Similarly, when unreachable lock owner or lock server is recovered, the appropriate changes in corresponding UL table and LS table are made.

3.5 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 which can be loaded and unloaded dynamically. Our implementation uses a part of system RAM and an additional hard disk for caching function. There is no local storage and all I/O operations are remote operations going through the network.

4. Performance Evaluations

4.1 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 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

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
91366U4	ATA-5	13.6G	2MB	7200	4.18	Up to 33.7	9.0	Maxtor
AS010a1	Ultra ATA/100	10.2G	2MB	7200	4.17	16.6	8.5	Maxtor

Our experimental settings are shown in Figures 4 and 5. 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 1 and the characteristics of individual disks are summarized in Table 2.

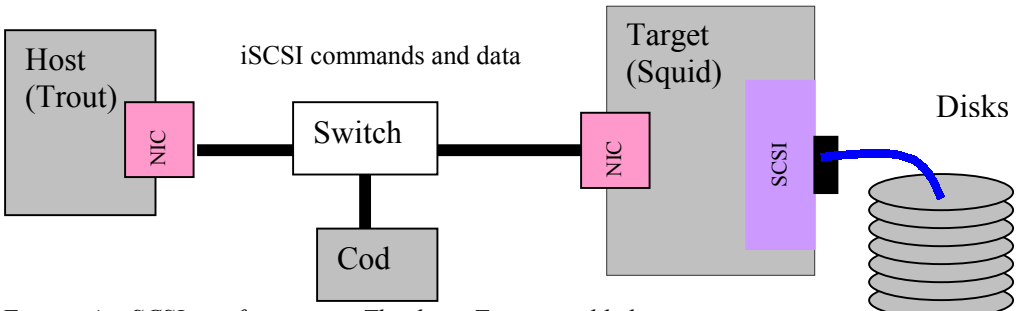


Figure 4: iSCSI configuration. The host Trout establishes connection to target, and the target Squid responds and connects. Then the Squid exports hard drive and Trout sees the disks as local.

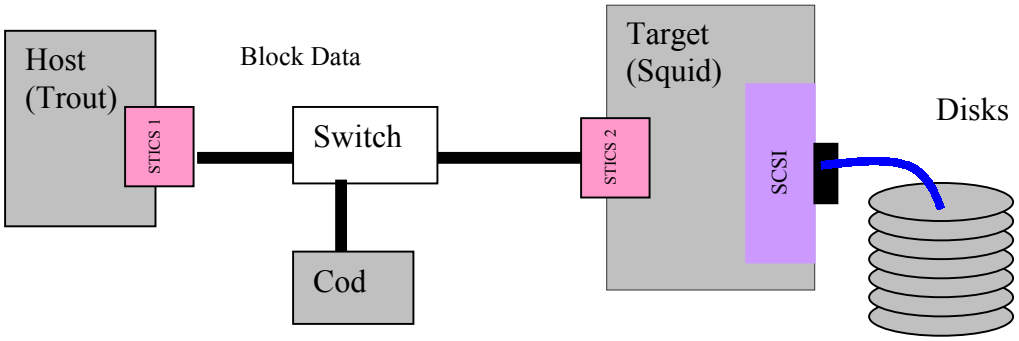


Figure 5: STICS configuration. The STICS cache data from both SCSI and network.

For iSCSI implementation, we compiled and run the Linux iSCSI developed by Intel Corporation [15]. 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.

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.

4.2 Benchmark program 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 real world trace and benchmark program.

The benchmark we used to measure system throughput is PostMark [16] which is a popular file system benchmark developed by Network Appliance. It measures performance in terms of transaction rates in an ephemeral small-file environment by creating a large pool of continually changing files. "PostMark was created to simulate heavy small-file system loads with a minimal amount of software and configuration effort and to provide complete reproducibility [16]." 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.

4.3 Measured results and discussions

4.3.1 Throughput

Our first experiment is to use PostMark to measure the I/O throughput in terms of transactions per second. In our tests, PostMark was configured in two different ways. 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.

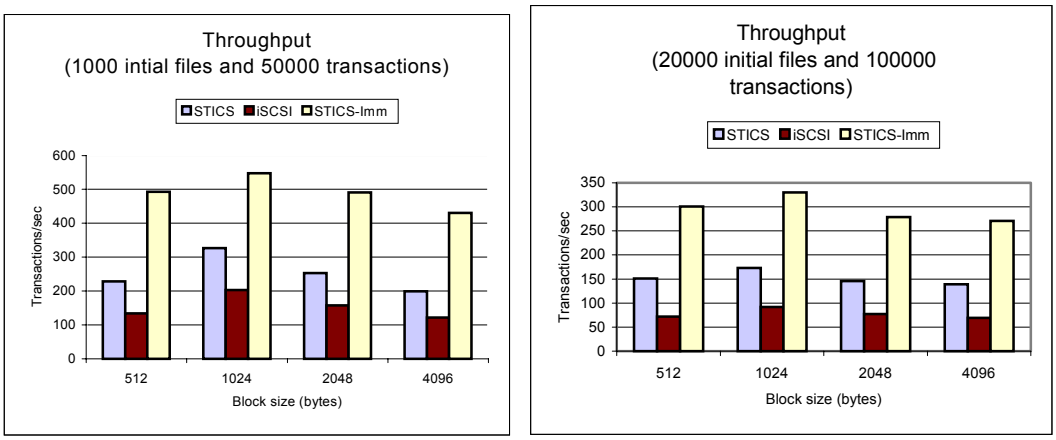


Figure 6: PostMark measurements (100Mbit network).

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.

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

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.

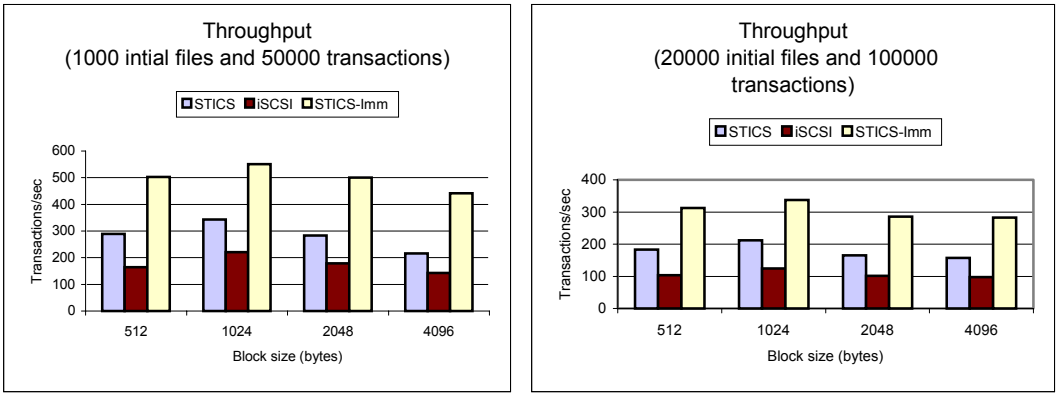
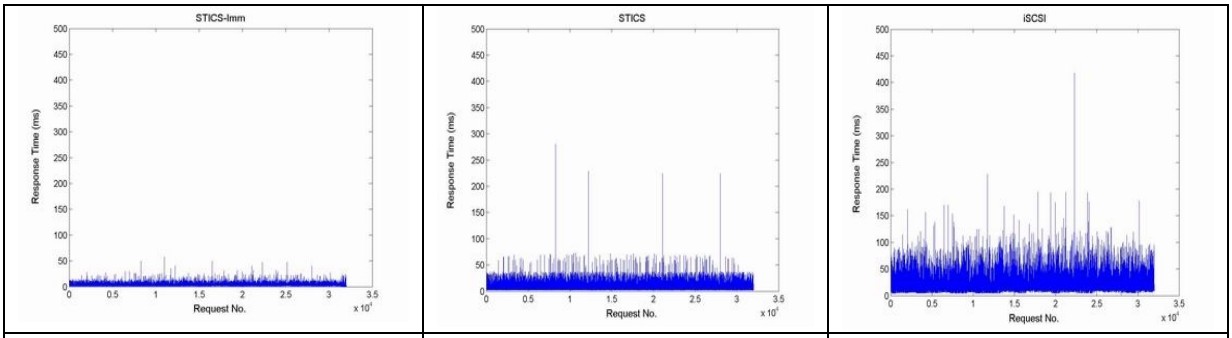


Figure 7: PostMark measurements (Gigabit network).

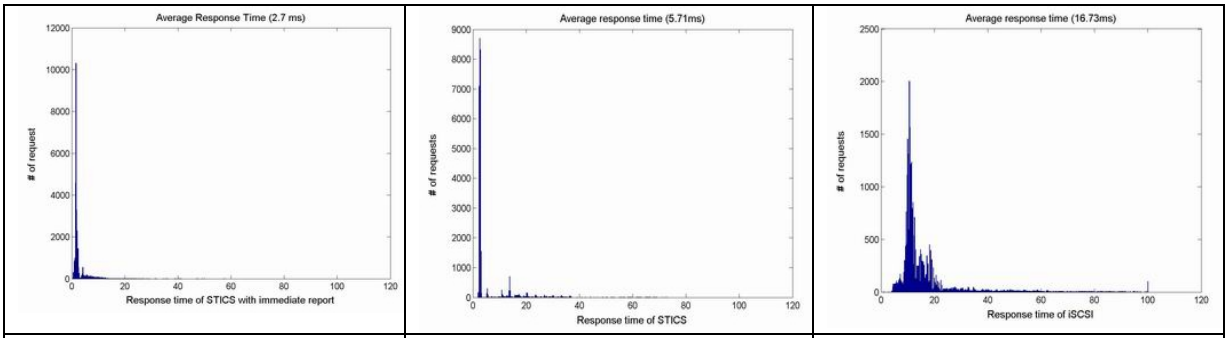
4.3.2 Response times

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 8 for STICS with immediate report (Figure 8a), STICS with report after complete (Figure 8b) and iSCSI (Figure 8c). 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 8: Response times for EMC-tel trace. Each dot in this figure shows the response time of an individual I/O request.



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

Figure 9: Histograms of I/O response times for trace EMC-tel.

We also plotted histogram of request numbers against response times in Figure 9. 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 9a, 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 9b, 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.

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.

4.4 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 10 GB disk is well under $100 while a typical SAN costs over tens of thousands dollars, implying a very small 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 [24], 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 card	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 Storage 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 [24] 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.

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.

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.

5. Related Work

Existing research that is most closely related to STICS is Network Attached Storage (NAS) [9,10,32], a research project at Carnegie Mellon University. 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 Ethernet plug. The main difference between NAS and SAN is that NAS provides storages at file system level while SAN provides storages at block device level. Another difference is that NAS is attached to the same LAN as the one connecting servers accessing storages, 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 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* [21, 36], 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) [12,15,33] 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* [26] 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 [33]. 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 [34,40]), Disk Caching Disk (DCD [13]), 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 [14,25,41]. 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.

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 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 Grants from National Science Foundation. We would like to thank EMC Corporation for providing trace files to us.

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] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

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

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

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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, October 3-7, 1998.

[27] V. Milutinovic and M. Valero, "The Evolution of Cache Memories," *Special Issue on Cache Memory IEEE Transactions on Computers*, pp. 97-99, February 1999.
[28] D. Nagle, G. Ganger, J. Butler, G. Goodson, and C. Sabol, "Network Support for Network-Attached Storage," *Hot Interconnects'1999*, August 1999.
[29] Nishan System white paper, "Storage over IP (SoIP) Framework – The Next Generation SAN," *URL: http://www.nishansystems.com/techlib/techlib_papers.html*, June 2001.
[30] 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.
[31] B. Phillips, "Have Storage Area Networks Come of Age?" *IEEE Computer*, Vol. 31, No. 7, 1998.
[32] E. Riedel, G. Faloutsos, G. Gibson and D. Nagle, "Active Disks for Large-Scale Data Processing," *IEEE Computer*, Vol. 34, No. 6, June 2001.
[33] J. Satran, et al. "iSCSI draft standard," *URL: http://www.ietf.org/internet-drafts/draft-ietf-ips-iscsi-06.txt*, April 2001.
[34] 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.
[35] 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.
[36] 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.
[37] University of New Hampshire Interoperability Lab iSCSI consortium, *URL: http://www.iol.unh.edu/consortiums/iscsi/*, Oct. 2001.
[38] 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.
[39] 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.
[40] 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.
[41] 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.
[42] Working Draft, "Information Technology: The SCSI Architecture Model-2 (SAM-2)," Revision 14, T10-1157-D, *URL: ftp://ftp.t10.org/t10/drafts/sam2/sam2r14.pdf*, September 2000.