Design and Evaluation of Cache Coherence in Distributed STICS – SCSI-To-IP Cache Storage[1]

Jian Li, Xubin He and Ming Zhang
Department of Electrical and Computer Engineering
University of Rhode Island, Kingston, RI 02881, USA
{lijian, hexb, mingz}@ele.uri.edu

ABSTRACT

STICS is a novel protocol cache storage architecture that couples reliable and high-speed data caching with low-overhead conversion between SCSI and IP protocols. This paper studies the cache coherence issue in distributed STICS system. A cache coherence protocol for private STICS cache scheme is designed and optimized. Modeling analysis of both private cache scheme and shared cache scheme are investigated. The network traffic overhead in shared cache scheme can be less that the one caused by cache coherence protocol in private cache scheme, when host hit rate is low, the remote memory access latency is closer to local memory access latency, and the number of hosts scales up. In addition, the space cost of lock service in RAM also has negative effect on private cache scheme. Simulation evaluation of a two-host-STICS-system is studied under two typical workloads, OLTP and Web Access. It shows that private cache scheme performs well in web access workload since web access workload is read dominant and has locality. In OLTP workload, shared cache scheme out-performs private cache scheme when STICS cache hit rate is less than 63% in high-speed network. The comparison between private cache and shared cache in STICS can be useful to other distributed networked storage systems.

1 Introduction

Cache is essential to achieve reasonable performance, it does, however, have cache coherence problem. Inconsistent view happens when multiple copies of the same data exist in different

[1] This research is supported in part by National Science Foundation under Grants CCR-0073377 and MIP9714370.

places (not necessarily in cache only) simultaneously. In this paper, only data consistence issue is discussed. Inconsistence caused by process migration and others are not considered.

STICS (SCSI-To-IP Cache Storage) is a novel protocol cache storage architecture that couples reliable and high-speed data caching with low-overhead conversion between SCSI and IP protocols [1,2]. Through efficient caching algorithm and localization of certain unnecessary protocol overheads, STICS can significantly improve performance, reliability, manageability, and scalability over current iSCSI systems. Analogous to "cache memory", which bridges the speed gap between CPU and memory, STICS is the first-ever "protocol cache storage" to bridge the protocol gap between SCSI and IP. With STICS, it is possible to build more efficient SAN (Storage Area Network) over IP. Figure 1 shows a STICS system in a typical SAN implementation over IP. Instead of using a specialized network or specialized switch, STICS connects a regular host server or a storage device to the standard IP network. For example, STICS 1 is directly connected to the SCSI HBA of Host 1. It can also act as a cache and bridge to allow Host 1 to access, at block level, any storage device connected to the SAN such as NAS (Network Attached Storage), STICS 2, STICS 3, etc. In order to allow a smooth data access from Host 1 to the SAN, STICS 1 provides SCSI protocol service, naming service, and IP protocol service, etc.

In a distributed STICS environment, questions remain such as where to put the cache service, near the host (private cache or STICS target) or storage device (shared cache or STICS initiator), and how to maintain data consistency. Cache hit rate varies with different application data access patterns, cache structures and cache policies. Low cache hit rate eliminates the advantage of private cache over shared cache. In current high-speed networks, such as Gigabit and 10 Gigabit LAN, the network operation latency of remote memory access is getting closer to local memory access. The overhead of network traffic in shared cache scheme can be less than the one caused by cache coherence protocol in private cache scheme. We investigates the difference between two cache schemes: private STICS cache and shared STICS cache. A cache coherence protocol

for private STICS cache was proposed and optimized. Modeling and simulation evaluation of the two cache schemes are studied. The contribution of this work can be classified as follows:

- Caching for data storage and the arising cache coherence issue are usually handled by the corresponding distributed file system in a distributed operating system. Changing OS results in difficulty in porting to other systems. Since STICS is built at device driver or I/O card hardware level in a computer network environment, change to existing system is minimized. No similar research effort was done before.
- The STICS cache structure is an asynchronous L2 cache, as indicated in Section 2. Since STICS is a cache for data storage, it functions as an extended disk subsystem from the host point of view. It also offloads disk management utility from file system. In this sense, the cache coherence of STICS migrates the design experience from both multiprocessors and Distributed File Systems (DFS).
- Quantitative analysis of private cache and shared cache in computer network environment are deployed. In modern high-speed computer networks, remote memory access latency is getting closer to local memory access latency. Hit rate for storage cache is relatively low. Both hit rate and network latency need to be studied before choosing an appropriate cache scheme for a large scale STICS system. This study can be useful to other distributed systems.

The rest of this paper is organized as follows. Next section presents two cache schemes, private cache and shared cache, in a distributed STICS system, as well as the cache coherence protocol for private cache scheme. Section 3 is modeling and performance evaluation of the two proposed cache schemes. Numerical results obtained from simulation experiments are also discussed in this section. Previous research related to the work is discussed in Section 4. Section 5 concludes and mentions possible future work.

2 Design of Cache Coherence Scheme in Distributed STICS

2.1 Cache Structure in a STICS Building Block

Besides a SCSI interface and a network interface, a STICS building block consists of three main components to form its cache function, as show in Figure 2:

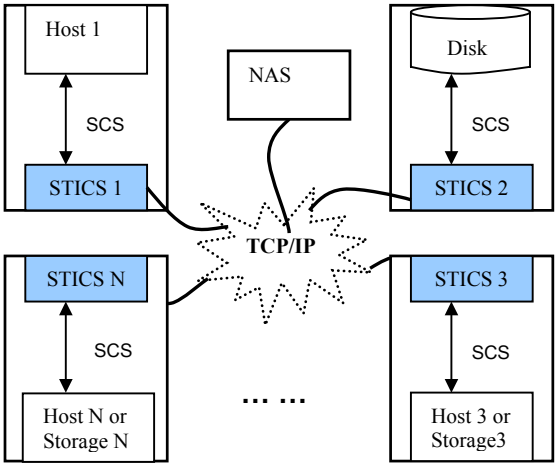


Figure 1: Distributed STICS system overview.

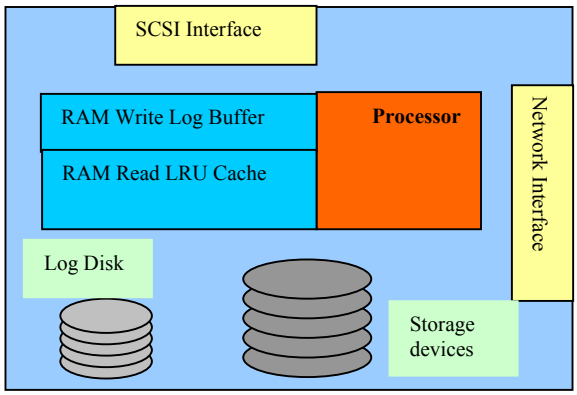


Figure 2: STICS building block architecture.

- An intelligent processing unit: This processing unit has an embedded processor and a RAM. A specialized log-structured data system [3], standard SCSI protocol, and IP protocol run on the processing unit. The RAM in a STICS is divided into two major parts: a small write log buffer (called Log Buffer thereafter), and a big read LRU cache (called LRU Cache thereafter) with prefetch function. The rest of the RAM is dedicated to some other modular services, such as lock service and management service, which will be discussed later in the paper. Note that the LRU cache is not overlapped with the file system cache in the host, because STICS is a separate intelligent unit to the host.
- A log disk: The log disk is a sequential access device. It is used to cache the destaged data from the Log Buffer. The Log Buffer and the log disk form a L2 cache similar to DCD [4]. Whenever the log disk is idle, dirty data are written from the Log Buffer into

the log disk. At this time, the data in the log buffer is invalidated, but another copy will keep valid in the LRU cache. Such an asynchronous L2 cache is similar to RAPID [5].

- Storage device: The regular storage device can be a disk, a RAID, or Just-Bunch-Of-Disks (JBOD). This storage device forms the basic storage component in a networked storage system.

2.2 Private STICS Cache vs. Shared STICS Cache

When put the cache service near the host, the corresponding STICS building block is a private cache; near the storage, a shared cache. The difference between private cache and shared cache is: Private cache has faster cache access time and less network data traffic, but it has data consistency problem because multiple copies of the same data exist. Private cache scheme faces the cost of cache coherence protocol, while shared cache scheme does not. However, shared cache scheme can not reduce iSCSI handshaking as much as private cache scheme, because each single SCSI initiator command has to go over TCP/IP network before being cached. Study on the tradeoff is necessary before determine where to put the STICS caching service.

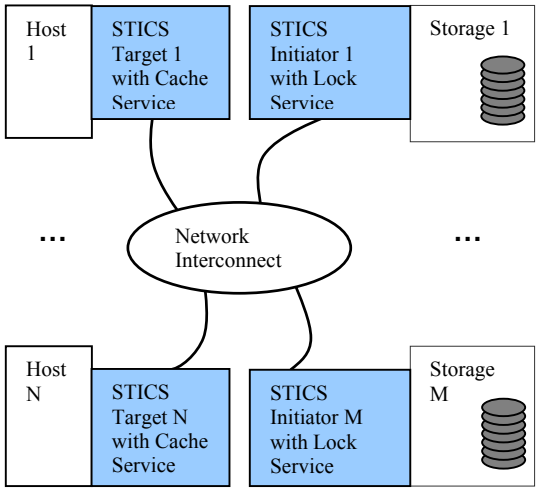
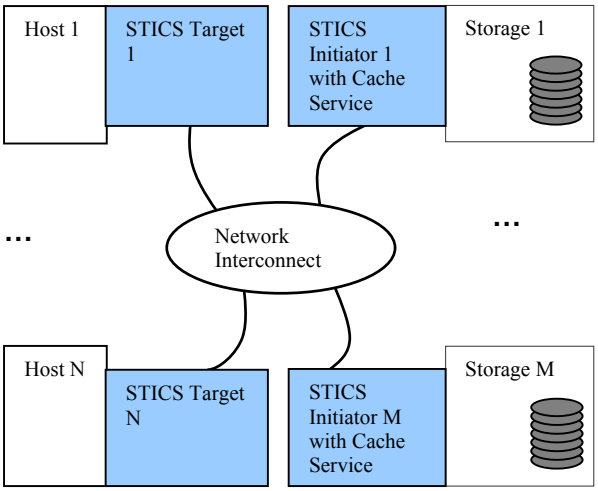


Figure 3: Private cache scheme. (a)



Shared cache scheme. (b)

The private STICS cache scheme is illustrated in Figure 3a and the shared STICS cache scheme in Figure 3b. The building structures of the two schemes are the same except that: in private cache scheme, the STICS target is with cache service enabled and the STICS initiator with lock

service enabled (for cache coherence purpose); in shared cache scheme, the STICS initiator is with cache service enabled while lock service is not necessary. Note that each STICS building block is equipped with modular services. Depending on where a STICS is located, it can enable and disable services as needed. When one service crashes, it will not affect other services. This makes the system more fault-tolerant and flexible. A remote accessible STICS management interface is now under-developing which will be added to the STICS environment soon.

2.3 Cache Coherence Protocol for Private STICS Cache

Cache coherence problem only happens with private caches. In a private-cache STICS environment, cache coherence protocol is based on the local consistency (LC) model [6], which helps to minimize the meta-data network traffic caused by cache coherence protocol. The STICS system data processing is configurable per-disk-segment granularity, currently ranging from 32KB to 256KB. The cache coherence is fine grained to the same configurable level. Such coherence granularity, similar to xFS, is to avoid false sharing [7]. The protocol borrows ideas from xFS, GFS [8] and Frangipani [9]. We use shared-read/exclusive-write locks (tokens) to implement the necessary synchronization. For each STICS cache service, the read/write procedure with shared-read/exclusive-write locks is very simple to implement, even though data may exist in four different places:

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

disk (if it is shared) before complying. It can retain its cache entry if it is downgrading the lock, but must invalidate it if releasing the lock.

There are four types of lock operation: request, grant, revoke, and release. Lock upgrade and downgrade operations are also handled with these four message types. As a lock server is resident on a STICS attached to storage, it is the lock server's responsibility to pass the dirty data to the request STICS cache together with the corresponding lock. A STICS does not pass dirty data to another STICS directly, which is different from xFS and GFS, because we want to keep recovery process simple.

The correctness of STICS cache coherence protocol can be proved by considering all the possible cases for a read operation. The shared-read/exclusive-write lock scheme serializes each operation and guarantees the data returned by any read operation belongs to the set of latest updated data, because either multiple readers (including one) or single writer exists, not both. Theoretical analysis and experiments are done to prove the theory. Because of limited space, this part is omitted in the paper.

Several techniques are utilized to reduce the overhead of STICS cache coherence protocol. The distributed lock server scheme can be naturally implemented with First Write policy [10], i.e., when data is written to an empty disk segment the lock status of this disk segment is automatically maintained by the lock server attached to the data storage. Different from GFS, distributed lock servers are only responsible for locks of the attaching data storage. They pass dirty data to both the requesting STICS and the data storage, to eliminate extra request to the data server. Consistency related information can be adaptively batched and piggybacked, with new lock requests from the STICS to the lock server or with response vice versa. The lock is sticky to further reduce network traffic [11].

The distributed lock servers maintain shared-read/exclusive–write lock status tables. Lock servers flush lock status to its local disks periodically for checkpoint purpose. Each tuple in a table contains a 4-Byte Disk Segment ID, a 4-Byte lock owner's GLN (Global Location Number),

and 1-Byte lock information. Currently we allow up to 4 million cache entries in a lock status table, the total size of which is set to 36 MB. Such a table can hold lock information for up to 1 TB data. The lock server uses 30-second socket timeout to deal with STICS failure and to reduce the space of lock status table. When a STICS' response times out, the lock server purges the corresponding lock status table entries and inserts a new tuple to an un-reachable STICS table. It also reports failure to STICS management facility. The un-reachable STICS table only contains 4-Byte STICS GLN entries. We currently set one thousand entries to the un-reachable STICS table. The total size of the table is 4 KB. Therefore, the total space cost of lock service is less than 40 MB. When a 'lost' STICS shows up again, the lock server purges the entry in the un-reachable STICS table and asks such STICS to reset its cache entries before processing its lock request. With the coordination of the two meta-date hash tables, the space cost of the lock service is minimized. The pseudo codes of the lock service are listed in Figure 4. Note that a 30-second timeout interrupt is applied to each socket communication to handle STICS failure. A socket timeout is also applied at the private STICS cache side, however timeout only causes random retries. When timeout number exceeds a certain number, the STICS cache will report lock server failure to STICS management facility. A 9-Byte tuple lock status table is also maintained by each private STICS cache. It is similar to the table at lock servers except that the GLN number records the lock server's GLN not the lock owner's.

```
doReadLockRequest
{
  if (not in un-reachable STICS table) {
    Check what is the Lock Status for the requested Disk
    Segment;
    if (CurrentState = write) {
      Downgrade CurrentSTICSCache lock to Read with
      timeout interrupt;
    }
    NewLockState = Read;
    Update lock state on lock server;
    Grant lock to the RequestSTICSCache with timeout
    interrupt;
  }
  else {
    Ask RequestSTICSCache to reset its cache entries with
    timeout interrupt;
    Purge RequestSTICSCache entry in un-reachable table;
  }
  Return success;
}
```

(a) Pesudo code for read lock processing at lock server

```
doWriteLockRequest
{
  if (not in un-reachable STICS table) {
    Check what is the Lock Status for the requested Disk
    Segment;
    if (CurrentState != free) {
      Revoke lock from CurrentSTICSCaches other than
      RequestSTICSCache with timeout interrupt;
    }
    NewLockState = Write;
    Update lock state on lock server;
    Grant lock to the RequestSTICSCache with timeout
    interrupt;
  }
  else {
    Ask RequestSTICSCache to reset its cache entries with
    timeout interrupt;
    Purge RequestSTICSCache entry in un-reachable table;
  }
  Return success;
}
```

(b) Pesudo code for write lock processing at lock server

Figure 4: Pesudo code for lock service.

Deadlock and livelock may occur when some operations need same set of locks at the same time. For example, two atomic operations both need two same write locks, when each operation holds only one of the lock and will not release its lock, dead lock occurs. To avoid such probability, a two-phase approach is implemented:

- Phase 1: A STICS tries to get all the locks it needs for an operation and is willing to release any lock it has required 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 release the locks and loops 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.

3 Modeling and Evaluation of Cache Coherence in Distributed STICS

3.1 Modeling and Analysis

Since both private and shared STICS caches use the same cache policy, they have the same number of total disk accesses. Disk access latency is usually an order (or more) of magnitude larger than remote memory access latency, and two order of magnitude larger than local memory access latency [12, 13]. Therefore, in the preliminary model, we exclude the effect of disk accesses and focus on the effect of local and remote memory access latencies only.

The average response time per request for shared cache, assuming that at the shared cache the time to process a client request is equal to the time to process a request to data disk, is:

$$T_{sc} = T_c + T_{net} + (2 - P_h) \times T_s \qquad (1)$$

Similarly, the average response time per request for private cache is:

$$T_{pc} = T_c + T_p + (1 - P_h) \times ((P_r \times P_{ciw} + P_w \times P_{onn}) \times (T_{net} + T_p) + 2 \times T_l + T_{net}) + P_h \times P_w \times P_{war} \times (T_l + T_{net}) \qquad (2)$$

The probability of lock request for a private cache in one transaction after cache is warmed up, is:

$$P_l = (1 - P_h) \times (P_w \times P_{oir} + P_r \times P_{nl}) + P_h \times P_w \times P_{war} \quad (3)$$

In the formulas above, T_c, T_p, T_s, T_{net}, and T_l are client latency, private cache latency, shared cache latency, network latency (time of flight plus data transfer time), and lock server latency respectively; $P_h = 1\text{-}P_m$ is the cache hit rate; $P_w = 1 - P_r$, is the probability of a write request; P_{ciw} and P_{cir} are the probabilities that the current lock status of the requested disk segment is write and read respectively; P_{onn} is the probability that the STICS that holds the lock of the disk segment is not the STICS that requests for the lock; P_{nl} is the probability that no lock is available for the requested disk segment; P_{war} is the probability that a STICS holding a read lock asks for a write lock for the same disk segment.

To compare shared STICS cache scheme and private STICS cache scheme, two typical workloads are studied:

- OLTP (Online Transaction Processing) workload: The STICS randomly access a working set of N blocks with uniform probability of reading and writing each block [14]. There is a linear relationship between the aggregate LRU cache stack depth and the cumulative hit fraction.
- Web Access workload: The STICS reads from a working set of N blocks, where the probability of reading the *ith* block is proportional to $1/i^{\alpha}$, yielding a Zipf-like distribution. Here we only consider the case when $\alpha=1$ [15]. This is the behavior of web access where very few writes happens [16]. For small cache stack depth, as the STICS LRU cache stack increases, the hit ratio increases dramatically.

Assuming all local memory access are the same, We set $T_c = T_p = T_s = T_l = T_{lm} = 1$, and let $R = T_{net} / T_{lm} >= 1$. For a two-STICS-private-cache system, the formulas above can now be simplified as follows:

$$T_{sc} = 3 - P_h + R \quad (1a)$$

$$T_{pc} = \frac{1}{8}\left(-10P_h \times R + 11R - 18P_h + 35\right) \qquad \textit{for OLTP workload} \qquad (2a)$$

$$T_{pc} = -P_h \times R + R - 2P_h + 2 \qquad \textit{for Web workload} \qquad (2b)$$

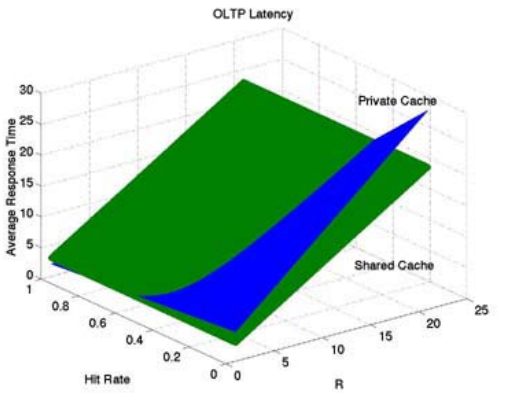
We compare the average response time per request of shared cache scheme and private cache scheme under the two workloads in Figure 5. In Figure 5, hit rate is ranging from 0 to 1, R is ranging from 1 to 21. The unit of average response time is T_{lm}. It shows that, when hit rate and R are low shared cache performs better than private cache. Shared cache also has more gain under OLTP workload than under Web Access workload. Figure 6 compares the average response time per request as a function of LRU cache stack depth and R (T_{net}/T_{lm}). Note that the response time of both shared cache and private cache increase dramatically when cache stack depth is approaching zero. Figure 7 shows the intersection line where the response time of private STICS cache equals to the one of shared STICS cache. Because the active data working set in web access workload is easily to be captured by caches (high hit rate), private cache scheme (client caching) is a better choice. However, since hit ratio in storage cache is usually low, especially in OLTP workload, it is better to choose shared cache scheme by eliminating the overhead of coherence protocol. As shown in Figure 7, in a slow network the shared cache scheme has the same performance as private cache scheme when hit rate is around 0.3. As network latency getting smaller, it is better to use shared cache even when hit rate is as high as 63%. The hit rate can be even higher when host number is getting larger.

Lock request probability for private cache in a two-STICS-private-cache system is:

$$P_l = \frac{1}{2} - \frac{3}{8}P_h = \frac{1}{8} + \frac{3}{8}p_m \qquad \textit{for OLTP workload} \qquad (3a)$$

$$P_l = 0 \qquad \textit{for Web workload} \qquad (3b)$$

It suggests that: In OLTP, lock request probability is proportional to cache miss rate; In web access, lock request probability is zero after the active data working set is warmed up, i.e., all the data has been accessed at least once.



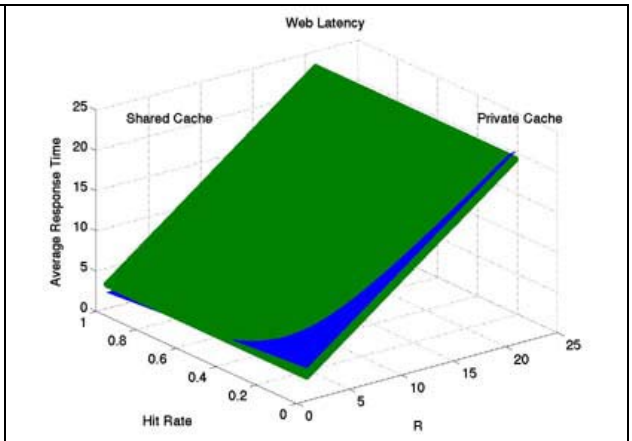
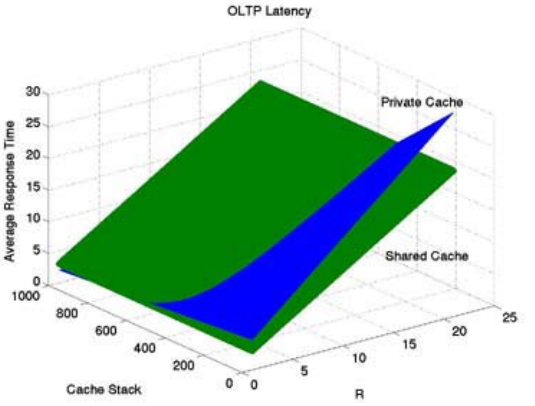


Figure 5: Latency comparison between private cache (blue) and shared cache (green) under OLTP workload (a) and Web workload (b), with respect to hit rate and R (T_{net}/T_{lm}).



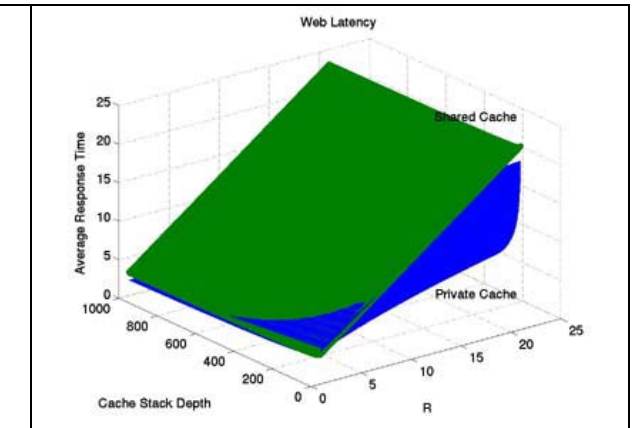


Figure 6: Latency comparison between private cache (blue) and shared cache (green) under OLTP workload (a) and Web workload (b), with respect to LRU cache stack depth and R (T_{net}/T_{lm}). (Active working data set = 1000 cache lines)

3.2 Simulation Results and Discussion

Currently STICS is implemented in Linux OS. A cache coherence protocol simulator called CCSim is developed to compare the feasibility of private cache and shared scheme for STICS. CCSim is a set of user space program running on each STICS building block. In the experiments, 4 Gateway E-1400 PCs are connected via a 100Mbps LAN. The experiment layout for both schemes is in accordance with Figure 3. To compare with our model, we studied a case where two hosts are connected to two data storages. Direct-mapping cache is simulated for both private

cache and shared cache scheme in OLTP and web access workload respectively. The experiment system configuration is shown in Table 1. In the experiment: The total number of transactions is 100000, active data working set is 5K cache lines, cache line (data block size) is ranging from 128B to 8KB. This arrangement is not enough for storage system performance evaluation. However, since we are only interested in the overhead of cache coherence protocol without disk data access, we believe such experiment setup is enough to get valuable insight. It is proved correct by the experiment results in Figure 8 to Figure 11.

Computer Name	**System Configuration**	**Roles in Privated Cache Scheme**	**Roles in Shared Cache Scheme**
PC1	Gateway E-1400 PC: Celeron 500MHz, 128 MB RAM, 3c905b 10/100Mb NIC	Client 1, STICS Private Cache Server 1	Client 1
PC2		Client 2, STICS Private Cache Server 2	Client 2
PC3		Lock Server 1, Data Server 1	STICS Shared Cache Server 1, Data Server 1
PC4		Lock Server 2, Data Server 2	STICS Shared Cache Server 2, Data Server 2

Table 1: Experiment system configuration.

Figure 8 shows that when hit rate is high (94% in the figure), the response time of shared cache scheme is more sensitive to data block size per transaction. This is because: in shared cache scheme every data need to be transferred to and from the shared cache over the network no matter hit or miss, while in private cache scheme data can be cached locally and coherence protocol only transfer small packets of meta data, which is independent to the data block size. Figure 9 shows when miss rate is around 65%, the response time of private cache is equal to the one of shared cache. This result is expected by the modeling result in Figure 7, because in a 100Mb LAN remote memory access latency is much higher than local memory access latency. Note that under a random read/write OLTP workload, high miss rate is common. Figure 10 proves the correctness of modeling analysis in Figure 6b. It shows that when LRU cache stack depth decreases after a turning point, the response time for both private cache and shared cache increase dramatically. As the network is relatively slow and lock requests rate is low in web access workload (read dominant), private cache scheme outperforms shared cache scheme. Figure 11

also proves the correctness of our preliminary model for lock requests in the two-client case, as the two curves overlap.

For a fixed cache structure, different application workloads result in different cache hit rates. Shared cache scheme works well in low hit rate environment. Note that the cache hit rate is relatively low for storage and data access over network. In a high-speed network, such as Gigabit and 10Gigabit LAN, as remote memory access latency is getting closer to local memory access latency, the overhead of shared cache is minimal. As the host number is getting larger shared cache scheme will perform better, because more lock contention in private cache scheme occurs. The space cost of lock service in RAM also has negative effect on the performance of private cache scheme. All the above factors will affect the location decision of cache services in STICS systems.

4 Related Work

Cache coherence scheme has been implemented throughout all the other system levels in distributed systems. Distributed file systems (DFS) were once the most widely used distributed systems. Most of them borrows ideas of cache coherence protocol from multiprocessors [17, 18]. For example, xFS's cache coherence protocol is similar to those in hardware DSM systems. However, aspects of a distributed file system require different protocols that cannot be ported from DSM system directly [10]. STICS, as discussed in Section 2, encounters more specific problems in its new architecture environment.

Features of an xFS [7, 10] file system cache coherence protocol are: It utilizes a token-based (sometimes called lock-based) cache consistency scheme similar to AFS, i.e., a directory-based invalidate cache coherence protocol with callback; It manages write-ownership based cache consistency on a per-block rather than per-file basis; it support host-to-host data transfer.

GFS is mainly implemented for Fiber Channel (FC) based SAN [8, 19]. It can also be used in IP based Network Block Device (NBD) storage. Currently no iSCSI based storage has been

implemented yet. In a GFS, superblock of the metadata structure is not distributed across the resource groups, which may become a bottleneck of performance and a single point failure. GFS uses atomic read-modify-write operations on disk-resident metadata (locks) to maintain file system consistency. The GFS lock abstraction allows GFS hosts to implement callbacks and data transfer between each other, which is similar to xFS.

Cooperative caching middleware for cluster-based servers [20], supports read-only request stream. Global memory service for workstation clusters [12, 21] is a single, unified, but distributed memory management algorithm at the lowest level of the OS. Its cache coherence is handled by upper-level software, such as NFS. Cache coherence in a global memory management for a multi computer system [13], is maintained by hardware interconnect. Buffer Server [22], which is an I/O node near clients, functions as a cache and proxy. It is basically a shared cache scheme.

5 Conclusions and Future Work

This paper studies the cache coherence issue in distributed STICS environment. When put the cache service near the host, the corresponding STICS building block is a private cache; near the storage, a shared cache. Private cache may cause data consistency problem. A cache coherence protocol for private cache scheme is designed and optimized. Two typical workloads, OLTP and web access, are analyzed and simulated in a case where two host are connected to two data storages. It shows private cache scheme performs well in web access workload since web access workload is read dominant and has locality. In OLTP workload, shared cache scheme out-performs private cache scheme when such storage cache hit rate is less than 63% in high-speed network. As the host number is getting larger shared cache scheme performs better, because more lock contention in private cache scheme occurs. In addition, the space cost of lock service in RAM also brings negative effect on the performance of private cache scheme. Careful study is necessary to decide the location of cache services in a large-scale STICS systems, with respect to

various network environment and application workloads. The comparison between private cache and shared cache in STICS can be useful in other distributed network attached storage systems.

In the future, we will consider modeling of multi-clients with queuing delay. We plan to utilize bigger data chunks to test real storage network performance. The cost of lock service will be studied in detail.

References

[1] X. He, Q. Yang, and M. Zhang, "Architecture and Performance Potential of STICS -- SCSI-To-IP Cache Storage," ICPP'2002.

[2] Q. Yang and X. He, "STICS -- SCSI-To-IP Cache Storage," in File Pending. USA, 2001.

[3] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," ACM Transactions on Computer Systems, vol. 10, pp. 26-52, 1992.

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

[5] Y. Hu and Q. Yang, "DCD -- Disk Caching Disk: A New Approach for Boosting I/O Performance," In Proceedings of the 23rd Annual Int'l Symposium on Computer Architecture (ISCA), Philadelphia, PA, May, 1996.

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

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

[8] S. R. Soltis, T. M. Ruwart, and M. T. 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.

[9] C. A. Thekkath, T. Mann, and E. K. Lee, "Frangipani: A Scalable Distributed File System," In Proceedings of ACM Symposium on Operating Systems Principles, 1997.

[10] R. Y. Wang and T. E. Anderson, "Experience with a Distributed File System Implementation," University of California at Berkeley CSD-98-986, 1998.

[11] A. Adya, "Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions," in Department of Electrical Engineering and Computer Science. Boston: Massachusetts Institute of Technology, 1999, pp. 198.

[12] G. M. Voelker, E. J. Anderson, T. Kimbrel, M. J. Feeley, J. S. Chase, A. R. Karlin, and H. M. Levy, "Implementing Cooperative Prefetching and Caching in a Globally-Managed Memory System," In Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, 1998.

[13] D. Milojicic, S. Hoyle, A. Messer, A. Munoz, L. Russell, T. Wylegala, V. Vellanki, and S. Childs, "Global Memory Management for a Multi Computer System," In Proceedings of the 4th USENIX Windows Symposium, Seattle, WA, 2000.

[14] "TPC Benchmark C, Standard Specification Revision 3.5," Transaction Processing Council, October 1999, http://www.tpc.org/cspec.html, Accessed August, 2001.

[15] P. Cao, L. Fan, G. Phillips, S. Shenker, and L. Breslau, "Web Caching and Zipf-like Distributions: Evidence and Implications," In Proceedings of Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), New York City, NY, March 21-25, 1999.

[16] A. Veith, E. Riedel, S. Towers, and J. Wilkes, "Towards Global Storage Management and Data Placement," Hewlett Packard Laboratories HPL-SSP-2001-1, March 2001.

[17] 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.

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

[19] M. Tilstra, "GFS HOWTO," Sistina Software, Inc. June 18th. 2001.

[20] F. M. Cuenca-Acuna and T. D. Nguyen, "Cooperative Caching Middleware for Cluster-Based Servers," In Proceedings of the 10th International Symposium on High Performance Distributed Computing (HPDC), August, 2001.

[21] M. J. Feeley, W. E. Morgan, F. H. Pighin, A. R. Karlin, and H. M. Levy, "Implementing Global Memory Management in a Workstation Cluster," In Proceedings of the 15th Symposium on Operating Systems Principles, 1995.

[22] D. Anderson, K. Yocum, and J. Chase, "A Case for Buffer Servers," In Proceedings of IEEE Workshop on Hot Topics in Operating Systems (HOTOS), April, 1999.