An Efficient Data Sharing Scheme for iSCSI-Based File Systems *

Dingshan He and David H.C. Du
Department of Computer Science and Engineering
DTC Intelligent Storage Consortium
University of Minnesota
{*he,du*}*@cs.umn.edu*

Abstract

iSCSI is an emerging transport protocol for transmitting SCSI storage I/O commands and data blocks over TCP/IP networks. It allows storage devices to be shared by multiple network hosts. A fundamental problem is how to enable consistent and efficient data sharing in iSCSI-based environments. In this paper, we propose a suite of data sharing schemes for iSCSI-based file systems and use ext2 as an example for implementation. Finally, we use simulations to verify the correctness of our designs and to study the performances.

1. Introduction

iSCSI [2] combines two popular and mature protocols in the data storage area and the network communication area - SCSI and TCP/IP. Small Computer System Interface (SCSI) enables host computer systems to perform I/O operations of data blocks with peripheral devices. iSCSI extends the connection of SCSI from traditional parallel cables, which could only stretch to several meters, to ubiquitous TCP/IP networks. iSCSI encapsulates and reliably delivers SCSI Protocol Data Units (PDUs) over TCP/IP networks.

iSCSI is expected to expand the coverage of System Area Networks (SAN). One major feature of SAN is to provide a shared pool of storage resources for multiple accessing hosts. As storage devices are not directly attached to any hosts, they can be easily shared. However, data sharing is going beyond sharing storage devices. With data sharing, a single piece of data could actually be shared by multiple clients. The integrity of data content has to be enforced. Therefore, concurrency control mechanism is necessary when multiple hosts could read/write a single piece of data concurrently.

*This project is partially supported by members of DTC Intelligent Storage Consortium (DISC) at UMN, and gifts from Intel and Cisco.

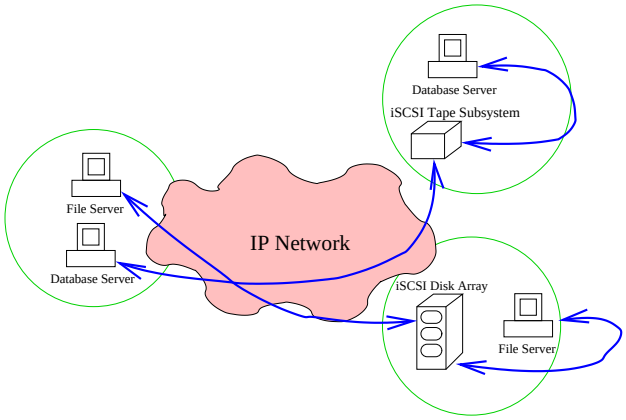

Figure 1. iSCSI network architecture scenario

When using iSCSI-based storage subsystems, application servers (as initiators) and their shared storage subsystems (as targets) could be connected by TCP/IP networks over long distances as depicted in Figure 1. Applications like file systems on the application servers are not aware of the fact that iSCSI-based storage subsystems are accessed and these storage subsystems are potentially shared with other application servers. Remote iSCSI devices are mounted at mounting points of application servers' local file systems and corresponding file system modules are loaded to manage data blocks. The file systems for mounted iSCSI devices are not adapted for iSCSI at all. Although these mounted file system modules in IP hosts are able to use their existing file system locks locally, multiple IP hosts still can not guarantee consistent data sharing. Therefore, our design is to coordinate the accesses of multiple iSCSI initiators.

Since iSCSI-based architectures could be deployed over WANs, latency introduced by large physical distance will be a severe restriction on performance. To improve performance, caching at application servers is necessary. In addition to hiding physical latency, caching at application

servers can also reduce load at iSCSI targets. However, it incurs the problem of cache consistency, which is the consistency between cached data on initiators (application servers) and data on targets (iSCSI-based storage subsystems). In our design, we will enforce strong consistency on cached data by using a callback cache similar to the Coda file system [6]. An iSCSI target keeps track of cached physical blocks on all connected iSCSI initiators. The iSCSI target forces the iSCSI initiators to discard their stale copy, when it is going to modify a physical block.

Metadata are also concurrently accessed, and cached by multiple initiators. However, a general solution for both metadata and normal file data would be inefficient since metadata and normal file data are different in terms of access patterns. We design different mechanisms for them separately as discussed in Section 3.1 and Section 3.2.

Finally, most existing SAN file systems, such as Lustre, only provide UNIX file sharing semantics. However, the UNIX file sharing semantics is not suitable for transactional execution of a sequence of operations. In the transaction file sharing semantics, there should be a consistent view of any involved data throughout the execution of a transaction. Deadlocks are possible, so detection and resolution of deadlocks are considered. In addition, rollback capability is required in case a transaction is unable to complete due to deadlocks.

The rest of this paper is organized as following. In Section 2, we will summarize related work. In Section 3, we are going to give an overview of our design and implementation. In Section 4, we present the simulation results over ns-2 simulator to study the performance of our design.

2. Related Work

The performance of iSCSI-based storage subsystems is studied by Lu and Du in [3]. It is shown that iSCSI storage with Gigabit connection could have performance very close to directly attached FC-AL storage, and iSCSI storage in campus network can achieve reasonable performance restricted by available bandwidth. Tang et al [5] have study performance of software-based iSCSI security using IPSec and SSL. In application side, researchers start to consider using iSCSI to implement hierarchical web proxy server and remote mirroring and backup.

Several distributed or clustered file systems have designed different data sharing schemes. GFS uses Device Lock mechanisms, which has been included in SCSI 3 specification as Dlock command. The IBM Distributed Lock Manager (DLM) is an implementation of the classic VAX Cluster locking semantics. Another simpler implementation is the DLM in Lustre with introduction of object concept. Our work is mainly different from these designs in network environment and locking granularity. We explicitly take possible long network latency into design consideration. Our locking granularity is at physical block level.

Table 1. Compatibility of metadata locks

	M_S	M_X
M_S		+
M_X	+	+

3. Design Overview

Our proposed scheme for data sharing in iSCSI-based file systems consists of two major parts. The first part is a concurrency control mechanism to coordinate multiple concurrent accesses for shared data. The second part is a cache consistency control mechanism. We assume that no single operation will involve data from more than one iSCSI target/LUN.

Roselli et al [1] found that the percentage of metadata reads is much larger than metadata writes. In order to take advantage of this fact, our design allows iSCSI initiators to cache shared locks (referred to as semi-preemptible shared locks [4]) on metadata objects.

On the other hand, normal data are organized into files. The access patterns for files are application dependent. Therefore, in addition to integrity of shared data, we are trying to design a general locking scheme to achieve high concurrency and to reduce maintenance costs of locks. Unfortunately, these two goals are conflicting with each other. Fine granularity of locks are preferred to maximize concurrency, meanwhile coarse granularity reduces number of locking requests and memory space used to maintain locks. Our design is trying to balance between these two conflicting goals. The detail of our hierarchical locking scheme will be discussed later in Section 3.2.

3.1. Locking scheme for metadata

The locks for metadata are applied on metadata objects. We define following 5 kinds of metadata objects: 1) directory files, 2) normal file inodes, 3) super block, 4) inode bitmap blocks, and 5) data-block bitmap blocks.

For each metadata object, there are two possible kinds of locks: M_S and M_X. M_S lock gives shared access to the requested metadata object. M_X lock gives exclusive access to the requested object. The compatibility of the locks is shown in Table 1, where '+' indicates incompatibility.

The M_S lock is semi-preemptible. An initiator is allowed to hold an M_S lock until the lock manager asks it to release that M_S lock. A callback mechanism is used to force the holder to release the lock, when some initiator requests M_X on the same metadata object. The holding

Table 2. Compatibility of hierarchical locks

	D_S	D_X	D_IS	D_IX
D_S		+		+
D_X	+	+	+	+
D_IS		+		
D_IX	+	+		

initiator only comply the request when it has no conflicting usage of the object.

An *M_X* lock for a metadata object is requested by an initiator when it is going to modify the metadata object. The *M_X* locks will not be cached at the initiators, so initiators have to contact targets every time. An *M_X* lock is always released immediately after the involved operations have finished.

Another possible operation on an *M_S* lock is to upgrade it to a *M_X* lock. This happens when an metadata object is first read and cached locally, and later a write request for the same metadata object arrives at the same initiator.

3.2. Locking scheme for normal data

Normal data are organized into files. In order to balance between high concurrency and high resource consumption, we design a two level hierarchical locking scheme. The upper level is an entire file and the lower level contains fix-sized block groups.

There are 4 possible locks applicable to nodes of such hierarchy.

- *D_IS*: intention shared access; allowing explicitly locking on descendant nodes in *D_S* or *D_IS* mode; no implicit locking to the sub-tree.
- *D_IX*: intention exclusive access; allowing explicitly locking on descendent nodes in *D_X*, *D_S*, *D_IX*, or *D_IS* mode; no implicit locking to the sub-tree.
- *D_S*: shared access; implicit *D_S* locking to all descendants.
- *D_X*: exclusive access; implicit *D_X* locking to all descendants.

Intention mode is used to indicate that compatible locks are going to be requested at finer level and thereby prevents incompatible non-intention locks (*D_S* and *D_X*) on upper level. Table 2 gives the compatibility of lock modes, where '+' means conflict.

Locks are always requested from root to leaves. On the other hand, locks should be reversely released from leaves to root. Intention modes are not applicable to leaf nodes.

3.3. Cache consistency control

Physical blocks fetched over networks are cached in iSCSI initiators' buffer caches. Buffer caches will be checked first when a physical block is requested. In order to avoid revalidating consistency of cached data blocks every time, we employ a mechanism based on callback. A callback record will be set up on iSCSI target side when a physical block is read out. When an iSCSI initiator is going to write a physical block, it first sends a SCSI CDB with write request. The iSCSI initiator will wait for a R2T response before starting transmitting data. When an iSCSI target receives a SCSI CDB with write request, it will check callback records for the requested physical blocks. If there are outstanding callback records, callback requests will be sent to those iSCSI initiators to ask them to purge the requested physical blocks out of their buffer caches. A iSCSI target will not send R2T response until it receives confirmations for all callback requests that it sent out.

3.4. Transaction file sharing semantics

In our design, file-accessing operations are grouped into transactions. Every transaction will be assigned a unique transaction id within the session between an iSCSI initiator and an iSCSI target.

Deadlocks are going to happen since we are supporting transactions. Due to the nature of random access of file data, it is difficult to prevent deadlocks from happening. Therefore, we use deadlock detection mechanism to detect deadlock when they have happened. When detecting a deadlock, a victim transaction will be selected and rolled back. The mechanism to detect deadlocks is to find a loop among transactions and locks.

3.5. Implementation components

Figure 2 shows the architecture overview of our implementations. We insert new modules in to both iSCSI initiators and iSCSI targets. Our implementation is based on the ext2 file system. The metadata and file data stored on storage devices are intact.

In iSCSI initiators, vfs is used between the upper level system call layer and the lower level iSCSI layer. we have inserted following two modules into the kernel of iSCSI initiators.

- **iSCSI client module** is actually a modified ext2 file system module. It manages transactions and various metadata and normal data locks.
- **Initiator cache manager module** manages a dedicated buffer cache for the iSCSI client module. It supports callback mechanism to assure cache consistency.

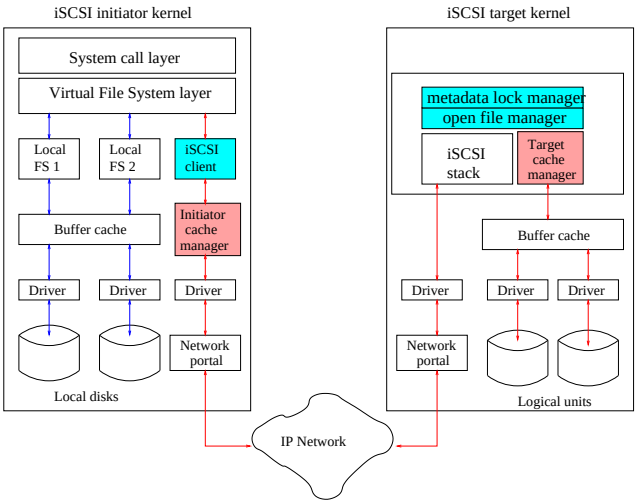

Figure 2. Overview of the architecture

An iSCSI target is responsible for maintaining active transactions, maintaining opened files, supporting callbacks for cached physical blocks, and so on. In iSCSI targets, we have inserted following three modules into iSCSI targets' kernel.

- **Metadata lock manager module** manages lock requests for metadata objects mentioned in Section 3.1. It handles deadlock detection and resolution caused by metadata locking requests.
- **File lock manager module** manages transaction requests, file open/close requests, and block group lock requests. It is also responsible for deadlock detection and resolution.
- **Target cache manager module** maintains callback records for physical blocks cached at iSCSI initiators. When there is a SCSI CDB with write command triggering callbacks, this module is also responsible for suspending the write command until all confirmations for callback requests are received.

4. Simulation Results

We use network simulator ns-2 to simulate network environments. Our schemes are implemented as modules running on host nodes in ns-2. The implementation is based on the ext2 file system as described in Section 3.5. We implement application-level iSCSI initiators and iSCSI targets, which contain components as presented in Section 3.5.

Table 3 shows the parameters we used for SCSI disk modules in all of our simulations. These parameters are following the specification of Seagate Cheetah 15K.3 family disk drives. However, no cache of disk modules is assumed in our simulations. Once a Read/Write command leaves the waiting queue on an iSCSI target for execution, the delay for Read/Write access one block of data is computed as $Delay_{Read/Write} = AverageLatency + AverageSeekTime_{Read/Write} + BlockSize/InternalTransferRate$.

In our simulations, we let iSCSI drivers on iSCSI initiators send a SCSI CDB for every physical block. iSCSI targets process requests, including locking requests and read/write requests, sequentially.

Table 3. Parameters of SCSI disk modules

Average Latency	2.0 msec
Average Read Seek Time	3.6 msec
Average Write Seek Time	3.9 msec
Internal Transfer Rate	62 Mbytes/sec[1]

4.1. Scheme Overhead

In order to investigate the overhead of our concurrency control and cache consistency scheme, we run simulations for single sequential writing of a single file. The operations are run as a single transaction as defined in Section 3.4. The file size we used in these simulations is 100MB. For writing of each physical block, we assume that each physical block should be read from its iSCSI target first and then written back. We have run this simulation for several different network configurations. Due to space limit, we only show the result that we get under one configuration. Figure 3 shows the composition of transaction time under this configuration. We use different size for physical blocks and different size for block groups defined in Section 3.2. For certain block group size, the total transaction time will decrease as the physical block size increasing. This is because larger physical block size requires less number of transmission and hence saves propagation delay. On the other hand, when using the same physical block size and varying block group size, the time spent on normal data locks decrease as we increase block group size. Larger group size would require less number of locking requests.

4.2. Effectiveness of Caching

In our design, physical blocks are cached in iSCSI initiators' buffer cache to improve performance. Our next set of simulations is trying to understand the effectiveness of such caching as the environment of iSCSI extending from LAN to WAN. In addition, we also try to investigate what are the major factors affecting effectiveness of caching and how.

In order to reflect file access patterns of real world, we use trace data generated from modified TPC-C benchmark

[1] average of 49 to 75 Mbytes/sec

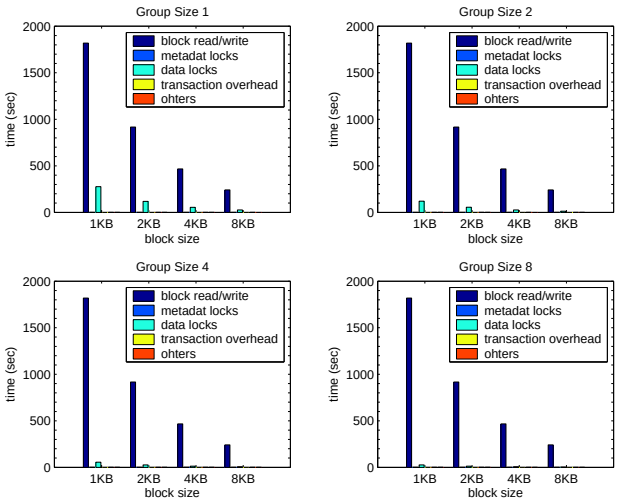

Figure 3. Composition of total transaction time for sequential access with bandwidth=100Mbps and latency=1msec

of the Transaction Processing Performance Council (TPC). The TPC-C benchmark is an OnLine Transaction Processing (OLTP) benchmark for database systems. We adapt this benchmark to generate file access traces. TPC-C benchmark involves a warehouse management database with 9 relation tables. We view each relation table as a file storing fix-sized records consecutively. TPC-C benchmark has 5 different transactions in SQL. For each transaction, we only trace the location of real reading or writing of records in the table files. We totally ignore additional database metadata such as index for keys in real database systems. For a transaction involving one or more than one table files, all table files are opened with proper mode before any reading or writing of data blocks.

Network bandwidth and latency are two potential factors that could affect effectiveness of iSCSI initiator caching. We set up a configuration of TPC-C with 4 warehouses. Each warehouse has 10 distinct districts. There are a number of customers registered to a district. Initially, we generate information to load the 9 table files. After loading initial data, the size of these 9 files range from over 380B to 120MB. We collect trace data from one client terminal, which is bond to one of the 4 warehouses. There are 200 transactions generated from this client terminal. The distribution of these transaction is 45% new order, 43% payment, 4% order status, 4% delivery, and 4% store level. In this set of simulation, we use 4K as physical block size and 1 as block group size. In Table 4, we show the comparison of with and without iSCSI initiator caching. The network bandwidth is 100Mbps and the network latency is 1msec. There are 724888 blocks accessed from cache. We also run simulations for other network configuration, which

Table 4. Comparison of w/ and w/o iSCSI initiator caching bandwidth=100Mbps latency=1msec

Time (sec)	w/ cache	w/o cache
reading blocks	29.5	5785.5
total	50.2	5806.3

show the performance will be intolerable without caching as iSCSI-based systems extend to WAN.

Our cache consistency scheme employs callback mechanism. A SCSI write command will be blocked at an iSCSI target until all response for callback requests are received. Therefore, access patterns for blocks will affect effectiveness of caching and performance of caching consistency control scheme. Still using the aforementioned TPC-C configuration, we run simulations three times with 2, 3, and 4 client terminals, respectively. Each simulation is run for 3600 seconds. In each simulation, only one client is repeatedly writing 10 physical blocks. For the other clients, they are repeatedly reading the same 10 physical blocks. Each reading and each writing is a single transaction, so no deadlock could happen. The result show that as the number of concurrent readers increases, the single writer spends more time on getting block from iSCSI target. This is caused by two reasons. First, with higher concurrency, writing command conflicts more with reading command. Secondly, with more reader, there is a higher chance that when a write command is sent, a copy of the requested block is cached on some other clients.

References

[1] D. Roselli, J. Lorch, and T. Anderson. A comparison of file system workloads. In *In Proceedings of USENIX Technical Conference*, pages 41–54, San Diego, California, June 2000.

[2] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner. *iSCSI*. IP Storage Working Group, January 2003.

[3] Y. Lu and D. Du. Performance study of iscsi-based storage subsystems. *IEEE Communication Magazine*, 41(8):76–82, 2003.

[4] R. Burns, R. Rees, and D. Long. Semi-preemptible locks for a distributed file system. In *In proceedings of the 2000 International Performance Computing and Communication Conference(IPCCC)*, Phoenix, AZ, February 2000.

[5] S. Tang, Y. Lu, and D. Du. Performance study of software-based iscsi security. In *Proceedings of First International IEEE Security in Storage Workshop*, pages 70–79, 2002.

[6] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.