A Failure Recovery Mechanism for Distributed Metadata Servers in DCFS2

Zhihua Fan, Jin Xiong, Jie Ma
National Research Center for Intelligent Computing Systems, China
Institute of Computing Technology, Chinese Academy of Sciences
fanzh@ncic.ac.cn

Abstract

Distributed metadata servers are required for cluster file system's scalability. However, how to distribute the file system metadata among multiple metadata servers and how to make the file system reliable in case of server failures are two difficult problem. In this paper, we present a journal-based failure-recovery mechanism for distributed metadata servers in the Dawning Cluster File System—DCFS2. The DCFS2 metadata protocol exploits a modified two-phase commit protocol which ensures consistent metadata updates on multiple metadata servers even in case of one server's failure. In this paper we focus on the logging policy and concurrent control policy for metadata updates, and the failure recovery policy. The DCFS2 metadata protocol is compared with the two phase commit protocol and some virtues are shown. Some results of performance experiments on our system are also presented.

1. Introduction

With the increasing popularity of cost-effective cluster computing in the recent years, various cluster file systems, such as Sistina's GFS[1], IBM's GPFS[2], and the Lustre file system[3] are developed. Traditional failure recovery mechanism for file systems is the fsck method, which checks all file system metadata and recovers them to a consistent state[5]. Fsck is robust, but is too time-consuming to be applied to large file systems over hundreds of gigabytes in volume. A better method is the journal file system, which logs all updates of the file system, including file data and metadata updates, and replay the log contents when recover. Some cluster file systems use the traditional journal technology, e.g. GFS. Some cluster file systems only log metadata updates, e.g. GPFS. Some cluster file systems use a journal file system to maintain metadata, e.g. Lustre. These cluster file systems either have no metadata server, e.g. GFS and GPFS, or a single metadata server, e.g. Lustre.

DCFS2 is an IP-SAN based cluster file system designed for the DAWNING-4000A super-server which is an ongoing research project at the NCIC in China. Unlike other cluster file systems, DCFS2 is based on multiple metadata servers to support scalability, and metadata are stored on each server's local file system.

In DCFS2, a metadata operation usually consists of a series of sub-operations. For instance, to create a regular DCFS2 file will include following steps: first, to add an entry in the parent directory; second, to allocate a free inode, and set its values; third, to modify the inode bitmap. The parent directory and the new allocated inode are not necessary on the same metadata server. So a metadata operation may need the collaboration of multiple servers. If an operation is completed by a single metadata server, it is called a local operation; else it is called a distributed operation. If a server breaks down, the metadata updates of a local operation may be partially lost if they are in the metadata cache, which leads to inconsistent metadata on disk. Similarly, a distributed operation may lead to inconsistent metadata among multiple servers. These two inconsistencies are called local inconsistency and distributed inconsistency respectively.

The DCFS2 metadata protocol uses the Write-Ahead-Log protocol[6] to settle the local inconsistency problem. As for distributed inconsistency, a modified two-phase commit protocol is used. Typically, atomic commitment protocols, e.g. two-phase commit protocol[7], are used to settle the distributed inconsistency problem in Database Systems. DCFS2 metadata protocol is carefully designed to make each metadata operation atomic, deadlock-free and rollbackable, Moreover the two-phase commit protocol are tailored to suit to DCFS2 metadata processing,

* This work is supported by the National '863' High-Tech Program of China (No. 2002AA1Z2102 and No. 2002AA104410).

IEEE COMPUTER SOCIETY

which reduces communication messages and synchronous logging I/O.

The reminder of this paper is organized as follows. The architecture and metadata management of DCFS2 is briefly introduced in section 2. The design considerations and implementation details of the failure recovery mechanism in DCFS2 are described in section 3. In section 4, the differences between DCFS2 metadata protocol and the two-phase commit protocol are discussed. In section 5, our experiment results are shown. Finally, some remarks are given in section 6.

2. Background

2.1. Dawning-4000A Super-server

DAWNING-4000A super-server is a high-performance Linux cluster of SMPs, which is being developed in NCIC China. It will consist of more than 512 Operon-based SMP nodes, and its peak speed will be more than 10Tflops. It will be also grid-enable with grid-enabling components, such as GridKVM, GridKey, GridRouter, GridView, GridGateway. DCFS2 is the cluster file system for DAWNING-4000A, which is a global file system shared among all cluster computing nodes with high performance.

2.2. DCFS2 Architecture

Different from its predecessor DCFS1[4], DCFS2 is an IP-SAN based cluster file system. DCFS2 is designed for high availability, high scalability, high performance and high manageability.

DCFS2's architecture is depicted as figure 1. It's based on a client-server design. File system metadata is managed separately from the data by the metadata servers. All file system clients access metadata from the metadata servers, and directly access data from the IP-SAN storage servers.

There are four components in DCFS2: one SSM server, a group of MGR servers, file system clients and one CSA daemon. SSM is a server for allocating and retrieving space on the IP-SAN storage servers for data. MGR servers manage the file system metadata. All clients have a uniform namespace and application can use system call to access the DCFS2 files. CSA is a graphic tool to be used for configuration and management of DCFS2, including notifying the failure of one server.

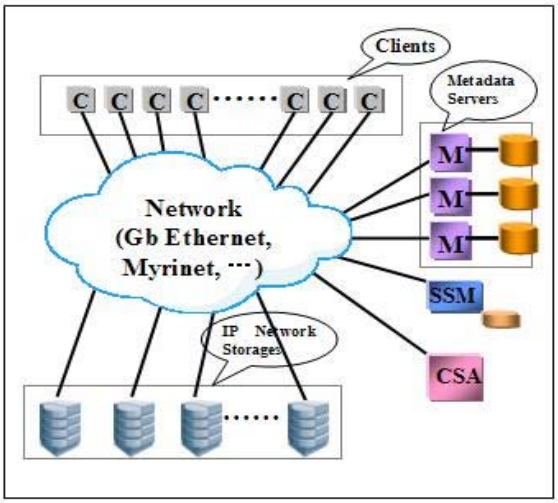

Figure1. The Architecture of DCFS2

2.3. Metadata Management in DCFS2

The namespace is organized as figure 2. The first level sub-directories of file system root are distributed to each metadata servers. When one metadata server is at high load, the rebalance mechanism will let its new created sub-directories to another low load server.

The DCFS2 file system's metadata is stored as files on the local file system on the metadata server. There are six kinds of metadata files: inode file, inode-bitmap file, directory file, mapping file, symlink file and super block file. The inode file stores the DCFS2's file or directory attributes such as the owner, file size, modify time etc. The inode-bitmap file indicates that the inodes

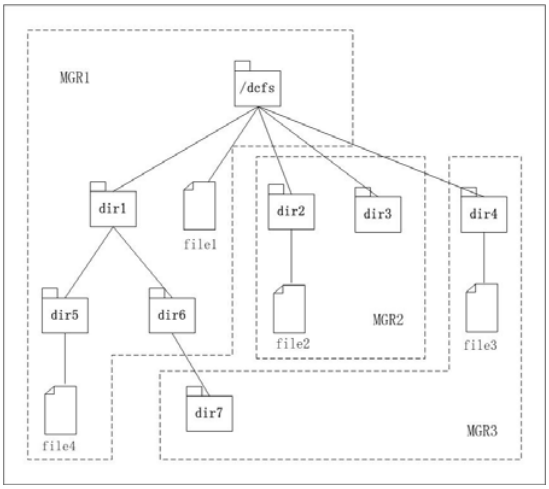

Figure2. The namespace of DCFS2

IEEE COMPUTER SOCIETY

in inode file are used or free. The dir file contains all entries in this dirctory. The mapping file records the position of the file data blocks on the IP-SAN devices. The symlink file records the symlink name. The super block file records the file system super block information.

In our system, there are four file system operations which are related to two metadata servers: MKDIR, RMDIR, LINK and RENAME. For example, as the figure 2 shows, the inode of the newly created directory "dir7" is not located on the same metadata server as the parent directory "dir6" which has the name entry in the directory file.

To achieve good performance, we use inode cache, inode bitmap cache and directory cache on metadata servers. The log buffers are separate to the metadata buffers. The Write-Ahead-Log protocol requires the log buffers are always written to disk before the metadata buffers. The logging hierarchy is showed as figure 3.

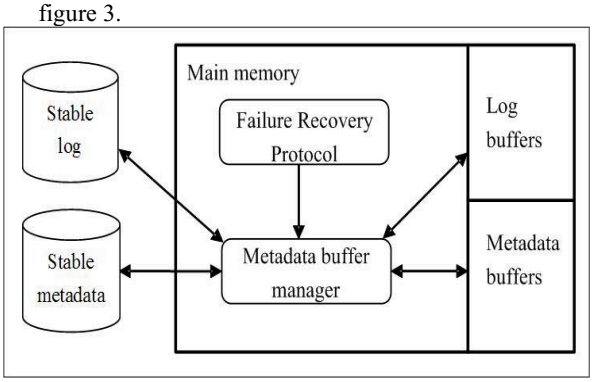

Figure3. The logging hierarchy

3. Metadata Consistency Protocol

3.1. Failure-free Protocol

As introduced in the previous section, we have 2 kinds of cases to lead to the metadata inconsistency after an unexpected server failure. One is that an operation modifies multiple metadata contents in the metadata cache at one MGR server and the cache is not written to disk as an integrity in case of unexpected MGR failure. The other one is that an operation modifies multiple metadata contents at multiple MGR servers and some updates are lost on one of these servers.

The first case is easy to be settled. Just to use the Write-Ahead-Log protocol can settle it. If the log is recorded to disk as an integrity, then we can redo it on the metadata files. Of course, our log can redo "repeatly", thus whether the metadata file is modified already or not.

In DCFS2, there are four kinds of operations which can update metadata on multiple metadata servers: MKDIR, RMDIR, LINK, RENAME. MKDIR operation is executed on the server which stores the parent directory's inode information and directory entries and on the server which stores the new directory's inode information and directory entries. The RMDIR operation is executed on two servers similarly. The LINK operation is executed on the server which stores the new parent directory's inode information and directory entries and on the server which stored the original object's inode information. The RENAME operation is special, for it may be executed on more than two metadata servers. For instance, we rename one "dir1" from directory "pardir1" to directory "pardir2" with the new name "dir2". The directory entry "dir1" is removed from the directory "pardir1" on server A; the directory entry "dir2" is added to the directory "pardir2" on server B; the renamed directory needs to modify its directory entry ".." on server C. These three servers are not necessarily the same.

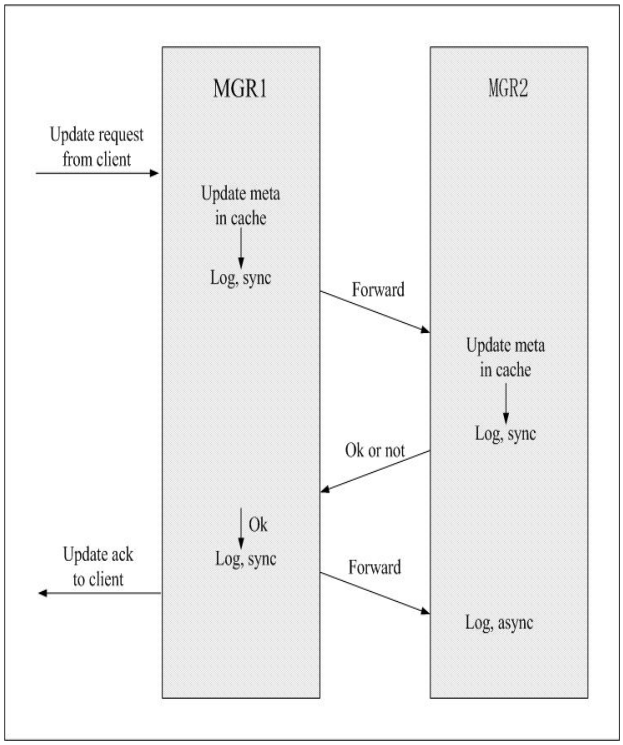

Figure4. The failure free protocol between two MGR servers at the commit case

IEEE COMPUTER SOCIETY

In DCFS2, this kind of operation is restricted to be executed on no more than two metadata servers by ignoring the update of directory entry "..". It's not harmful because the DCFS2 clients can give the right directory entry of ".." by itself.

The uniform metadata update protocol on two metadata servers is depicted as figure 4.

On each server, the update is done in cache first, then the update information is written to log synchronously. The updated metadata can be written to disk later when the flush thread is triggered periodically. Figure 4 only shows the commit case. Actually, if MGR2 can't complete the request successfully due to intension of memory or deadlock avoidance etc, MGR1 should undo previous update and log the failure information after it receives the message from MGR2. The last forward message is necessary because MGR2 needs to know whether MGR1 has received the commit information and MGR2 can forget its previous log on itself after the last forward message is received.

3.2. Recovery Protocol

In DCFS2, the failure recovery mechanism only considers to settle the single point of failure. The recovery protocol is under the following assumptions:

1. At most one metadata server crashes once in failure. If more than two metadata servers crash, we have to stop the DCFS2 and use the fsck utility to recover the metadata consistency.
2. The message is not delivered or lost on the way only due to one metadata server's crash. If a network partition appears, we only expect that it lasts a short time, thus our communication protocol can ensure the message can be received when the network is good again.

Therefore, following only gives the recovery process in various failure cases of one metadata server:

1. MGR1 fails before its first log is written to disk. MGR1 can not see anything from the log when it restarts, so we need do nothing and the metadata update in cache of the request is simply lost for that the Write-Ahead-Log protocol ensures the metadata update is not written to disk.
2. MGR1 fails after its first log is written to disk but before its second log is written to disk. MGR1 sends a negotiation request to MGR2. If MGR2 tells MGR1 that MGR2 is to commit, then MGR1 redo according to its first log; otherwise, if MGR2 tells that it has no information or is to abort, MGR1 undo according to its log. The undo can be executed repeatedly, so it doesn't care about that whether the metadata update is written to disk.
3. MGR1 fails after its second log is written to disk. MGR1 just needs to redo or undo according to the decision information of the second log. After MGR1 recovers completely, MGR2 check whether it receives the last message. If not, it needs to write its second log to end this request.
4. MGR2 fails before its first log is written to disk. After MGR2 restarts and finishes its own recovery, MGR2 sends a message to notice MGR1 that MGR2 has finished local recovery, then MGR1 finds that this request is not done completely and it just needs to undo it and writes its second log to end.
5. MGR2 fails after its first log is written to disk but before its second log is written to disk. MGR2 needs either to redo according to its log or to do nothing if the log records that it is to abort. MGR2 also sends the decision of commit or abort to MGR1 even if it has sent already before the failure. MGR1 does nothing if the decision is received before; MGR1 executes redo or undo according to the message if the decision is not received before.
6. MGR2 fails after its second log is written to disk. MGR2 does like the fifth case except that it doesn't need to send the message of decision to MGR1 anymore.

3.3. Concurrency Control

When one server forward one request to another server after it has done the metadata update on itself, it can do another request from clients or other metadata servers. So we should handle the concurrency control problem.

In DCFS2, we only lock the directory before we add or delete directory entries in one directory. The read-only request and the update request which does not change the namespace structure such as CHMOD, don't need to acquire the lock.

The problem is how to detect the deadlock. For instance, one process renames the file A from dir1 on MGR1 to dir2 on MGR2; at the mean while, another process renames the file B from dir2 to dir1. The former locks dir1 on MGR1 and waits for the lock of dir2 on MGR2, and the later locks dir2 on MGR2 and waits for the lock of dir1 on MGR1. It causes a deadlock. In DCFS2, we don't detect this kind of deadlock but use a simple method to prevent deadlock

IEEE COMPUTER SOCIETY

to occur. If one request gets a lock on one MGR and needs to wait for another lock on another MGR, this request just returns unsuccessfully. We don't expect that this kind of cases is always to be occurred.

4. Comparison with Two Phase Commit Protocol

The two-phase commit protocol is a common used atomic commit protocol and it's suitable to more than two participants. So these commit protocol is not always good in some special cases. In DCFS2, there are as most two participants involved in one request. Here we compare the protocol in DCFS2 to two-phase commit protocol.

Figure 5 depicts the flow chart as we use two-phase commit protocol in our file system. We use one of the participants as the coordinator. First, the coordinator needs to write a start log synchronously. Then it informs two servers including itself to prepare to commit. When both of them give the decision to commit, it gives the global decision to commit, writes to log synchronously and notify to participants. When it receives the acknowledgement, it writes an end log. This log can be written asynchronously.

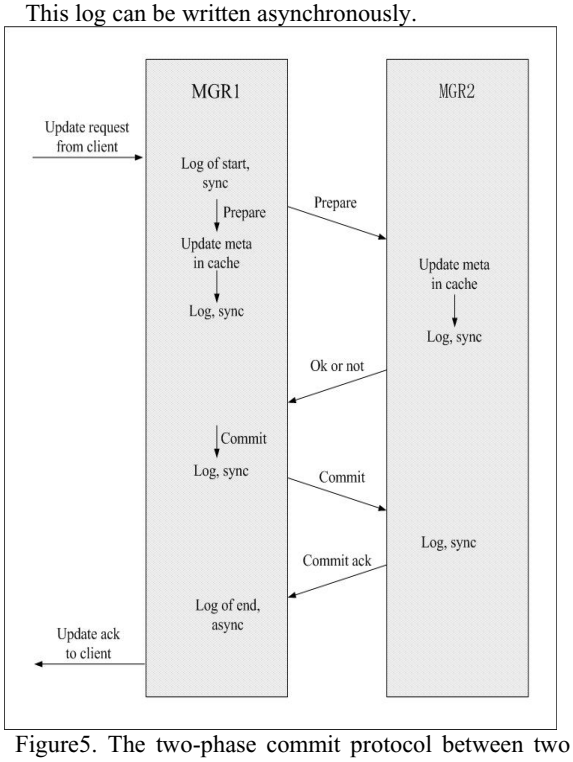

Figure5. The two-phase commit protocol between two metadata servers

First, as a coordinator, it needs to write synchronous log before it notify participants to prepare to commit. Second, when the other participant gives the decision to commit, it doesn't know the global decision, and it needs to wait for the global decision. Third, the coordinator can't forget all information when it writes log of the global decision because the participant may need to ask it after a failure. So the coordinator needs to wait for the last acknowledgement.

In DCFS2's protocol, when MGR2 gives the decision to commit or abort, it knows the global decision. And when MGR1 receives MGR2's decision, MGR1 just needs to notice MGR2 that MGR2 can forget the decision information (MGR1 will not ask MGR2 the decision any more), and MGR1 doesn't need to wait for the response from MGR2 any more.

Table 1 gives a comparison between our protocol in DCFS2 and two-phase commit protocol. Our protocol can save one synchronous log write, one asynchronous log write and 2 messages in the critical path.

Table 1 Comparison with two phase commit protocol

	Total Log Write (sync, async)	Log Write in Critical Path (sync, async)	Total Messages	Messages in Critical Path
DCFS2	**(3,** 1)	**(3, 0)**	**3**	**2**
2PC	(5, 1)	(4, 1)	4	4

5. Performance Experiments

Our current testing platform is four Linux workstations. Each node runs Linux 2.4.18-3, with one PIII 866 CPU, one 1G memory and one 9G SCSI disk. These nodes are interconnected through 100M Ethernet. The CSA and SSM daemon run on one node, two MGR daemons run on another two nodes, and on the left node runs the Client daemon.

First, we compare the distributed operation's performance between the protocol in DCFS2 and the two-phase commit protocol. The performance without writing log in DCFS2 is also given, thus we can see how much of the performance enhancement while we don't take care of the fault tolerant function. Here we use the rename operation to execute on two metadata servers. The total count of rename is various from 128 to 2048. The result in figure 6 shows that the time in our protocol is only 76% of the two-phase commit protocol. And more than half of the time is spent on writing log.

Secondly, because the distributed operation doesn't always appear in DCFS2, we want to see how the ratio of distributed operations effects on the performance.

IEEE COMPUTER SOCIETY

Our testing program is to create one directory tree with that the depth is 5 and every directory has 4 sub-directories and 8 files in it. The total number of directories created is 1364 and the total number of files is 2728. During this process, there are only two directories created with distributed MKDIR. So we inject some distributed operations such as rename with various ratios. Figure7 shows that the 10 percent of distributed operations spend nearly 30 percent of the total time. Moreover, in the local operations the pathname even extends to the 5th depth and the pathname lookup takes much overhead, while the injected distributed operations are all in the second depth. So the distributed operations are not expected to be much in DCFS2.

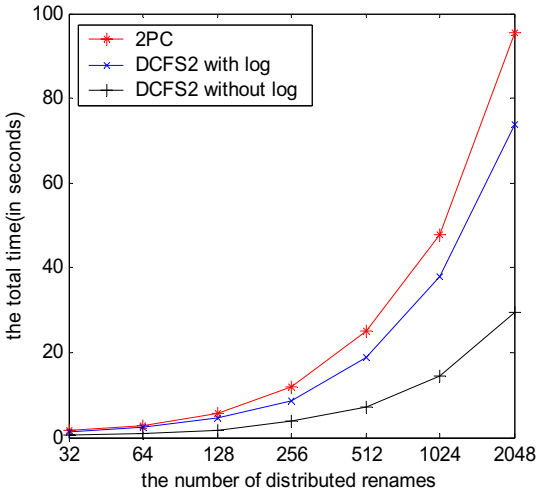

Figure6. Performance of distributed rename operation

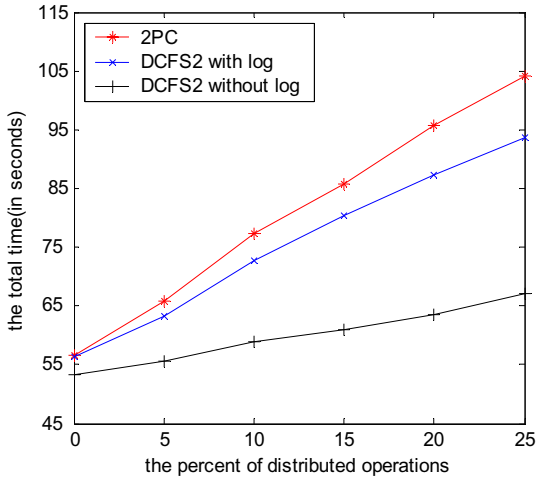

Figure7. Performance of creating a directory tree with various ratios of distributed operations

6. Conclusions and Future Work

This paper mainly introduces the failure recovery mechanism of DCFS2, including the failure-free protocol, the recovery protocol and the concurrency control. It also compares the commit protocol between two metadata servers in DCFS2 with the two-phase commit protocol, and some virtues of our protocol are presented such as saving communication messages and synchronous logging.

The experiment results verify that the performance of our protocol in DCFS2 is better than the two-phase commit protocol. And we can see from the results that the overhead of distributed operations is very high. It indicates that we should reduce such operations as possible in DCFS2 metadata protocol to improve performance. More careful performance analysis is needed to see the performance impact of the logging.

Currently, if the metadata server stores the information of the root of DCFS2 file system crashes, we need to wait this server to recover. We plan to backup the root information on another server thus the clients can access metadata through root on other normal servers. We also plan to stores the metadata and log on network storage. Thus when one server can't restart to rejoin, its metadata and log can be managed by a new metadata server.

References

[1] K. W. Preslan, A. P. Barry, J. E. Brassow, R. Cattelan, A. Manthei, E. Nygaard, S. V. Oort, D. Teigland, M. Tilstra, M. T. O. and G. M. Erickson, and M. Agarwal. *Implementing journalling in a linux shared disk file system.* In Proc. of the Eighth NASA Goddard Conference on Mass Storage Systems and Technologies, 2000.

[2] F. Schmuck and R. Haskin. *GPFS: a shared-disk file system for large computing clusters.* In Proceedings of FAST '02, January 2002.

[3] *Lustre: A Scalable, High-Performance File System*, White Paper, Cluster File Systems, Inc., Nov 11, 2002.

[4] Jin Xiong, Sining Wu, Dan Meng, Ninghui Sun, Guojie Li, *Design and Performance of the Dawning Cluster File System,* CLUSTER'03, December 01-04, 2003

[5] Marshall Kirk McKusick, *Fsck - The UNIX File System Check Program.* 4.2BSD, Computer Systems Research Group, UC Berkeley, 1985.

[6] K. Rothermel, C. Mohan. *ARIES/NT: A Recovery Method Based on Write Ahead Logging for Nested*

IEEE COMPUTER SOCIETY

Transactions. In P. M. G. Apers, G. Wiederhold (Eds.). Proc. 15th VLDB : 337-346, Amsterdam 1989.

[7] Gray, J. and Reuter, A., *Transaction Processing: Concepts and Techniques,* Morgan Kaufman, 1993.

[8] Mohan, C., Lindsay, B., and Obermarck, R., *Transaction Management in the R* Distributed Data Base Management System.* ACM Transactions on Database Systems, Vol. 11(4): pp. 378-396, 1986.

[9] M. T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems,* Prentice Hall, 1991.

[10] M. Ji, E. Felten, R. Wang, and J. P. Singh. *Archipelago: an island-based file system for highly available and scalable Internet services.* In USENIX Windows Systems Symposium, August 2000.

[11] http://www.lustre.org

IEEE COMPUTER SOCIETY