Validation Rules based on Concurrency Control (VRCC) For Network File System

Hau Dim Vung, Dr. Khin Than Kyi
Computer University, Kalay
hdimvung@gmail.com, khinthankyi@gmail.com

Abstract

File systems are central parts of any modern operation system, and are expected to be both fast and exceedingly reliable. Network file System (NFS) is one of the components in the distributed file system. This paper implements the system to manage file concurrency control using the Optimistic Concurrency Control method and Validation rules in NFS. In this system, Validation by using the read-write conflict rules to ensure that the scheduling of a particular transaction is serially equivalent with respect to all other overlapping transaction. It includes collection of processes, possibly running on different operating system and machines, sharing common file system and up-to-date file to make read and write. To assist in performing validation, each transaction is assigned a transaction when it enters the validation phase. If the transaction is validated and completes checked and is aborted, or if the transaction is read only, the number is released for reassignment.

Keywords: Distributed File System, Network File System, Optimistic Concurrency Control

1. Introduction

A key component of any distributed system is the file system. File systems are central parts of any modern operating system, and are expected to be both fast and exceedingly reliable. The file system typically requires that filenames adhere to certain conventions and that files possess certain attributes which are interpreted in a specific way.

Distributed file systems whose primary purpose is to emulate the functionality of a non-distributed file system for client programs running on multiple remote computers. A well-designed file service provides access to files stored at a server with performance and reliability similar to, and in some cases better than, files stored on local disks.

Network file System (NFS) is one of distributed file system. It can act both client and server. It allows every machine to be a client and server at the same time and allows for fast, seamless sharing of files across a network.

Concurrency Control is process of managing simultaneous execution of transaction in a shared database, to ensure the serializability of other clients simultaneously accessing or changing the same file. There are two main categories of concurrency control mechanisms: Optimistic and Pessimistic.

This paper focus on Optimistic Concurrency Control (OCC) method associated with Validation rules. Optimistic Concurrency Control (OCC) allows transactions to proceed without restriction until the close transaction. The up-to-date information of the file is provided by using OCC. When one transaction performs write operation, the others can't do reads operations. So concurrency problem are already controlled.

The rest of this paper is organized as follow. Section (2), represent the related work. In section (3) explain Distributed File System of background knowledge of this paper. We show Network File System of method of this paper in section (4). The next section (5), also present Concurrency Control and Optimistic Concurrency Control in section (6). Both section (7) and (8), explain the propose architecture and algorithm used in proposed architecture respectively. In section (9), design the process of the proposed Validation based on Concurrency Control (VRCC) proposed. Finally we conclude this paper in section (10).

2. Related Work

Prof. Dr. J. Becker, el. at [2] considered optimistic concurrency control. This paper design alternatives were presented which are superior to the original approach in several aspects and then a special validation scheme for read transactions was introduced. Then the integration of version into the optimistic approach was discussed briefly. Finally, it presented a solution for the starvation problem. The proposed algorithm is easy to integrate because it is only based on optimistic concepts. The main advantage of the proposed solution is that it distinguishes between serious and non-series

conflicts and only rolls back a transaction in case of a serious conflict.

A. Adya, R. Gruber, B. Liskov and U. Maheshwari [1] described an efficient optimistic concurrency control scheme for use in distributed database systems in which objects are cached and manipulated at client machines while persistent storage and transactional support are provided by servers. The scheme provided both serializability and external consistency for committed transactions; it used loosely synchronized clocks to achieve global serialization. The paper also presented a simulation study that compares the scheme to adaptive callback locking, the best concurrency control scheme for client-server object-oriented database systems studied to date. The study showed that our scheme outperforms adaptive callback locking for low to moderate contention workloads, and scaled better with the number of clients.

H. T. KUNG and J. T. ROBINSON [3] has been done on locking approaches to concurrency control in practice two control mechanisms are used: locking and backup. To investigate solutions to concurrency control, backup mechanism is relied almost entirely on it. The optimistic methods presented in this paper are orthogonal to locking methods. Transactions are controlled by having them wait at certain points. Serial equivalence can be proved by partially ordering the transactions by first access time for each object. The major difficulty in locking approaches is deadlock, which can be solved by using backup. They also presented two families of concurrency controls with varying degrees of concurrency. These methods may well be superior to locking methods for systems where transaction conflict is highly unlikely.

3. Distributed File System

In the distributed file system, the *file service* specifies what the file system offers to its clients and describes the primitives available, what parameters they take and what the actions they perform. File service enables programs to store and access remote files exactly as they do local ones, allowing users to access their files from any computers in an intranet. The *file server*, that helps implement the file service, is a process that runs on some machine. The *persistent storage* at servers reduces the need for local disk storage and enables economics to be made in the management and archiving of the persistent data owned by an organization. *Other services*, such as the name service, the user authentication service and the print service, can be more easily implemented when they can call upon the file service to meet their needs for persistent storages [3].

Distributed file systems have become popular in a variety of environments. They facilitate the sharing of information between users, increase scalability, mobility, and maintainability. There are a various file systems. These are Andrew file system (AFS), Sun's Network file system (NFS). NFS is widely used than AFS. So we focus on NFS in this paper.

4. Sun's Network File System

NFS provided transparent access to remote files for client programs running on UNIX, Linux and other systems. Typically, every computer has NFS client and server modules. Each computer in an NFS network can act as both a client and a server, and the files at every machine can be made available for remote access by other machines [3].

An important goal of NFS is to achieve a high level of support for hardware and operating system heterogeneity. All implementations of NFS support the NFS protocol - a set of remote procedure calls that provide the means for clients to perform operations on a remote file store. The NFS protocol is operation-system-independent.

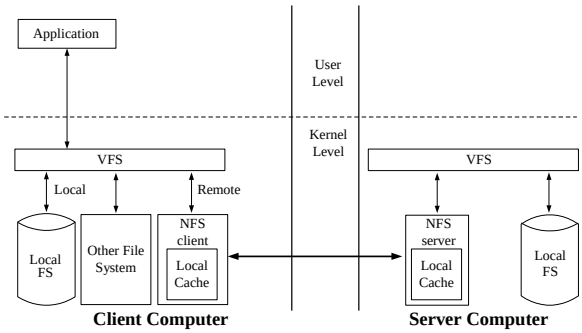

Figure 1. Architecture of NFS

.

5. Concurrency Control

Concurrency control is the management of contention for data resources. A concurrency control scheme is considered *pessimistic* when it locks a given resource early in the data-access transaction and does not release it until the transaction is closed and *optimistic* when locks are acquired and released over a very short period of time at the end of a transaction.

Pessimistic and optimistic concurrency schemes require different transaction isolation levels. Enterprise beans that participate in the same transaction and require different concurrency control

schemes cannot operate on the same underlying data connection.

In this paper, we focus on the Optimistic Concurrency Control method. So we briefly explain it.

6. Optimistic Concurrency Control

Under an optimistic scheme, locks are obtained immediately before a read operation and released immediately after-wards. Update locks are obtained immediately before an update operation and held until the end of the transaction.

The objective of optimistic concurrency is to minimize the time over which a given resource would be unavailable for use by other transactions. This is especially important with long-running transactions, which under a pessimistic scheme would lock up a resource for unacceptably long periods of time.

Optimistic concurrency control divides a transaction into a read phase, a validation phase, and a writing phase. During (a) the read phase, a transaction performs writes on local buffers and no checking takes place, (b) validation phase, the system does synchronization checking, and (c) the write phase, the local writes are made global [3].

It assigns each transaction a unique timestamp at the end of its read phase. A transaction T_i is validated if one of the following conditions can be established for all transactions T_j with later timestamps:

- Transaction T_i completes its write phase before transaction T_j begins its read phase.
- Transaction T_j does not read any of the items written by T_i and transaction T_i finishes its write phase before transaction T_j begins its write phase.
- Transaction T_j does not read or write any items written by T_i.

T	T_i	*Rule*
write	*read*	1 T_i must not read objects written by *T*
read	*write*	2 *T* must not read objects written by T_i
write	*write*	3 T_i must not write objects written by *T* and *T* Must not write objects written by T_i

Figure 2. Validation of transaction (T)

7. The VRCC System Architecture

This paper proposed a system for Network File System (NFS). The system is constructed by composed of five components. These are

1) Virtual File System (VFS)
2) Local File System (Local FS)
3) NFS Client
4) NFS Server
5) VRCC mechanism

Since NFS provides access transparency, user programs can issue file operations for local or remote files without distinction.

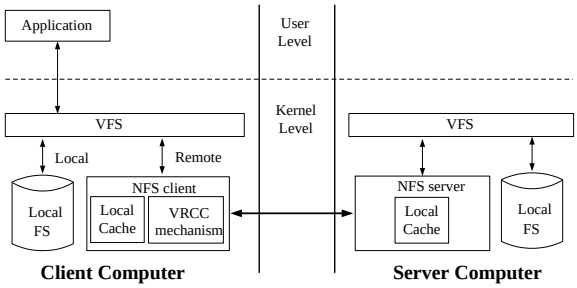

Figure 3. Architecture of the proposed system

7.1. Virtual File System

Virtual File System distinguishes between local and remote files and translates file identifiers used by relevant file systems. In addition, VFS keeps track of the file systems that are currently available both locally and remotely, and it passes each request to the appropriate local system module.

.

7.2. Local File System

Local File System checks the rights allowed for the user's identify in the access control list against the mode of access requested in the open system call. If the rights match the mode, the file is opened and the mode is recorded in the open file state information.

7.3. NFS client module

NFS client module cooperates with virtual file system. The NFS clients transfer blocks of files to and from the server and caching the blocks in the local memory whenever possible. It shares the same buffer cache that is used by the local input-output system. But since several clients in different host machines may simultaneously access the same

remote file, a new and significant cache consistency problem arises.

7.4. NFS server module

NFS server module resides in the kernel on each computer that acts as an NFS server. The NFS client and server modules communicate using remote procedure calling.

7.5. VRCC mechanism

Validation-based Concurrency Control (VRCC) mechanism is controlled concurrency control protocol using the technique called Validation rules. This protocol is provided with timestamp system. The controlled algorithms based on the Concurrency Control method associated with Validation rules are explained in detail the section (8).

8. Algorithm used in Proposed Architecture

A concurrency timestamp system is a higher typed communication object, and has been shown to be a powerful tool to solve many concurrency control problems. One of the requirements of the system is to determine the temporal order in which the objects are written. The processes assign timestamp to their respective transactions in such a way that the timestamp reflect the real-time order in which they are written to. These systems must support three algorithms, namely read operation, write/update operation and VRCC procedure algorithm. A read, write/update operation algorithm generate a new timestamp to each transaction and performs the concurrency control procedure, and a VRCC procedure algorithm enables a process to perform the operation in which all the users are requested. We are concerned with those systems where operations can be executed *concurrently*, in an overlapped fashion.

Algorithm for read operation

1) Broadcast input (Filename, read operation) to all clients
2) Wait for reply message
3) If reply message is ‘read’ then Call the VRCC procedure
4) Else Input (another file)
5) End if

The read operation algorithm provides the reading file to and from the other processes. This includes initializing a process by first sending an input (line 1). After a reply is received, it checks the reply message. If both the reply message and the input are read, the VRCC procedure is performed (line 3). If the user’s operation is not correct, it enter the another file as input (line 4).

Algorithm for write/update operation

1) Broadcast input (Filename, write/update operation) to all clients
2) Wait for reply message
3) If reply message is ‘valid’ then Call the VRCC procedure
4) Else Input (another file)
5) End if

Similarly, write/update operation algorithm provides the reading file to and from the other processes. This includes initializing a process by first sending an input (line 1). After a reply is received, it checks the reply message. If both the reply message and the input are read, the VRCC procedure is performed (line 3). If the user’s operation is not correct, it enter the another file as input (line 4).

Algorithm for VRCC procedure

1) Check operation after receiving the input
2) If operation is read operation then
   Check Filename exist in the local cache
   If Filename exist in the local cache then Read File
   Else Read File from the server
   End if
3) Else
   Check Filename exist in the local cache
   If Filename exist in the local cache then
      Write File as the updated file in the local cache
      Broadcast Filename with updated time to the server
   Else
      Read File from the other node
      Write File as the updated file
      Store File in the local cache
      Broadcast Filename with updated time to the server
   End if
4) End if

The VRCC procedure algorithm performs the actions taken by read, write/update algorithms which can be collected the information from the inputs. The collected information will be sent with the relative timestamp. After receiving the input, it checks the operation. If the operation is read, it also checks the filename. If the filename exist in the local cache, that file can be read from the user. If not, that file is read from the server. Similarly if the operation is write/update and the requested filename exist in the local cache; it processes the same operation as the read operation processes. If the operation is write/update and the requested filename doesn't exist in the local cache, it read that file from other node, write/update into it and temporally store as updated file in the local cache. And then broadcast it with updated time to the server.

This system uses the Concurrency Control Method. In client computer, the functions as well as the request messages such as read, write and delete files are controlled by the server. Therefore, these request messages are sent to the server. When sending to the server, the server calculates the access times. And then client computer that sent to server early request messages to reply. When experimenting three computers (one for server and two for clients) with this system, access time of the computer A is 1.2463ms and that of computer B is 1.2464ms. That is why the difference between the accesses of the two computers is 0.0001ms. It is too small time slice. So these access times (0.0001ms) is a negligible amount. Therefore it can be said that the message is reached simultaneously to both computers. It means that it can perform the concurrency for the above reasons; this system certainly does Concurrency

9. The process of the proposed Validation based on Concurrency Control (VRCC)

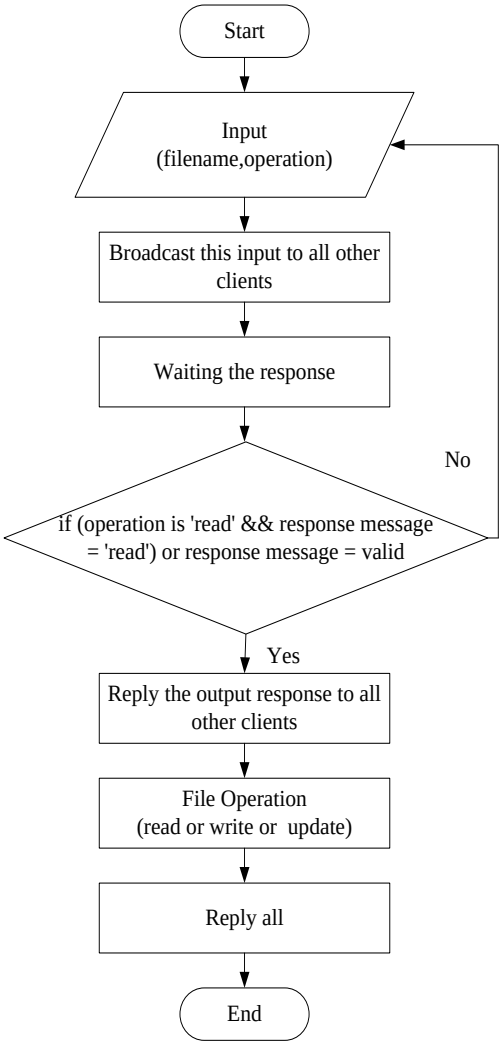

Figure 4. System Design for VRCC

10. Conclusion

Distributed File System is responsible for the organization, storage, retrieval, naming, sharing and protection of files. File systems are designed to store and manage large numbers of files, with facilities for creating, naming and deleting files. The need for concurrency control for access to shared data in many applications is widely accepted. So we contribute the algorithm to control the concurrency.

The contribution towards the development of this algorithm is to increase disk space availability by avoiding duplication. The Transparent access can be done into the files. If the client or server is crashed, this architecture can be recovered again. We also expect our synchronized distributed file system support the needed facilities for simultaneous file access from the clients in the network file system. By using simulation theory, it can be observed that the result of DFS is efficient. But we don't solve the consistency and security problems in this algorithm.

11. References

[1] A. Adya, R. Gruber, B. Liskov and U. Maheshwari, " Efficient Optimistic Concurr-ency Control Using Loosely Synchronized Clocks", Proceedings of the ACM SIGMOD International Conference on Management of Data, San Jose, CA, May 1995.

[2] Prof. Dr. J. Becker, Prof. Dr. H. L. Grob, Prof. Dr. K. Kurbel, Prof. Dr. U. Müller-Funk, Prof. Dr. R. Unland, Prof. Dr. G. Vossen, "Optimistic Concurrency Control Revisited", Institut für Wirtschafts-informatik der Westfälischen Wilhelms-Universität Münster, Grevener Str. März 1994.

[3] H. T. KUNG and J. T. ROBINSON, "On Optimistic Methods for Concurrency Control", ACM Transactions on Database Systems, Vol. 6, No. 2, June 1981, Pages 213-226.