Achieving Efficient Pathname LOOKUP in File Server Group

Feng-jung Liu[1] Chu-sing Yang[2] Yao-kuei Lee[3]

[1] Dept. of Management Information Systems, Tajen University, 907 Pingtung, Taiwan
[2] Dept. of Computer Science and Information Engineering, National Cheng Kung University, Tainan, 701, Taiwan
[3] Dept. of Marketing & Distribution Management, Tajen University, 907 Pingtung, Taiwan

Abstract

With the increasing growth of Internet users and the Web contents, it has led to much attention on scalability and availability of file system. Hence the ways to improve the reliability and availability of system, to achieve the expected reduction in operational expenses and to reduce the operations of management of system have become essential issues. In FSG system, it improved the reliability of file system through replication to handle the effects of failures. An efficient consistency control protocol is previously proposed to ensure the consistency among replicas. In this paper, we leveraged the concept of intermediate file handle to cover the heterogeneity of file system and designed a mechanism, named Multi-component LOOKUP to solve the inefficiency problem of full pathname LOOKUP, to reduce the number of RPC requests going across the network and to allow a client to resolve a full path name in one operation. Above all, simplicity is our main design consideration.

Keywords: File Server Group, file handle, Multi-component LOOKUP.

1. Introduction

A basic technique for improving reliability of file system is to mask the effects of failures by replication. Many distributed File Systems, such as intermezzo[1], Coda[2], Deceit[3], RNFS[4], Pangaea[5] and FSG[6,8,9,16] implemented reliable file system services through software replication approach Within them, the FSG, RNFS and Deceit are NFS-based systems. JetFile[7], Coda and FSG are the instances of multicast-based file systems. Due that NFS client accesses a file using a file handle obtained from the server as a result of a LOOKUP operation. Thus, how to lookup the pathname efficiently becomes an essential issue for improving the efficiency.

Many reports such as NFSv4[13] and WebNFS[14, 15] have been proposed to solve the inefficiency problem of full pathname LOOKUP. It is named multi-component LOOKUP, MCL. With such a mechanism, file servers allow a client to resolve a full path name in one operation to reduce the number of RPC calls going in the network. Normally the NFS (v2 and v3) LOOKUP request takes a directory *fHandle* along with a directory name, and returns the *fHandle* of each subdirectory name. If a client needs to evaluate a pathname that contains a sequence of components, then beginning with the directory *fHandle* of the first component it must issue a series of LOOKUP requests one component at a time. For instance, on evaluation of the path "a/b/c", the system will generate separate LOOKUP requests for each component of the pathname "a", "b", and "c". The server is expected to evaluate the entire pathname and return a *fHandle* for the final component “c”.

2. Overview of File Server Group

The implementation of the file server group, FSG [6,8,9,16] is based on NFS and interacts by underlying IP multicasting. In designing system, the collection of replicated servers is treated as a group. Each group is assigned a group IP address. The IP address will be used by the underlying multicast protocol to deliver messages to all servers in this group. With multicast communication [10,11,12] it is possible to implement distributed systems without any explicit need to know the precise location of data. Instead, peers find each other by communicating over agreed upon communication channels. To find a particular data item, it is sufficient to make a request for the data on the agreed upon multicast channel and any node that holds a replica of the data item may respond to the request. This property makes multicast communication an excellent choice for building a system that replicates data.

As shown in Fig. 1, it illustrates the system model of FSG. A user on the client machines uses the "mount" command to connect to the sever group as the general UNIX mount command. The only difference between a UNIX mount and the proposed “mount” command is that the "host:pathname" parameter is replaced by "multicast IP address:pathname". The nodes in this model are not limited to be homogeneous processors.

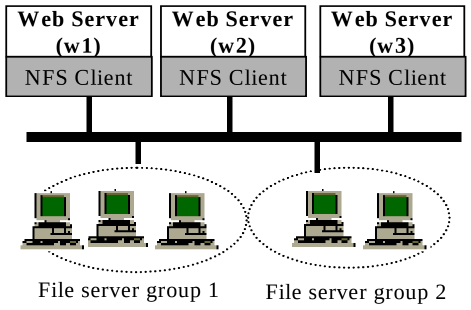


Fig. 1: System Model.

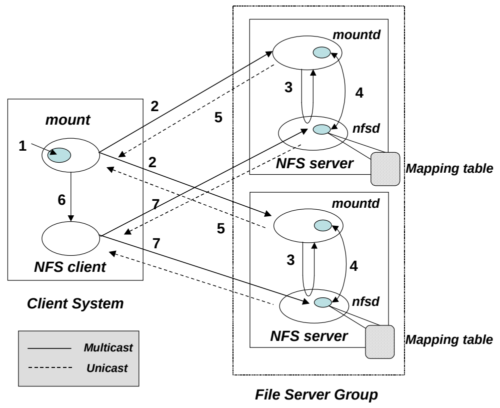


Fig. 2: The scenario of a mount procedure in FSG.

In Fig. 2, it illustrates the executions of a mount procedure in FSG. Within the executions, the concept of the *I_fHandle* will be discussed in the incoming section. The scenario is explained below:

1. The client generates an *I_fHandle* for mount point.
2. The client sends the mount request, carried *I_fHandle* to server group.
3. Each server in that same group creates a *fHandle* for the mount point.
4. The *mountd* process in server sends *I_fHandle* and *fHandle* to *nfsd* process.
5. The *mountd* process replies "ok" to client.
6. This *I_fHandle* is handed over to NFS client.
7. The client issue RPC calls to server/server group. On server side, the server transforms the *I_fHandle* into the real *fHandle* before executing the request.

2.1. Intermediate File Handle

NFS file handles, *fHandle*, are normally created by the server and used to identify uniquely a particular file or directory on the server. The client does not normally create file handles or have any knowledge of the contents of a *fHandle*. The traditional content of a file handle is composed of device number, the inode number, and a generation number for the inode. Clearly, it is machine-dependent, so the concept of intermediate file handles, *I_fHandle*, is proposed previously in [6,8,9] to mask the heterogeneity of file systems. The *I_fHandle* consists of 4 items, client's IP address, a mount number, a sequence number and an incremental number. The *Client_IP_addr* is used to distinguish different clients. The *Seq_number* and the *Mount_number* respectively represent different files in the mount directory and the different mount. The *Inc_number* is to support the multi-component LOOKUP operation [13,14,15].

```
//The structure of I_fHandle
struct I_fHandle {
   unsigned long Client_IP_addr; // Client IP address
   unsigned int Mount_number;   // the order of
   different mount
   unsigned long Seq_number;// different files in the
   mount directory.
   unsigned int Inc_number; //different components in
   MCL
   char dummy[20];  // Dummy
}
```

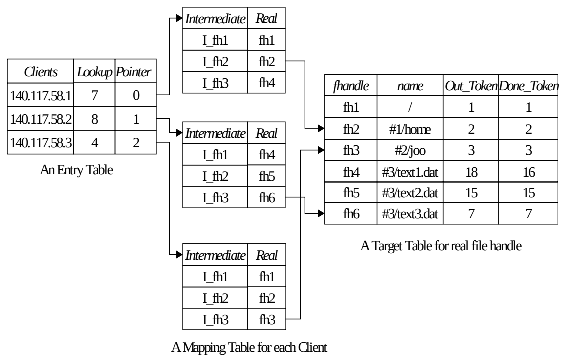
For the consistency of *I_fHandle* in server group and the generation of the unique *I_fHandle* in each server, each client must acquire a token from the sequencer before issuing a LOOKUP request to the server group.

2.2. The Mapping Table

In order to provide the mapping between the *fHandle* and the *I_fHandle*, each server in the FSG has to maintain its own mapping table. To the client, the replicated servers are grouped as a server machine with highly reliable disk storage. When a client would like to access these files in the server group, it uses the *I_fHandle* instead of *fHandle* to identify the object that the operations are applied to and multicast its request to the server group. While a server receives this request, the *I_fHandle* is retrieved from the message and is translated into the actual *fHandle* available to itself through a mapping table.

Each server in the same FSG maintains a mapping table to map *I_fHandle* into corresponding *fHandle*. While a client tries to mount a remote directory, it has to issue firstly a mount command to the server group. As receiving the mount request from a client, the server creates an Entry Table for the client as shown at the most left hand side of Fig. 3. Within the Entry Table, the LOOKUP column is used to keep the latest token for LOOKUP requests. To ensure that the unique and consistent *I_fHandle* be generated in each server, the LOOKUP operations must be performed sequentially.

As to the mapping table for each client in the middle of Fig. 3, it contains two items, *I_fhandle* and *fhandle*. In general, a file server uses the *fhandle* to locate the corresponding information in the target table. A file server used the *Out_Token* field within the target table to keep the latest updated tokens of each files and the name field to represent the file/directory name. The *Done_Token* field is deployed to record the maximum token of completed requests for the implementation of consistency control scheme. The related details were discussed in the paper [9].



Clients	*Lookup*	*Pointer*
140.117.58.1	7	0
140.117.58.2	8	1
140.117.58.3	4	2

An Entry Table

Intermediate	*Real*
I_fh1	fh1
I_fh2	fh2
I_fh3	fh4

Intermediate	*Real*
I_fh1	fh4
I_fh2	fh5
I_fh3	fh6

Intermediate	*Real*
I_fh1	fh1
I_fh2	fh2
I_fh3	fh3

A Mapping Table for each Client

fhandle	*name*	*Out_Token*	*Done_Token*
fh1	/	1	1
fh2	#1/home	2	2
fh3	#2/joo	3	3
fh4	#3/text1.dat	18	16
fh5	#3/text2.dat	15	15
fh6	#3/text3.dat	7	7

A Target Table for real file handle

Fig. 3: The structure of a Mapping Table.

2.3. The Ordering Control Scheme

In the FSG system, the client multicasts update requests to these servers in the server group. Since multiple clients might issue requests to the same file at the same time, the *"dual-counter"* synchronization mechanism [6] is proposed to manage concurrent updates before. And, the variants were proposed in [8,9,16] to improve the efficiency of FSG.

Within each server group, a server is designated as a "sequencer". The sequencer is responsible for assigning a unique token, *Out_Token*, for each update requests. The *Out_Token* consists of 2 Tokens, one for the turn and the other for the dependency. A token consists of the generation number, major sequence number and minor sequence number. The generation number is used to distinguish from different generation of sequencer. The major sequence number is subject to different GETTOKEN request and the minor sequence number represents the number of tokens that works along with the major sequence number for a GETTOKEN request. Studies [8,9,16] have shown the related discussions. For space limitation, the details do not be repeated here.

While a client receives an update request including LOOKUP request, it first multicasts the GETTOKEN request to get a token from the sequencer. In order to make the GETTOKEN request idempotent, we added a redundancy field to prevent the duplicated GETTOKEN request. This redundancy field keeps the last token, which the client required recently. Because the GETTOKEN procedure is stop-and-wait method, the sequencer can distinct from these duplicated LOOKUP requests by this field.

2.4. Multi-component LOOKUP Operation

Firstly, let’s consider the proposed Reliable File Server Group (FSG) Session without Multi-component LOOKUP, MCL, support. An example of a client reading the /web/index.html file in the exported directory /pub is shown below. It needs 8 RPC calls for resolving the /web/index.html pathname besides the mount command is necessary.

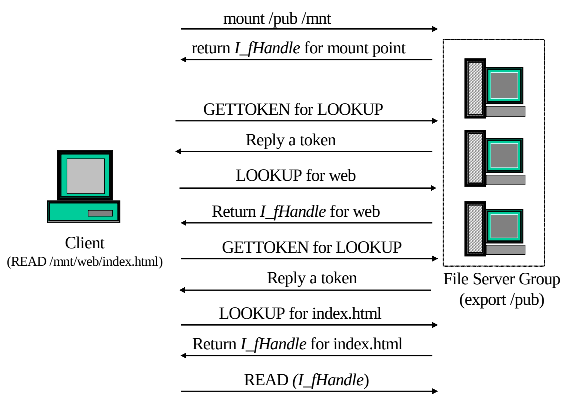


Fig. 4: An example of a FSG Session without Multi-component LOOKUP.

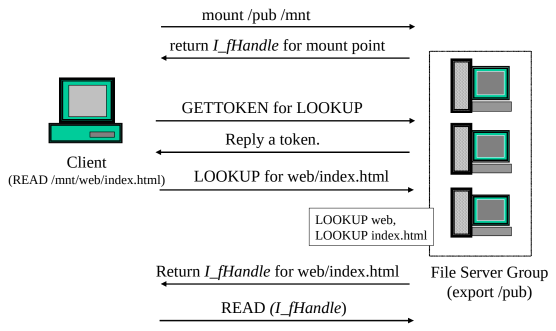


Fig. 5: An example of a FSG Session with Multi-component LOOKUP support.

As to the construction of the *I_fHandles* for the MCL operation, the *Seq_number* item of the *I_fHandle* is assigned with the token number and the *Inc_number* item of the *I_fHandle*, initialized with 0, increases for each component LOOKUP. As shown in Fig. 5, the “web” and “index.html” objects own the same *Seq_number* but different *Inc_number*, 0 and 1 separately. With such a MCL mechanism, the number of RPC calls to resolve the pathname is reduced to 4.

3. Conclusions and Future Works

Previously, the concept of intermediate file handle is proposed to cover the heterogeneity of replicated file system. In this paper, we follow the concept of intermediate file handle to improve the efficiency of FSG with the feature of multi-component LOOKUP, MCL. With MCL mechanism, file servers allow a client to resolve a full path name in one operation. It is able to reduce the number of RPC calls and to achieve efficient pathname LOOKUP in the FSG.

4. References

[1] Peter J. Braam, "File Systems for Clusters from a Protocol Perspective", http://www.inter-mezzo.org

[2] M. Satyanarayanan, J.J. Kistler, P.Kumar, M.E. Okasaki, E.H. Siegel and D.C.Steere "Coda: A highly available file system for a distributed workstation environment" IEEE Transactions on Computers, 39(4), pp.447-459, April 1990

[3] A. Siegel, K. Birman and K. Marzullo. "Deceit: A flexible distributed file systems" In Summer 1990 USENIX Conference, pages 51-61, Anaheim, CA, June 1990

[4] M.M. Leboute and Taicy Weber, "A reliable distributed file system for UNIX based on NFS", UFRGS, Brazil, IFIP International Workshop on Dependable Computing and Its Applications (DCIA 98) January 12 - 14, 1998, Johannesburg, South Africa

[5] Yasushi Saito and Christos Karamanolis, "Pangaea: a symbiotic wide-area file system," ACM SIGOPS European Workshop, Sep 2002.

[6] C. S. Yang, S. S. B. Shi and F. J. Liu, "The Design and Implementation of a Reliable File Server", Newsletter of the Technical Committee on Distributed Processing, summer 1997.

[7] Bjorn Gronvall, Assar Westerlund, and Stephen Pink. "The design of a multicast-based distributed file system". In Proc. of Operating Systems Design and Implementation, pages 251-264, 1999.

[8] F.J.Liu and C.S.Yang, "THE DESIGN AND ANALYSIS OF A HIGHLY-AVAILABLE FILE SERVER GROUP", IEICE Transactions on Information and System, Vol.86-E, No.11, pp. 2291-2299, 2003

[9] F.J.Liu, C.S.Yang and Y.K.Lee, "The Design of An Efficient and Fault-tolerant Consistency Control Scheme in File Server Group", IEICE Transactions on Information and System, Vol.E87-D No.12, pp.2697-2705, 2004.

[10] S.Deering, Host Extensions for IP Multicasting, RFC 1112, Internet Engineering Task Force, 1989.

[11] S.Floyd, V. Jacobson, C.Liu, S. McCanne, L.Zhang, A Reliable Multicast Framework for Light-weigh Sessions and Application Level Framing, IEEE/ACM Transactions on Networking, 5(6), Dec. 1997.

[12] K.Birman, A.Schiper, P.Stephenson, Light-weight Causal and Atomic Group Multicast, ACM Transactions on Computer Systems, 9(3), Aug. 1991.

[13] The NFS Version 4 Protocol.

[14] Callaghan, B., "WebNFS Client Specification," RFC 2054, October 1996. http://www.ietf.org/rfc/rfc2054.txt

[15] Callaghan, B., "WebNFS Server Specification," RFC 2055, October 1996. http://www.ietf.org/rfc/rfc2055.txt

[16] Fengjung Liu and Chu-sing Yang, "Improving Concurrent Write Scheme in File Server Group", (ICA3PP-2005) Lecture Notes in Computer Science, pp.1-10, Vol. 3719, 2005.