

國立交通大學

電機與控制工程學系

碩士論文

適用於低網路頻寬的個人化行動檔案系統

Design and Implementation of a Personal Mobile File System
for Low Bandwidth Networks



研究生：許正道

Student: Cheng-Tao Hsu

指導教授：黃育綸 博士

Advisor: Dr. Yu-Lun Huang

中華民國九十八年五月

May, 2009

適用於低網路頻寬的個人化行動檔案系統

Design and Implementation of a Personal Mobile File System
for Low Bandwidth Networks

研 究 生：許正道

Student: Cheng-Tao Hsu

指導教授：黃育綸 博士

Advisor: Dr. Yu-Lun Huang

國 立 交 通 大 學

電機與控制工程學系

碩士論文

A Thesis

Submitted to Department of Electrical and Control Engineering

College of Electrical Engineering

National Chiao Tung University

in partial Fulfill of the Requirements

for the Degree of

Master

in

Department of Electrical and Control Engineering

May, 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年五月

適用於低網路頻寬的個人化行動檔案系統

學生：許正道

指導教授：黃育綸 博士

國立交通大學電機與控制工程學系(研究所)碩士班

摘要

隨著網路的迅速發展，行動網路上的檔案分享成為近幾年的熱門應用。愈來愈多的研究圍繞著日漸成熟的行動網路，同時也設計出各式各樣的檔案系統供使用者使用。與傳統的有線網路相比，行動網路有著頻寬較為有限、訊號品質較不穩定以及通訊成本的考量等特性，多數的研究在於利用離線操作的方式來解決網路連線品質的問題，而較少考慮通訊成本。較低的通訊成本，可以提升檔案系統在操作的反應以及效率。本論文設計一個新的行動檔案系統 -- MoFS。除了支援離線操作之外，MoFS 的設計中也同時考慮如何降低通訊頻寬需求量等問題。MoFS 的使用者端透過使用者層的檔案系統程式庫、本地端快取、使用者認證與檔案傳輸加密等方式，實現一個安全、容易部署且支援離線操作的行動檔案系統使用端。MoFS 利用新設計的伺服器通知 (Server Notification) 功能，由伺服器負責通知檔案狀態的變動情形，可藉以降低使用者端因頻繁詢問檔案同步狀態所浪費的網路頻寬。最後本研究亦透過數個實驗，分析比較數種分散式網路檔案系統 (包括 NFS、Coda 等) 在不同硬體平台 (x86、ARM 等) 的效能表現。實驗結果證明 MoFS 能有較快的檔案存取處理速度、較好的讀寫處理能力與較低的網路頻寬消耗。尤其在計算能力與系統資源有限的 ARM 嵌入式處理平台上，MoFS 能比其他分散式網路檔案系統有更好的效能表現。

Design and Implementation of a Personal Mobile File System for Low Bandwidth Networks

Student: Cheng-Tao Hsu

Advisor: Dr. Yu-Lun Huang

Department of Electrical and Control Engineering

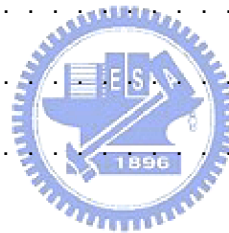
National Chiao Tung University

Abstract

With the rapid growth of Internet technologies, file sharing has become one of the most popular applications in recent years, especially on mobile networks. However, limited bandwidth, unstable signals and high communication cost can erode the popularity of mobile file sharing applications. In this research, we design a new mobile file system, MoFS, to provide mobile file services with low-bandwidth consumption. MoFS is a structured user-space file system that supports disconnected operations and implements a server-side notification to reduce the bandwidth requirement resulting from the frequent status checks. MoFS does not modify the server side but simply provides a dynamic-link library for applications. We realize our design on both x86- and ARM-based platforms (Openmoko Freerunner). We also conduct a series of experiments to demonstrate and compare the performance among various network file systems, including NFS, Coda, etc. The experiment results show that MoFS has faster processing time, better read/write throughputs and lower communication costs.

Contents

摘要	i
Abstract	ii
Table of Contents	iii
List of Figures	vi
List of Tables	vii
Chapter 1 Introduction	1
1.1 Background	1
1.2 Contribution	2
1.3 Synopsis	3
Chapter 2 Related Work	4
2.1 Traditional Network File System	4
2.2 Network File System Supporting Mobility	5
2.3 Filesystem in User-space (FUSE)	8
2.4 SSHFS	9
2.5 Summary	10
Chapter 3 New Mobile File System: MoFS	13
3.1 Architecture	13
3.2 MoFS Client	14
3.2.1 File Operation Manager (FOM)	14
3.2.2 Data Manager (DM)	15



3.2.3	Network Manager (NM)	15
3.2.4	Upload Manager (UM)	16
3.2.5	Version Manager (VM)	17
3.3	MoFS Server	17
3.4	Supported Operations	19
3.4.1	Basic File Operations	20
3.4.2	Advanced File Operations	22
Chapter 4	Implementation	23
4.1	MoFS Client Application	24
4.1.1	File Operation Manager (FOM)	24
4.1.2	Data Manager (DM)	25
4.1.3	Network Manager (NM)	26
4.1.4	Upload Manager (UM)	27
4.1.5	Version Manager (VM)	27
4.2	MoFS Server Application	27
4.3	Installation	28
Chapter 5	Experiments and Analyses	29
5.1	Preliminaries	29
5.1.1	Hardware	29
5.1.2	Networks	30
5.1.3	Tools	30
5.2	Environment Setup	31
5.3	Experiments	32
5.3.1	Operation Latencies	32
5.3.2	Read/Write Throughputs	35
5.3.3	Communication Costs	38

Chapter 6 Conclusion

42

References

43



List of Figures

2.1	Architecture of NFS.	5
2.2	Architecture of Coda.	6
2.3	Architecture of AshFS.	8
2.4	Work Flow of FUSE.	9
3.1	Architecture of MoFS.	13
3.2	Architecture of MoFS Client.	15
3.3	The Flow of Checking the Network States	16
3.4	Architecture of MoFS Server.	18
3.5	Work Flow of Consistency Manager	19
4.1	Architecture of MoFS	23
4.2	Hash Tree Structure	26
5.1	Experimental Environment	31
5.2	File Operations Frequencies of Laptop	33
5.3	File Operations Frequencies of MoFS/Smartphone	35
5.4	Read/Write Throughputs of Laptop	37
5.5	Read/Write Throughputs of MoFS/Smartphone	38
5.6	Experiment flow for Communication Cost	40



List of Tables

2.1	Comparisons of related work	12
3.1	ls	20
3.2	open and close	21
3.3	read and write	21
5.1	File Operations Frequencies of MoFS/Laptop (1/Sec)	33
5.2	File Operations Frequencies of MoFS/Smartphone (1/Sec)	35
5.3	Read/Write Throughputs of MoFS/Laptop (KB/Sec)	36
5.4	Read/Write Throughputs of MoFS/Smartphone (KB/Sec)	37
5.5	Communication Costs of MoFS/Laptop (KB)	41



Chapter 1

Introduction

With the rapid growth of Internet technologies, file sharing becomes one of the most popular applications in recent years, especially on mobile networks. People like to share their musics, video clips, documents, and even personal schedules on the Internet. However, the limited bandwidth, unstable signals, and high communication costs erode the popularity of file sharing applications on mobile networks.

1.1 Background



In 1987, A research group, under the direction of of M. Satyanarayanan, studied on mobile network file systems and designed the Coda file system [1][2]. Coda is a distributed file system for large scale local area networks. With a shared data repository, users can entirely rely on local resources when that repository is partially or totally inaccessible. Coda also supports disconnected operations, and has good performance through the design of client side persistent caching. For every command, a Coda client program needs to check the file status with the Coda file server and thus require more bandwidth consumption.

In 1998, NFS/M [3] extends NFS to support mobility and disconnected operations. Based on the NFS protocol, NFS/M works seamlessly with the existing NFS server. Such a design may make it easy to be deployed. Different from conventional NFS systems, NFS/M implements a proxy server and a cache manager in the NFS client application to support disconnected

operations. The modification of NFS client application requires kernel patches to rebuild a new client program. The deployment of NFS/M is then restrained.

For easy development and deployment, Filesystem in User-space (FUSE) [4] provides libraries and APIs to construct a user-space file system [5]. With a user-space implementation, FUSE allows simple installation (no patches and no kernel rebuilds) and is usable by non-privileged users. Currently, many file systems, such as SSHFS [6][7], AshFS [8], httpfs [9], etc, are implemented based on the FUSE libraries and APIs.

Based on FUSE, SSH Filesystem (SSHFS) realizes a file client application based on the SSH File Transfer Protocol. Since most SSH servers already support FTP protocol, no modification is required on the SSHFS servers. On the other hand, using an SSHFS client application is similar to the usage of an SSH client program. It is as easy as logging into an SSH server.

Also taking FUSE as its basis, AshFS [8] shots for a user-space personal file system. AshFS supports some advanced features including disconnected operations and automatic synchronization. Since AshFS is designed for the personal use, it simplifies some complex mechanisms used in conventional distributed file systems. Such a simplification may reduce the bandwidth consumption and computing powers. However, AshFS suffers from the same problem as that in Coda, more bandwidth is required for frequent status checks.

1.2 Contribution

In this research, we design a new mobile file system, named as MoFS. MoFS is a structured user-space file system. With a persistent cache reserved at the client program, MoFS also supports disconnected operations. In a MoFS client program, FUSE is used to execute file operations, and an SSH client program is adopted for secure data transfer. MoFS implement a

server-side notification to reduced the bandwidth requirements resulted from the frequent status checks. For easy installation, MoFS does not modify the SSH server program but simply provides a dynamic link library for the MoFS server daemon.

1.3 Synopsis

The thesis is organized as follows. We review existing file systems in Chapter 2. Then, we describe the architecture of the proposed mobile file system (MoFS) in Chapter 3. Chapter 4 explains the implementation of MoFS, and we show some experiments and the performance of MoFS in Chapter 5. In the end, we conclude the thesis in Chapter 6.



Chapter 2

Related Work

Because of the population of Internet, people sharing files on network become more and more familiar. For this situation, the network file system needs some features: mobility, security, and conveniency. The mobility includes wireless support, bandwidth, disconnect operation, file caching. The security includes user authentication, data encryption, and access protection. The conveniency includes deployment effort, and friendly using. There are many researches focus on network file system. We can divide them into two categories: traditional network file system and network file system supporting mobility. In this paper, we define "server side change" as "give some individual patch to the general file server".



2.1 Traditional Network File System

The network file system distributed without mobility supports is defined as "Traditional Network File System". The Network File System (NFS) is a network file system protocol defined in RFC 1813 (NFSv3 [10]). It is the first widely deployed transparent network file system. Using NFS not only improves the storage for the users but also provides well remote backup. Therefore, many people shared files by using NFS. It use RPC to communicate between server and client, but it does not support disconnect operation and does not have any cache. No security property is also its disadvantage. We illustrate the architecture of NFS with Figure 2.1. When taking a file request, the request goes from shell to VFS to invoke the request in the kernel

module of NFS client. NFS client transmits the request to NFS server over the internet by RPC. Next, NFS server requires the real data through VFS at server side, then returns them back to client side.

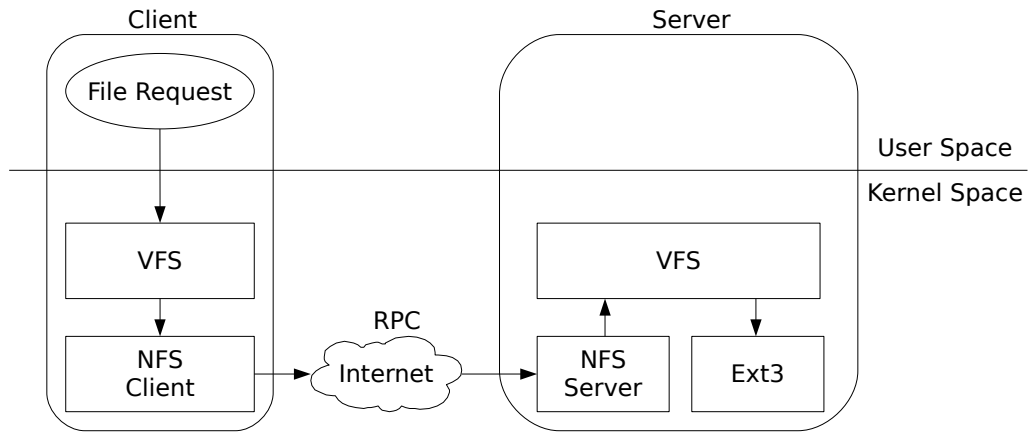


Figure 2.1: Architecture of NFS.



2.2 Network File System Supporting Mobility

Mobility becomes one of the major issues in designing distributed network file systems. To support mobility, we need to consider many criteria, such as limited bandwidth, battery-powered mobile devices, secure communications, etc. This section discusses existing file systems supporting mobility. The first two file systems, Coda [2][1] and MAFS [11], create brand new architecture to support mobility, while the last two (NFS/M [3] and AshFS [8]) implement new client side programs to support disconnect operations and meanwhile keep their file servers unchanged.

- Coda

Coda is a distributed file system with server replication and disconnected operation. To provide a shared storage repository of higher availability, Coda uses server replication,

storing copies of files on multiple servers. For clients does not have network with server, Coda provides disconnected operation to let the client can work continuously. Coda contains three blocks, two at client: Coda kernel module and Venus, one at server: Vice, as we show in Figure 2.2 Assume client and server are connected, here comes a file command, ``cat" for example. The system call comes into kernel catching by VFS which notices the action executes in Coda file system directory. VFS through Coda kernel module communicates with Venus, Coda cache manager, by ``/dev/cfs0". Venus use RPC2 protocol communicates with Vice, Coda file server, over the network. Then, the informations are returned by the same path. In this flow, we know that Coda uses the RPC2 protocol library on client-server communication, but RPC2 is not very familiar cause deployment difficultly. In addition, Coda has fully reintegration and replication methods, means that it needs enough control messages. Using too many control messages may cause more protocol overhead, then increasing the communication cost.

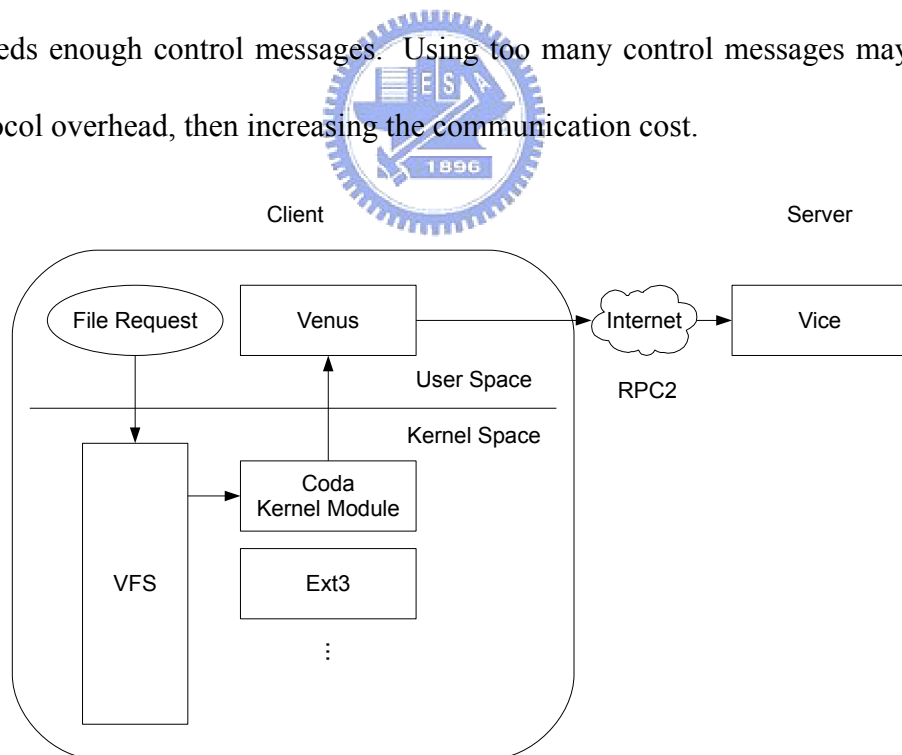


Figure 2.2: Architecture of Coda.

- MAFS

MAFS (Adaptive Mobile File System) is a distributed file system designed to cope with floating available bandwidth. It designs a new protocol, Adaptive RPC for client-server communication. By using Adaptive RPC, MAFS propagates file modifications asynchronously at all bandwidth levels. It uses RPC priorities to reduce interference between read and write traffic at low bandwidth. To ensure the application adapts itself to the available bandwidth, lower bandwidth translates into longer delays for lower-priority RPCs. The write RPC has the lowest priority and has longer delay at low bandwidth. This way may cause file inconsistency. MAFS incorporate a new invalidation-based update propagation algorithm, SIRP, to ensure the propagation.

- NFS/M

NFS/M extends NFS client without modifying the NFS server. It supports data prefetching, disconnect operation and data reintegration. The original NFS is not suitable for mobile computing applications. The reason is that NFS assumes the communication network is fast and reliable. But the real situation is network connection may not even available in some areas. NFS/M uses Cache Manager and Proxy Server to maintain local disk cache. For the file consistency issue, NFS/M used mathematical expression to show how conflict occurred and how to resolve the conflict.

- AshFS

AshFS is a lightweight mobile file system. It is extended from SSHFS [7] but supports disconnect operations. It used the SSH [12][13][14][15][16] server for the file services, and developed a new client program to handle file cache, disconnect operation, file staleness, conflict when reintegration, communication state, and transmission optimization. The implementation of AshFS leverages a user-space filesystem library, called FUSE [4]. Figure2.3 shows the work flow of AshFS. Taking ``ls" command (list files) as an example,

the command goes from shell to VFS to invoke the ``ls" function of the specified file system. Since FUSE is used as the underlying library, the invocation goes back to the user space. Then, the ``ls" function of libfuse is invoked. Next, the AshFS client application either obtains file information from server by SSH client (if the internet is reachable), or shows the file information existed in the local data cache (if network is disconnected.)

Since AshFS uses a stateless server without keeping client state information, it needs to periodically check the staleness of files. The periodical checks result in high communication costs.

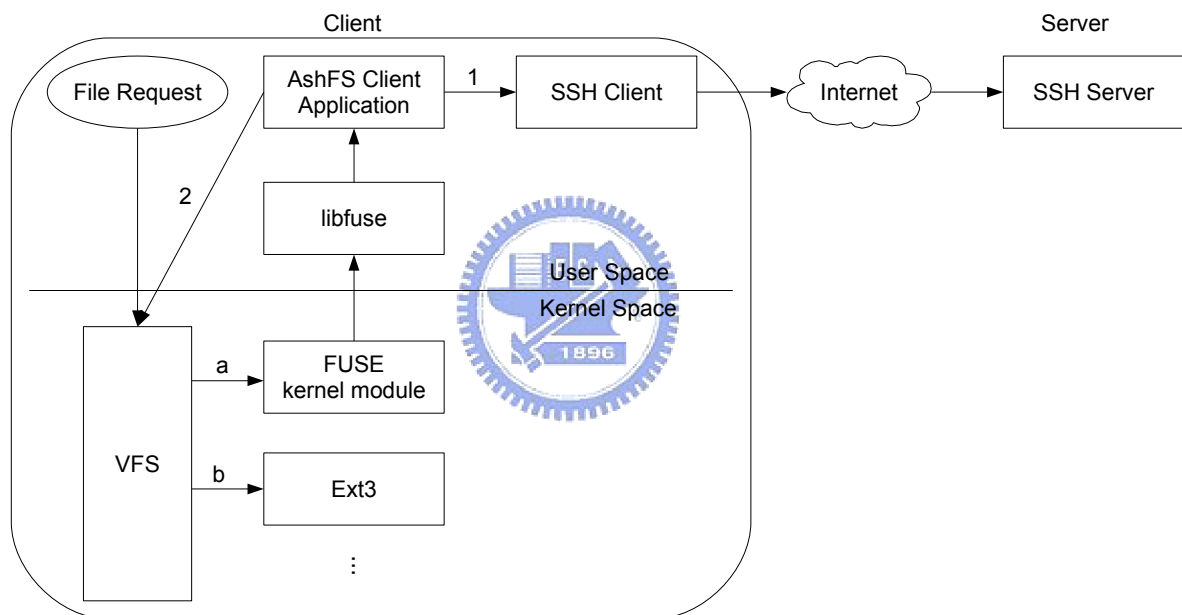


Figure 2.3: Architecture of AshFS.

2.3 Filesystem in User-space (FUSE)

A file system has to provide file operation functions. In general, file system operations are made in the kernel space. However, the performance bottleneck of network file systems is

the network speed. We use FUSE to make a file system in the user space. The work flow of FUSE shows as Figure 2.4. When a file request, like stat, comes from user, it enters the VFS in the kernel. The VFS identifies that the request is for FUSE file system, and the VFS passes the request to the FUSE kernel module. The FUSE kernel module transfers the request to user space FUSE library, by allowing the library to access a special device `/dev/fuse`. Finally the FUSE library communicates with the user space application to know how to answer the request.

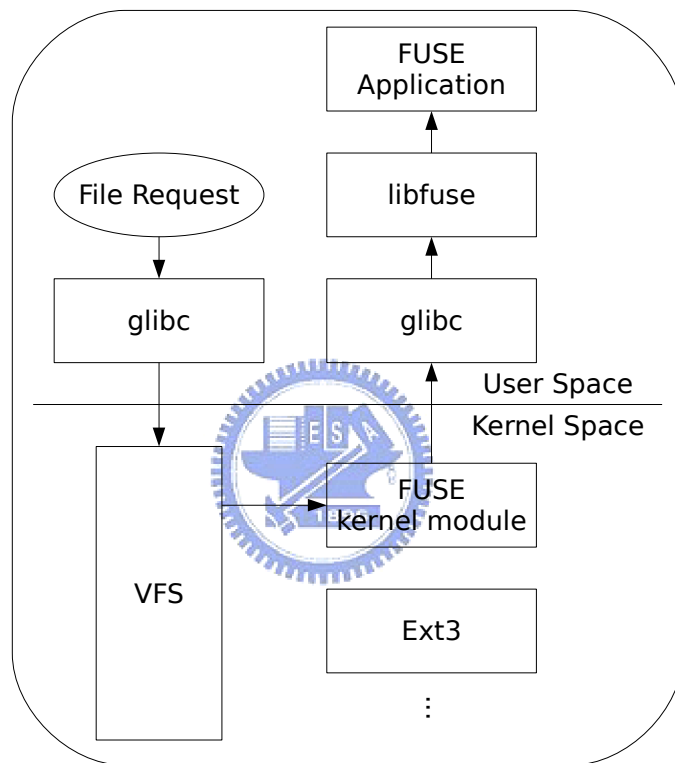


Figure 2.4: Work Flow of FUSE.

2.4 SSHFS

SSHFS is a file system client based on the SSH File Transfer Protocol. Since most SSH servers already support this protocol it is very easy to set up: i.e. on the server side there's nothing to do. On the client side mounting the filesystem is as easy as logging into the server

with ssh. On the local computer where the SSHFS is mounted, the implementation makes use of the FUSE kernel module.

2.5 Summary

In this section, we compare existing file systems in terms of mobility, security, and convenience. A file system supporting mobility need to have a persistent cache, can work at the disconnection, and suitable for wireless environment. User authentication and data encryption are the security issue of a mobile file system. If a network file system uses the standard protocol, it increases the convenience of the deployment.

- Persistent Cache

The file system uses cache to hoard the needing files into local disk. When the file system wants to access a file, it can access the file in the cache directly. Directly accessing the cache can reduce the slow efficiency caused by network. Here, ``persistent" means the caching files are not just temporary storing in local disk. Even if users reboot the client, the caching files are still hoarding in the local disk.

- Disconnected Operation

In the mobile network, users may not connect with Internet all the time. However, users still hope that they can work with the remote file at disconnection. Hence, disconnected operation is a very important method for mobile network. In addition, after the network resumes, file reintegration and file consistency between client and server are another important issue [17][18][19].

- Wireless Supporting

The network that mobile device usually use by users is wireless network, like 3G, GPRS,

or WiFi. Since the environment and the network interface card, the quality of the wireless network is floating at many time. For unstable wireless network, the mobile file system have many special method to improve the efficiency of the usage of network.

- User Authentication

The privacy and security of the files in the file system become more and more important. Only the file owner has right to access the file. The mechanism of identification allows only users who have the right can access the file.

- Data Encryption

In mobile network, transfer data using the wireless network is a very common way. However, transferring files through the wireless network is easy to be eavesdropped and causes confidential information to be disclosed. Using data encryption to transfer files can ensure the security and privacy of files. It is necessary for mobile network.



- Standard Protocol

Using the standard protocol can increase the feasibility and portability of the system. It is easier to integrate with existing system and easier to expand for future development. If a network file system uses the standard protocol, it increases the convenience of the deployment.

Table 2.1 shows the comparison of characteristics about NFS, SSHFS, Coda, MAFS, NFS/M, and AshFS. First, it shows that Coda, MAFS, NFS/M, and AshFS are well supporting for mobility. These four file system have different cache control mechanisms to support disconnected operation. In contrast, NFS and SSHFS only provide remote access in connection. Since NFS protocol only provides IP mask to protect connection source, NFS and NFS/M can not give files safe protection. Even though MAFS has user identification, it does not encrypt the tran-

ferring data to cause information be stolen. Coda considers security issue, it has safe protection for data transmission. SSHFS and AshFS use the well-know secure shell protocol so that the data transmission is very reliable and securable. MAFS uses self-definition network protocol. NFS, NFS/M, SSHFS, and AshFS use general standard protocol. Using standard protocol can increase the feasibility of system. FUSE only provides an interface for developer to construct an user-defined file system. Since FUSE is not a complete file system, we does not list FUSE in Table 2.1.

Table 2.1: Comparisons of related work

	NFS (nfsd)	SSHFS (sshd)	Coda (Vice)	MAFS (MAFS Server)	NFS/M (nfsd)	AshFS (sshd)
Persistent Cache	No	No	Yes	Yes	Yes	Yes
Disconnected Operation	No	No	Yes	Yes	Yes	Yes
Wireless Supporting	No	No	Yes	Yes	Yes	Yes
User Authentication	No	Yes	Yes	Yes	No	Yes
Data Encryption	No	Yes	Yes	No	No	Yes
Standard Protocol	Yes	Yes	No	No	Yes	Yes

Chapter 3

New Mobile File System: MoFS

3.1 Architecture

We show the architecture of mobile file system, MoFS, in Figure 3.1. For easy deployment, we use FUSE to construct a user-level file system so that we do not modify the kernel source code. For decreasing the communication cost of checking file staleness, we use a server side program, MoFS server application. For transmission security, we use SSH protocol for communication between client and server.

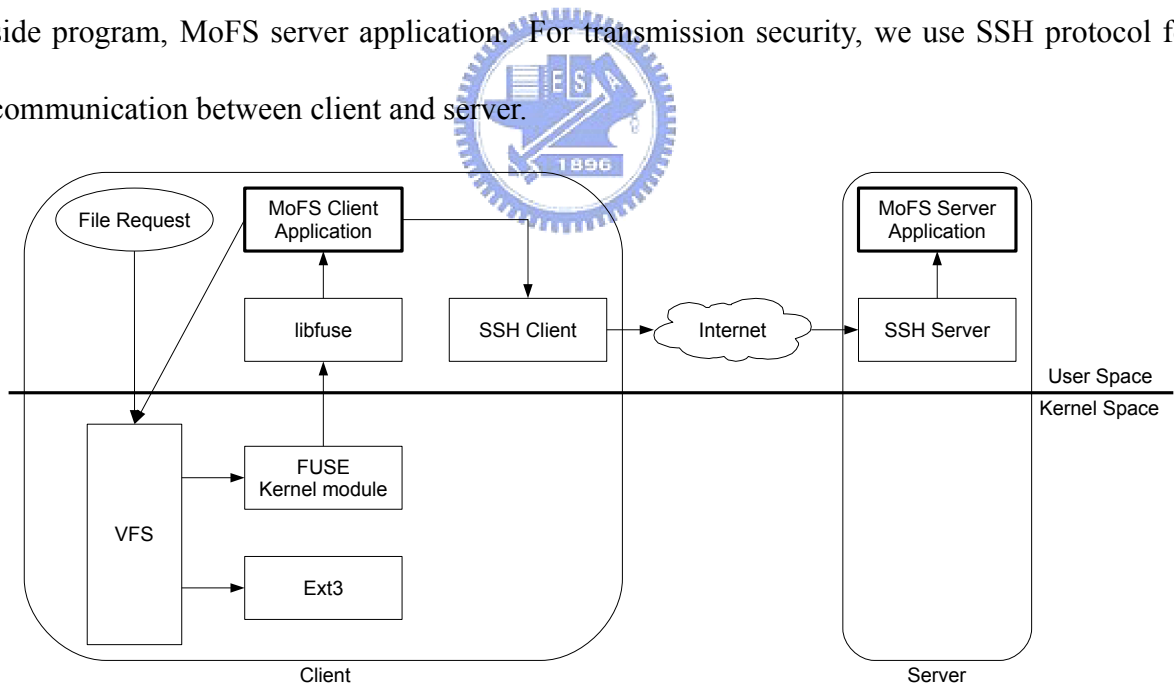


Figure 3.1: Architecture of MoFS.

3.2 MoFS Client

Figure 3.2 shows that MoFS Client is composed of libfuse, MoFS Client Application, SSH Client, in user space, and VFS, FUSE kernel module, Ext3, in kernel space. VFS, FUSE kernel module, libfuse, and MoFS Client Application construct a user-level file system for easy deployment. VFS, Ext3, and MoFS Client Application construct a local cache to use in disconnected operation. MoFS Client Application and SSH Client protect the transmission data securely with SSH protocol. The behavior of VFS, FUSE kernel module, Ext3, libfuse, and SSH Client is similar to their action in AshFS. MoFS Client Application is a client program to maintain file caching, network status, and client requests. MoFS Client Application is composed of File Operation Manager, Data Manager, Network Manager, Upload Manager, and Version Manager. The relationship of these managers shows as Figure 3.2. The File Operation Manager (FOM) implements file system operations, like open, read, write, close, and so on. The Data Manager (DM) maintains local file cache and metadata information. The Network Manager (NM) monitors network status and executes network commands. The Upload Manager (UM) optimizes files in the upload queue. The Version Manager (VM) controls the version of the local file cache.

3.2.1 File Operation Manager (FOM)

In our design, the FOM implements file system operations, like open, read, write, close, and so on. The FOM is a implementation of the API which provide by FUSE library. When creating a new file or reading/writing a file, it will communicate with the Data Manager for getting file's information. When the request relates to network, get file's metadata from server for example, it pushes the request into upload queue of Upload Manager.

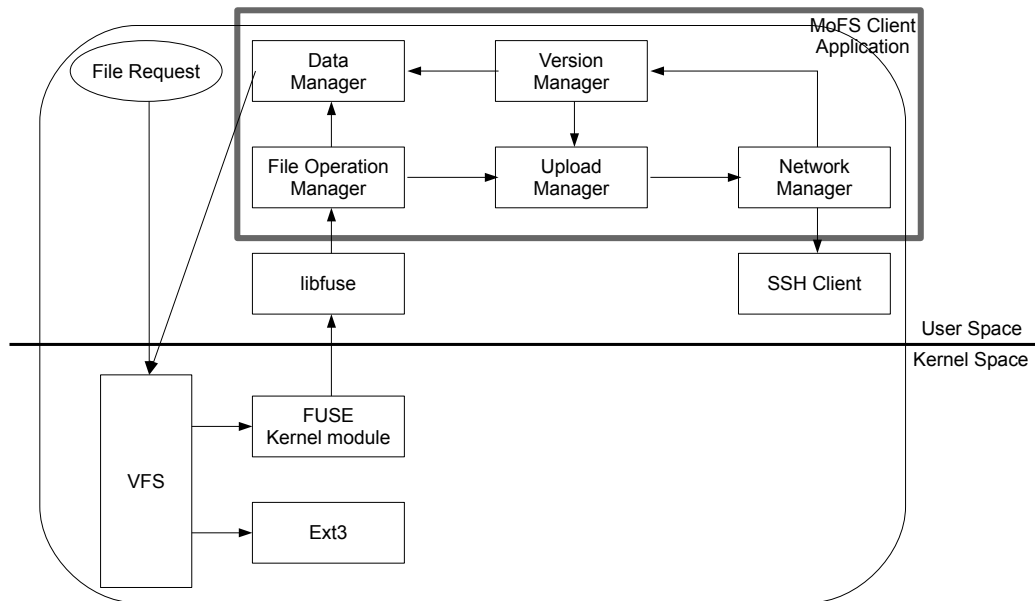


Figure 3.2: Architecture of MoFS Client.

3.2.2 Data Manager (DM)

The DM maintains local file cache and metadata information. For using less network packages, we do not download all the files in server. When we list the server directory, we only get the stat structure of files in the server directory. In this method, we can download the files what we actually need. At the first time we open the file, the file start to download. Another mechanism for decreasing communication packages is writeback-on-close. We only upload the modified file to server when the file is closed.

3.2.3 Network Manager (NM)

The NM monitors network status and executes network commands. We divide the connection status, like AshFS, into three states: Strong, Weak, and Disconnect. We define the three states by using ICMP packets as showing in Figure 3.3 When we execute any file operators, like open, we need to check the network status to know if we can send any commands to the file server. First, we send an ICMP packet and waiting for it's reply. If the reply packet returns

in 100ms, we define the network status in Strong State. Otherwise, we send an ICMP packet again. The second time for waiting the reply, we set the timeout as twice than the first ICMP packet. If the reply does not timeout, we define it in Weak State. Provided that the timeout still happens, does not reply in 200ms, we define the network status in Disconnect State.

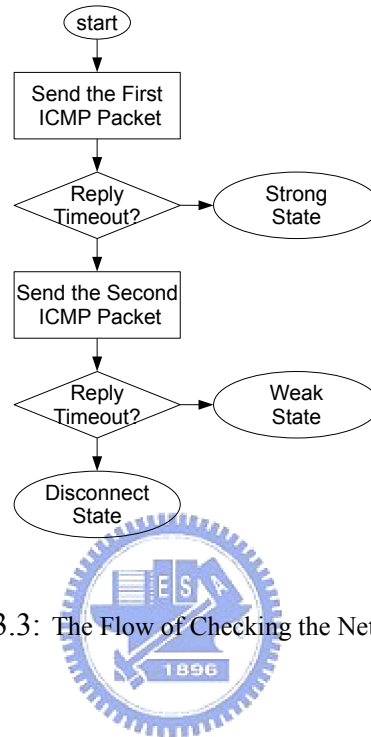


Figure 3.3: The Flow of Checking the Network States

3.2.4 Upload Manager (UM)

The UM optimizes files in the upload queue. We want to decrease the usage of communication, so we design an upload optimizer, like AshFS. When using a file system, people may do some redundant works, like creating a file and deleting the file in short period of time. The main action of the optimizer is to reduce the redundancy. The UM uploads the file to the server only at the Strong State, so the optimizer has different behaviors at different network status. At the Strong State, if there has no any other file operations, no download files or no fetch metadata, the UM upload files immediately. But at Weak State and Disconnect State, the files put into an upload queue waiting for the Strong State to come. At these two states, the UM monitors whether the file is removed by user. If user removes the file, the UM delete the entry of the file

in upload queue.

3.2.5 Version Manager (VM)

The VM controls the version of the local file cache. In AshFS, when a reintegration occurs, system resolves the staleness between server and client. At this situation, file may be have some conflicts. AshFS use the ``rename" mechanism to solve conflict programs. The "rename" mechanism changes old file to a new name with timestamp, and changes the latest file to the real name. It is confused at viewing the directory by using "rename" mechanism. The real files are overwhelmed by other renamed files. Version control is another choice to decrease this situation. Using version control mechanism, we put the older version file into the repository. If we discover that the file in repository is the real file we need, we can revert the file from the repository by using version control mechanism. In version mechanism, we only save different part in old version. By using ``diff" we decrease the disk usage comparing with rename mechanism in AshFS. However, version control mechanism may cause operation overhead at resolving content conflict.

3.3 MoFS Server

MoFS Server is composed of SSH Server and MoFS Server Application (Figure 3.4). SSH Server is a well-know shell server with security issue. MoFS Server Application is a server program to record the file state for client. MoFS Server Application only has one component, Consistency Manager (CM). The CM keeps file consistency between client and server.

To comply with existing file systems, AshFS does not change the server side. It means that AshFS does not patch existing server or create a new server. In AshFS, server side only uses

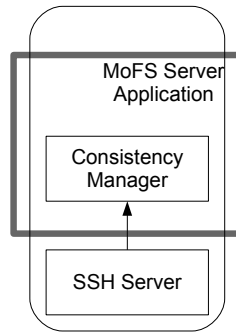


Figure 3.4: Architecture of MoFS Server.

original ssh server for operating. No server side change is a good method for easy deployment and portability, but it wastes communication cost when validating the files in cache. Every operation of AshFS needs to validate the files in cache. The cost of validation will be very large. There are two different methods can decrease the effect of validation. One is to change the server program, and the other one is to create a relay process. Changing the server program involves patching and rewriting. Patching means we modified a current existing server code, and then rebuild the server program. Rewriting means we do not use any existing server, but we write new server code instead. Regardless of giving a patch to the original server or rewriting a new server program, it will be inconvenient to setup the server side. Changing the server program conflicts to our expectation. The other option is to create a relay process. Relay process is a server side program. It transmits the messages between server and client, but not only bypass the messages. On the progress, relay process records the file requests from client. Relay process will check files status continuously. It will return the updating message to client, if the server side files are updated. This method does not modify the original server program, but makes a little change to server side. Users can still use the original server, and execute a small process in a easy way. Relay process does not modify the server code, but also can decrease the communication cost.

In our design, relay process is called Consistency Manager (CM). The work flow of CM shows in Figure 3.5. First the CM waits for client connection. When a request comes from

client, CM determines the request. If the request is a download request, the CM checks whether the client accesses the file or not. The CM adds the information or updates the file into record pool. Finally, server transfers the file to client. If the request is an upload request, the CM receives the file right away. And then checking the record pool makes sure that whether the file is on tracking. If there is a client has used the file, the CM sends a message to announce the client that the files has been changed.

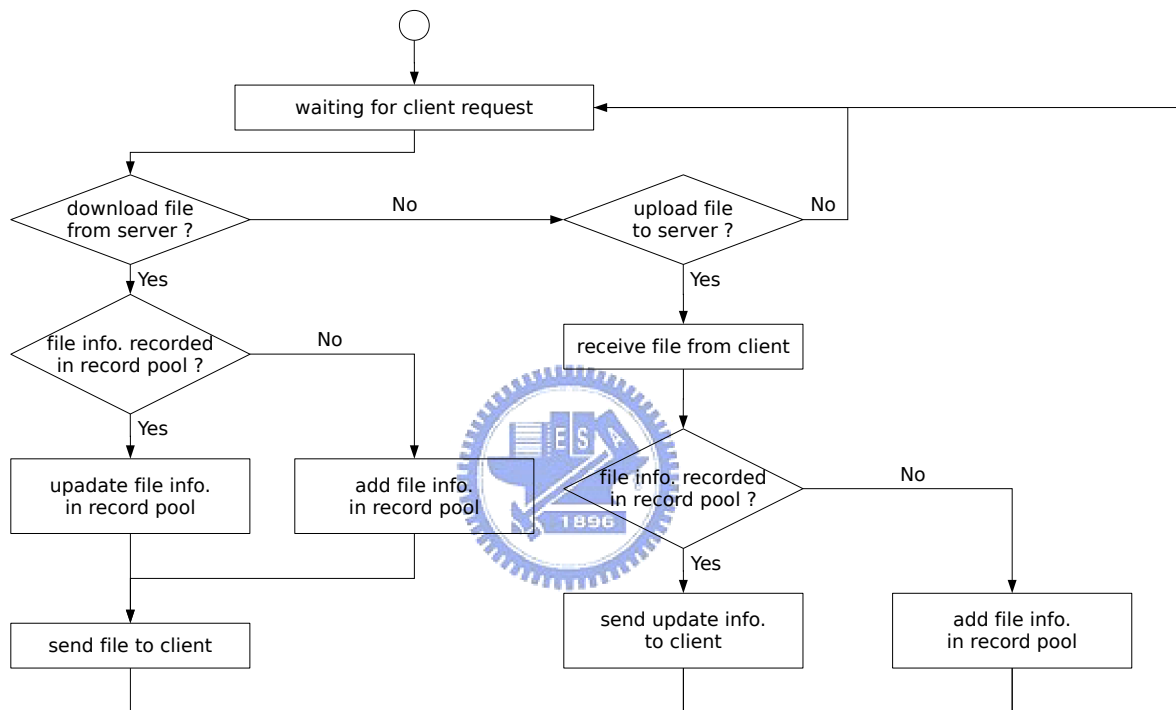


Figure 3.5: Work Flow of Consistency Manager

3.4 Supported Operations

MoFS supports lots of operations at client and server. The first kind of operation is the basic file operation, like open, read, write, and so on. All the file systems have to provide the basic file operations for I/O. The second kind of operation is the advanced file operation only provided by MoFS.

3.4.1 Basic File Operations

MoFS supports all the file operations provide by kernel, consisted of ls (list), open, read, write, close, rename, rm (remove), chmod (change file mode), chown (change file owner), mkdir (make directory), and rmdir (remove directory). We implements those operations at FOM in MoFS Client Application. The file requests of ls, open, read/write, and close are the most important operations at file I/O usage.

Table 3.1 shows the behavior of ``ls". The ls command is to list the contents of the directory. As an user executes ``ls" in the shell at MoFS mount point, VFS transmits the request to FUSE kernel module. Since FUSE is used as the underlying library, the invocation goes back to the user space. Then, the ``ls" function of libfuse is invoked. Next FOM asks DM whether the metadata of this directory is in metadata cache. If metadata cache has the metadata of require directory, returning the information to the user. Otherwise, FOM adds a request in upload queue of UM to require the metadata of required directory. NM and SSH Client bypasses the request to SSH Server. Finally, the metadata of required directory gets from the file server and then caches in local memory for next usage.

Table 3.1: ls

Operation	Description	Actions
ls	Lists the directory information	<ol style="list-style-type: none">1. Find the metadata in cache.2. Check if the metadata is valid.3. Enqueue the ``get metadata request" to the upload queue of UM.

Table 3.2 shows the behavior of ``open" and ``close". These two operations are very im-

portant at file access. The ``open" file operation is the begin of every file operations for a file or device. In MoFS, only at the time users need the file (open the file), then client download the file into cache by DM. The ``close" file operation is the end of every file operations for a file or device. In MoFS, only at the time users close the file and also write some data into the file, the client upload the file from cache by DM.

Table 3.2: open and close

Operation	Description	Actions
open	Open a file or device	1. Find the metadata in cache. 2. Check flags for download and write.
close	Close a file descriptor	3. Enqueue the requests into the upload queue of UM.



Table 3.3 shows the behavior of ``read" and ``write". The ``read" and ``write" operations in MoFS work similar to the general operation in Ext3. MoFS only operates the files in cache until close the files.

Table 3.3: read and write

Operation	Description	Actions
read	Read data by a descriptor	1. Find the metadata in cache. 2. Access the file in the cache.
write	Write data to a descriptor	3. Set write flag if it is a write operation.

3.4.2 Advanced File Operations

There are some operations created by MoFS. MoFS provides some advanced file operations for improving the usability. The ``mofs remove cache" command deletes the cache file in the local cache manually. The ``mofs dhash" command checks the hash table usage to determine the type of hash key.



Chapter 4

Implementation

The Chapter 3 explains the design of MoFS, in this chapter we explain how do we realize the MoFS. As we show in Figure 4.1, there are many necessary components for MoFS, including FUSE and SSH client/server. FUSE provides a simple connection between kernel space and user space for a file system. After Linux kernel 2.6, FUSE has been included in the source code of Linux kernel. SSH is very famous on remote control. We have to setup SSH Server at the file server, and SSH Client at all user client side. In next two sections, we show how to implement MoFS Client Application and MoFS Server Application.

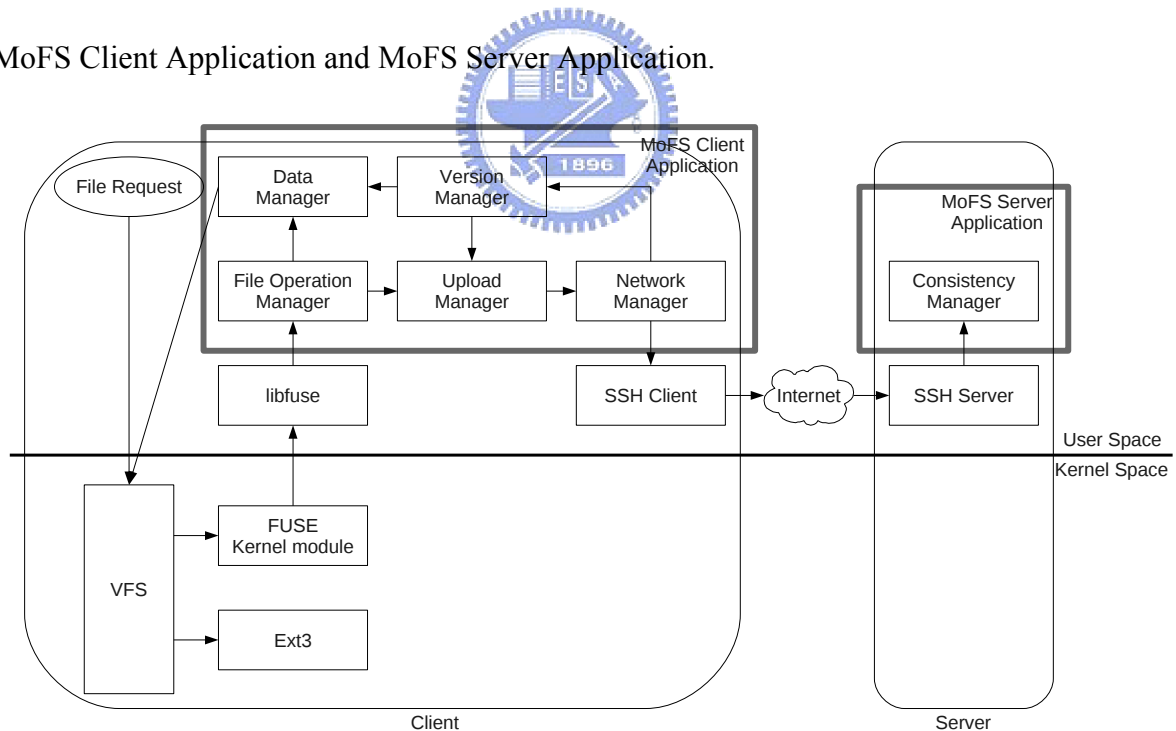


Figure 4.1: Architecture of MoFS

4.1 MoFS Client Application

MoFS Client Application has five components that we mention before. The File Operation Manager (FOM) implements file system operations, like open, read, write, close, and so on. The Data Manager (DM) maintains local file cache and metadata information. The Network Manager (NM) monitors network status and executes network commands. The Upload Manager (UM) optimizes files in the upload queue. The Version Manager (VM) controls the version of the local file cache.

4.1.1 File Operation Manager (FOM)

The FOM is a implementation of the API which provide by FUSE library. It implements the file system operation function and maps the functions to FUSE. In Table 4.1, we shows the relationship between shell commands / function calls and file system operations. The command ``ls" (list directory contents) for example, after enter the ``ls" command in shell, it calls the ``opendir" file system operation to check whether the directory is existing or not. Next, it read all contents name of the directory into a buffer for showing results. At the end of reading the directory, it calls ``readdir" to close the session of reading. Finally, showing the details of contents in buffer, by using ``getattr" file system operation with parameter of conetent name in buffer. By default, FUSE runs multi-threaded, so that we need to concern for mutual execution. Although solving the mutual execution may waste time, multi-threaded can increase the use of perception.

Table 4.1:

Shell command/Function Call	Used File System Operations
ls	opendir, readdir, relosedir, getattr
open	access, open / create / open + mknod, utimens
close	flush, release, utimens
write	truncate, write, utimens
read	read, utimens
rm	unlink

4.1.2 Data Manager (DM)

The DM maintains local file cache and metadata information. The usage of data structure influences the operation performance of the local file cache and metadata information. Using array is a basic way to maintain the cache, but array does not have good extensibility. Next, we think may be linked list is a good data structure for cache design. We analysis the behaviors of maintaining local file cache and metadata information. There are three main behaviors: insert, delete, and search. We also discover that before insert or delete any entry in the cache, we need to know wether the same named entry is in the cache or not. Hence, the ``search" behavior is the most frequent one of the three behavior. Clearly, linked list is not suitable for our design. We have to use a $O(1)$ data structure for implement. Using hash data structure is a choice. As not all the mobile device have large disk or memory. System space usage is another issue so that the hash size can not be too large. Generally, hash has two method to implement, linear probe and linear linked list. Linear probe has no extensibility and linked list has low efficient at large file search. Here we use the design of combination with hash and tree. If there are large amount

of metadatas in one bucket, using tree structure can let the time in $O(\log n)$.

The Figure 4.2 shows the combination with hash and tree. First, we use absolute path (whole directory and file name) as the hash key. At the insertion, the DM records how many nodes are used in every buckets. If the usage of buckets is not balance, it can change the hash key manually to improve the performance.

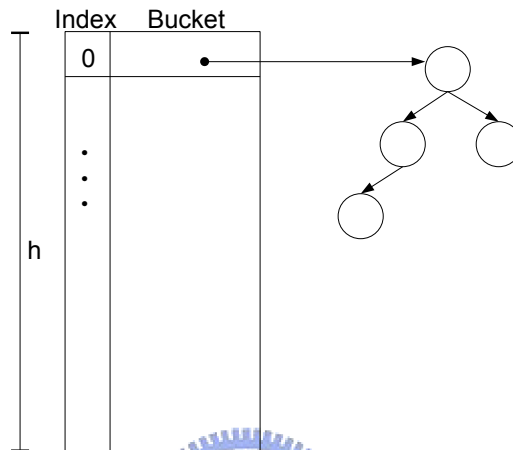


Figure 4.2: Hash Tree Structure

4.1.3 Network Manager (NM)

The NM is to check the network status between client and server. In our design, we use ICMP packets to identify the status into Strong, Weak, and Disconnect. The main purpose of the use of ICMP is to confirm the existence of network connect and the network quality. The ping utility is a famous tool to send ICMP ECHO_REQUEST to network hosts and like our purpose. We refer to the source code of ping utility to implement the status checking. The NM also provides a ssh wrap function and a sftp wrap function to execute the commands on file server.

4.1.4 Upload Manager (UM)

The UM queues the requests of file operations and transfers it to file server by NM. We construct four queues, each queue has their own priority. These four queues queuing different kind of requests, with the priorities for ``get metadata'', ``download object'', ``upload object'', and ``remove object''. As the ``get metadata'' queue has any requests, the UM executes them first. Before a request puts into ``remote object'' queue, the UM checks whether the same name file is in the ``upload queue''. If the request matches in ``upload queue'', the UM deletes the request which is in the ``upload queue'' and remove the file on server (if exist). By using the optimization, the UM can decrease the communication waste.

4.1.5 Version Manager (VM)

The VM controls the version of the local cache. As the server notify comes back from the CM, VM downloads the different part of file between client and server. And VM revises the file to the latest version. The VM also maintains a small command daemon for the MoFS advanced commands.


4.2 MoFS Server Application

Only one component is implemented in MoFS Server, the Consistency Manager (CM).

The CM is a relay process to record the access state of the files at server. There are two ways to realize the record method. The first method is to create a small client-server program which intercepts the SSH session from MoFS client. After essential operations, the small client-server program resend the session to the real SSH server then do the real procedure. But this implementation is just like to build a new server at the server side. This method decreases the

convenience of deployment. The second method is to use `LD_PRELOAD` [20], the environment variable under UNIX-like operating system. `LD_PRELOAD` instructs the loader to load additional libraries into a program, beyond what is specified when it is compiled. It allows users to add or replace functionality when they run a program and can be used for beneficial purposes. We implement a shared library at server side to let the `sftp-server` loading it. The shared library catches two functions, `open()` and `socket()`. The `open` function is called when client download or upload a file. At the same time, the shared library logs the informations. The `socket` function is called when the `sftp` connection be created. When a connection be created, it check the server file repository periodically for file staleness.

4.3 Installation



The requirements of a MoFS client are `libfuse`, FUSE kernel module, SSH client program, and MoFS Client Application. Generally, FUSE kernel module and SSH client program are the default packages of Linux distributions so that for installation users only need to setup `libfuse` and MoFS Client. The requirements of MoFS server are SSH server program (SSH daemon) and MoFS Server Application (shared library). SSH server program is an important remote shell for Linux servers so that the installation of server is only to modify the configuration about `sftp-server` (let the `sftp-server` preloads the shared library).

The requirements of Coda are RPC2 library, Light Weight Processes (`lwp`) library, Recoverable Virtual Memory (`rvm`) library, Coda kernel module, and Coda (Vice and Venus). Although in many package maintenance systems of different Linux distributions have the packages of RPC2 library, `lwp` library, and `rvm` library, the installation dependency should be carefully considered.

Chapter 5

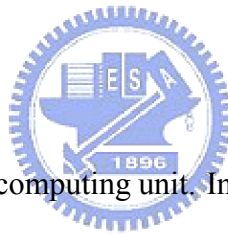
Experiments and Analyses

5.1 Preliminaries

5.1.1 Hardware

We construct a testing environment to evaluate MoFS performance. The hardware specifications are listed as follow:

- File Server



The file server is a PC-based computing unit, Intel Pentium 4 3.0GHz CPU with 512 MB of RAM using an internal 3.5 inch SATA 7200rpm 80GB disk with Gigabit Ethernet NIC. The operating system is Debian 5.0 Linux kernel 2.6.26.

- File Client

A file client program can be executed on a variety of devices, including laptop and smart-phone.

- Laptop (Client A)

Intel Core2 Duo CPU T7300 2.00GHz with 512MB of RAM using an internal 2.5 inch SATA 7200rpm notebook hard drive with 802.11abg NIC. The operating system is Debian 5.0 Linux kernel 2.6.26.

- Smartphone (Client B)

Samsung S3C2442B ARM920T 400MHz with 128MB of RAM using 256MB NAND flash and 512MB microSD with 802.11b/g SiP-M. The operating system is Fyp (Debian 5.0) Linux kernel 2.6.28 (Openmoko Neo FreeRunner).

5.1.2 Networks

We use three kinds of networks to construct the experimental environment. Clients can connect with the file server by Ethernet, wireless LAN, or the mobile network.

- Ethernet: In our experimental environment, the file server and file client connect to Internet by a 10/100 Mbps switch.
- Wireless LAN: We use an Atheros 5212 802.11b/g WLAN interface on a PC to be the AP.
- Mobile: We can use CHT GSM/3.5G mobile network.



5.1.3 Tools

We use some tools to evaluate the capability of file operation, read/write throughput, and communication cost.

- `Bonnie++` [21][22] is an industry-standard file system benchmark used to benchmark ideal performance in a uniform and repeatable way. We utilize `Bonnie++` to evaluate file operations frequency and file read/write throughput.
- The `iftop` [23][24] listens to network traffic on a named interface and displays current bandwidth usage by pairs of hosts. We use `iftop` to measure the efficiency in the use of communication cost.

- The `netem` provides Network Emulation functionality for testing protocols by emulating the properties of wide area networks. We use `netem` to emulate GPRS network between File Server and Laptop.

5.2 Environment Setup

We construct a testing environment to evaluate MoFS performance as shown in Figure 5.1. In this experimental environment, we construct several networks, including LANs, 802.11 wireless networks, and 3rd generation mobile networks, all connected to the Internet. We setup several MoFS clients in these networks. The MoFS clients, with a comprehensive computing power, can access the MoFS server via the Internet. The first kind of clients is PC-based computing unit.

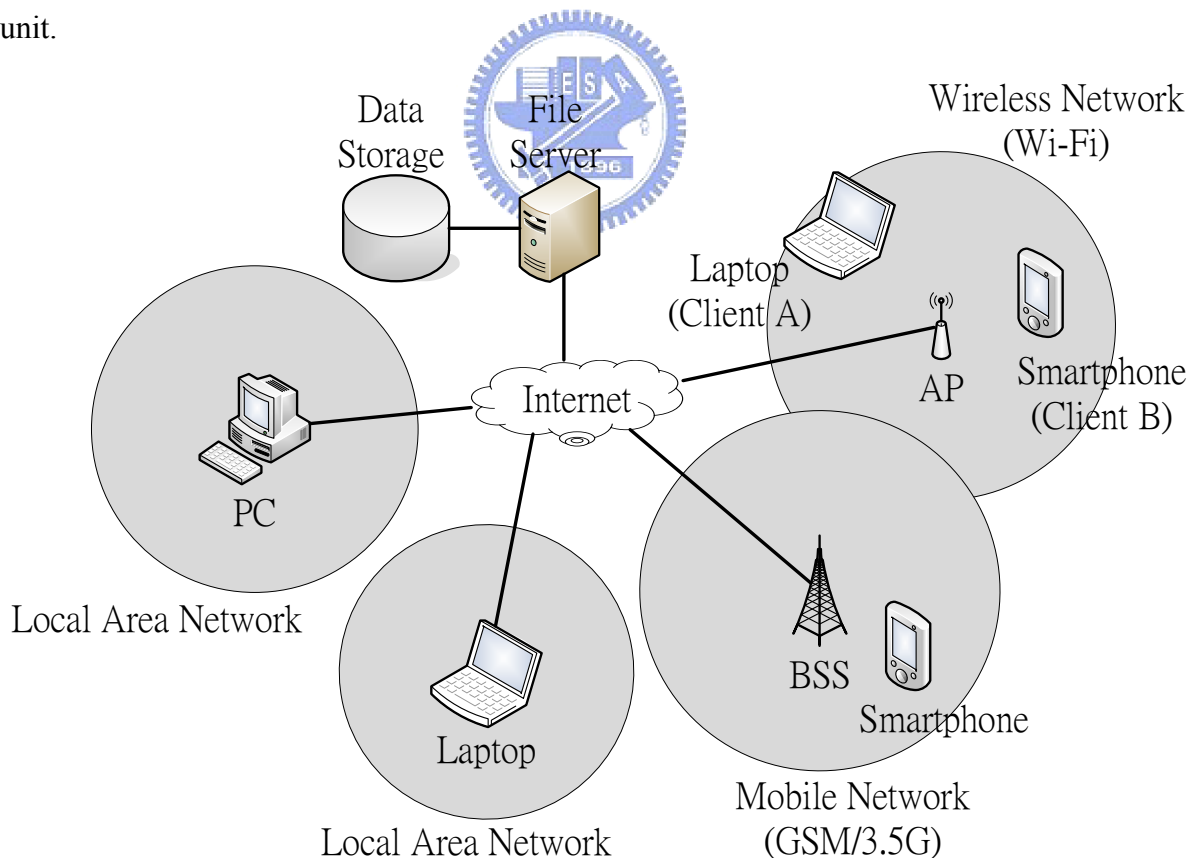


Figure 5.1: Experimental Environment

5.3 Experiments

The main concern of all file systems is general file operation. The mobility is just a feature of a mobile file system. Original file operation can not be reduced by adding mobility, so that we need to know the file operation performance of MoFS. The network communication cost is also an important issue that we want to realize. In our experiments, we intend to analyze the performance about operation latencies, read/write throughputs, and communication costs. We compare different file system to show their difference. However, FUSE just provides file operations libraries and APIs, not a fully file system. The ``FUSE" represents a simple file system just mirrors the root directory to mount point by using FUSE libraries and APIs.

5.3.1 Operation Latencies

In the first part, we use three operations (create, stat, and remove) to measured the operation latencies on laptop and smartphone.



- Exp1.1 : Latencies of MoFS/Laptop

To measure the latency of operations for laptop by wireless network, we connect the laptop to the AP. The connection bandwidth is between 800KBps to 1MBps. In Exp1.1, we use Bonnie++ to create 2048 files in 20 times, and the files sizes are between 0.5MBytes to 1MBytes.

In Table 5.1 and Figure 5.2, Ext3 and FUSE are local file systems, NFS is a traditional network file system, and Coda, AshFS, and MoFS are file systems supporting mobility. We can find that all the file systems do not faster than Ext3 at create and remove, because they are still the essence of ext3. AshFS and MoFS are based on FUSE libraries and APIs, so that their create frequency can not higher than FUSE. NFS propagates all operations by

the network directly, so the bottleneck of NFS is the network bandwidth so that it has lower frequency. Since MoFS cache the metadata in the memory, MoFS has the fastest frequency at the stat operation. FUSE still fetches the metadata from disk so that its frequency is lower than Ext3 and MoFS. Coda concerns at many properties and has complex design may cause its performance not very well. For remove operation, AshFS and MoFS put the work unit into a waiting queue, so that they does not execute the remove operations immediately. Since the remove operation behavior of AshFS and MoFS, they look like fast at remove operation, but actually their remove speed are more slower.

Table 5.1: File Operations Frequencies of MoFS/Laptop (1/Sec)

	Ext3	FUSE	NFS	Coda	AshFS	MoFS
Create	37.4	35.15	0	31	31.75	34.19
Stat	35.1	30.13	1	33.71	33.6	41.85
Remove	131.4	102.56	53	111.57	154.7	156.7

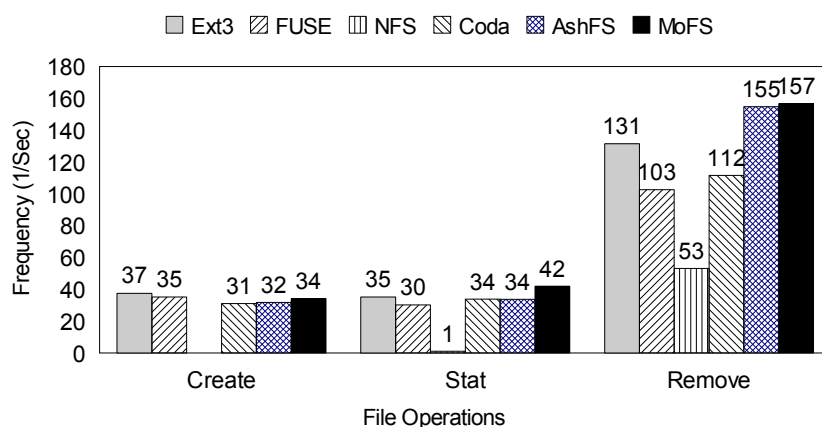


Figure 5.2: File Operations Frequencies of Laptop

- Exp1.2 : Latencies of MoFS/Smartphone

To measure the latency of operations for smartphone, we connect the smartphone to the AP. Since the wireless module (chip and antenna) of the smartphone has poor quality, the connect bandwidth is between 300KBps to 500KBps. The last experiment works on smartphone, with little disk size, so that we create the files with small size. The setting of Bonnie++ for smartphone is to create 2048 files in 20 times, and the files sizes are between 200KBytes to 500KBytes.

In Table 5.2 and Figure 5.3, Ext3 and FUSE are local file systems, NFS is a traditional network file system, and AshFS, and MoFS are file systems supporting mobility. Coda needs the supporting both in kernel space and user space. The kernel source code of Coda is pulled into kernel.org repository. Since the Coda user space program may need much disk size, Coda can not work under smartphone. Although the storage in smartphone is microSD card different to a hard disk, the results of smartphone are similar with the results of PC. Ext3, the local file system, is the highest performance file system for file operations. NFS propagates all operations by the network directly, so the bottleneck of NFS is the network bandwidth so that it has lower frequency. Since the lower computing power and the more threads in process, MoFS is not faster than FUSE and AshFS for stat operations and remove operations. MoFS does not has the greatest performance for file operations, but supporting mobility for file system.

Since Ext3 and Coda use kernel cache in memory, they can use memory more efficient by linux kernel. For the limited resource device, using MoFS can get good performance of local file operation and mobility.

In the duration of experiment, we discover that the amounts of cache file in AshFS have the limitation at about 5000 files. The large amounts of operations also crash the AshFS network communication. Since AshFS has limitation at the numbers of files and operations, Ashfs can

Table 5.2: File Operations Frequencies of MoFS/Smartphone (1/Sec)

	Ext3	FUSE	NFS	AshFS	MoFS
Create	14.45	9	5.8	7	6.5
Stat	27.35	20.8	9	18.5	15.7
Remove	115.65	71.8	29	54.25	52.1

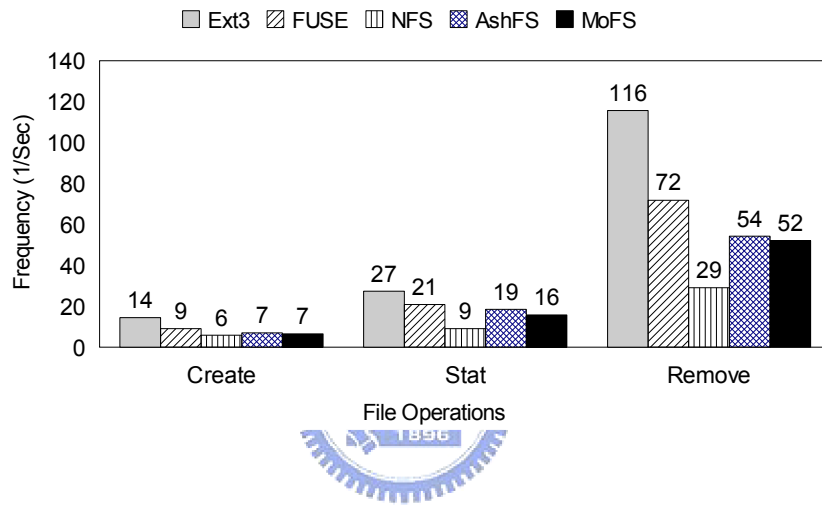


Figure 5.3: File Operations Frequencies of MoFS/Smartphone

only suit for a small scale environment.

5.3.2 Read/Write Throughputs

In the second part, we evaluate the read/write throughput of each file system on laptop and smartphone.

- Exp2.1 : Read/Write throughputs of MoFS/Laptop

To evaluate the read/write throughput for laptop by wireless network, we connect the laptop to the AP. The connection bandwidth is between 800KBps to 1MBps. In Exp2.1, we use Bonnie++ to write a 1GB file to evaluate the write speed of these file systems.

After write the file, Bonnie++ reads the file to evaluate the read speed of these file systems. In Table 5.3 and Figure 5.4, Ext3 and FUSE are local file systems, NFS is a traditional file system, and Coda, AshFS, and MoFS are file systems supporting mobility. We can find that all the file systems do not faster than Ext3 at read and write, because they are still the essence of ext3. AshFS and MoFS are based on FUSE libraries and APIs, so that their read/write throughput may not higher than FUSE. NFS propagates all operations by the network directly, so the bottleneck of NFS is the network bandwidth so that it has lower throughput. Since Coda writes the file into local cache first then propagates the file for a period of time delay, its throughput is similar to Ext3.

Table 5.3: Read/Write Throughputs of MoFS/Laptop (KB/Sec)

	Ext3	FUSE	NFS	Coda	AshFS	MoFS
Read	54852.6	54723	11205.1	51655.1	42773.5	54715.2
Write	39951.6	39086	10610.3	35835.2	36729	38891.2

- Exp2.2 :Read/Write throughputs of MoFS/Smartphone

Since the wireless module (chip and antenna) of the smartphone has poor quality, the connect bandwidth is between 300KBps to 500KBps. The last experiment works on smartphone, with little disk size, so that we write the files with small size. The setting of Bonnie++ for smartphone is to write a 40MB file to evaluate the write speed of these file systems. After write the file, Bonnie++ reads the file to evaluate the read speed of these file systems.

In Table 5.4 and Figure 5.5, Ext3 and FUSE are local file systems, NFS is a traditional network file system, and AshFS, and MoFS are file systems supporting mobility. Since the

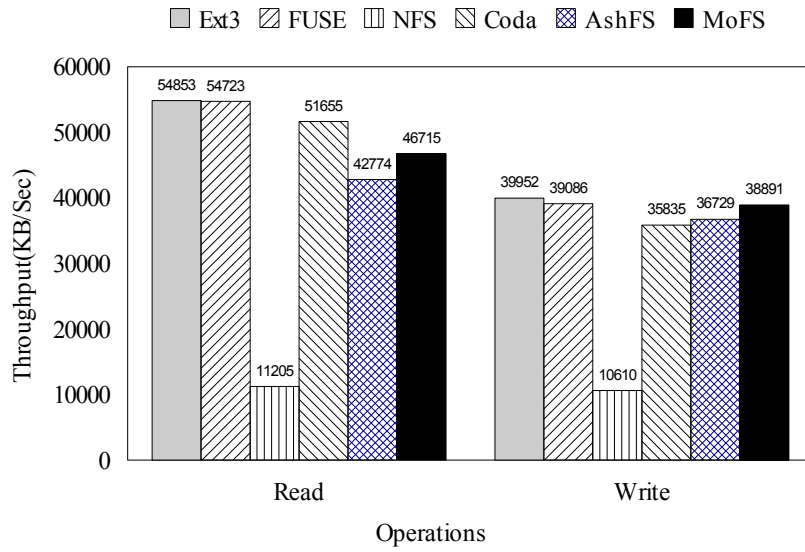


Figure 5.4: Read/Write Throughputs of Laptop

Coda user space program may need much disk size, Coda can not work under smartphone. Although the storage in smartphone is microSD card different to a hard disk, the results of smartphone are similar with the results of PC. In Exp2.2, NFS has better throughput than NFS in Exp2.1, because we use small size of file in Bonnie++. Since the lower computing power and the more threads in process, MoFS is not faster than FUSE and AshFS for stat operations and remove operations. MoFS does not has the greatest read/write through, but supporting mobility for file system.

Table 5.4: Read/Write Throughputs of MoFS/Smartphone (KB/Sec)

	Ext3	FUSE	NFS	AshFS	MoFS
Read	1056.4	783.4	563.8	475.67	631.8
Write	1323.6	1177	772.4	573.33	684

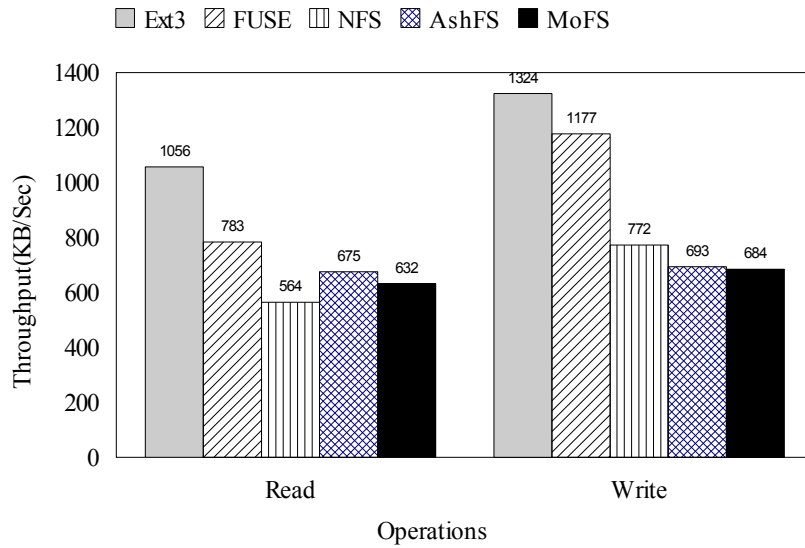


Figure 5.5: Read/Write Throughputs of MoFS/Smartphone

Since the large memory can cache more data for kernel, Ext3 and Coda have more cache hit. MoFS can works well at limited hardware platform, and supports mobility.



5.3.3 Communication Costs

In the third part, we calculate the communication costs (network workload) between one server and two clients.

- Exp3 : communication costs (network workload)

Since the wireless module (chip and antenna) of the smartphone has poor quality, the connect bandwidth is between 300KBps to 500KBps. The smartphone use an AP to connect to the Internet. We use `iftop` at file server to calculate the network communication of specific connection. At client, we execute a sequence of file operations including create, remove, read, write, and list, then record the total network usage.

$$S = \{(h_i, f_i, o_i, t_i)^+\}$$

- h_i :host
- f_i :file
- o_i :file operation
- t_i :time

Here is two clients (hosts) and one server. First, host 1 reads file 1, so that host 1 downloads file 1. After reading file 1, host 1 modifies some data then closes file 1, so that host 1 uploads file 1 to the file server. Since host 1 accesses file 1, file server records the information of host 1. Second, host 2 reads the same file, file 1, so that host 2 also downloads file 1. After reading file 1, host 2 modifies some data then close file 1. Then, host 2 uploads new file 1 to the file server. The file server gets the data of file 1 and records the information of host 2. The file server finds that host 1 also accesses file 1, so file server sends a notification automatically to host 1 to announce host 1 updating file 1.

In Table 5.5, NFS is a traditional network file system, AshFS, Coda, and MoFS are file systems supporting mobility. Since this experiment is the calculation of communication costs, we just compare with the network file systems. The connection bandwidth is between 800KBps to 1MBps on laptop. All the file system use the same testing sequence to measure the communication cost (like). NFS executes all the file operations during the network so that it has the most network usage. AshFS caches file in local cache but it need to check the file status frequently, this may exhaust the communication costs. Using ``rsync" is the other factor cause AshFS exhaust the communication. MoFS does not frequently check the file status, and resolve the network. Since MoFS does not implement

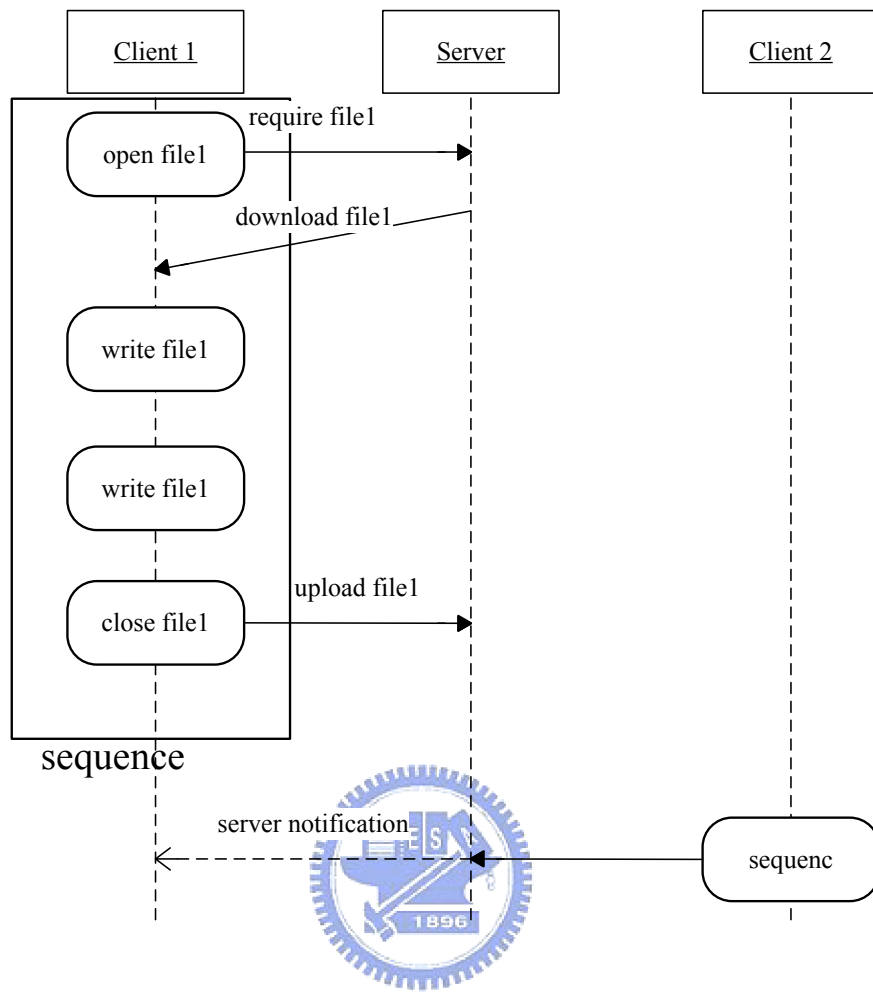
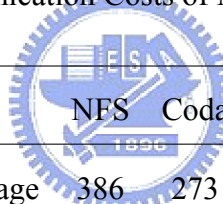


Figure 5.6: Experiment flow for Communication Cost

differential upload, the communication cost of AshFS is similar to the communication cost of MoFS. If we use differential upload instead of whole file upload, the communication cost of MoFS can be more less.

Table 5.5: Communication Costs of MoFS/Laptop (KB)



	NFS	Coda	AshFS	MoFS
Network Usage	386	273	233	225

Chapter 6

Conclusion

In this research, we design and implement a mobile file system, named MoFS. The new proposed file system, MoFS supports disconnected operations with a persistent cache and decreases the communication costs at file status notification from the MoFS server. Taking FUSE as its underlying file system library, MoFS requires no kernel patch in its deployment. Also, by integrating with the SSH protocol, MoFS can provide user authentication and file encryption to protect communication sessions between file servers and clients.

The MoFS client program has been implemented on both x86- and ARM-based platforms (Openmoko Freerunner GTA02). We made several experiments to compare MoFS and other file systems (Coda, AshFS, NFS, Ext3) in terms of processing latencies, read/write throughputs, communication costs. From the experiment results, we conclude that MoFS/x86 and MoFS/ARM have the best performance in retrieving file stats and removing files. For read/write throughputs, MoFS/x86 and MoFS/ARM have the best performance. Also, comparatively, we observe that less network traffic (communication costs) is required when running the MoFS client on Laptop. In short, MoFS has faster processing time, better read/write throughputs and lower communication costs.

References

- [1] 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*, vol. 39, no. 4, pp. 447--459, 1990.
- [2] P. J. Braam, "The Coda Distributed File System," *Linux Journal*, p. 6, 1998.
- [3] J. Lui, O. So, and T. Tam, "NFS/M: An Open Platform Mobile File System," in *Proc. 18th International Conference on Distributed Computing Systems*, 26--29 May 1998, pp. 488--495.
- [4] "FUSE: Filesystem in Userspace." [Online]. Available: <http://fuse.sourceforge.net/>
- [5] I. Voras and M. Zagar, "Network Distributed File System in User Space," in *Proc. 28th International Conference on Information Technology Interfaces*, 2006, pp. 669--674.
- [6] M. E. Hoskins, "SSHFS: super easy file access over SSH," *Linux Journal*, vol. 2006, no. 146, p. 4, 2006.
- [7] M. Szeredi, "SSH Filesystem," 2005. [Online]. Available: <http://fuse.sourceforge.net/sshfs.html>
- [8] L.-Y. Chang, "AshFS: A Lightweight Mobile File System Supporting Disconnected Operations," Master's thesis, Department of Electrical and Control Engineering, National Chiao Tung University, 2008.
- [9] Marionraven and T. M, "HTTPFS," 2006. [Online]. Available: <http://httpfs.sourceforge.net/>

- [10] B. Callaghan, B. Pawlowski, and P. Staubach, "NFS Version 3 Protocol Specification," RFC 1813 (Informational), Internet Engineering Task Force, Jun. 1995. [Online]. Available: <http://www.ietf.org/rfc/rfc1813.txt>
- [11] B. Atkin and K. Birman, "Network-Aware Adaptation Techniques for Mobile File Systems," in Proc. Fifth IEEE International Symposium on Network Computing and Applications NCA 2006, 2006, pp. 181--188.
- [12] S. Lehtinen and C. Lonvick, "The Secure Shell (SSH) Protocol Assigned Numbers," RFC 4250 (Proposed Standard), Internet Engineering Task Force, Jan. 2006. [Online]. Available: <http://www.ietf.org/rfc/rfc4250.txt>
- [13] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," RFC 4251 (Proposed Standard), Internet Engineering Task Force, Jan. 2006. [Online]. Available: <http://www.ietf.org/rfc/rfc4251.txt>
- [14] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Authentication Protocol," RFC 4252 (Proposed Standard), Internet Engineering Task Force, Jan. 2006. [Online]. Available: <http://www.ietf.org/rfc/rfc4252.txt>
- [15] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Transport Layer Protocol," RFC 4253 (Proposed Standard), Internet Engineering Task Force, Jan. 2006. [Online]. Available: <http://www.ietf.org/rfc/rfc4253.txt>
- [16] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Connection Protocol," RFC 4254 (Proposed Standard), Internet Engineering Task Force, Jan. 2006. [Online]. Available: <http://www.ietf.org/rfc/rfc4254.txt>
- [17] B. Cornell, P. Dinda, and F. Bustamante, "Wayback: A User-level Versioning File Sys-

tem for Linux," in Proceedings of Usenix Annual Technical Conference, FREENIX Track, 2004, pp. 19--28.

[18] A. Boukerche and R. Al-Shaikh, "Towards Building a Fault Tolerant and Conflict-Free Distributed File System for Mobile Clients," vol. 2, April 2006, pp. 6 pp.--.

[19] A. Helal, A. Khushraj, and J. Zhang, "Incremental Hoarding and Reintegration in Mobile Environments," 2002, pp. 8--11.

[20] D. Dwyer and V. Bharghavan, "A Mobility-Aware File System for Partially Connected Operation," SIGOPS Oper. Syst. Rev., vol. 31, no. 1, pp. 24--30, 1997.

[21] R. Coker, "The Bonnie++ Benchmark," 1999. [Online]. Available: <http://www.coker.com.au/bonnie++/>

[22] I. Dowse and D. Malone, "Recent Filesystem Optimisations on FreeBSD," in Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference. Berkeley, CA, USA: USENIX Association, 2002, pp. 245--258.

[23] P. Warren and C. Lightfoot, "iftop: Display bandwidth usage on an interface," Feb 12th 2006. [Online]. Available: <http://www.ex-parrot.com/pdw/iftop/>

[24] D. A. Bandel, "Focus on Software," Linux Journal, vol. 2002, no. 99, p. 12, 2002.

