A Distributed File System over Unreliable Network Storages

Kyunghee Oh
Cyber Security Research Division
Electronics and Telecommunications Research Institute
Daejeon, Korea
Email: khoh@etri.re.kr

Dooho Choi
Cyber Security Research Division
Electronics and Telecommunications Research Institute
Daejeon, Korea
Email: dhchoi@etri.re.kr

***Abstract*— Nowadays, an individual uses multiple ICT devices such as PCs, laptops, smart phones and others. And the content files are not dedicated to a specific device, but shared by the devices. One of the sharing services is the personal cloud computing. Users can backup, synchronize, share and manage their files with it. But most cloud systems have their own dedicated interfaces and it is not easy to use files in various applications. We propose a distributed file system which works with the legacy internet protocols. Applications on devices can share files with the general file i/o interface, and our system enhanced reliability of file storages in both aspects of failures of servers and security risks.**

***Keywords*— personal cloud storage, clustered file system, erasure code**

I. INTRODUCTION

An individual uses multiple ICT devices such as PCs, laptops, smart phones and others. And the contents are not dedicated to a specific device, but shared by the devices. To share content files, they should be uploaded to and downloaded from a file server. One of that services for sharing files between devices is personal cloud computing. Many internet companies provide those services. There are Apple iCloud, Google Drive, Microsoft SkyDrive, Dropbox, Amazon Cloud and many others. Users can backup, synchronize, share and manage their files with them. But most cloud systems have their own dedicated interfaces, so that other applications cannot access the contents directly. It restricts the user's choice of applications. The third party applications are also restricted by the storage service provider.

For the enterprise environment, many distributed storage systems are competing. HDFS, MooseFS, iRODS, Ceph, GlusterFS, and Lustre are some of them [1]. Internally, personal cloud service systems include such distributed file system. The performance is very important in storage systems for the enterprise environment to handle large amounts of data. For that reason, well-defined dedicated protocols and systems for file storages are adopted. And the systems should be managed carefully to maintain its availability.

The distributed file system for enterprise environment has good performance, but the overall system is expensive to make up and not easy to maintain personally. Another weakness of cloud storages is single point of failure problem. Usually they provide a single service access point to a user, and the failure of access point will stop the entire system of a user from working.

We propose a distributed file system that is easy to maintain, and of which the interfaces to applications are so general that applications can access files transparently.

Our system is based on the standardized protocols which transfer files over the internet. It transfers files using the legacy internet protocols, and storage servers are also legacy servers. To get a file, the client tries to access multiple storage servers simultaneously. Although some of them are not accessible, the client can recover the file if there are enough number of available servers. So we don't have the single point of failure problem. Applications of client devices can access files through general file i/o interfaces. Only the file system software at the client device handles all of the complexity. It also assures the reliability of files against failures of servers and malicious security attacks.

II. DISTRIBUTED STORAGES

A. Distributed File System

A distributed file system (DFS) provides a permanent storage which consists of several federated storage resources. Clients can create, delete, read or write files in DFS. Unlike local file systems, storage resources and clients are dispersed in a network. Files are shared between users in a hierarchical and unified view. Files are stored on different storage resources, but appear to users as they are put on a single location. A distributed file system should be transparent, fault-tolerant and scalable. Users should access the system regardless where they log in from, be able to perform the same operations on DFSs and local file systems. Data manipulations should be at least as efficient as on conventional file systems. In short, the complexity of the underlying system must be hidden to users. Network and server failures may make data and services unavailable. If several users access data concurrently, data integrity and consistency would be broken. A fault tolerant system should not be stopped in case of transient or partial failures. Although the amounts of files and nodes increase dynamically and continuously, the system should efficiently manage them without much overhead [1].

One of applications using distributed file system is the high performance computing. It relies on distributed environments

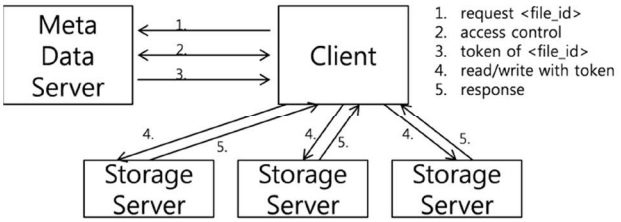

Figure 1. An example of DFS architecture

to process and analyze large amounts of data. As the amount of data increases, the need to provide efficient, easy to use and reliable storage solutions has become one of the main issues for scientific or cloud computing.

Fig. 1 depicts a typical example of DFS architecture. The client accessing a file gets the location data of the file from the meta data server and then it accesses the file. The client may not access the storage servers directly in some architecture. The meta data server may provide overall interfaces for file i/o to the client.

Peer-to-peer storage is a different kind of distributed storage. Users contribute their local storages to the peer-to-peer network, and share the storages. But P2P storage has problems to be solved. No technical solution ensures that nodes always cooperate. A malicious node can deny service when it is asked to serve other peers. For fair usage, out-of-band rules should be provided [2].

B. Personal Cloud

The personal cloud is not a tangible entity but some types of services that give the ability to store, synchronize and share content on a relative core, moving from one platform, screen and location to another [3].

Resources for personal cloud can be supplied by a service provider or users may setup their own devices to supply the resources.

A network-attached storage (NAS) devices may provide cloud service such that backups and archiving. Moreover users can access them at anytime from anywhere. Some more technologically proficient users create a home-made cloud system by connecting an external USB hard drive to a Wi-Fi router. This enables both wired and wireless computers to access the USB hard drive and use it for storage or for retrieving files a user needs to share on the network thereby acting like a cloud.

Online cloud, or public cloud, supplied by service providers, are constructed using pooled, shared physical resources and accessed by the Internet. Storage maintenance tasks are offloaded from users, but an individual or organization has little control over the system in which the online cloud is hosted.

Using cloud service may increase security risks. In cloud based architecture, data is replicated and moved frequently so attackable area increases. And as the cloud system is shared by many customers, unauthorized access may happen if the cloud service provider does not manage the system secure enough. Another risk of cloud service is that the service company may change the quality of service or go out of business.

III. FILE SYSTEM OVER UNRELIABLE NETWORK STORAGES

Each DFS scheme has its own characteristics for its purpose and usage environment. A cloud service provider or enterprise environment adopts enormous physical systems and dedicated protocol for DFS. The storage servers are distributed, but management authority is centralized.

Individuals may purchase the storage service from cloud storage providers. But it has some drawbacks as previously described. P2P storage also has deployment problem. And NAS provides a network file system, but it is not a distributed system and has the weakness of single point of failure problem.

We propose a distributed file system for personal usage. Our system does not have central management, and we don't use a specific dedicated protocol, so it is easy to deploy.

Compared with the scheme of Fig. 1, there is no meta data server in our scheme. Meta data are stored as a file in storage servers like other files. Storage servers are just legacy file servers such as FTP or WebDAV servers which may be governed by other authorities. Any complexity of the file system resides in the client system. We do not presume the reliability of storage servers. We assume that some of them may not be accessible sometimes or even that attackers may intrude into the storage servers.

Fig. 2 depicts the write and read process of a file. At the write process, a file is divided into chunks of some predefined size. As our scheme is based on the protocols transferring files, a chunk file is the unit of transferring data over the network, and the size of a chunk influences the performance of file i/o.

Next, chunks are encrypted. Although the storage servers are not secure, that is, others may get the data in storage servers, they cannot acquire useful information.

Encrypted chunks are coded for redundancy. We adopted Reed-Solomon erasure code [4][5], which is the most popular erasure code for various applications, especially for storage systems. A chunk splits into k blocks, and (n-k) parity blocks are generated, where n is the number of storage servers. By the choice of n and k, the redundancy overhead differs.

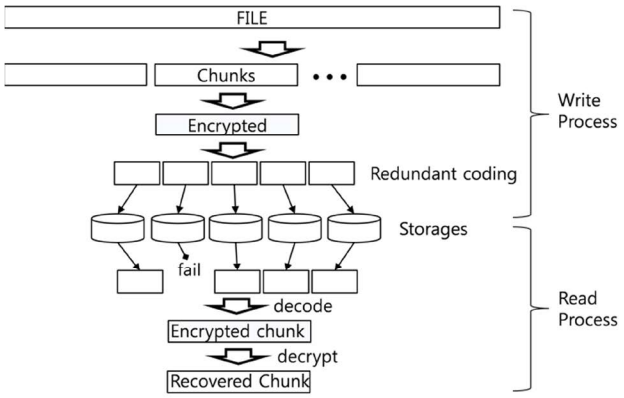

Figure 2. Write and read process

Each encoded block is transferred to each storage server. The blocks are transferred over the legacy file transferring protocols, such as HTTP or FTP. Any other protocols can be adopted if they can send and receive files. It is easy to deploy such storage servers, and usually users already have some accessible servers supporting such file transfer.

When the file system in the client device gets a request to read some data in a file, it accesses storage servers and gets the proper chunks. Although some storage servers are not accessible temporally, it can decode them into the encrypted chunk if there are at least k available blocks, and finally it decrypts it to get the original chunk.

Actually, because the file system manages local cache, read/write processes do not lead to the file transfer immediately. The write process influences only to the local cache files. User's explicit request to synchronize or unmounting the file system causes the actual file transfer. And any first read operation causes the transfer of file chunks, and it does not need to transfer chunks if they have already been cached.

IV. EVALUATION

Our implementation is available at GitHub repository [6]. The file system interfaces are built on FUSE [7], and it is mountable at Linux systems.

Table I and Fig. 3 describe the test environment. There are five storage servers, three WebDAV servers and two FTP servers. As the round trip time (RTT) shows, WebDAV servers are located a little far away. One of storage servers, Storage E is a low performance embedded device. Tests with two clients have been performed at Client D and Client E. Each of Client D and Client E is located at the same subnetwork of Server D and Server E. For our scheme, parameters of Reed-Solomon code are set to $n = 5$ and $k = 3$, that is, 3 message blocks and 2 redundant parity blocks are generated from a chunk. For example, a 10MB chunk will be encoded into five 3.3MB blocks and the redundant overhead is about 67%. For write process, every 3.3MB block is transmitted to each server concurrently. For read process, the client requests the blocks from servers concurrently, and it can make the chunk when it completes receiving any three blocks.

Fig. 4 is the result of a 10MB file transmissions comparing the transmission time over HTTPS/FTP and ours. The left half is the test result of Client D, and the right half is the result of Client E. Bars of *A-E* are the time to upload and download a file with each server over legacy HTTPS/FTP protocols and Bar *Avgs* are the average of *A-E*. Bar *Ours* are the time to transfer a file through our file system. Ours take similar or a little longer time to the average times. Considering our scheme includes encryption and encoding, the overall overhead is not so much.

Applications may read and write only some portion of a file, not the whole file. In such case, it does not need to transfer the whole file to read or write the portion. Our file system only transfers the chunks that include the requested portions of a file. As the chunk size is smaller, the response time gets smaller. Table II shows that the time to read only one byte (peek) from a 10MB-size file depends on the size of a chunk. Peek operations of applications take shorter time as the chunk size gets smaller. But as the chunk size gets smaller, the time to read or write the whole file increases very much. Fig. 5 shows it.

TABLE I. STORAGES IN THE TEST ENVIRONMENT

Nodes	***Type***	***RTT*** *(ping from Client D)*
Storage A	WebDAV, HTTPS	182.4 ms
Storage B	WebDAV , HTTPS	145.7 ms
Storage C	WebDAV, HTTPS	285.0 ms
Storage D	Desktop, Linux, FTP	0.27 ms
Storage E	Wrieless Router, OpenWRT, FTP	12.2 ms
Client D,E	Laptop, Linux	

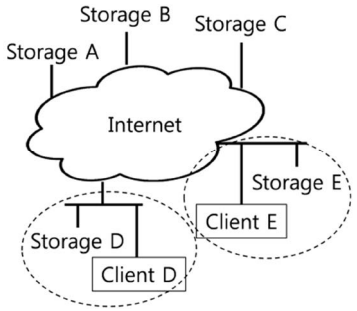

Figure 3. Test envrionment

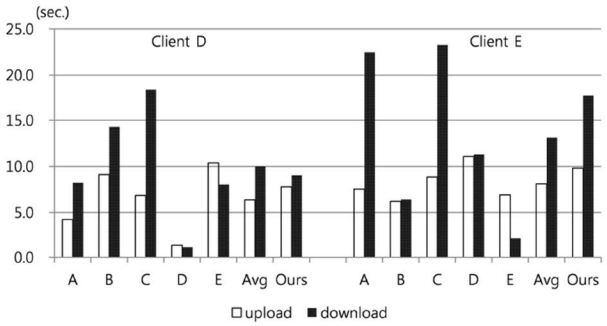

Figure 4. Upload & download time of 10 MB file

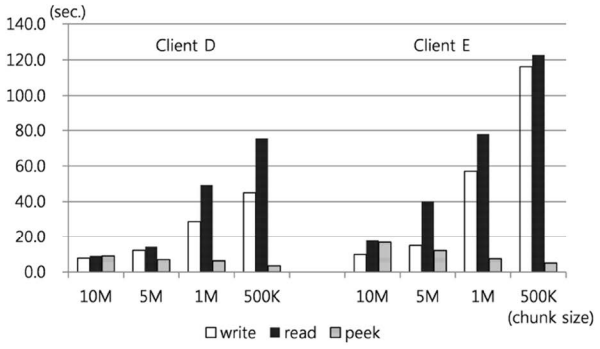

Figure 5. Write, read and peek time for each chunk size

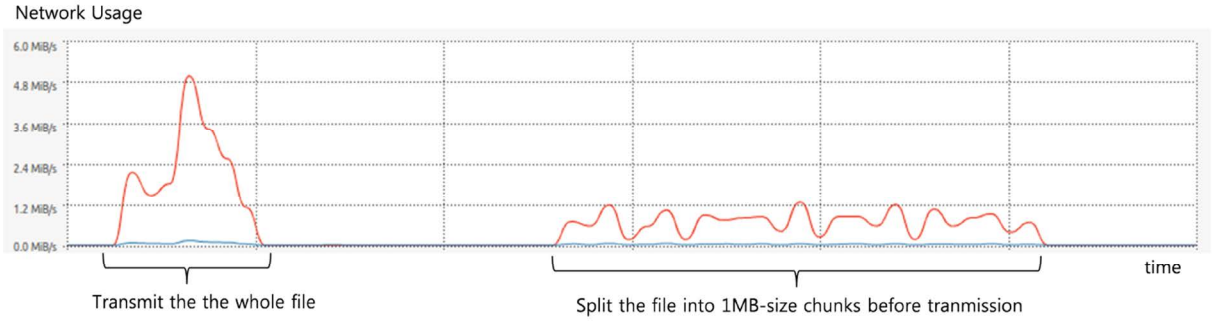

Figure 6. Network usage while writing a 10MB file

TABLE II. THE ACCESS TIME OF 10MB & 10KB SIZE FILES FOR EACH CHUNK SIZE

Operation (chunk size)	**Client D**		**Client E**	
	10MB	***10KB***	***10MB***	***10KB***
Read(10M)	9.02	1.13	17.79	1.51
Write(10M)	7.80	2.27	16.74	5.46
Peek(5M)	6.96	-	12.02	-
Peek(1M)	6.23	-	7.45	-
Peek(500K)	3.37	-	5.11	-

Smaller chunk size means more transfer connections over legacy protocols, of which overhead are not negligible. Most transferring protocols lie on TCP, which has connection setup steps and slow-start congestion control. Moreover, HTTPS lies on SSL that processes security handshake for every connection.

Fig. 6 is the network bandwidth usage graph when sending a file to storage servers with write process. It compares the bandwidth usage when the chunk size is large enough to contain the whole 10MB-size file and when a file splits into ten 1MB-size chunks.

When the whole file does not split into chunks, there are only five encoded blocks. So there are five TCP streams and the left big peak in Fig. 6 shows the bandwidth usage of five overlapped stream. On TCP, data are not transferred at the maximum speed at the start time. It gradually speeds up by TCP Slow-start control.

In the case that a file splits into ten chunks and each chunk are encoded into five blocks, there are 50 TCP streams for transmission. Each TCP stream sends about 333KB of data, and it is too small to reach the available maximum speed of the network. Finally, the overall time to send the whole file increases.

Table II also compares the transfer time of a big and a small files. The size of the bigger file is a thousand times larger than that of the smaller, but the difference of elapsed time is not so much.

We can conclude that the smaller chunk size shortens the response time of accessing portions of a file, but it causes a big overhead to access the whole file.

Our scheme is not for scientific computing or enterprise environment, but for personal file sharing over multiple devices. The test result shows that the file transfer delay overhead of our scheme is not so much against legacy file servers. But our scheme provides redundancy for recovery and availability. And it ensures the security even though the storages are not trusted.

For deployment, the only thing that a user should prepare is to install the file system software into their client devices and configure them to connect the legacy file servers. There is no need to configure servers or to negotiate with other peers.

V. CONCLUSION

The need to share files between devices is increasing. The files to share are not only application contents but also the configuration files of applications. A user can have consistent configurations for the same application in different devices. For example, Firefox web browser has the function of synchronize its user preference data with other devices. For this function, Firefox has its own synchronization protocol and there is a dedicated server. But, if the user data files are shared at the layer of file system, applications are synchronized automatically without any additional function of application program. There are various applications which can have the benefit of it. And moreover, applications themselves might be shared.

Network file system conveniently provides file sharing because the file system can be mounted to the client device and applications can access files through the general file i/o interface. There is no need to modify applications to access shared files. But it has some drawbacks compared to the local disk. File i/o takes a longer time, file servers may fail to access, and the files may be leaked to attackers.

Our scheme resolves the performance problem by using cache. Any files ever cached are accessed rapidly almost the same as local files. Distributed redundancy endures server failures, and encryption protects privacy.

Our scheme can be extended to share files between users. File servers may have different access control for read and write. For example, a WebDAV server is usually configured that only privileged users can write files into the server and others can read them. The process of sharing files might be as follows.

1) Each File is encrypted with its own key. Only the owner of the file can generate the key.

2) The owner stores encoded blocks into storage servers.

3) The owner publishes the list of access method (ex. URLs) of blocks with the encryption key to the file receiver.

4) If there is enough blocks the receiver can read, the receiver can recover the file.

For file sharing, access control is done by the service provider in commercial cloud service. In our scheme, access control is done by sharing the encryption key without any other authority.

ACKNOWLEDGMENT

This work was supported by the K-SCARF project, the ICT R&D program of ETRI(Research on Key Leakage Analysis and Response Technologies)

REFERENCES

[1] Benjamin Depardon, Ga el Le Mahec, Cyril Seguin, "Analysis of Six Distributed File Systems," [Research Report] 2013, pp.44. <hal-00789086>

[2] Dinh Nguyen Tran , Frank Chiang , Jinyang Li, "Friendstore: cooperative online backup using trusted nodes," Workshop on Social Network Systems, pp. 37-42, 2008

[3] Personal Cloud, https://en.wikipedia.org/wiki/Personal_cloud, Wikipedia, retreived Jul. 2015

[4] Plank, James S. "A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems." Softw., Pract. Exper. 27.9 (1997): 995-1012.

[5] RSCODE project, http://rscode.sourceforge.net/

[6] https://github.com/frcgang/Filesystem-over-Unreliable-Network-Storages

[7] Filesystem in Userspace, http://fuse.sourceforge.net/