

A File System Specialized for N-Screen Platforms

Illyoung Choi¹, Jinsuk Baek², Sangchul Kim³, Paul S. Fisher²

¹Minigate Co., Ltd, Seoul, Korea, iychoi@minigate.net

²Department of Computer Science, Winston-Salem State University, Winston-Salem, NC, USA, {baekj, fisherp}@wssu.edu

³Department of Computer Science and Engineering, Hankuk University of Foreign Studies, Korea, kimsa@hufs.ac.kr

Abstract— It is rapidly becoming more and more common that users interact multiple computing devices such as smart phones, smart televisions, tablet PCs, video game consoles, etc. This introduces a new situation which is called the N-screen environment which differs in several ways from the traditional computing environment. To date, there has not been a file system specifically designed for this environment. This paper introduces a new file system designed for the N-screen environment. Specifically, it is designed to run in a virtualization layer which allows for cross-platform operation. The file system is optimized for efficient synchronization between different devices while allowing for disconnected operation, on demand file loading, and encryption. In order to show applicability of the proposed file system, we mainly focus on the practical methods resolving frequently encountered problems on N-screen environment. Our experiment results show that the proposed file system improves file synchronization, which indicates the proposed method can be applied to write-once/read-many systems.

Keywords— *synchronization; N-screen; file systems; virtualization; FAT*

I. INTRODUCTION

As technology advances, many of the tasks traditionally done by the desktop computer are now being taken over by devices such as smart phones, tablets, ebook readers, gaming consoles, and network-connected televisions or blu-ray players. In the average household, the digital media and applications are now spread over many different devices. Unfortunately, data does not generally flow seamlessly among these devices. Rather, each device is like a walled garden, containing its own settings, applications, and data. Although it is possible to manually share data among devices, it requires the explicit direction of the user to do so, and it requires specialized applications, communicating with shared protocols. Even when possible [1], this only allows for the streaming of data between devices, and not copying the data onto the device. So although a user may be able to watch a video from his desktop computer using his cell phone while he is at home, when he leaves home, he will likely lose the ability to watch that video.

Even when the user can copy data between devices [2], there are still a host of issues which need to be managed. Synchronizing the data and tracking modifications to ensure that changes are preserved rather than accidentally overwritten is especially difficult when the devices are running different operating systems and applications. Further, many devices are using file systems which allow only for ASCII characters which causes difficulties when exchanging files with devices

which do allow for international file names. As such, it would be preferable if a uniform file system environment existed which spanned across the different devices.

As such, we propose a new file system which is specialized for the N-screen environment [3, 4]. This file system will handle international file names, data synchronization, data streaming, and disconnected operation. We expect that it will be run in a virtual environment allowing it to work effectively across different platforms potentially using different underlying media. It will not replace the existing low level file systems of devices, but instead run as a file image inside of one. However, in this task, it will be specially designed to avoid creating wasted space or storing redundant information.

The rest of this paper is organized as follows. Section 2 defines design goals of the new file system. In Section 3, we state how to realize the proposed file system. Section 4 discusses the expected performance. Finally, we conclude the paper and suggest future work in Section 5.

II. DESIGN GOALS

A There are many well-known and widely used file systems such as EXT2, EXT3, YAFFS, NTFS, FAT16 [5] and FAT32 [5]. While all existing file systems, equipped with various features, have provided efficient management mechanisms, we need to point out that those systems are designed for other operational purposes rather than focusing on the N-screen environment. Therefore, it is not feasible to apply existing file systems to N-screen environment. For example, many existing files systems mainly consider underlying hardware structures and their operations. This feature imposes unnecessary processing cost and disk space overhead when it is straightforwardly applied to the N-screen environment. The proposed file system minimizes the performance degradation by taking into account unique characteristics of the N-screen environment. Before we discuss practical matters, we should first lay out some design goals for the file system.

A. High Level

The N-Screen file system is a high level file system built to run in a virtual environment [6]. As such, it will manage a virtual file system which is stored in files inside a local, operating-system-specific file system. Therefore, it assumes that the underlying file system manages the direct interaction with the hardware. For example, managing the specifics of block size or dealing with bad sectors will be assumed to be handled by the underlying file system.

PBR																															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Disk ID																User UUID															
UPHV																Padding															
BPC				TCN				RDEN				FS				FCS				DCS				Padding							
Reserved																															
Reserved																															
Reserved(400 Bytes)																PHV															

Figure 1. PBR structure

B. Code/Data Separation

In the N-Screen file system, the primary source of applications will be through application stores [6, 7] or other centrally managed repositories of executable code. This allows us to treat applications differently from code. They do not need to be as thoroughly backed up and frequently cannot be shared across different platforms. Instead, we can reload them from the central repository as needed. As such, we differentiate in the file system between code, such as applications and the operating system, and data which is created or consumed directly by the user. Note that user configuration files should be considered part of the data component and not the code component.

C. Optimized for Infrequent Data Use

Any individual file is not always frequently used. As such, we can avoid unnecessary network traffic by delaying the synchronization of many large files. Instead, we can share these files in an on-demand manner.

D. Synchronization Friendly

We want our file system itself to assist in the process of tracking changes and file contents so that we can know which files have been changed and when without needing a separate tracking program and database.

E. Allows Varing Connectivity

Many of the devices involved are portable and may spend some time either disconnected from each other or connected via slow and/or expensive connections. The file system should be able to function even when some nodes are unavailable and should have different options for data sharing depending on the available bandwidth.

F. Satisfying General Requirements of Files Systems

It should also share some common goals with many local file systems such minimizing disk usage, supporting long file names and international characters, support encryption for security purposes, and minimize processing time.

III. REALIZATION

We have decided to base our new file system on simple and well known file systems, FAT16 and FAT32. However, we could also make similar changes to other file systems.

In FAT16 and FAT32, the disk is divided into four sections: the Partition Boot Record (PBR), File Allocation Table (FAT), Root Directory (RD), and Data Section (DS).

- The PBR is located on the first sector of the disk partition and includes basic information of the file system. It includes the first piece of code that the device will run after checking hardware components.
- In FAT 16 and FAT32, a partition is divided into cluster unit. As the typical cluster size is 4KB, each file may occupy multiple clusters depending on its size. The FAT maintains disk allocation information with a list of entries that map to each cluster on the partition.
- The RD section stores all of the file and directory entries of the root folder.
- The DS has sub directory information and actual data of files.

In its original form, the PBR has a large number of fields which are primarily describing the physical disk. For efficiency, we remove these, instead using the following fields:

- Disk ID
- User UUID
- User Password Hash Value (UPHV)
- Bytes Per Cluster (BPC)
- Total Cluster Number (TCN)
- Root Directory Entry Number (RDEN)
- FAT Size (FS)
- FAT Cluster Start (FCS)
- Data Cluster Start (DCS)
- PBR Hash Value (PHV)

We can support user private storage by using the Disk ID, User UUID, and UPHV. As shown in Figure 1, the amended PBR requires 512 bytes in total.

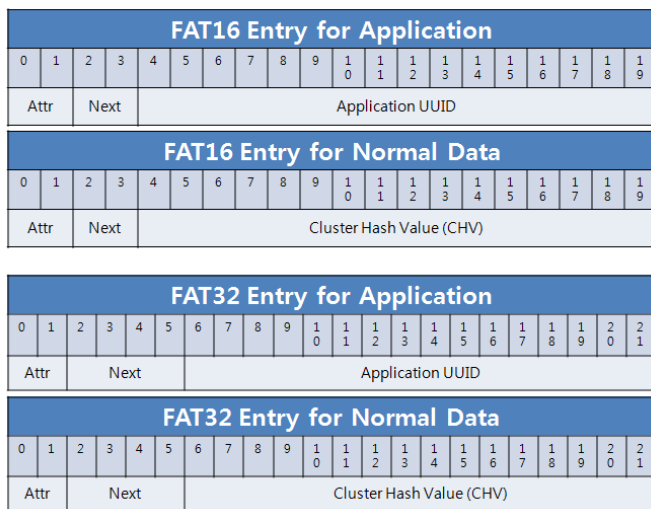


Figure 2. FAT16 and FAT32 entry structure

The FAT is also modified by adding additional attributes. Specifically, for each file we add an “Attribute” field and a second field which serves as either “Application UUID” or “Cluster Hash Value (CHV)”. The Attribute field contains a series of bits which can be set and additional space to hold unique identification values. In the Attribute field, we can set four flags including:

- ATTR_APP
- ATTR_DATA
- ATTR_INITIAL
- ATTR_ONDEMAND
- ATTR_DONE

If ATTR_APP is set, then the file is code or other parts of the application and the second field will be “Application UUID” and will hold a unique identifier for the application. If ATTR_DATA is set, then the file is user settings or data or other data which is not part of a download from a central repository and the second field will be CHV and will hold a hash of the data. FAT entries for both attribute flags are configured in Figure 2.

If ATTR_INITIAL is set, this indicates that the file is small and/or frequently used (for example, user settings). Therefore, the associated clusters should be synchronized every time. If ATTR_ONDEMAND is set, this indicates that the file is large and/or rarely used. As such, it should be synchronized only on demand rather than every time. There is also an additional flag, ATTR_DONE which is used to mark the results of the synchronization process.

Note that moving the hash information into a FAT table makes it easier for whole file system synchronization processes to quickly identify which files have changed. Rather than navigating a series of file entries, instead the FAT can be read directly. This feature allows the system to quickly initialize synchronization process with reduced network traffic volume.

In our system, we do not make any changes to the directory structure because it already supports unicode file names.

Lastly, to enhance data security, encryption should be applied to each cluster except for the PBR. We advise this rather than encrypting it on a file-by-file basis because this will minimize the changes to the encrypted file in the event that there are changes to the plaintext version of the file. Because only changed clusters will have their values modified, only those clusters will need to be synchronized.

IV. EXPECTED PERFORMANCE

In order to evaluate the performance of the proposed file system, we have developed separate test applications for file read/write/hash, and communication overhead. For accurate evaluation of system response time, the average number of CPU clock cycles is considered as a basis metric. We have run these test applications while counting the CPU clock cycles. Our target system is configured with 2.27GHz CPU and 4GB main memory, which is connected to Internet through a link with a capacity of 10Mbps. We also consider MD5 algorithm to generate fixed-size hash value.

Our modifications slightly increase the size of the FAT section as compared to the original FAT16 and FAT32, but not by enough to make a substantial impact on a modern sized disk drive. Tables 1 and 2 show the amount of space overhead for two base file systems having two different disk sizes. As FAT16 has short address field length, it cannot be used for huge file storage, such as those over 100MB.

FAT16-Based Modification (16MB)			FAT16-Based Modification (64MB)		
PBR	512 bytes	0.003%	PBR	512 bytes	0.0007%
RD	3584 bytes	0.02%	RD	3584 bytes	0.005%
FAT	86016 bytes	0.5%	FAT	331776 bytes	0.49%
DS	16687104 bytes	99.4%	DS	66772992 bytes	99.49%

TABLE 1. DISK SPACE REQUIREMENT FOR FAT16-BASED MODIFICATION

FAT32-Based Modification (64MB)			FAT32-Based Modification (1GB)		
PBR	512 bytes	0.0007%	PBR	512 bytes	0.00005%
RD	3584 bytes	0.005%	RD	3584 bytes	0.005%
FAT	364544 bytes	0.54%	FAT	5804032 bytes	0.54%
DS	66740224bytes	99.45%	DS	1067933696bytes	99.45%

TABLE 2. DISK SPACE REQUIREMENT FOR FAT32-BASED MODIFICATION

We expect that the small increase in space overhead will be outweighed by the substantially reduced communication cost and comparison cost during the synchronization process. Once the cold-start file system is assumed, the proposed file system, equipped with on-demand synchronization feature, requires much lower communication cost than those of other file systems. Of course, additional user interactions after the initialization process will introduce additional network traffic. However, we note that this additional network volume will not exceed the existing traffic difference, experienced at the initial synchronization process. Figure 3 shows this effect when 50% of total disk space is used for lower priority synchronization data.

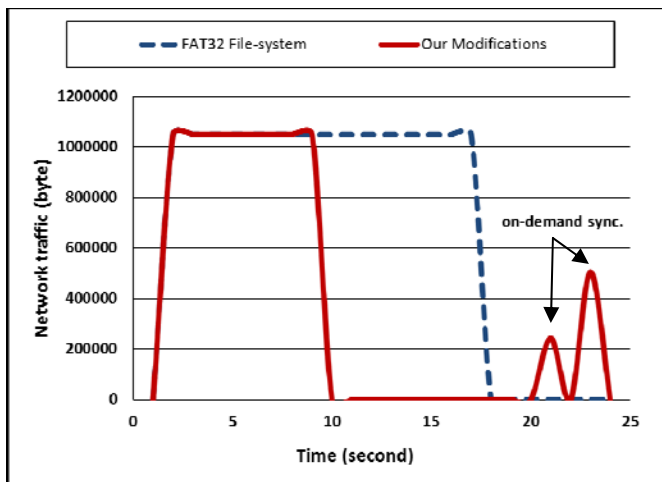


Figure 3. Network traffic cost for synchronization

The synchronization process always involves many comparisons between files. As such, the performance or capability of an entire file system can be limited by this bottleneck effect. The proposed file system resolves this problem by maintaining pre-calculated hash values of the file data in the FAT section. This feature avoids the significant overhead of scanning all of the file data and hashing during the synchronization process, which results in better synchronization performance. In addition, our cluster-based hash value allows the system to quickly recognize which cluster of the file has been modified. This consideration makes a significant impact on the following file merging process. Figure 4 shows the time cost of FAT32-based file-by-file basis comparison and the proposed cluster-by-cluster basis comparison. As we can see, the difference between two file systems becomes more prominent as file size increases.

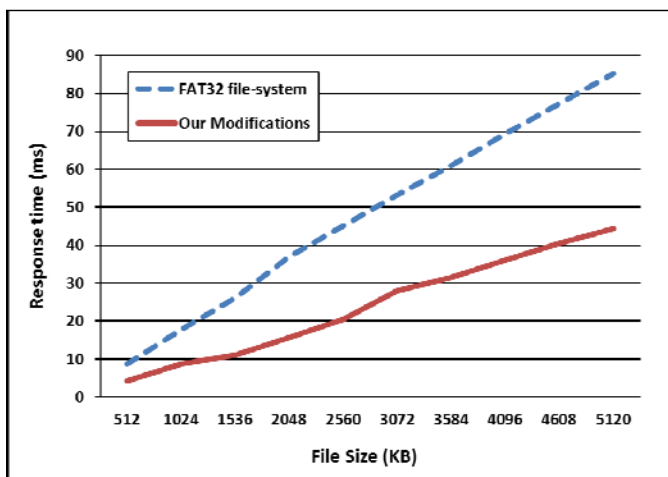


Figure 4. Comparison cost for file synchronization

We need to clarify that this benefit can be only achieved by preparing the pre-calculated hash values of the file data. Therefore, our system is required to generate the hash values at the file writing phase.

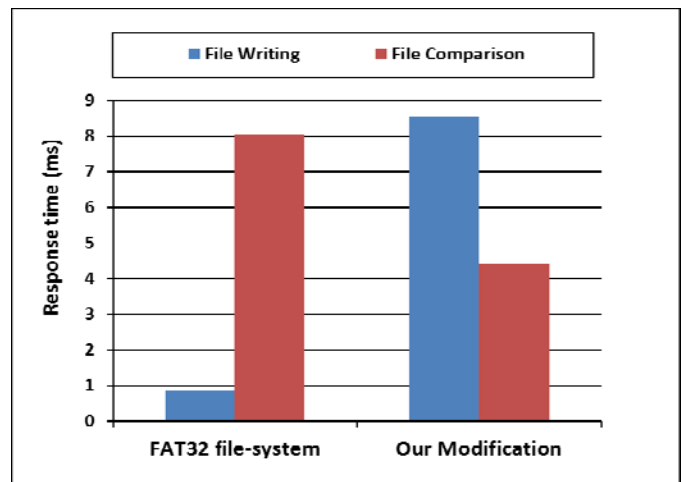


Figure 5. File writing/comparison time cost

Figure 5 shows file writing and comparison overhead for 512KB file. As we can see, the file writing process for the proposed file system consumes much more CPU clock cycles compared to the FAT32 while the file comparison process is showing completely opposite result. Note that it is a number of CPU clock cycles required for single file writing and comparison. It means the total cost is an improvement as long as the pre-calculated hash value is used at least two times. Fortunately, in N-screen environment, users typically and frequently move their data between devices. This process involves multiple file comparisons. Thus, the expected users' behavior will favorably affect the performance of the proposed file system. In addition, our cluster-based encryption/decryption further reduces the file writing cost, although it depends on the specific file sizes involved.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we have identified benefits which could potentially be realized from having a file system built for the N-screen environment. We have identified a set of design goals for such a file system and put forth one system which meets those design goals. In the future, we plan to refine our approach and compare it to other approaches for solving the same problem. Then we will implement a practical system for this file system and run further tests on this implementation.

REFERENCES

- [1] Digital Living Network Alliance, DLNA, <http://www.dlna.org/home/>, (last retrieved in September 2011).
- [2] W. Dees, and P. Shrubsole, "Web4CE: accessing web-based applications on consumer devices," *Proc. of the 16th Conference on World Wide Web*, pp. 1303–1304, August 2007.
- [3] V. Ramasubramanian, et al., "Cimbiosys: A platform for content-based partial replication," *Proc. of the 6th USENIX Symposium on Networked Systems Design and Implementation*, pp. 1–14, April 2009.
- [4] R. G. Guy, et al., "Rumor: Mobile data access through optimistic peer-to-peer replication," *Proc. of ER Workshop*, pp. 254–265, November 1998.
- [5] Microsoft Corporation, "FAT: General Overview of On-Disk Format," *Hardware White Paper*, May 1999.
- [6] Google, Chrome web store, <https://chrome.google.com/webstore/>, (last retrieved in September 2011).
- [7] Apple, App store, <http://www.apple.com/ipad/from-the-app-store/>, (last retrieved in September 2011).