Implementation of cross-platform mounting remote file systems

Maksim Elkin
Institute of Computer Science and Technology
Peter the Great Saint-Petersburg Polytechnic University
Saint-Petersburg, Russia
elkin.mm@edu.spbstu.ru

Nikita Voinov
Institute of Computer Science and Technology
Peter the Great Saint-Petersburg Polytechnic University
Saint-Petersburg, Russia
voinov@ics2.ecd.spbstu.ru

***Abstract* — Modern software solutions for cross-platform file sharing and file synchronization help users from all over the world to manage their files across multiple devices. Though it is quite easy to send a file from one device to another via email, messenger, file exchanger or cloud storage, we found it not convenient enough in some cases.**

We propose more user-friendly approach that helps user manage his files easier and faster. Its idea is to mount remote devices automatically so that user could just install an application once and then find files of remote devices in system explorer of user's device. Such an application requires cross-platform implementation in order to create experience of having all devices connected in a single network. Security considerations must be taken into account to ensure the security of transferring files between devices, and to prevent unauthorized access. Moreover, proposed application should contain viewers for popular file formats to give the user an ability to view any file of a remote device if other installed software cannot manage this.

The Android node of proposed distributed file system was designed and implemented.

Keywords— File sharing, File synchronization, Mounting file systems, Cross-platform mounting, Remote file systems, Distributed file system

I. INTRODUCTION

Storing information in the form of files on a computer has been an important part of human culture for almost half a century. With the appearing of personal computing devices, the task of organizing access to files on different devices of the same user becomes more and more relevant and more and more difficult. For instance, imagine some office worker who has a home computer with Microsoft Windows, a workstation running macOS, and a smartphone with Android. Let's suppose that he is in a meeting with a contractor in his office, and he urgently needs some contracts placed on the workstation and, at the same time, personal documents stored on a home computer. How can he access his files located on different devices running various operating systems?

Easy file access is not the only problem faced by many computer systems owners. The development of digitalization technologies leads to an increase in the number of files that a person needs to manage. The studies show that the amount of digitized information reached 4.4 Zettabytes in 2016[1]. Let's suppose that the mentioned employee has to deal with a variety of work and personal documents. Some of them have to be regularly changed, deleted and added. How to organize centralized file management on different devices so as not to bore the user with the routine actions of connecting to different devices and synchronizing files between devices?

Moreover, the security of storing data has always been a priority among users owning particularly valuable information. Data storage must be protected from unauthorized physical and logical access, devices must exchange data via secure network protocols. These are critical operating conditions for the system managing remote files.

Therefore, providing secure storing, easy access and convenient management of files on various devices of the same user is one of important problems of computer science nowadays.

The objective of the study is to improve user experience when working with files stored on various devices. In order to achieve it, the possibilities and disadvantages of existing solutions are reviewed, their impact on the user experience are analyzed in Section II. In Section III we propose a system that solves stated problem without leading to issues arising in similar systems. Functional specification of such a system will be formed. Section IV describes architecture of the system. Section V contains implementation details of a part of the system on Android platform. Section VI gives an example of use of the system. Section VII concludes the paper.

The proposed system can help owners of complex computer systems around the world facilitate the work with their files. This study can serve as a starting point for further research to improve user experience when working with large amounts of information on a regular basis.

II. DISTRIBUTED FILE SYSTEMS

The stated problem has been generally solved by various distributed file systems (DFS) - methods of storing and accessing files based in a client/server architecture [2]. It must be noted that in this paper we consider practical tools that are ready for use by a wide range of users, in contrast to related technologies, such as virtual file system (VFS), because it is just an abstraction layer on top of a more concrete file system [3]; network file system (NFS), as it is an application layer protocol of DFS [4]; parallel file systems (PFS), because they are a type of DFS that distributes file data across multiple servers and provides for concurrent access by multiple tasks of a parallel application [5]. So, let's review applied hardware and software designed to solve the stated problem.

A. Network-attached storage

Network-attached storage (NAS) is a file level computer data storage server connected to a computer network that provides data access to multiple number of clients [6]. Examples of such devices include products of companies like Synology, QNAP, Western Digital. A NAS device uses its own operating system and integrated hardware and software components to meet a variety of user's needs. NAS connects

to wireless router – enabling multiple users from multiple devices to access the files and data on the network [7].

The advantages of NAS for working with files are as follows:

- Network attached storage system helps to organize and save critical and confidential information in an efficient and accessible manner to valid persons [7]
- User has direct physical access to the NAS so he can set the level of physical data protection as high as he needs

However, using NAS as a universal tool for working with files may be complex to manage.

- The user is forced to buy a separate device in addition to bought ones
- NAS as a technology does not guarantee ease of access to files on various devices. To do this, it is required to install specialized NAS manufacturer software or third-party software, whose quality and functionality may vary
- Centralized file storage forces the user to download files from all devices to NAS, resulting in additional difficulties in managing files if there are a lot of them, and they are often added or become outdated and need to be deleted.

Thus, we can conclude that network-attached storage is well suited for secure storing of files, but not perfect if we want to achieve maximum ease of access and convenience of managing files stored on multiple devices.

B. Public cloud storages

A cloud-based synchronization system is used to store user's files in a central server, owned and governed by a certain entity (eg: an enterprise, or a small company). Users upload their files to this server from one device, and download them on another [8]. Examples of such a storage are Google Drive and Microsoft OneDrive. Let's consider benefits of cloud storage.

- Cloud services provide ample opportunities to view and edit files on various platforms using multifunctional applications
- The security of user data is guaranteed based on the security of the infrastructure of the service provider, many of which are the largest corporations in the world. The long-standing reputation and fault tolerance of servers create confidence that user's files are better protected than if they were on his personal devices

Cloud storages provide easier access to files from different platforms than NAS, but they also have some limitations, as listed in [8].

- Centralized storing introduces inconvenience of file management, as mentioned earlier
- Limitations of the size of available disk space may be critical for some users
- Storing data on remote servers of a third-party organization can be a fundamental problem if these data are secret or particularly valuable

We can infer that cloud storage provides easy access to user's files, but has potential issues with convenience of file management and with confidentiality of stored files.

C. P2P-based File Synchronization Systems

A peer-to-peer-based synchronization system, unlike a cloud-based synchronization system is a decentralized system wherein each peer in the network acts as both a server, as well as a client, to synchronize files between a user's authorized devices. In this system, files are broken down in encrypted pieces, and each peer uploads a certain number of pieces to, and downloads from, other nodes in the system, ensuring that the files are almost always available for synchronization, and that no one peer contains the complete file, thus enforcing privacy and security of the users' data [8]. Accordingly, in order to view the file, user must firstly download it to the device. Such systems, for example, Resilio Connect, solve many problems of cloud storages.

- A high level of privacy and security of data storage is provided. Files are stored on the user's devices, but even unauthorized physical access to one of the devices does not guarantee an opportunity to read the data
- The amount of free disk space is determined by the user
- Decentralized storage does not force the user to transfer files to a dedicated device so that they are visible on other devices. In order to have access to files, it is only required that the host nodes of these files are available on the network. This simplifies management of files if there are a lot of them, and they are tied to different devices.

However, the restrictions of peer-to-peer file synchronizers have the following impact on user experience.

- The lack of random access to the contents of the file leads to a long wait for the file to download if the ratio of its size to the speed of the Internet connection is too high. Moreover, the file may simply not fit on the user's device. And finally, the amount of Internet traffic consumed also matters, especially if someone need to view a small part of a large file (for example, a video clip), and the price of the traffic is significant for the user.
- The availability of the file depends not only on the device where it was originally added, but also on another hosting node. Accordingly, the risk of denial of access to the file is doubled.

We can conclude that peer-to-peer file synchronization systems provide relatively secure data storing and convenient file management, but have issues with ease of access to files.

D. Private cloud storages

The issues of distributed file systems have already been studied previously, and in order to resolve them, hybrid systems based on the principle of private cloud storaging were created, as was done by Jakub T. Mościcki and Massimo Lamanna in «Prototyping a file sharing and synchronization service with Owncloud» [9].

In such systems, shared files are stored on a dedicated user's computer, pre-configured to allow file clients to access files. Files are stored on the server's filesystem (primary

storage) in a transparent directory tree which reflects the client folder structure. It is not only easy to understand, but it also allows to access data directly on storage if needed. Access control is supported for sharing of files and folders. ACLs (Access Control Lists) are not propagated on the file-system level but are enforced when accessing files via the owncloud server [9].

File synchronization is provided by uploading the file from the client device to the server in the corresponding folder. Next, version control of the file is carried out so that both the server and the client have consistent contents. Similarly, content control of folders is supported: if the file was added to the synchronized folder on the server, then it is downloaded to the client device, and vice versa.

The system also provides ample features for working with files: a set of viewing and editing tools for some types of files, offline work, mobile devices support, seamless integration with major desktop environments.

Such systems provide a convenient solution to the problem of the research, because they overcome drawbacks of both public cloud storages and file synchronizers. The user data privacy is entrusted to the user himself, and the size of the available storage is also determined by the disk capacity of the user's computer, and is not limited by the tariff plans of public cloud storage. Moreover, such solutions provide the same convenient access to files and collaboration on them.

However, such systems still force the user to copy files to the server to synchronize them between devices. As we discussed earlier, such centralized file management is not convenient enough in some cases. If files were simply accessible from each device on which they are stored, the user would not have to upload them to a dedicated server.

The proposed system should work as a file server on each target platform. In this case, remote file systems can be entirely mounted, without the need for creating shared folders. Consider what additional features a system must have in order to overcome the shortcomings of similar systems and combine their advantages.

III. Functional requirements for the system of cross-platform mounting remote file systems

A. *File storing security*

In terms of file storing security, the system must meet the level of network-attached storage. In order to achieve this, it must fulfill the following requirements:

1.1. Files must be located on personal devices of user. Therefore, he can provide any required level of physical security of his data. In addition, he will not have to buy an additional device. Finally, the amount of available disk space is limited only by the capabilities of the user.

1.2. Devices must communicate over secure network protocols.

1.3. The system should provide authentication of users to differentiate access to devices.

1.4. Devices must be visible on the Internet in order to have access to them from anywhere in the world.

B. *Convenience of file management*

In terms of convenience of managing files, the system must match the level of peer-to-peer synchronization systems. This requires support for the following features:

2.1 The system must provide decentralized file storing. This means that the files must be stored on the device to which they were originally added. This eliminates the need for the user to manually transfer all files to a single repository, which takes as much time as many files.

2.2 The designed system must mount the entire remote file systems so as not to force the user to select special shared folders and transfer files there, which again is centralized storing, but within the same device.

2.3 The system must support standard file operations between devices and within the same device: copy, move, delete, rename etc.

2.4 Devices must exchange data directly, without intermediaries. This enhances privacy and transmission speed.

2.5 The user must be able to enable and disable external access to the file system on device.

C. *Ease of access*

In terms of ease of access to files, the system must match the level of cloud storages. In order to achieve this, it must also fulfill following requirements:

3.1. The system must have client applications on different platforms so that from any device it is possible to access the file system of any other user's device connected to the system. Access implies the ability to browse any of the folders on the remote system and work with files located there.

3.2. Client applications should provide viewing of files in the most convenient way. The user should be provided with ability to open files in preinstalled third-party viewers. On mobile platforms, it makes sense to add built-in file viewers of the most common formats in case third-party viewers are not available on the device.

3.3. Remote files should provide random access to the content in order to instantly start viewing large files without waiting for them to be downloaded to user's device.

3.4. The remote file system must be mounted in operation system as if it was just a folder on device. This helps to browse remote folders not only in the client application, but also in system and other file browsers.

3.5. Files should only be stored on the devices to which they were originally added. Thus, the availability of data will depend only on the availability of its direct source.

This set of high-level requirements can be treated as a functional specification that we can use to design the proposed system. Also, these requirements can be treated as criteria by which we can further evaluate the results in achieving the goal.

IV. SYSTEM DESIGN

The architecture of the proposed system is shown in Fig. 1

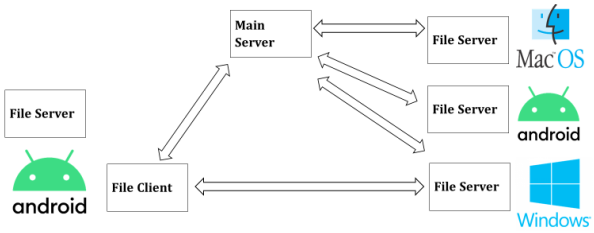

Fig. 1. Architecture of the system of cross-platform mounting remote file systems.

Applications on different platforms act as file clients and allow user to run a file server. A file server provides remote clients access to its file system. File clients allow browsing remote file servers, and perform all other functions specified in the functional specification.

For the entire system, a common virtual file system interface is defined that file clients can use. Each file server implements this interface. Because the capabilities of different file systems are different, the interface includes state of the concrete file system and its capabilities. Since the interaction between the file client and the file server is via the Internet, this interface is essentially a remote API, the same as the API of any other remote service on the Internet. Through separate requests to the file server, the client receives information about the state of its file system and can make changes to it.

To speed up the interaction, both the client and server must support the establishment of a permanent peer-to-peer connection and a pool of such connections for multiple simultaneous asynchronous operations. After establishing a connection, the user can directly manage remote files. The quality of access to them will depend only on the speed of the Internet connection on the file server and file client sides. Moreover, if both the client and the server are registered in the same local network, the data exchange must occur at the speed of this local network.

Requests should be processed by the file server only if they came from an authorized source. If the entire distributed file system was decentralized, the user would have to specify server address, each time he connects to the server. The problem is that the IP address of the file server can change when its Internet Service Provider changed (when using mobile networks and replacing SIM card); when the serving base station changed (if device was physically moved to the area covered by another base station); on demand of the Internet Service Provider (if the user does not have a dedicated static IP); on demand of router, operating system or third-party firewall, or for other reasons. The need to administer each of the servers reduces the system's usefulness in achieving its goal - to improve the user experience of working with remote files. Moreover, the very need to find out the current IP address of the desired server, and then enter it can be difficult for users who are quite new to computer technologies. The main use case for them will only be to view some remote file, so we need to simplify this process as much as possible.

Therefore, to automatically administer file servers and ensure the security of data exchange, a specialized dedicated server is used, common to all users. In fig. 1. it is designated as the Main Server. It is intended for user authentication, authorization of file clients and registration of file servers in the system. After receiving data about file servers from the main web server, the file client can establish a peer-to-peer connection with any of them. Thus, the user only needs to go through the familiar authentication process using a login, password or other credentials.

Certainly, the main web server acts as a single point of failure, that is, when it stops working, the entire distributed file system starts to work partially. Already established connections between the file client and the file server continue to function, but it is not possible to obtain data about the servers and their addresses on the network. Thus, with a short-term failure of the main web server, the current file operations will not be canceled, and the user may not even notice the problems encountered. Long-term failures of the main web server should be eliminated using standard practices of supporting server infrastructure, such as duplication of equipment, load balancing, and others. Other configuration of the main server's execution environment, the selection of the appropriate hardware and software is up to the system developers.

Having a common web server opens up the possibility of using cloud storage technologies for further work on the system. For example, adding storage servers will allow user to expand the amount of available disk space or use backup and automatic recovery of user-selected files, folders, or entire file systems.

The proposed system can be treated as functioning if the main server, file client, and at least one file server is functioning. Application that works both in file client mode and in file server mode is a Subsystem of the described system. There are three Subsystems shown on the Fig. 1: Android Subsystem, Windows Subsystem and macOS Subsystem. In this paper we describe the implementation of the Android Subsystem. In order to maximize ease of access to files, it is necessary to implement similar subsystems for Windows, MacOS, Linux, iOS and other platforms.

V. IMPLEMENTATION OF SUBSYSTEM ON ANDROID

Physically, the subsystem looks like an application for a smartphone, as shown in Fig. 2.

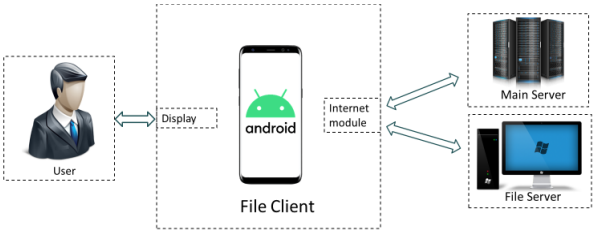

Fig. 2. Android subsystem of cross-platform mounting remote file systems.

The user interacts with the application through the display of his smartphone. The application interacts with servers through the Internet connection module.

To fulfill the requirements of the functional specification, the application must contain many software modules, each of which implements some functionality. Inside the application, the exchange of information between modules and the user is carried out by program classes structured as a multilayer architecture Model-View-ViewModel (MVVM) [10], as shown in Fig. 3.

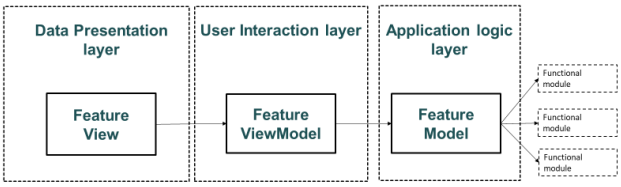

Fig. 3. File client's architecture layers.

The best user experience is provided by applications with maximum responsiveness of the graphical user interface, which cannot be achieved without asynchronous exchange of information. Therefore, the transfer of the user's intentions is carried out by means of a direct call to the classes, but the data are displayed according to the Observer design pattern. The used class (to which the dependency arrow points, as shown in Figure 3) stores information about its state in a special object (Observable), and the class that uses it (from which the dependency arrow comes out, the Observer) subscribes to changes in this state. The Observable stores a list of observers and notifies them when its state has changed [11]. This eliminates the need for the used class to have a reference to the Observer, and to callback objects. Thus, each class has links only to those classes that it itself uses. This approach is called the rule of unidirectional dependencies, it helps to split responsibilities of classes, increase cohesion and reduce coupling [12].

Each application feature visible to user has a corresponding class on the View, ViewModel and Model layers. The Model layer contains the logic of the application, on the basis of which it interacts with functional modules and forms the state of this feature in the system. The ViewModel layer transfers user actions to the Model layer and, based on its state, forms the state of the user interface. The View layer displays this state on the smartphone screen.

Let's review functional modules of file client, describe principles of their work and technologies used to implement them. Structure of Android Subsystem is shown in Fig. 4.

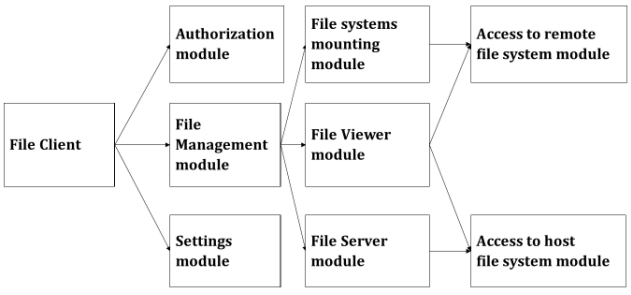

Fig. 4. Functional modules of Android Subsystem

Work with the system starts with authentication of user by email and password (requirement 1.3 of the functional specification, hereinafter referred to as FS). The authorization module automatically communicates with the main server to grant user access to remote file systems. Additionally, a one-time password is sent to the email, if this is specified in the settings. The settings module also stores other user preferences.

After authorization, the file management module receives from the main server a list of file servers registered in the system and available at the moment. Data exchange with servers using the HTTPS protocol (FS 1.2) is implemented in the C ++ dynamic library, an application in the Kotlin language interacts with it through the JNI interface. Information about the server contains data for connecting to it via the Internet (FS 1.4). After selecting a file server, a peer-to-peer connection (FS 2.4) is established and contents of the remote file system root are displayed.

The remote file system access module uses file system interface that remote file server implements. This eliminates the need to interact with the central server and allows to store files directly on the device (FS 1.1, 2.1, 3.5). Accordingly, a file server itself manages scope of access and file permissions, but for more convenient file management, the root of file server should contain all file disks (FS 2.2). Similarly, the host system access module also implements a file system interface so that a user can access files on the device he uses. The user can start file server and then other file clients of this user will also get access to these files (FS 2.5).

The list of files and other user interface elements are based on the components of both the Android OS and AndroidX library [13]. Clicking on some folder displays contents of this folder (FS 3.1), clicking on some file launches built-in viewer, if available for the format of this file (FS 3.2), otherwise the user is prompted to select another viewer from the list of already installed programs. The file context menu allows you to view information about this file, or apply standard file operations to it (FS 2.3).

The file viewer module contains tools for viewing files of the following MIME types:

- text/* - using Android WebView [14]
- image/* - using Glide library [15]
- audio/*, video/* - using ijkplayer library [16]
- application/pdf - using Android PdfRenderer [17]

Viewing files is carried out with restrictions imposed by various versions of the Android OS and third-party libraries. Viewers use the interface of random access to the contents of the file so that the user can quickly navigate to the desired part of the file (FS 3.3).

The file systems mounting module allows user to view remote file systems not only in the application, but also in the system or other file browsers (FS 3.4). On the Android platform, this is implemented using the Android Content Provider [18] and Android Storage Manager [19]. FUSE technology, used to access remote files through file descriptors of the operating system, is only supported on Android 8.0 and higher. Therefore, on older Android devices,

a separate server with socket access is launched to provide external viewers with random access to remote file.

Part of the system of cross-platform mounting remote file systems is implemented on the Android platform. The subsystem fulfills all the requirements of the functional specification.

VI. PROCESS OF MOUNTING REMOTE FILE SYSTEM

An example of usage of the implemented system is shown in the following screenshots. We launched the system on both a smartphone with Android OS and a computer with Windows OS. Both subsystems work as a file client and as a file server. In Fig. 5, we can see the window of Windows Explorer, which displays all the disks in the file system and also the mounted file system of the phone (redmi-note-7).

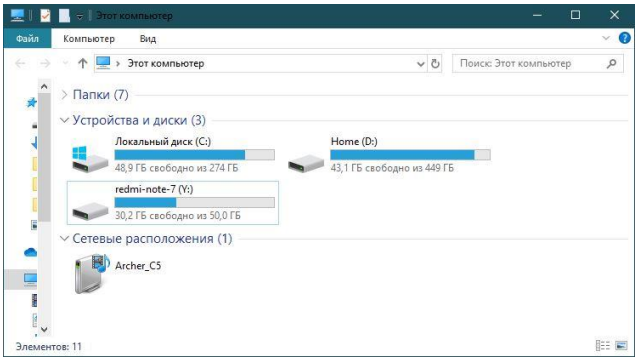

Fig. 5. Mounted remote Android file system

Fig. 6. Mounted remote Windows file system

Similarly, on the smartphone we can browse mounted file system of PC on Windows. Fig. 6 shows launched system file manager on Android 8.0.

Disk space of computer (‘C:’ and ‘D: Home’ folders in Fig. 6) can be browsed so user can work with files on PC as if they were stored on a smartphone. At the same time, the subsystem on Windows additionally mounts the folder with user documents for faster access (‘Max’ folder in Fig. 6).

VII. CONCLUSION

The described system is designed to improve user experience when working with remote files. In order to achieve this goal, it solves the problem of providing secure storing, easy access and convenient file management on various devices of the same user. However, the convenience, security and ease are relative and subjective concepts, therefore, to compare the described approach with others, we use a qualitative research method - we consider the functionality of similar systems and principal, unresolved issues that the user could potentially encounter when using these systems. We contend that a system that combines the advantages of other systems and solves their fundamental issues is a qualitatively better system in terms of achieving the goal.

In Section III the requirements for such a system were specified and in Section V it was indicated that the implemented system meets all the requirements. Therefore, we contend that it surpasses similar solutions in terms of improving user experience when working with remote files. In other words, we contend that mounting remote file systems is actually more convenient for user than transferring or synchronizing files between them.

Further evaluation of user experience requires an extensive and long survey of users after the market introduction of technology. Nevertheless, in the era of many various personal devices with different operating systems, the problem of convenient file management will always remain relevant.

REFERENCES

[1] W. Xia et al., "A Comprehensive Study of the Past, Present, and Future of Data Deduplication," in Proceedings of the IEEE, vol. 104, no. 9, pp. 1681-1710, Sept. 2016.

[2] M. Jayanand, M. A. Kumar, K. G. Srinivasa, and G. M. Siddesh, “Big Data Computing Strategies,” Handbook of Research on Securing Cloud-Based Databases with Biometric Applications, pp. 72–90, 2015.

[3] E. V. Sharma, E. M. Varshney, and S. Sharma, Design and implementation of operating system. Bangalore: University Science Press, 2010.

[4] A. B. Karuvally, B. Hameem, A. J. Sundar and J. P. Joseph, "Enhancing Performance and Reliability of Network File System," 2018 International CET Conference on Control, Communication, and Computing (IC4), Thiruvananthapuram, 2018, pp. 317-321.

[5] R. Filgueira, M. Atkinson, Y. Tanimura, and I. Kojima, “Applying Selectively Parallel I/O Compression to Parallel Storage Systems,” Lecture Notes in Computer Science Euro-Par 2014 Parallel Processing, pp. 282–293, 2014.

[6] A. Lanka and A. Gargeyas, "Remotely Accessible, Low Power Network Attached Storage Device," 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT), Coimbatore, 2018, pp. 1083-1088.

[7] K. Shaikh, S. Soni, V. Shah, K. Thakkar, and K. Belwalkar, “Network Attached Storage,” International Research Journal of Engineering and Technology, vol. 06, no. 01, pp. 1413–1416, Jan. 2019.

[8] Z. Mehdi and H. Ragab-Hassen, “File Synchronization Systems Survey,” Computer Science & Information Technology (CS & IT), 2016.

[9] J. T. Mościcki and M. Lamanna, “Prototyping a file sharing and synchronization service with Owncloud,” Journal of Physics: Conference Series, vol. 513, no. 4, p. 042034, Nov. 2014.

[10] W. Sun, H. Chen, and W. Yu, “The Exploration and Practice of MVVM Pattern on Android Platform,” Proceedings of the 2016 4th International Conference on Machinery, Materials and Information Technology Applications, 2016.

[11] Liu Jicheng, Yin Hui and Wang Yabo, "A novel implementation of observer pattern by aspect based on Java annotation," 2010 3rd International Conference on Computer Science and Information Technology, Chengdu, 2010, pp. 284-288.

[12] M. Fowler, “Reducing coupling,” IEEE Software, vol. 18, no. 4, pp. 102–104, 2001.

[13] D. Zelenchuk, “AndroidX Test Library,” Android Espresso Revealed, pp. 271–280, 2019.

[14] P. Hazarika, Rahul Raj CP and S. Tolety, "Recommendations for Webview Based Mobile Applications on Android," 2014 IEEE

International Conference on Advanced Communications, Control and Computing Technologies, Ramanathapuram, 2014, pp. 1589-1592.

[15] Bumptech, “Glide,” GitHub. [Online]. Available: https://github.com/bumptech/glide. [Accessed: 30-Mar-2020].

[16] Bilibili, “ijkplayer,” GitHub. [Online]. Available: https://github.com/bilibili/ijkplayer. [Accessed: 30-Mar-2020].

[17] Google, “PdfRenderer,” Android Developers Documentation. [Online]. Available: https://developer.android.com/reference/kotlin/android/graphics/pdf/PdfRenderer. [Accessed: 30-Mar-2020].

[18] W. Jackson, “Android Content Providers: Datastore Concepts,” Android Apps for Absolute Beginners, pp. 415–472, 2017.

[19] Google, “StorageManager,” Android Developers Documentation. [Online]. Available: https://developer.android.com/reference/android/os/storage/StorageManager. [Accessed: 30-Mar-2020]