applied sciences

MDPI

Article

An Active File Mode Transition Mechanism Based on Directory Activation Ratio in File Synchronization Service

Mingyu Lim

Department of Smart ICT Convergence, Konkuk University, 120 Neungdong-ro, Gwangjin-gu, Seoul 05029, Republic of Korea; mlim@konkuk.ac.kr; Tel.: +82-2-2049-6270

Abstract: In this paper, we propose an active file mode change mechanism in which the file synchronization system of cloud storage automatically changes files in a directory of a client to the online or local mode by considering tradeoff between local storage usage and file access time according to directory activation ratio. When the directory activation ratio rises above a certain threshold, the proposed active file mode change mechanism selects online mode files in this directory based on file access delay time and local storage usage and changes them to the local mode to reduce file access delay of active IoT clients. When the directory activation ratio falls below the threshold, the active file mode change mechanism selects the local mode files based on the last access time and local storage usage and changes them to the online mode to increase available local storage. Experimental results show that the proposed active file mode change mechanism can control when and how much the client can reduce and increase the local storage usage and the file access delay by changing file mode parameters according to the requirements of various IoT devices.

Keywords: active file mode change; directory activation ratio; file synchronization; local mode file; online mode file

check for updates

Citation: Lim, M. An Active File Mode Transition Mechanism Based on Directory Activation Ratio in File Synchronization Service. *Appl. Sci.* **2023**, *13*, 5970. https://doi.org/10.3390/app13105970

Academic Editors: Ce Li and Bei Guan

Received: 1 March 2023
Revised: 26 April 2023
Accepted: 11 May 2023
Published: 12 May 2023

Copyright: © 2023 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

A cloud file synchronization service keeps the contents of files on a local device and in remote cloud storage the same. With a file synchronization service, users can always keep their files on multiple devices up to date through cloud storage, and they can also use a backup function. As file synchronization services become more common, the capacity of local storage and remote cloud storage increases, and the capacity of the two storages become different. When the local storage of a new client device has a smaller capacity than the existing cloud storage, the new device cannot accommodate all the files in the cloud storage. When local storage runs out, users must free up space on local storage by deleting or moving files that are no longer needed.

To solve the problem of lack of local storage, commercial cloud file synchronization services such as Dropbox and OneDrive provide the functionality to change files from local mode to online mode. The local mode of a file is a normal mode in which file contents exist in both the local and cloud storage. On the other hand, an online mode file keeps the original content only in cloud storage. In the local storage, the online mode file deletes file contents and keeps only file attributes. Therefore, an advantage of the online mode file is that local storage can be saved by deleting only the file contents while maintaining the file attributes on the local device. However, when a client accesses an online mode file, it has a disadvantage in that the access delay time may be long because it must first download the entire file content from the cloud storage and change it to the local mode. The larger the file size, the longer the latency to access these online mode files. When there are many local mode files, local storage is used as much as the file size allows, and when there are many online mode files, file access latency occurs at the cost of saving local storage. In other words, there is a tradeoff between local storage usage and file access time.

Existing research on file synchronization [1–7] focuses on methods to improve synchronization performance to efficiently process large files for many users, but do not support online and local mode change functions for synchronized files. Some commercial cloud file synchronization services [8–11] provide the functionality to change the online and local modes of files, but this function only allows users to manually change the mode of selected files on an on-demand basis. There are also cloud storage and distributed file systems [12–20] and other multi-tiered storage systems [21–30] that use similar approaches to the proposed file mode transition mechanism by prefetching or caching files among storages. However, as they focus more on the performance of file access in costly high-layered storages, cache hit ratio or prediction accuracy are the most important performance metric.

In this paper, we propose an active file mode change mechanism that actively changes the files selected in the directory by the file synchronization system to online or local mode according to the directory activation ratio. The active file mode change mechanism first defines the directory activation ratio, which quantifies how active the files in the directory are. If a file is accessed within a reference period, the file activation ratio is 1, which is the maximum value. If the file has not been accessed for a certain period, the file activation ratio starts to drop in proportion to the period. The directory activation ratio is the average value of the activation ratio of all files in the directory. Files in a directory with a low activation ratio waste local storage. The active file mode change mechanism periodically monitors the directory activation ratio, and when this ratio falls below a certain threshold, it selects among the files in the directory and changes them to the online mode in order to increase the available space. The files to be changed to the online mode are selected among the local mode files in consideration of the local storage usage ratio and the most recent access time. Conversely, the more frequently files in the directory are accessed, the higher the directory activation ratio. The active file mode change mechanism selects the online mode files in the directory and changes them to the local mode when the directory activation ratio is higher than a certain threshold. Online mode files to be changed to the local mode are selected in consideration of the local storage usage ratio and file access delay time. As the proposed file mode transition mechanism aims for a file synchronization system where client and server storages are not hierarchical but at the same user layer, our mechanism is not sensitive to accurate selection of local mode files. Instead, we focus more on the balance between client storage usage and file access delay that can be different according to user requirements. Experimental results show that the proposed active file mode change mechanism can control when and how much the client can reduce and increase the local storage usage and the file access delay by changing file mode parameters according to the requirements of various client devices.

The remaining sections of this paper are structured as follows. In Section 2, we describe the existing file synchronization mechanisms and other storage systems, and analyze them in terms of the file mode support. Section 3 introduces our file synchronization framework, which is the basis of this research. Section 4 describes the process of manually changing the file mode to the online and local modes in detail. Section 5 introduces the proposed active file mode change mechanism, which is the core part of this paper. In Section 6, we conduct file access experiments using the existing manual file mode change mechanism and the proposed active file mode change mechanism and analyze the results according to directory activation and deactivation scenarios. Section 7 summarizes the advantages of the proposed mechanism and discusses remaining research issues. Section 8 concludes this paper.

2. Related Works

2.1. File Synchronization Systems

Rsync [1] is an algorithm and tool that synchronizes the client's file with the remote server's file. The rsync algorithm has the advantage of reducing the data transfer cost for synchronization by using delta encoding to transmit only the differences between the two files. Rsync is also a standard Linux utility, and many other synchronization tools have

been developed based on it. The file synchronization framework of this paper was also developed based on the rsync algorithm. However, the rsync algorithm does not contain information about the file mode, such as whether the file is in online or local mode.

Research has been performed aiming to improve file synchronization performance in cloud storage [2–7]. Andriani et al. [2] proposed a cloud storage synchronization architecture called Cloud4NetOrg. It is designed mainly for users collaborating with corporate mass storage rather than personal storage. Cloud4NetOrg configured a two-level cache and private network to improve the synchronization performance. Drago et al. [3] presented the results of analyzing the synchronization design method of commercial cloud storage services through measurement experiments and network packet analysis. Li et al. [4] proposed an update-batched delayed synchronization (UDS) mechanism to solve the problem that synchronization performance is significantly degraded when a file is frequently modified with a small amount of modification. When a file modification event occurs, UDS does not immediately synchronize with cloud storage several times, but rather waits until the total amount of modified data reaches a threshold and then synchronizes that data all at once. Han et al. [5] proposed a safe and reliable synchronization method called MetaSync by integrating existing cloud storage services. MataSync proposes the pPaxos algorithm, which is a client-based modification of the existing Paxos [31] algorithm, and proposes a data replication algorithm that reduces the cost of maintaining the consistency of replicated data in order to synchronize file modifications consistently among multiple cloud services. Lopez et al. [6] proposed a StackSync architecture that provides elastic file synchronization based on a lightweight message queue framework called ObjectMQ. The elastic file synchronization is an efficient method of providing resources such as servers or message queue objects according to synchronization overhead, by increasing or decreasing resources. Li et al. [7] defined the Network Traffic Usage Efficiency (TUE) in the synchronization process. They measured and analyzed this TUE value through experiments targeting commercial cloud storage services and presented an efficient method to use the synchronization traffic. As described above, various methods for improving file synchronization performance have been proposed, but they did not consider the costs of different file modes in terms of the local storage and file access delay.

For commercial cloud storage services such as Dropbox, OneDrive, and Google Drive, specific file synchronization methods are not disclosed. Instead, there has been research [3,8] that indirectly analyzed them through service usage experiments and traffic analysis. In addition, technical documents [9–11] provide the characteristics of each cloud storage service. Dropbox and OneDrive store the original files only on the server under the names of Smart Sync and Files on Demand, respectively, and the client maintains only the meta information of the files to save client storage space. However, these methods have a limitation in that the user manually selects the files and changes them to either online or local mode.

2.2. Cloud Storage and Distributed File Systems

Research related to cloud storage and distributed file systems has also been conducted [12–20]. Among legacy distributed file systems are the Andrew File System (AFS) [12] and Coda File System [13]. AFS used file transfer and cache to improve the performance of accessing remote file at servers. Coda also uses the same cache approach, and further provides disconnected operation when a client–server connection has failed. Bessani et al. [14] proposed a Shared Cloud-backed File System (SCFS) that provides strong consistency to solve the problems of reliability, durability, and file sharing inefficiency of existing cloud storage systems. Ghemawat et al. [15] introduced the Google File System (GFS), which is a scalable distributed file system suitable for large-scale data processing applications. Additionally, GFS works on low-cost hardware while providing fault-tolerance technology and providing superior performance to large clients. Muniswamy-Reddy et al. [16] proposed a method of providing not only data but also historical information (Provenance) related to data creation and modification to cloud storage. This includes informa-

tion such as when the data was created and from which preceding data it was modified. Duan et al. [17] proposed CSTORE, a cloud storage for the data management of many individual users rather than the management of large-scale data. CSTORE strengthened data security by providing an independent namespace for each user and avoided data conflicts in the process of multiple logins and file modification of the same user based on log records. In addition, CSTORE used a method to avoid data duplication by managing data at the block level. Shvachko et al. [18] deals with the Hadoop Distributed File System (HDFS), which focuses on the distributed processing method of large-scale data. In HDFS, thousands of servers in a large cluster are designed to reliably distribute and store large-scale data, and to continuously transmit large-scale data sets to user applications at high speed. Chen et al. [19] proposes a prefetching mechanism by converting file path information into word vector and applying it to RNN to detect file access pattern in GlusterFS [20]. These research projects focused on improving the performance of the file access and operation but did not deal with the local storage and the file access delay issues through the online and local mode changes of files.

2.3. Multi-Tiered Storage Systems

Soundararajan et al. [21] proposes prefetching data blocks by analyzing the block access pattern per application context in a storage area network. It analyzes the frequency of access order of data blocks and detects correlation of access history. AMP [22] prefetches metadata based on affinity of metadata access patterns in distributed storage, where metadata servers and file servers are separated.

In the active archive system [23,24], it is possible to repeatedly schedule the archiving start point to be performed at a specific time. The archive target file can be set by the elapsed time since the last access or can be based on file attribute values such as file size, file owner, and file type. Because the active archive system restores files to main storage when accessing a link (or an online mode file), there is a recovery delay for all archived file accesses. Amazon S3 Intelligent-Tiering [25] organizes storage with three or four layers and moves old files from higher layer to lower layer storage. When a file in the lower layer storage is accessed, it moves to the top layer (Frequent Access Tier) storage. As with the previous active archive systems, Amazon S3 also has the same problem of restoring files when they are accessed. Cherubini et al. [26] proposed a file prefetching method for multi-tiered archive storage systems. They select target files by applying machine learning to the metadata of accessed files to predict future access patterns.

File prefetching also has been researched in the multi-tiered storage of high performance computing (HPC) systems [27,28]. Alturkestani et al. [27] proposed multilayered buffer storage (MLBS), which is a data management method in three layered storages of super computers for HPC. MLBS prefetches data if higher layered storage has an empty buffer slot. Selection of prefetched files is based on whether the application processes in LIFO or FIFO. In the research of Qian et al. [28], a HPC client of hierarchical storage management (HSM) architecture uses its SSD as cache. Files to be cached are selected using file access information such as open/close events, client NID, and job ID.

Some research approaches prefetching data in a more fine-grained manner [29,30]. In the research of Khot et al. [29], they consider the typical access patterns of different file types when file pages are prefetched to the cache in the OS level. Prefetch window size is adjusted according to the file type. Devarajan et al. [30] proposed a method to prefetch file segments in multi-tier storages, which is called HFetch. HFetch specifically focuses on a write-once-read-many (WORM) data access model of scientific data workflow. After HFetch detects file access, it prefetches the next file segment by analyzing the global view of the file access pattern with updated file offset, length, and timestamp.

Multi-tiered storage systems that support file cache or prefetch are similar to the transition to the local mode of the proposed mechanism, although their target domains are different. While caching takes place in an on-demand manner, prefetching occurs in advance before the file is accessed. Table 1 summarizes their comparison in various aspects.

In the existing multi-tiered storage systems, cache hit ratio and prefetching accuracy are the main important performance factors because the cache and prefetch target is costly high-layer storage (main purpose and storage hierarchy columns in Table 1). In the proposed file synchronization system, on the other hand, as client and server storages are not hierarchical and they are main storages used by a user, the selection accuracy of local mode candidate files is not relatively important. Therefore, the proposed file mode change mechanism uses a directory activation ratio as the criterion for changing a file to the local mode, which does not require as much computing resources and historical information as prefetching methods for multi-tiered storage systems. Instead, we focus more on the balance of importance between client storage usage and file access delay, which can be different according to user requirements (transition direction priority column in Table 1). For example, a file can move between local or online mode, even in the same states of directory activation and local storage usage, according to user's preference.

Table 1. Comparison of proposed mechanism vs. multi-tiered storage systems.

	Storage Category	Consistency	Main Purpose	Storage Hierarchy	Transition Target	Transition Type	Transition Criteria	Transition Direction Priority
Proposed method	Cloud storage	Strong	File synchronization	No	File	Local/online mode	Directory activation ratio, download time, storage usage	Yes
[21]	Storage Area Network	Weak	Fast access	No	Data block	Prefetch	Block access pattern	No
[22]	Distributed storage	Weak	Fast access	No	Metadata	Prefetch	Metadata access pattern	No
[23,24]	Archive	Weak	Backup	Yes	File	Restore	On demand	No
[25]	Cloud storage	Weak	Backup	Yes	File	Restore	On demand	No
[26]	Archive	Weak	Backup	Yes	File	Prefetch	File access pattern	No
[27]	HPC storage	Weak	Fast access	Yes	Data	Prefetch	Data access order	No
[28]	HPC storage	Weak	Fast access	Yes	File	Cache	File access information	No
[29]	Local storage	Weak	Fast access	Yes	File page	Prefetch	File access pattern	No
[30]	Local storage	Weak	Fast access	Yes	File segment	Prefetch	File access pattern	No
[19]	Distributed File System	Weak	Fast access	No	File	Prefetch	File access pattern	No

Because the existing multi-tiered storage architecture is complementary to the file synchronization system, we expect that it can be applied to the client or server storage of the file synchronization system to improve file access performance.

3. File Synchronization Framework

This section describes the file synchronization framework added as a module to the existing Communication Framework (CM) [32]. CM is a communication framework that provides various application-level communication services required for client–server application development. Application developers can easily implement functions such as file transfer, social network service, and message transfer, as well as basic services such as server login and user management, by using the application programming interface (API) provided by the CM library. Since CM provides various options for each communication service through API parameters and configuration files, developers can choose different service policies according to application requirements. Among various services, the CM file synchronization service is systematically connected with existing CM services such as user management and file transfer, making it easy to develop a personal cloud storage service.

The file synchronization process proposed by the CM file synchronization framework was described in detail in our previous work [33], and this paper focuses on the mechanism for manually or actively changing the file mode within the file synchronization service. In

this paper, we assume that one client and server are considered for the file synchronization target, and that one-way synchronization that synchronizes the server files based on the client's file is used. We do not yet cover further extensions such as bidirectional synchronization, multiple logins from the same client, and file sharing with other clients.

Figure 1 shows the structure of CM file synchronization service. The client and server applications each use CM as the underlying communication framework. The file synchronization common module is a CM module commonly used by both the client and the server, and is responsible for detailed synchronization processing, synchronization event management, synchronization, and file mode information management. The directory-monitoring task is a module running as a separate thread to detect changes in the synchronization directory on the client-side CM. When the client stops synchronization, this monitoring task is also stopped, and when the synchronization is restarted, the monitoring task is also restarted. The active file mode change task is responsible for actively changing the file mode according to the synchronization directory activation status in the client-side CM. The details about changing the file mode are described in detail in the next section. The block checksum generation task is a module that selects files to be synchronized with a separate thread in the server-side CM and creates a block checksum for each file. This module works by creating a thread for each client only while performing synchronization with the client. The client can start or stop the file synchronization service after logging into the server. When the client starts the file synchronization service, CM synchronizes the files in the client's synchronization directory with the server. The server-side CM manages a separate synchronization directory for each client. After synchronization with the server is completed, the client CM continues to monitor the client's synchronization directory and repeats synchronization with the server whenever an event such as file addition, modification, or deletion occurs.

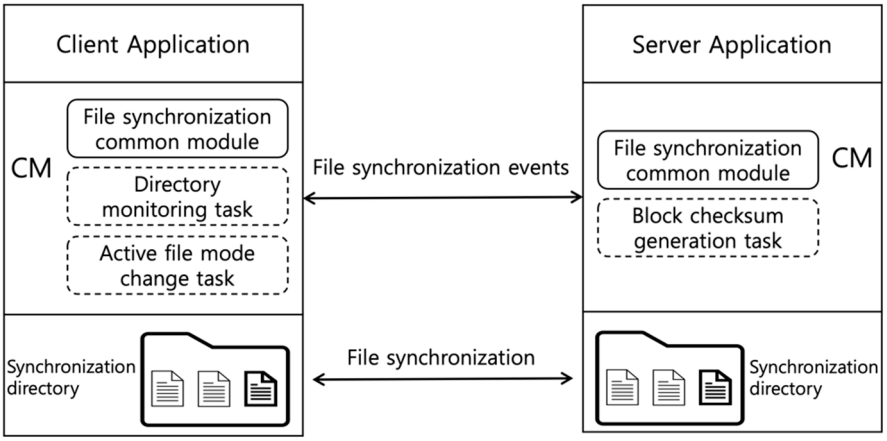


Figure 1. CM file synchronization structure.

Figure 2 shows the core classes related to file synchronization in client and server-side CM. Other CM modules are classes that support services and inter-node communication other than the file synchronization and are simply shown because they are not core modules of this paper. File synchronization common classes play the same role in the CM server and client. CMFileSyncManager provides detailed methods required by the file synchronization algorithm, such as the start and end of file synchronization, file block checksum calculation, synchronization completion confirmation, and file mode change. CMFileSyncEventHandler is responsible for handling events when the client and server receive them from another party during the file synchronization process. The file synchronization event is defined by inheriting the CMFileSyncEvent class, and the transmission and handling of the event used in this paper will be described in more detail in the file mode change process in the next section. CMFileSyncInfo manages various pieces of information required by the client and server in the file synchronization process. The information required by the client includes a

list of files in the synchronization directory, a list of block checksums for each file sent by the server, whether the synchronization has been completed for each file, and a list of files in the online mode. The information managed by the server includes the file information list sent by the client, the file block checksum information for each client, and the server-side synchronization file list for each client.

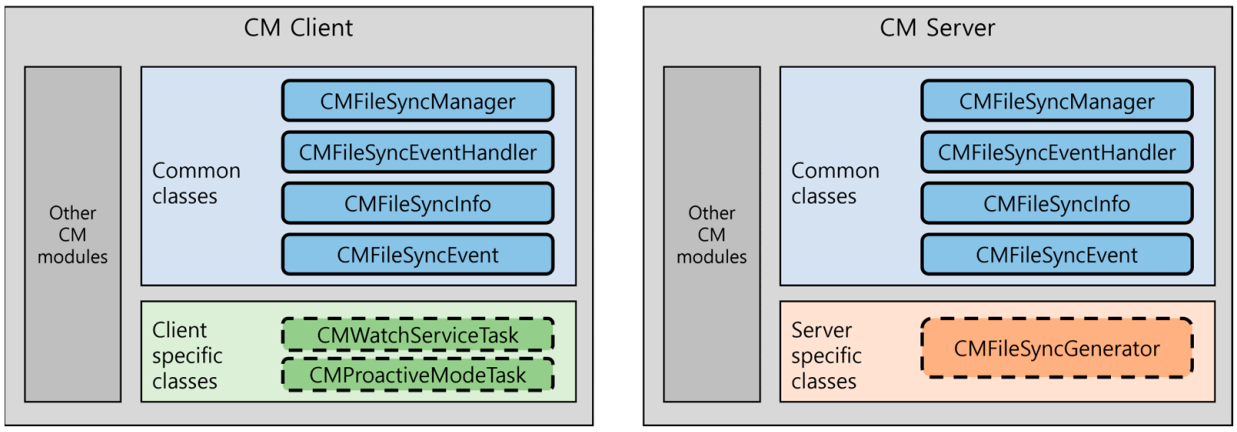


Figure 2. CM file synchronization modules.

CMWatchServiceTask is a client-specific class for file synchronization that implements the directory-monitoring task of Figure 1. This class is created as a separate thread object when file synchronization starts and monitors file changes in the client directory. When an event such as adding a new file, modifying an existing file, or deleting a file occurs, CMWatchServiceTask starts synchronizing the modified file with the server. When the file synchronization is stopped, the CMWatchServiceTask thread object is terminated, and the monitoring task stops. CMProactiveModeTask is a client-specific class for active file mode change, which implements the active file mode change task of Figure 1. This class is also created as a separate thread object, periodically calculating the directory activation ratio, and determining whether to change the file mode of files in the directory. We describe the details of the active file mode change mechanism in a different section. CMFileSyncGenerator is the class of file synchronization server that implements the block checksum generation task of Figure 1. When a client starts synchronization and sends a list of files, the server creates a CMFileSyncGenerator object for this client in a separate thread. This object compares the file list sent by the client with the files the server contains, selects the files to be synchronized, generates block checksum lists of the files to be synchronized, and transmits this information to the client. When the CMFileSyncGenerator thread sends the block checksum information to the client, it is terminated, and the rest of the synchronization process is performed by the file synchronization common classes.

4. Online and Local File Modes

In the file synchronization service, the client can change the file mode of the files in the synchronization directory. The file mode is assumed to be the local mode or the online mode. In the local mode, file data exists in both the client's local storage and the server's remote storage, and both files are synchronized. The client can directly access the local mode file to read or update it. As soon as the local mode file is updated, the server-side file is also synchronized to reflect the updated content.

In the online mode, file content exists only in the remote storage of the server. The local storage of the client maintains only file attributes and has no file content. Since the file data does not exist on the client, the online mode has the advantage of saving the local storage space. However, when a client tries to access an online mode file, it should first download the original file data from the server, which increases the file access delay. Therefore, the online mode is used to save the client's local storage space. Commercial

synchronization services such as Dropbox's smart sync and OneDrive's Files On-Demand service also support the local and online modes according to user's request. The file mode change service provided by the CM file synchronization framework also allows users to manually change the file mode. This section describes the file mode change process in detail.

4.1. Manual Online Mode Change

To manually change a synchronized file to the online mode, the client starts by selecting a list of files to change. In the CM file synchronization framework, when a client calls the manual online mode change service, it passes a list of files selected by the user as an argument. The sequence diagram in Figure 3 shows the subsequent online mode change process. Since file mode change is also a kind of synchronization process, if file synchronization is currently in progress, the file mode change request does not proceed. If synchronization is not currently in progress, the file mode can be changed, and the client changes the current state to synchronization in progress to prevent conflicts with subsequent separate synchronization requests. The client also stops the directory-monitoring task so that it does not detect file change events. After stopping the synchronization service, the client sends a list of files requested for the online mode change (CMFileSyncEventOnlineModeList) to the server and performs the server's confirmation procedure. If the number of files to change mode exceeds the maximum size of the event to be transmitted, the client divides the request list into multiple events and transmits them. The server checks whether the original file of the requested file exists or not. When receiving a response event (CMFileSyncEventOnlineModeListAck) transmitted by the server after confirmation, the client changes the checked files to the online mode. To change to the online mode, the client truncates the file size to zero and maintains only the file attributes, which frees up a space equivalent to this file size in the local storage. The client-side file synchronization framework maintains an online mode file list to manage online mode file information. When all requested files are changed to the online mode, the client notifies the server that the online mode change has ended with a CMFileSyncEventEndOnlineMode event. When the server sends a response event (CMFileSyncEventEndOnlineModeAck) to it, the client finally changes the current state to the synchronization complete status and restarts the directory-monitoring task.

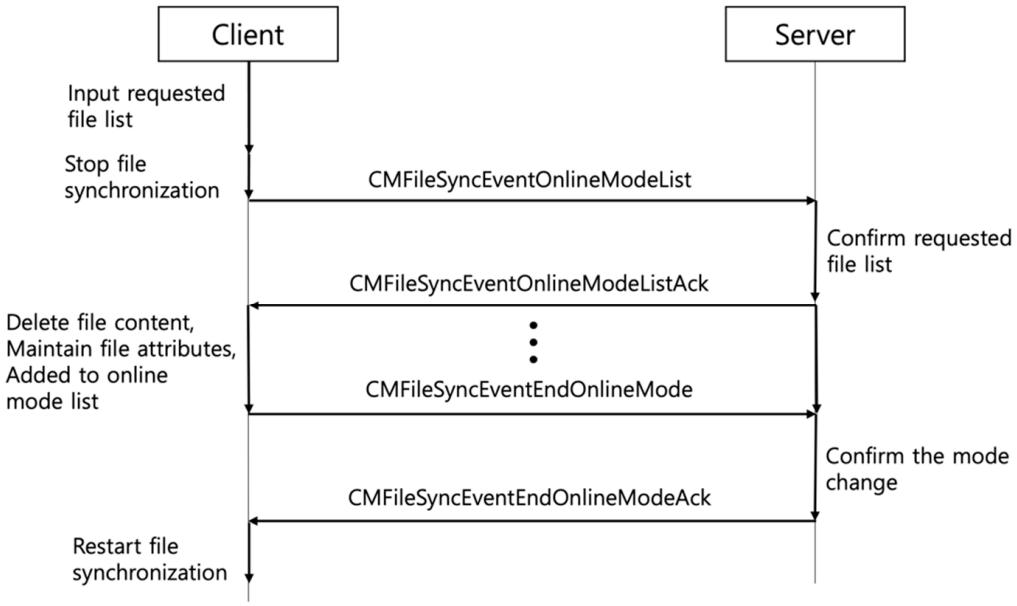


Figure 3. Online mode change process.

When a client deletes an online mode file, the CM file synchronization framework handles it in the same way as the deletion of an existing local mode file during the synchro-

nization process. Since the client must first change to the local mode to access an online mode file other than a deletion event, the server ignores change events other than the deletion for an online mode file.

4.2. Manual Local Mode Change

To change the online mode file back to the local mode, the client selects the list of online mode files to change and requests the local mode change service of the CM file synchronization framework. Figure 4 is a sequence diagram to change the requested online mode file list to the local mode. Since the local mode change process is also a synchronization process, such as the online mode change, the file mode change request is canceled if the file synchronization is currently in progress. If the current synchronization is not in progress, the local file mode change can proceed, so the client changes the current state to synchronization in progress and stops the synchronization-monitoring task. The client then sends the requested file list to the server (CMFileSyncEventLocalModeList) to receive confirmation. The server checks whether the requested files exist, sends a response event (CMFileSyncEventLocalModeListAck) to the client, and then starts the file transfer procedure. The file transfer uses CM's file transfer service [34]. Whenever the client receives a file transfer completion event (END_FILE_TRANSFER) from the server, it checks whether this file is the requested file of the local mode change. If the transferred file is the target of local mode change, the client moves the file to the synchronization directory to overwrite the online mode file, and the file attributes maintain the values of the online mode file. After changing the file to the local mode, the client deletes the file information from the online mode file list to complete the mode change procedure of this file. When all requested files have been changed to the local mode, the client sends a mode change completion event (CMFileSyncEventEndLocalMode) to the server. The server sends a response event (CMFileSyncEventEndLocalModeAck) after checking whether the number of files processed for mode change is correct. When the client receives a response event from the server, it restarts the synchronization service that was stopped as the last operation of mode change procedure.

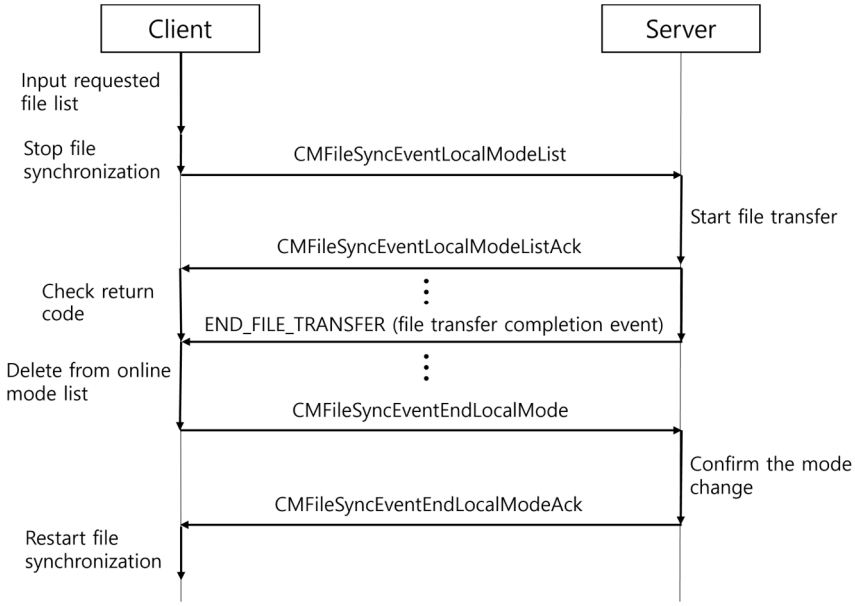


Figure 4. Local mode change process.

5. Active File Mode Change Mechanism

The basic idea of the proposed active file mode change mechanism is to measure the activation level of a directory and actively change the file mode according to this value. When a directory is active, it means that the files in this directory are frequently accessed

by clients based on the current time. Conversely, when the degree of directory activation is low, it means that the files in this directory are accessed less frequently. If the file is accessed frequently, the file should be kept in the local mode rather than the online mode to shorten file access time. On the other hand, if we do not access files well, we can save local storage capacity by keeping these files in the online mode until we access them.

To quantify the degree of directory activation, in this paper, we define the directory activation ratio (*DAR*) as an index indicating how often files in a directory are accessed up to the current time. The directory activation ratio is again calculated as the average value of the file activation ratio (*FAR*) of all files in the directory. The file activation ratio is defined as follows:

$$FAR = \min\left\{\frac{DSLAT}{T_c - T_l}, 1\right\} \ (0 \leq FAR \leq 1) \tag{1}$$

where T_c is current time and T_l is the last access time. *DSLAT* is the acronym of "duration-since-last-access threshold" of a file. This threshold represents the maximum period since the last access to a file that it can remain highest active until the next access. For example, if the *DSLAT* is 7 days, the file activation ratio remains at its maximum value of 1 until 7 days have passed since the last time the file was accessed. When the elapsed time from the last access to the next access exceeds 7 days, the file activation ratio starts to be less than 1. When the file is accessed, the file activation ratio is set back to its maximum value of 1, and this value is maintained for the threshold of 7 days. If the elapsed time without the next access crosses the threshold, the activation ratio thereafter decreases in proportion to the elapsed time. After finding the activation ratio of all files in the directory, the directory activation ratio is defined as the average value, as follows:

$$DAR = \frac{\sum_{i=0}^{N-1} FAR_i}{N} \quad (0 \leq DAR \leq 1) \tag{2}$$

where, N is the number of all files in the directory and FAR_i is the activation ratio of the i-th (i is a 0-based index) file. If a sub-directory exists in the directory and the i-th file type is a directory, the activation ratio of the corresponding sub-directory is used instead of the file activation ratio. If the directory is empty, the directory activation ratio is initialized to zero.

The active file mode change mechanism periodically updates the directory activation ratio to determine if it is necessary to change the file mode. Figure 5 is a flow chart showing the process in which the active file mode change mechanism determines whether to change the file mode. The active file mode change mechanism first calculates the directory activation ratio and checks if this value is less than *OMT*. *OMT* stands for an online mode threshold and serves as a criterion for determining whether the active file mode change mechanism starts an operation of changing files to the online mode. If the directory activation ratio is less than *OMT*, the active file mode change mechanism determines that the deactivation degree of the directory exceeds the reference value and starts the operation of changing the file mode to the online mode. If the directory activation ratio is greater than or equal to *OMT*, the next check is whether it is greater than the *LMT*. *LMT* stands for a local mode threshold and is a criterion for determining whether the active file mode change mechanism starts an operation to change files to the local mode. If the directory activation ratio is greater than *LMT*, the active file mode change mechanism determines that the activation level of this directory exceeds the reference value and starts to change the file mode of selected files to the local mode. The online mode threshold (*OMT*) is less than or equal to the local mode threshold (*LMT*), and if the directory activation ratio is between *OMT* and *LMT*, the active file mode change mechanism maintains the current file mode.

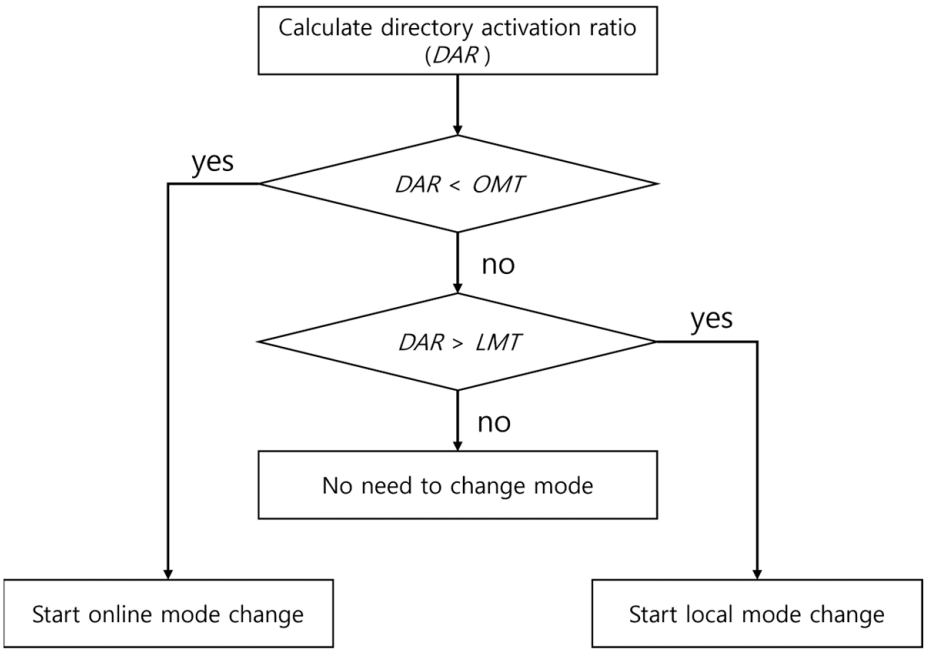


Figure 5. Flow chart of determining file mode change.

Figure 6 is a flow chart showing the process of changing the files to the online mode. The low directory activation ratio means that the files are accessed infrequently, and we can make more available local storage by changing these files to the online mode. To this end, the active file mode change mechanism first calculates the used storage ratio (*USR*). *USR* is the ratio of the currently used capacity to the total storage capacity allocated to the synchronization directory. If the *USR* is less than or equal to the used storage ratio threshold (*USRT*), then there is no need to change the file to the online mode yet because it means that there is still enough local storage capacity available. In this case, the operation is aborted. If the *USR* is greater than the threshold, it means that the available space needs to be increased further because the current local storage capacity is being used above the threshold. In this case, the active file mode change mechanism selects files to be changed to the online mode. Since the target of the online mode change is the file with the oldest last access time, the current local mode file list is sorted in ascending order of the last access time, and the file is changed to the online mode in order from the oldest file. Whenever a local mode file is changed to the online mode, the local storage usage ratio (*USR*) decreases by the file size, and the online mode change operation is repeated until this value is less than or equal to the threshold value (*USRT*). Here, the *USRT* is the criterion for determining whether the active file mode change mechanism needs the online mode change operation from the viewpoint of local storage capacity, and the criterion for determining to what level the *USR* should be reduced through the online mode change. For example, if the *USRT* is 0.5, the active file mode change mechanism starts the online mode change when the *USR* exceeds 50% and ends the online mode change operation when *USR* falls below 50%. If the *USRT* is 0, the active file mode change mechanism changes all files in the directory to the online mode. If the *USRT* is 1, no online mode change operation is performed on any file regardless of the *USR* value.

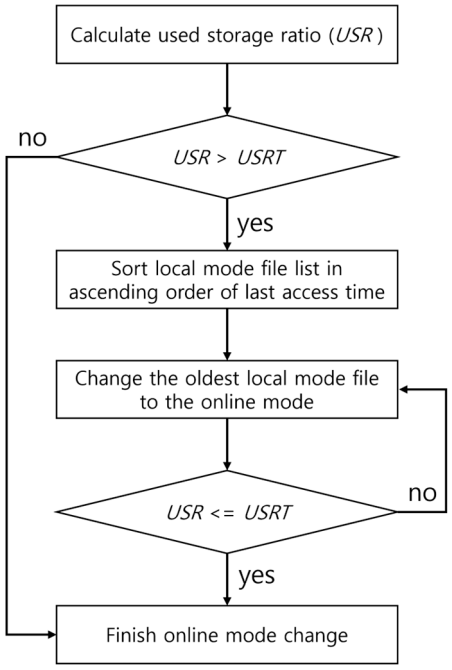


Figure 6. Flow chart of online mode change.

When the directory activation ratio becomes greater than the local mode threshold (*LMT*), the active file mode change mechanism initiates a task of changing the online mode files in the directory to the local mode. Figure 7 is a flow chart showing the local mode change process. The local mode change process first calculates the used storage ratio (*USR*) of local storage and compares it with the used storage ratio threshold (*USRT*), which is similar to the online mode change process. If the *USR* is greater than the threshold (*USRT*), the active file mode change mechanism determines that the local storage is already being used above the threshold and stops the local mode change operation. If the local storage capacity is sufficient, the next task is to select the online mode files for the local mode change. If the client wants to access a large online mode file, the access time is long because the file access needs a delay to download the file from the server. If each user has a maximum amount of access time he/she can tolerate, and if an online mode file of a size that requires a download time longer than this maximum time is preferentially changed to the local mode, the client can keep the average access time within the allowable range. To this end, the active file mode change mechanism may set a maximum allowable access delay threshold (*MADT*). To calculate the minimum size of a file to be changed to the local mode with the *MADT* value, the client first measures the input throughput from the server. Input throughput is a measure of the speed at which the current client receives data from the server in megabytes per second (MBps). Multiplying the input throughput by the *MADT* gives the minimum size of the file that requires the maximum allowable access time. Among the online mode files, the active file mode change mechanism excludes those of which original size is smaller than the minimum size from the local mode change target. This is because these files are downloadable from the server within the maximum allowable access time when the client accesses it. If the *MADT* is less than or equal to 0, the active file mode change mechanism ignores the minimum size criterion of a file to be changed to the local mode, and all online mode files are eligible to be changed. The active file mode change mechanism sorts the online mode files larger than the minimum size in descending order based on the last access time, and the most recently accessed file is the first to be changed. The active file mode change mechanism does not change all selected online mode files to the local mode, but only changes to the local mode until the used storage ratio (*USR*) becomes greater than the threshold (*USRT*) by considering the available space of the local storage. Although the active file mode change mechanism changes all selected files to the

local mode, the *USR* could be still smaller than the *USRT*. That is, there may be still enough local storage space. In this case, the mechanism additionally can change online mode files smaller than the minimum size to the local mode.

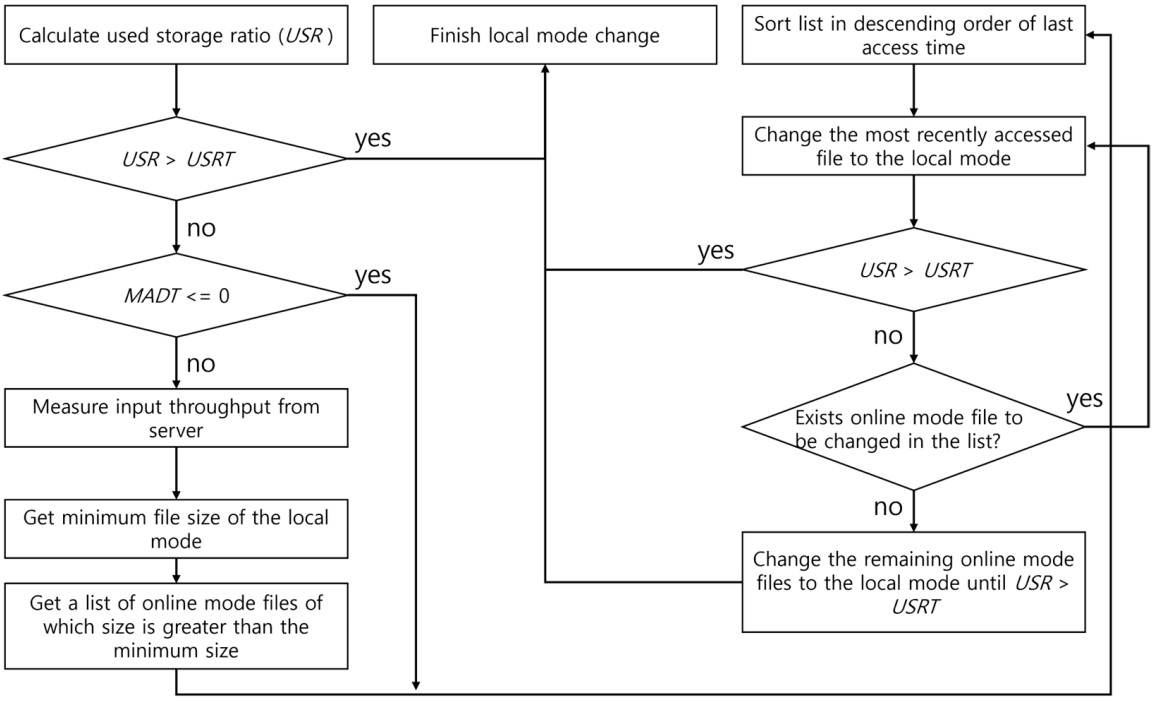


Figure 7. Flow chart of local mode change.

6. Performance Evaluation

In this section, we describe the experiments performed to compare and analyze the performance of the active file mode change mechanism proposed in this paper and the existing manual file mode change mechanism. Because the proposed active file mode change mechanism dynamically changes the local mode and online mode of a file according to factors such as directory activation ratio, used storage ratio, and file access time, we conducted experiments with the following two scenarios. In the first directory deactivation scenario, the client periodically accesses the files in the directory to keep the directory active, then stops accessing the files after a certain period, changing the directory to the inactive state. In the second directory activation scenario, conversely, the client does not access any file keeping the directory inactive, and then starts accessing the files after a certain period to increase the directory activation ratio. Through experiments with these two file access scenarios, we compared and analyzed the differences between the two methods by measuring the local storage usage ratio and file access time.

6.1. Experimental Environments

The specific experimental environment commonly applied to the two file-access scenarios mentioned above is as follows. The access experiment was conducted with 10 test files of the same size generated with 1 MB of random bytes. Actual 1 s is assumed to be 1 h in the experimental environment, and the total file access experiment period is 6 days. That is, the actual time to perform one experiment of the file access scenario is 144 s (= 24 × 6). The file access operations performed during the first 3 days and the last 3 days of the experiment differ depending on the scenario, which will be described in detail in the experimental environment of each scenario. Table 2 shows the default values of parameters used by the active file mode change mechanism in both experimental scenarios.

Table 2. Default Values of Parameters of Active File Mode Change Mechanism.

Parameters	Default Values
Directory activation monitoring period (*DAMP*)	11 h
File synchronization storage (*FSS*)	10 MB
Duration since last access threshold (*DSLAT*)	36 h
Online mode threshold (*OMT*)	0.5
Local mode threshold (*LMT*)	0.5
Used storage ratio threshold (*USRT*)	0.5
Maximum access delay threshold (*MADT*)	0

Directory activation monitoring period (*DAMP*) is the period at which the active file mode change mechanism calculates the directory activation ratio. Every 11 h after the start of the file access experiment, the active file mode change mechanism calculates the directory activation ratio and determines whether to change the file mode based on this value. File synchronization storage (*FSS*) represents the total amount of local storage allocated for file synchronization and is required when calculating the storage usage ratio in the experiment. This value may be the total capacity of the client's local storage, or it may be set to an arbitrary value smaller than this. In this experiment, we set the default value as 10 MB. Duration since last access threshold (*DSLAT*) is the threshold of time elapsed since the last access of a file, indicating the length of time the file activation ratio remains at its maximum value of 1 since the last access of the file. In this experiment, we set the default value to 36 h, which means that the file activation ratio remains at 1 for a day and a half (36 h) after the file is accessed. If there is no file access after this threshold time, the file activation ratio starts to decrease in proportion to the time elapsed after this threshold.

Online mode threshold (*OMT*) is the online mode threshold, which is a criterion for determining whether the active file mode change mechanism initiates the online mode change operation. If the directory activation ratio is less than the *OMT*, the active file mode change mechanism starts to change the selected files to the online mode. In this experiment, we set the *OMT* to 0.5. Local model threshold (*LMT*) is the local mode threshold. This threshold is the criterion by which the active file mode change mechanism decides whether to change files to the local mode. If the directory activation ratio is greater than the *LMT*, the active file mode change mechanism starts the local mode change operation. As with *OMT*, we also set the *LMT* to 0.5 in this experiment. Used storage ratio threshold (*USRT*) is the local storage usage ratio threshold. The *USRT* plays a role in regulating the local storage usage ratio so that the active file mode change mechanism can change files to the online or local mode under the *USRT* limit. In this experiment, we set the default value of the *USRT* to 0.5. For example, when the active file mode change mechanism changes files to the online mode, the change is only performed until the local storage usage is less than or equal to 5 MB, which is 50% of the total amount of 10 MB. When the mechanism changes files to the local mode, it limits the changes to only until the local storage usage is greater than 5 MB. Maximum access delay threshold (*MADT*) is used as a criterion for determining the minimum size of an online mode file to be changed to the local mode. In this experiment, we set the *MADT* to 0, which means that the active file mode change mechanism only considers the last access time, not the minimum size, when selecting a file to change to the local mode.

To proceed with the experiment, we implemented the proposed active file mode change mechanism as well as the existing manual file mode change method as services of our file synchronization framework. We also developed a synchronization server and client prototypes that utilize this framework. The synchronization server and client used in the experiment were connected by wireless LAN (Wi-Fi), and the specifications of each device are described in Table 3.

Table 3. Device Specifications of Synchronization Server and Client.

	Server	Client
OS	macOS Monterey	Windows 11 Pro
CPU	2.4 GHz, Intel Core i5	2.3 GHz, Intel Core i7
Memory	16 GB	16 GB
Storage	SSD 512 GB	SSD 1 TB

6.2. Directory Deactivation Experiment

In the first scenario, the directory deactivation experiment performs periodic file access operations during the first 3 days and performs no file access during the second 3 days. That is, the first 3 days are the directory activation period, and the last 3 days are the directory deactivation period. Starting the experiment with the synchronization directory empty, the file access operation during the first directory activation period repeats two operations: adding new files and updating existing files. The cycle for adding new files is 5 h and the cycle for updating existing files is 2 h. To add new files, we created 10 test files in the temporary directory in advance and copied them one by one to the synchronization directory at each addition cycle. We updated existing files one by one at every update cycle, starting with the files newly added to the synchronization directory. The file update method proceeds differently depending on the file mode. Updating a local mode file overwrites the same test file in the temporary directory by copying it back to the synchronization directory. In case of an online mode file, the update operation first changes the file to the local mode.

In this experiment, we measured the local storage usage ratio and the file access delay during the directory transition from active to inactive state. To this end, whenever a file access (new addition or update) operation is performed during the experiment period, we recorded the current local storage usage and the file mode of the accessed file. We record the mode of the file we accessed to get the cost of accessing the file. When accessing a local mode file, there is no download cost, whereas when accessing an online mode file, there is a cost to download the entire file from the server to change it to the local mode. In this experiment, since the file sizes are all the same, the file access cost was measured by the number of times the online mode file was accessed, not the actual download time. We conducted directory deactivation experiments under the same conditions mentioned above for the manual file mode change mechanism and the proposed active file mode change mechanism. In the case of the active file mode change mechanism, to observe the effect of changing parameters on the local storage usage ratio and file access cost, we conducted the experiment by changing three thresholds, including elapsed time since last access threshold (*DSLAT*), online mode threshold (*OMT*), and used storage ratio threshold (*USRT*). The local mode threshold (*LMT*) must be greater than or equal to the online mode threshold (*OMT*). In this experiment, we set the *LMT* and the *OMT* to the same value. During the entire experimental period of the directory deactivation scenario, we analyzed only the local storage usage changes, not the file access costs, as clients mostly accessed only the local mode files.

Figure 8 shows the change in local storage usage when the manual file mode change mechanism is applied to the directory deactivation scenario. Starting with the directory empty, we can see that the local storage increases for each new addition of 1 MB-sized test file over the first 3 days. After the 3rd day of the experiment (after about 68 h), even during the period of no file access, the storage usage continues to maintain 100% of 10 MB. Because the manual file mode change mechanism keeps the original mode unless the user manually changes the file mode, a low directory activation ratio without file access is a waste of local storage.

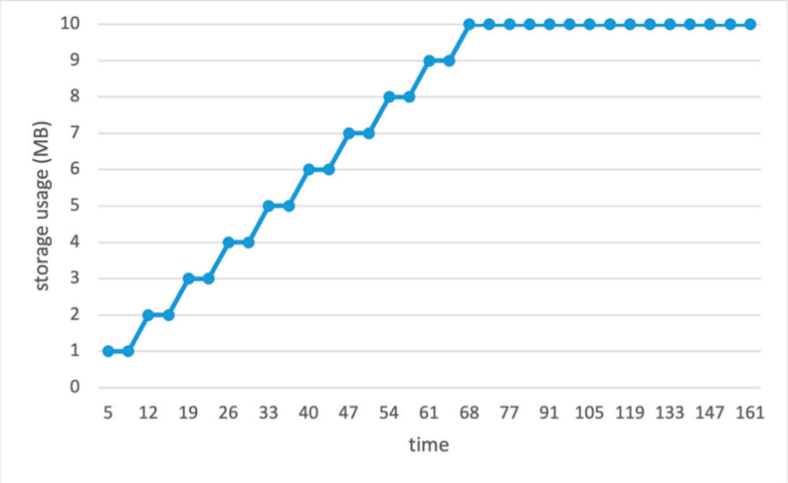


Figure 8. Change of local storage usage of manual file mode change mechanism in directory deactivation scenario.

Figures 9–11 show how the local storage usage changes according to various thresholds of the active file mode change mechanism in the directory deactivation scenario. Unlike the manual file mode change mechanism, in all three graphs, we can see that the storage usage decreases after a certain point in time. There is also a characteristic that the timing or rate of decrease in storage usage is different depending on the type and value of the thresholds. In Figures 9 and 10, the storage usage starts to decrease at a later time, whenever the two thresholds of the *DSLAT* and *OMT* are increased, respectively. Because the *USRT* default value is 0.5, storage usage is reduced only to 5 MB, which is 50% of the total amount of storage (10 MB). Figure 9 (*DSLAT*) and Figure 10 (*OMT*) are similar in appearance, but the roles of the two thresholds are different. *DSLAT* is how long a file can remain active even if it is not accessed. The larger the *DSLAT* value, the slower the inactivation starts without accessing the file, and the slower the directory activation rate decreases. Since the directory activation ratio is an index used by the active file mode change mechanism to determine whether to change the file mode or not, if the time at which the directory activation ratio decreases is delayed, the time to change the file mode to the online mode is also delayed. That is, in the directory deactivation scenario, the higher the *DSLAT* value, the slower the time at which the active file mode change mechanism determines that the file mode needs to be changed to the online mode. As a result, as shown in Figure 9, the higher the *DSLAT* value, the slower the file is changed to online mode (the local storage usage decreases). For example, when the *DSLAT* is 6 h (dslat-6), the online mode change starts after about 47 h, whereas when the *DSLAT* is 36 h (dslat-36), the online mode change starts only after 140 h of the experiment.

As shown in Figure 10, the *OMT* also affects when the local storage usage decreases, that is, when files are changed to the online mode. If the directory activation ratio is less than the *OMT*, the active file mode change mechanism determines that the directory is inactive and starts changing files to the online mode. Therefore, as the *OMT* is smaller, the directory activation ratio must be smaller than this value for the active file mode change mechanism to determine that the directory is inactive. That is, the online mode change operation starts late. For example, if the *OMT* is 0 (omt-0), the online mode change operation does not start until the end of the experiment because the directory activation ratio cannot be less than 0. If the *OMT* is 1 (omt-1), the online mode change starts after 47 h, when the directory activation ratio becomes less than 1. At this point, since the addition of new files and the change of existing files to the online mode are performed simultaneously, we can see that the local storage usage maintains the same value, and then increases and decreases repeatedly. To summarize the difference between the roles of the *OMT* and the

DSLAT, *DSLAT* is a factor influencing the calculation of the directory activation ratio, and *OMT* is a criterion for judging whether to start the online mode change operation.

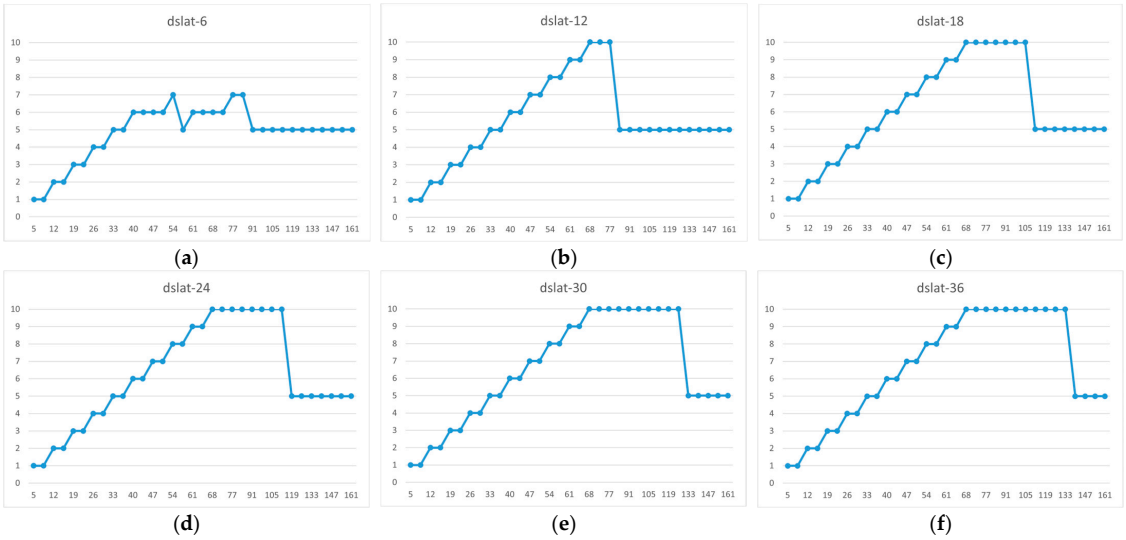


Figure 9. Change of local storage usage of active file mode change mechanism according to *DSLAT* in directory deactivation scenario. (**a**) dslat-6. (**b**) dslat-12. (**c**) dslat-18. (**d**) dslat-24. (**e**) dslat-30. (**f**) dslat-36.

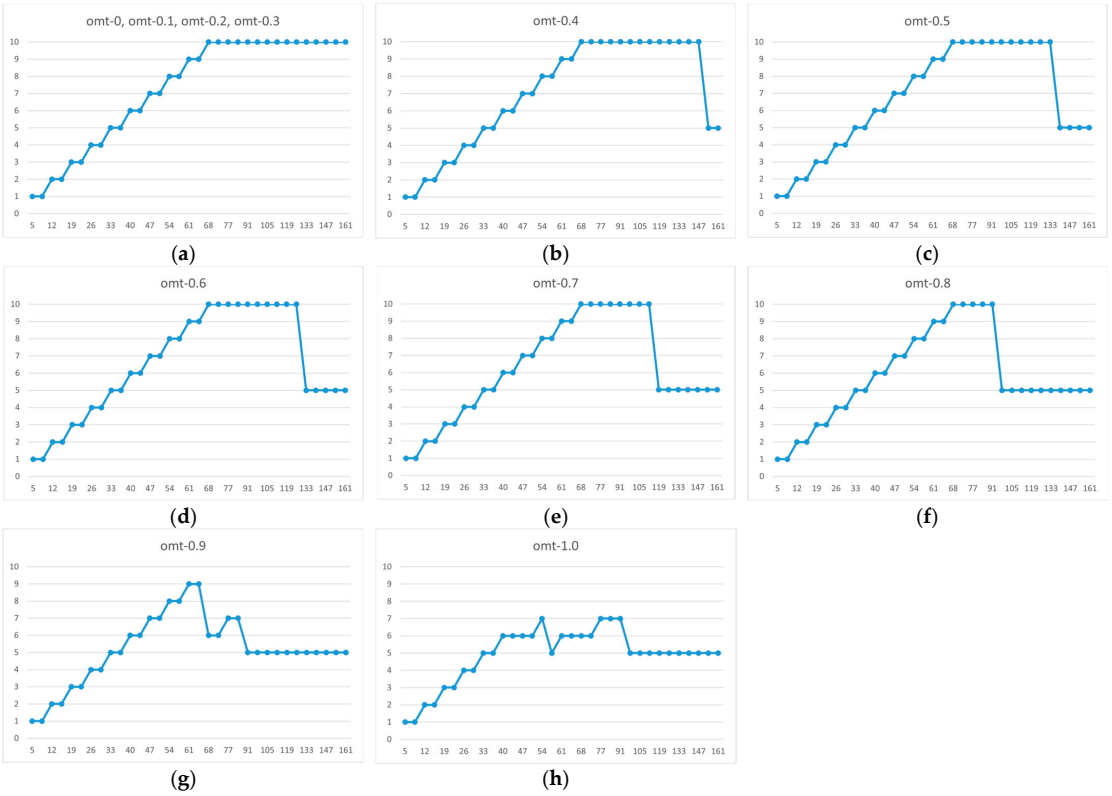


Figure 10. Change of local storage usage of active file mode change mechanism according to *OMT* in directory deactivation scenario. (**a**) omt-0~omt-0.3. (**b**) omt-0.4. (**c**) omt-0.5. (**d**) omt-0.6. (**e**) omt-0.7. (**f**) omt-0.8. (**g**) omt-0.9. (**h**) omt-1.0.

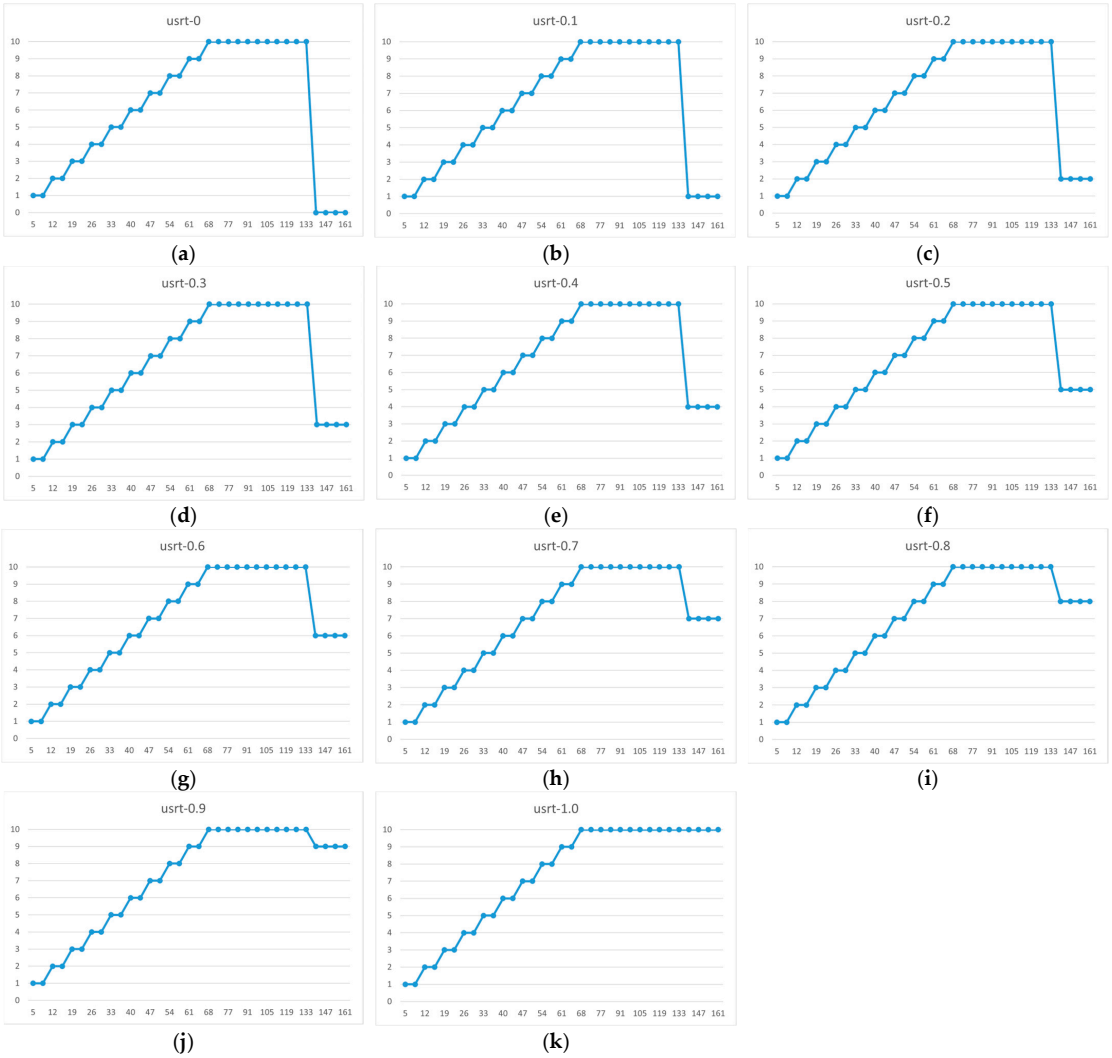


Figure 11. Change of local storage usage of active file mode change mechanism according to *USRT* in directory deactivation scenario. (**a**) usrt-0. (**b**) usrt-0.1. (**c**) usrt-0.2. (**d**) usrt-0.3. (**e**) usrt-0.4. (**f**) usrt-0.5. (**g**) usrt-0.6. (**h**) usrt-0.7. (**i**) usrt-0.8. (**j**) usrt-0.9. (**k**) usrt-1.0.

Figure 11 shows the change in the local storage usage according to *USRT*. The *USRT* plays a role in limiting the scope of file mode change from the point of view of local storage. In the directory deactivation experiment, when the active file mode change mechanism changes files to the online mode to free up local storage space, it changes to the online mode until the local storage usage ratio does not go below the *USRT*. A high *USRT* value means that the synchronization system can use a lot of space on local storage; therefore, even if the directory activation ratio is low, it will not make much storage available. For example, if the *USRT* is 0.8, files are changed to the online mode within the range where the local storage usage ratio is not less than 80%. If the *USRT* is 1, it means that up to 100% of the local storage usage is allowed. In this case, the active file mode change mechanism does not perform the online mode change operation and operates the same as the manual file mode change mechanism. On the other hand, if the *USRT* is 0, the active file mode change mechanism changes all files to the online mode because the local storage usage can be reduced to 0%.

6.3. Directory Activation Experiment

The second scenario is a directory activation experiment opposite to the first scenario. During the first 3 days of the entire experimental period (6 days), no file access is performed,

and the directory is kept inactive. Then, the directory is activated by performing periodic file access during the latter 3 days. The experiment starts with all 10 test files in the synchronization directory in the online mode. During the initial directory deactivation period, we recorded only the current local storage usage for each file access cycle. In the latter half of the directory activation period, each file is accessed one by one in a 7-h cycle. The file access method depends on the file mode, as with the first experiment. For the local mode file, we copied the same test file in the temporary directory to the synchronization directory. For the online mode file, we first changed it to the local mode.

In this experiment, we measured the local storage usage and the file access cost while the directory is transitioned from the inactive to active state. To analyze the effect of changing parameters on the local storage usage and file access cost, the active file mode change mechanism conducted the experiment by changing the three threshold values of elapsed time since the last access threshold (*DSLAT*), local mode threshold (*LMT*), and used storage ratio threshold (*USRT*). In the case of the *LMT*, as in the previous experiment, it was set to the same value as the *OMT*.

Figure 12 shows the results of measuring changes in the local storage usage when the manual file mode change mechanism is applied to the directory activation scenario. Since there is no access operation during the initial period of directory deactivation, all files in the directory remain in the online mode while keeping the local storage usage to a minimum. During the latter period of directory activation, the experiment begins periodically accessing files one by one in sequence. Since all files are in the online mode when accessed for the first time, the local storage usage increases in the order in which files are changed to the local mode. After all 10 files have been accessed, the storage usage becomes 10 MB, which is 100% of the total storage. The manual file mode change mechanism minimizes the local storage usage during directory deactivation periods when all files are in the online mode, but each time the file is accessed, the access cost increases as the original file is downloaded from the server to change to the local mode.

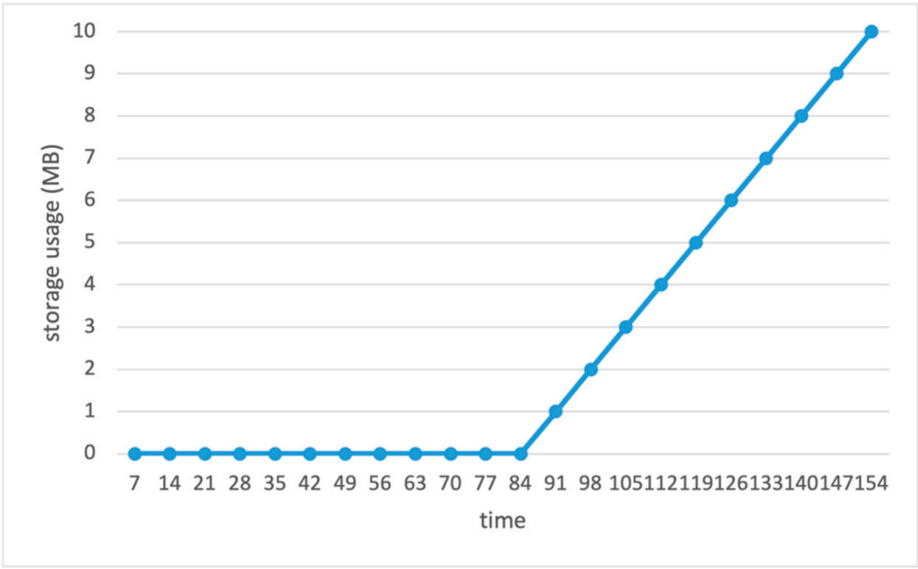


Figure 12. Change of local storage usage of manual file mode change mechanism in directory activation scenario.

Figure 13 shows the change in the local storage usage of the active file mode change mechanism according to the change of the *DSLAT*, which is the elapsed time threshold since the last access. First, looking at the initial directory deactivation period, unlike the manual file mode change mechanism, the storage usage is already about 5 MB and 6 MB. We can see that about half of the files have already been changed to the local mode when the experiment starts. As soon as the experiment started with all test files in the online

mode, the local mode change operation was performed because the directory activation ratio calculated by the active file mode change mechanism was greater than the local mode threshold (*LMT*) default value of 0.5. Because the *USRT* was set to the default value of 0.5, the files were changed to the local mode until the storage usage ratio became greater than 50%. During the directory deactivation period, the larger the *DSLAT* value, the longer the files remain active, so the initial local mode state (storage usage of 6 MB) is also maintained. As the *DSLAT* value decreases, the directory activation ratio decreases quickly, and the active file mode change mechanism starts early to change the local mode files back to the online mode. For example, when the *DSLAT* is 24 h (dslat-24), storage usage decreases after 63 h from the start of the experiment, and when the *DSLAT* is 6 h (dslat-6), storage usage decreases after 21 h. Since the experiment performed the change to the online mode within the range where the storage usage does not go lower than the *USRT* default value of 0.5 (50%), the storage usage is kept at a minimum of 5 MB.

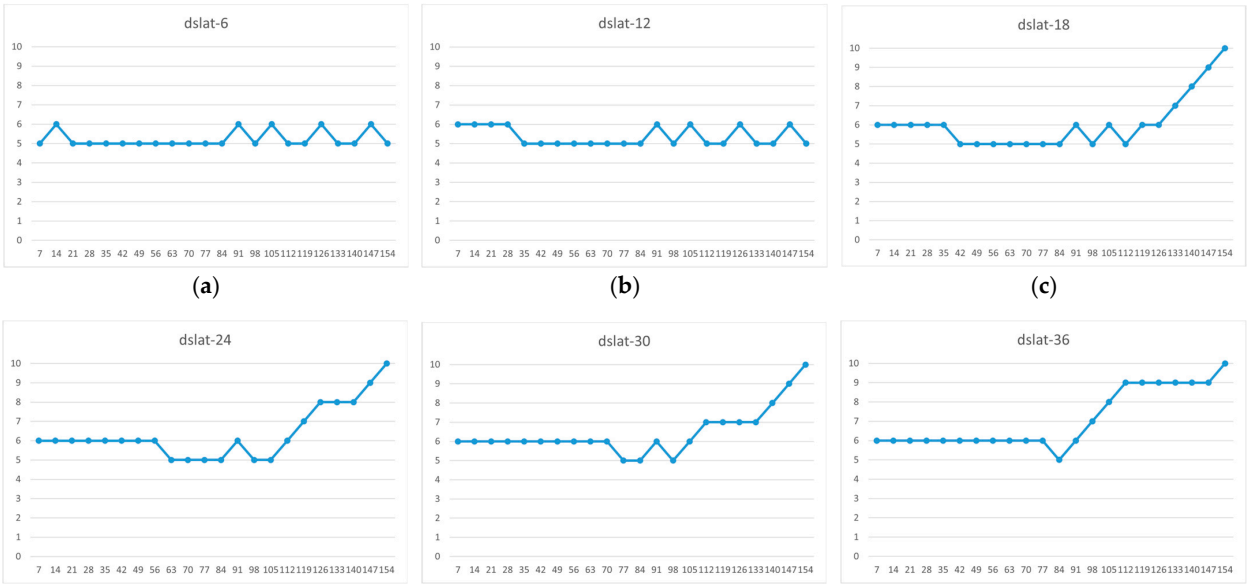


Figure 13. Change of local storage usage of active file mode change mechanism according to *DSLAT* in directory activation scenario. (**a**) dslat-6. (**b**) dslat-12. (**c**) dslat-18. (**d**) dslat-24. (**e**) dslat-30. (**f**) dslat-36.

When the directory activation period is reached, the file access operation begins. As the online mode file is accessed, the local storage usage gradually increases as it changes to the local mode. The *DSLAT* value of the directory activation period affects when storage usage starts to increase. If the *DSLAT* value is large, the chance of changing to the online mode is less because the local mode file maintains an active state for a long time, and the online mode files are accessed one by one and change to the local mode, increasing storage usage. For example, if the *DSLAT* is 36 h (dslat-36), we can see that the local storage usage continues to increase after 84 h. If the *DSLAT* value is small, the local mode files that are not accessed are quickly inactive, these files are changed to the online mode, and the local storage usage is reduced. On the other hand, when the online mode files are accessed, they change to the local mode and increase the local storage usage. As the local mode file and the online mode file change the mode together, the local storage usage is maintained at a certain level. For example, if the *DSLAT* is 6 h and 12 h (dslat-6, dslat-12), the local storage usage keeps repeating between 5 MB and 6 MB. As the *DSLAT* value increases, if the local mode change rate becomes larger than the online mode change rate, the local storage usage starts to increase.

In Figure 13, in terms of local storage usage, the active file mode change mechanism continues to use more than 5 MB of storage during the experimental period, but the increase

point varies according to the *DSLAT* value. In the manual file mode change mechanism, the storage usage is minimal during the directory deactivation period, and the usage increases linearly as soon as the activation period starts. Since it is difficult to determine which mechanism is better based on the local storage usage alone, we also compared the number of online mode file accesses in terms of file access cost as shown in Figure 14. The manual file mode change mechanism has 10 online mode file accesses, which means that all 10 test files accessed for the first time were in the online mode. The active file mode change mechanism shows a decrease in the number of online mode file accesses as the *DSLAT* value increases. This is because the larger the *DSLAT* value, the more files are active, and the probability that such files are changed to the local mode before access increases. If the *DSLAT* value is small, the number of files in inactive state increases, and the probability that such files are changed to the online mode increases, and the number of online mode file accesses also increases. In other words, if the *DSLAT* value is small, the local storage usage can be maintained at a certain level, but the file access cost increases accordingly.

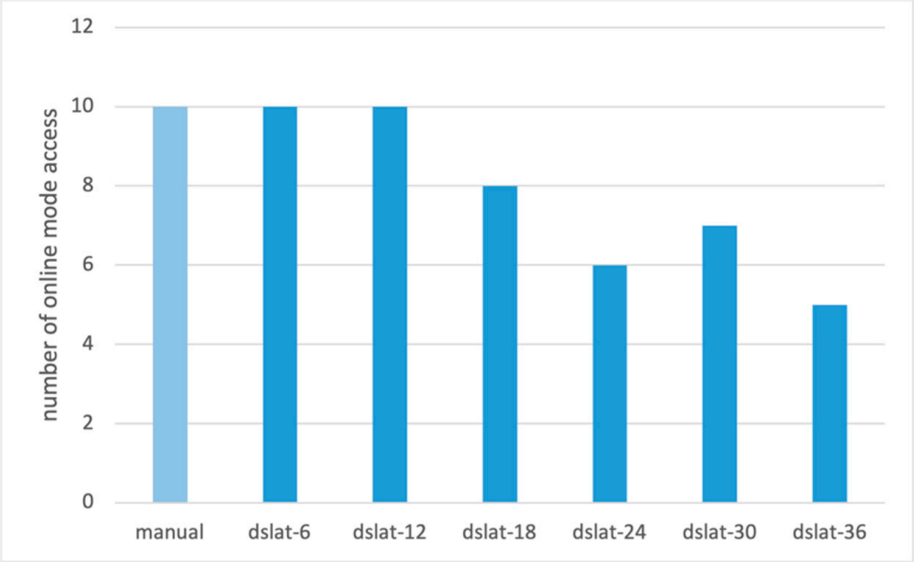


Figure 14. Change of the number of online mode file accesses according to *DSLAT* in directory activation scenario.

Figure 15 shows the change in the local storage usage according to the local mode threshold (*LMT*). Since the *LMT*s from 0 to 0.4 have the same results, we showed the measured values from 0.4. As in Figure 13 (*DSLAT*), because the active file mode change mechanism performs the local mode change at the start of the directory activation experiment, the storage usage in Figure 15 starts at 6 MB that exceeds 50% of the total capacity (because the *USRT* default is 0.5). During the initial directory deactivation period, even if files are changed to the online mode, the local storage usage is maintained at a minimum of 5 MB. Just as the local storage usage graphs according to the *DSLAT* and the *OMT* were similar in the first directory deactivation scenario, the graphs according to the *DSLAT* and the *LMT* are similar in the directory activation experiment. During the initial directory deactivation period, the higher the *LMT* value, the higher the reference value for the active file mode change mechanism to determine that the directory is activated. Therefore, the online mode change occurs early, and then the storage usage decreases. For example, when the *LMT* value is 1 (lmt-1.0), the storage usage decreased after 42 h of the experiment start. In the case of a smaller *LMT*, the usage decreases later. During the latter period of directory activation, file access operations are initiated. At this time, the larger the *LMT* value, the stronger the tendency of the active file mode change mechanism to keep the files in the online mode, and the storage usage does not increase well. This is because the rate of change to the local mode by file access and the rate of change to the online mode by the

active file mode change mechanism are similar. As the *LMT* becomes smaller, the storage usage tends to increase early because the active file mode change mechanism weakens the tendency for files to change to the online mode. For example, if the *LMT* is 0.4 (lmt-0.4), the storage usage increases after 91 h. whereas if the *LMT* is 0.7 (lmt-0.7), the storage usage starts to increase after 126 h.

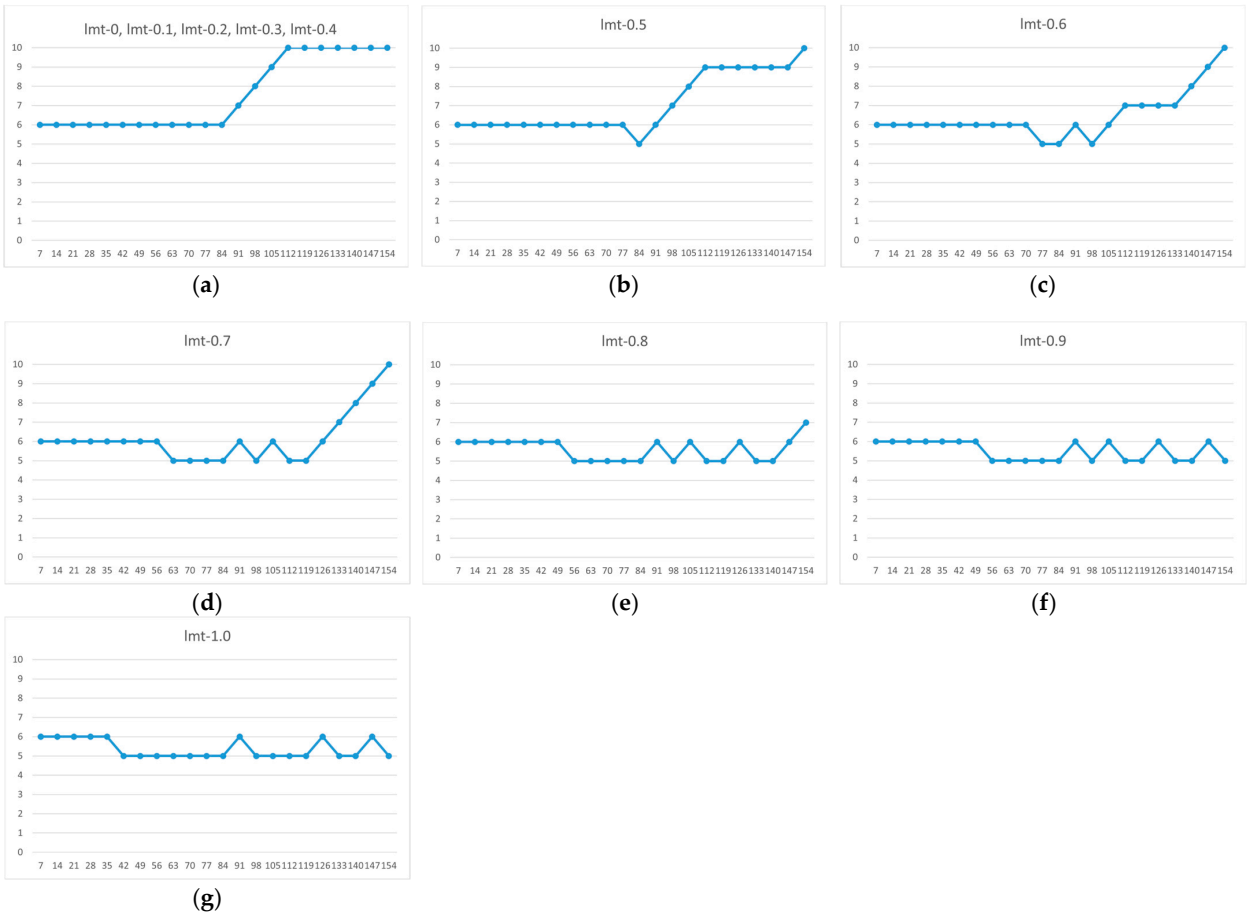


Figure 15. Change of local storage usage of active file mode change mechanism according to *LMT* in directory activation scenario. (**a**) lmt-0~lmt-0.4. (**b**) lmt-0.5. (**c**) lmt-0.6. (**d**) lmt-0.7. (**e**) lmt-0.8. (**f**) lmt-0.9. (**g**) lmt-1.0.

As shown in Figure 15, in terms of the local storage usage only, the larger the *LMT* value, the better it seems, because it maintains a relatively low storage usage, and it is difficult to judge which method is better compared to the manual file mode change mechanism. However, as shown in Figure 16, the results may be different when looking at the file access cost. If the *LMT* value is large, the local storage usage decreases because the number of online mode files is relatively large. However, as the number of online mode files increases, the probability that the accessed file is in the online mode also increases, and the access cost increases. The manual file mode change mechanism is good in terms of the local storage usage but results in high file access cost. The active file mode change mechanism allows us to choose the more important factor between the local storage usage and file access cost and adjust the threshold accordingly. If the user wants to reduce the local storage usage and delay its increase as much as possible, he/she can increase the *LMT* value. If the user has many large files and wants to quickly change to the local mode to reduce file access costs, he/she can reduce the *LMT* value.

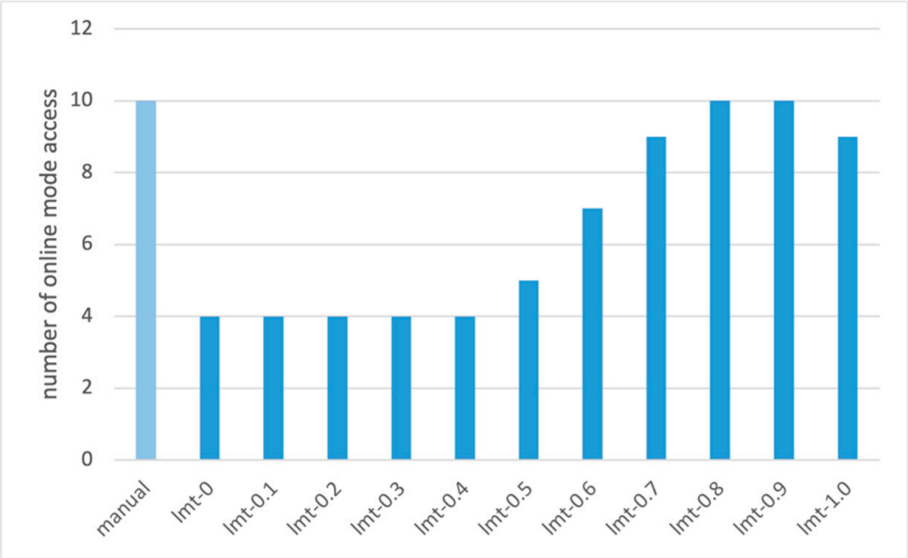


Figure 16. Change of the number of online mode file accesses according to *LMT* in directory activation scenario.

Figure 17 shows the change in the local storage usage of the active file mode change mechanism according to the various *USRT* values in the directory activation experiment. *USRT* is the lower or upper bound reference value for local storage change when the active file mode change mechanism changes a file to the online mode or local mode. For example, if the *USRT* value is 1 (usrt-1.0) during directory deactivation period, the active file mode change mechanism can use 100% of local storage when changing to the local mode. Therefore, since the active file mode change mechanism does not change the files to the online mode, all test files are changed to the local mode from the start of the experiment and this mode is maintained. Conversely, if the *USRT* value is 0 (usrt-0), only one file can be changed to the local mode because the active file mode change mechanism terminates the mode change operation when the storage usage exceeds 0% when changing the file to the local mode. Furthermore, if it determines that the directory is inactive, it changes all files to the online mode so that the storage usage is below 0%. During the directory activation period, as the file access progresses, the local storage usage increases when the online mode files are accessed. The larger the *USRT* value, the smaller the number of online mode files, so the storage usage becomes high. That is, from the viewpoint of the local storage usage, the smaller the *USRT* value, the better the performance.

Figure 18 measures the number of online mode file accesses according to changes in the *USRT*. As we can see from the graph, in terms of file access cost, the higher the *USRT* value, the better the performance. If the *USRT* value is large, more local storage can be used, so the number of online mode file accesses decreases as the proportion of the online mode files decreases. Therefore, as with *DSLAT* and *LMT*, users of the active file mode change mechanism can adjust the *USRT* value according to the important factors between the local storage usage and the file access cost. The *USRT* can control the upper and lower limits of the local storage usage but cannot control when storage usage increases or decreases as do the *DSLAT* and *LMT*.

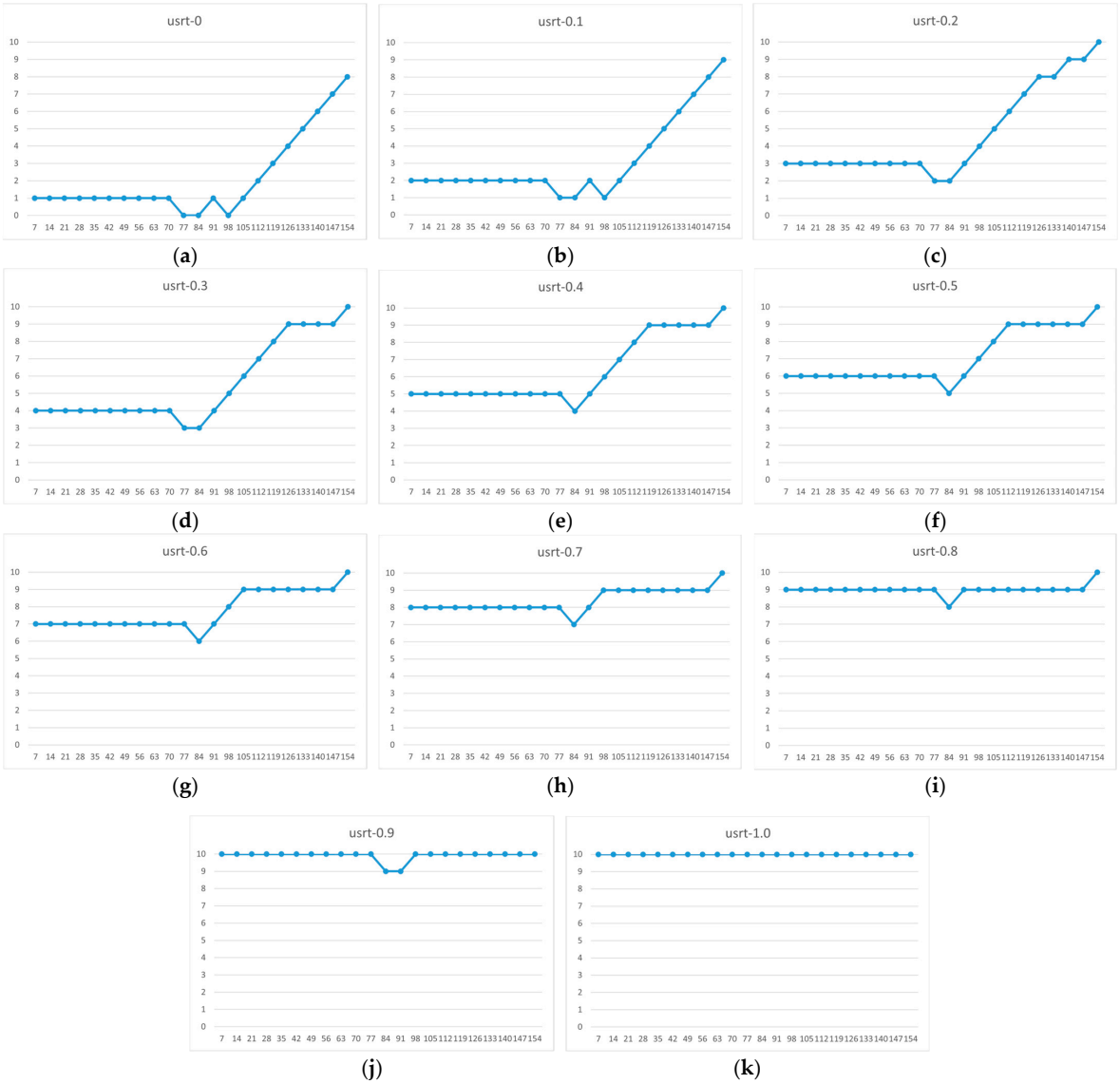


Figure 17. Change of local storage usage of active file mode change mechanism according to *USRT* in directory activation scenario. (**a**) usrt-0. (**b**) usrt-0.1. (**c**) usrt-0.2. (**d**) usrt-0.3. (**e**) usrt-0.4. (**f**) usrt-0.5. (**g**) usrt-0.6. (**h**) usrt-0.7. (**i**) usrt-0.8. (**j**) usrt-0.9. (**k**) usrt-1.0.

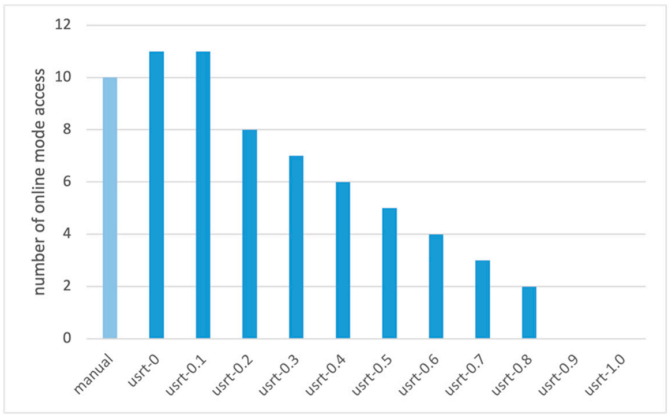


Figure 18. Change of the number of online mode file accesses according to *USRT* in directory activation scenario.

7. Discussion

The most distinctive feature of the proposed method in this paper is that the system automatically changes the file mode between the online mode and the local mode while monitoring the client's directory activation ratio. By controlling various parameter values, it is possible to flexibly adjust the time to change to the online and the local modes and the range of files to be changed. Table 4 summarizes the role of each parameter in the active file mode transition mechanism.

Table 4. Parameters and roles of active file mode transition mechanism.

Parameters	Roles
DSLAT	To control how long a file or a directory keeps the fully active state
OMT	To control when the change to the online mode starts.
LMT	To control when the change to the local mode starts.
USRT	To control when the change to the online or local mode stops.
MADT	To control minimum file size to be changed to the local mode

An additional consideration here is to find the optimal parameter values to maximize the overall performance. In this paper, we investigate the file mode change performance in terms of two aspects of local storage usage ratio and file access delay, which are in a trade-off relationship with each other. For example, as the *OMT* value increases, the local storage usage ratio improves because local mode files are easily changed to the online mode. On the other hand, as the number of online mode files increases, the average access delay of these files increases. Conversely, as the *OMT* value decreases, the local storage usage ratio increases, but the access delay decreases. When the user considers these two criteria with equal weight, we need a method for finding optimal parameter values. Additionally, the user may wish to assign different weights to the two criteria. Some users may deem access delay as the more important factor compared to the local storage usage ratio, and vice versa. If an appropriate parameter value can be presented according to the weight of each performance criterion, we can extend the active file mode transition mechanism to a user customized service.

8. Conclusions

In this paper, we proposed an active file mode change mechanism that automatically changes the file mode to online or local mode by considering the local storage usage ratio and file access delay time according to the directory activation ratio in the file synchronization service for cloud storage. Most research works regarding file synchronization do not support file mode change functionality [1–7]. In the existing manual file mode change method [8–10], the file mode is not changed without the user's request. On the other hand, the active file mode change mechanism selects files according to the directory activation ratio and changes them to the online or local mode, thereby maintaining an appropriate local storage usage ratio and file access latency. In addition, the client can adjust the file mode change timing and local storage usage ratio by changing various thresholds according to various requirements of normal clients and IoT devices. In future research, we plan to find the optimal parameter value that maximizes synchronization performance according to different weights of the local storage usage ratio and file access delay. We also plan to further analyze the performance of the active file mode change mechanism in more diverse file access scenarios.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2021R1F1A1047032).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Tridgell, A. Efficient Algorithm for Sorting and Synchronization. Ph.D. Thesis, Australian National University, Canberra, ACT, Australia, 1999.
2. Andriani, G.; Godoy, E.; Koslovski, G.; Obelheiro, R.; Pillon, M. An Architecture for Synchronising Cloud File Storage and Organisation Repositories. *Int. J. Parallel Emergent Distrib. Syst.* **2019**, *34*, 538–555. [CrossRef]
3. Drago, I.; Bocchi, E.; Mellia, M.; Slatman, H.; Pras, A. Benchmarking Personal Cloud Storage. In Proceedings of the 2013 Conference on Internet Measurement Conference, Barcelona, Spain, 23–25 October 2013. [CrossRef]
4. Li, Z.; Wilson, C.; Jiang, Z.; Liu, Y.; Zhao, B.Y.; Jin, C.; Zhang, Z.; Dai, Y. Efficient Batched Synchronization in Dropbox-like Cloud Storage Services. In Proceedings of the ACM/IFIP/USENIX 14th International Middleware Conference, Beijing, China, 9–13 December 2013. [CrossRef]
5. Han, S.; Shen, H.; Kim, T.; Krishnamurthy, A.; Anderson, T.; Wetherall, D. MetaSync: File Synchronization Across Multiple Untrusted Storage Services. In Proceedings of the USENIX Annual Technical Conference, Santa Clara, CA, USA, 8–10 July 2015.
6. Lopez, P.G.; Sanchez-Artigas, M.; Toda, S.; Cotes, C.; Lenton, J. StackSync: Bringing Elasticity to Dropbox-like File Synchronization. In Proceedings of the ACM/IFIP/USENIX 15th International Middleware Conference, Bordeux, France, 9–13 December 2014. [CrossRef]
7. Li, Z.; Zhang, Y.; Liu, Y.; Xu, T.; Zhai, E.; Liu, Y.; Ma, X.; Li, Z. A Quantitative and Comparative Study of Network-level Efficiency for Cloud Storage Services. *ACM Trans. Model. Perform. Eval. Comput. Syst.* **2019**, *4*, 1–32. [CrossRef]
8. Drago, I.; Mellia, M.; Munafo, M.M.; Sperotto, A.; Sadre, R.; Pras, A. Inside Dropbox: Understanding Personal Cloud Storage Services. In Proceedings of the 2012 Internet Measurement Conference, Boston, MA, USA, 14–16 November 2012. [CrossRef]
9. Dropbox Smart Sync. Available online: https://www.dropbox.com/business/smartsync (accessed on 29 September 2022).
10. OneDrive. Available online: https://onedrive.live.com/ (accessed on 29 September 2022).
11. Google Drive. Available online: https://www.google.com/drive/ (accessed on 29 September 2022).
12. Howard, J.; Kazar, M.; Menees, S.; Nichols, D.; Satyanarayanan, M.; Sidebotham, R.N.; West, M. Scale and Performance in a Distributed File System. In Proceedings of the eleventh ACM Symposium on Operating Systems Principles, Austin, TX, USA, 8–11 November 1987. [CrossRef]
13. Kistler, J.J.; Satyanarayanan, M. Disconnected Operation in the Code File System. *ACM Trans. Comput. Syst.* **1992**, *10*, 3–25. [CrossRef]
14. Bessani, A.; Mendes, R.; Oliveira, T.; Neves, N.; Correia, M.; Pasin, M.; Verissimo, P. SCFS: A Shared Cloud-backed File System. In Proceedings of the USENIX Annual Technical Conference, Philadelphia, PA, USA, 19–20 June 2014.
15. Ghemawat, S.; Gobioff, H.; Leung, S.T. The Google File System. *Oper. Syst. Rev. ACM* **2003**, *37*, 29–43. [CrossRef]
16. Muniswamy-Reddy, K.K.; Macko, P.; Seltzer, M. Provenance for the Cloud. In Proceedings of the 8th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, 23–26 February 2010.
17. Duan, H.; Yu, S.; Mei, M.; Zhan, W.; Li, L. CSTORE: A Desktop-oriented Distributed Public Cloud Storage System. *Comput. Electr. Eng.* **2015**, *42*, 60–73. [CrossRef]
18. Shvachko, K.; Kuang, H.; Radia, S.; Chansler, R. The Hadoop Distributed File System. In Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies, Incline Village, NV, USA, 3–7 May 2010. [CrossRef]
19. Chen, H.; Zhou, E.; Liu, J.; Zhang, Z. An RNN Based Mechanism for File Prefetching. In Proceedings of the 18th International Symposium on Distributed Computing and Applications for Business Engineering and Science, Wuhan, China, 8–10 November 2019. [CrossRef]
20. GlusterFS: A Scalable Network Filesystem. Available online: http://www.gluster.org/ (accessed on 21 April 2023).
21. Soundararajan, G.; Mihailescu, M.; Amza, C. Context-Aware Prefetching at The Storage Server. In Proceedings of the USENIX 2008 Annual Technical Conference, Boston, MA, USA, 22–27 June 2008.
22. Lin, L.; Li, X.; Jiang, H.; Zhu, Y.; Tian, L. AMP: Affinity-Based Metadata Prefetching Scheme in Large-Scale Distributed Storage Systems. In Proceedings of the Eighth IEEE International Symposium on Cluster Computing and the Grid, Lyon, France, 19–22 May 2008. [CrossRef]
23. ArchiverFS. Available online: https://www.mlteksoftware.com/Products/ArchiverFS/Index.aspx (accessed on 28 March 2023).
24. DataCore. Available online: https://www.datacore.com/blog/active-archive-object-storage/ (accessed on 28 March 2023).
25. Amazon S3 Intelligent-Tiering. Available online: https://aws.amazon.com/de/blogs/storage/automatically-archive-and-restore-data-with-amazon-s3-intelligent-tiering/ (accessed on 4 April 2023).
26. Cherubini, G.; Kim, Y.; Lantz, M.; Venkatesan, V. Data Prefetching for Large Tiered Storage Systems. In Proceedings of the IEEE International Conference on Data Mining, New Orleans, LA, USA, 18–21 November 2017. [CrossRef]
27. Alturkestani, T.; Tonellot, T.; Ltaief, H.; Abdelkhalak, R.; Etienne, V.; Keyes, D. MLBS: Transparent Data Caching in Hierarchical Storage for Out-of-Core HPC Applications. In Proceedings of the IEEE 26th International Conference on High Performance Computing, Data, and Analytics, Hyderabad, India, 17–20 December 2019. [CrossRef]

28. Qian, Y.; Li, X.; Ihara, S.; Dilger, A.; Thomaz, C.; Wang, S.; Cheng, W.; Li, C.; Zeng, L.; Wang, F.; et al. LPCC: Hierarchical Persistent Client Caching for Lustre. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, Denver, CO, USA, 17–19 November 2019. [CrossRef]
29. Khot, T.; Mathew, V.; Shenoy, P. *Adaptive Filetype Aware Prefetching*; Department of Computer Sciences, University of Wisconsin: Madison, WI, USA, 2010.
30. Devarajan, H.; Kougkas, A.; Sun, X. HFetch: Hierarchical Data Prefetching for Scientific Workflows in Multi-Tiered Storage Environments. In Proceedings of the IEEE International Parallel and Distributed Processing Symposium, New Orleans, LA, USA, 18–22 May 2020. [CrossRef]
31. Lamport, L. The Part-time Parliament. *ACM Trans. Comput. Syst.* **1998**, *16*, 133–169. [CrossRef]
32. Lim, M. C2CFTP: Direct and Indirect File Transfer Protocols between Clients in Client-server Architecture. *IEEE Access* **2020**, *8*, 102833–102845. [CrossRef]
33. Lim, M. A File Synchronization Framework Based on Rsync Protocol for Cloud Storage Services. *Trans. Korean Inst. Electr. Eng.* **2022**, *71*, 1164–1175. [CrossRef]
34. Moon, Y.; Lim, M. An Enhanced File Transfer Mechanism Using Additional Blocking Communication Channel and Thread for IoT Environments. *Sensors* **2019**, *19*, 1271. [CrossRef] [PubMed]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.