Active File Mode Transition Mechanism in File Synchronization

Subjects: Computer Science, Information Systems

Contributor: Mingyu Lim

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.

active file mode change directory activation ratio file synchronization

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][2][3][4][5][6][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][9][10][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][13][14][15][16][17][18][19][20] and other multi-tiered storage systems [21][22][23][24][25][26][27][28][29][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.

2. Active File Mode Transition Mechanism in File Synchronization

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 research 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][3][4][5][6][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][10][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][13][14][15][16][17][18][19][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 information 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,

researchers 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	No

	Storage Category	Consistency	Main Purpose	Storage Hierarchy	Transition Target	Transition Type	Transition Criteria	Transition Direction Priority
							pattern	
[19]	Distributed File System	Weak	Fast access	No	File	Prefetch	File access pattern	No

em, it is
rove file

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.
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.
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.
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.
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.
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.
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.
13. Kistler, J.J.; Satyanarayanan, M. Disconnected Operation in the Code File System. ACM Trans. Comput. Syst. 1992, 10, 3–25.
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.
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.
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.
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.
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.
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.
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.
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.
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.
31. Lamport, L. The Part-time Parliament. ACM Trans. Comput. Syst. 1998, 16, 133–169.