Block-level Cloud Tiering System for XFS

Mokshad Vaidya
Department of Information Technology
Pune Institute of Computer Technology
Pune, India

Vaibhav Mahajan
Department of Information Technology
Pune Institute of Computer Technology
Pune, India

Soham Kulkarni
Department of Information Technology
Pune Institute of Computer Technology
Pune, India

Gargi Mhaskar
Department of Information Technology
Pune Institute of Computer Technology
Pune, India

Shweta Dharmadhikari, PhD
Department of Information Technology
Pune Institute of Computer Technology
Pune, India

Swarnika Maurya
Software Engineer
Veritas Technologies LLC
Pune, India

ABSTRACT

Data storage and management in modern computing environments pose significant challenges, with a growing need for efficient and cost-effective solutions. Cloud tiering has emerged as a promising option, enabling efficient management and optimization of data placement across the local and cloud tier on the basis of various criteria such as access frequency, performance requirements, and cost considerations. XFS is a high-performance file system primarily used in Unix-like operating systems that can efficiently handle large amounts of data. This paper discusses the implementation of block-level tiering in XFS File System.

General Terms

Cloud Computing, Machine Learning

Keywords

Cloud Tiering, Storage Management, Cold Data, XFS

1. INTRODUCTION

With the massive amounts of data being stored by all organizations nowadays, it has become crucial to optimize the storage of that data to reduce cost and latency while improving on the throughput, reliability and availability. One of the strategies for storage optimization is data tiering. It is a data storage strategy that defines different storage tiers and shifts data between tiers to fulfil the requirements of an organization depending on the usage. Cloud tiering involves storing frequently accessed data in fast cloud storage while less used data is assigned to slower and cheaper storage. Advantages of cloud tiering also include more granular control over cloud resources and rapid data recovery in case of emergencies such as natural disaster or sudden system failure. Extending this need for storage optimization to the file system present on compute clusters and on-premise servers, we can utilize the primary, more expensive storage more judiciously by shifting a bulk of the data to secondary storage services such as AWS S3 and Google Cloud Storage, which implement a Pay-As-You- Go model with high durability. Some considerations need to be made while deciding on the type of cloud tiering that would be most suited to a particular organization. These considerations are as follows:

- *Block-Level vs File-Level Tiering*: File-level tiering operates with greater granularity, enabling precise control over which files are transferred to lower-cost storage tiers based on their access patterns and significance. This method maximizes savings without binding organizations to specific storage vendors or technologies. On the other hand, block-level tiering functions at a more granular level, relocating data in fixed-size blocks between tiers according to usage patterns. Although block-level tiering offers advantages like faster data retrieval, file- level tiering delivers more flexibility and cost-efficiency, particularly in diverse environments that utilize both on- premises and cloud storage. [7]
- *Identification of Cold Data*: Accurately identifying cold blocks is essential when implementing cloud tiering. Either the user can be given complete control over which data should be designated as cold data. Another method is imposing a deadline on data, beyond which the data would be declared to be cold. For example, data not accessed within a week or a month can be categorized as cold data. Otherwise, cold blocks can be detected using algorithms like Least Recently Used (LRU) and AI- based ensemble models, which take into account factors such as access frequency, file size, and user information. By analyzing metadata, it is possible to identify cold data by considering both the frequency and recency of access.
- *Primary Storage Utilization and Secondary Tier Read-Write Costs*: Cloud tiering should optimize primary storage use without data loss, preventing both overuse and under use. Methods to ensure this include setting a maximum utilization limit for primary storage, beyond which cold data is tiered, or defining a hot-cold storage ratio. The costs of reading from and writing to secondary storage must also be considered. A policy should address frequently accessed data in the secondary tier, migrating it back to primary storage. Monitoring cloud tier access patterns helps determine which data to return to primary storage, factoring in current primary storage utilization, migration costs, and secondary tier read costs.

XFS is a high-performance, journaling file system for Linux, known for its scalability and near-native I/O performance, even across multiple storage devices. It is favored by large-scale enterprises with substantial data and performance needs due to its significant advantages.

In this paper, we have demonstrated a Proof of Concept for implementing block level cloud tiering in XFS. The system uses IOCTL (Input/Output Control) and system commands for deallocating blocks of data and transferring them to the cloud tier along with a machine learning algorithm to distinguish between cold and hot data. The system operates in the background so the user will be unaware of where their data is actually stored. The proposed solution is also modularized so clients can, for example, plug-in the hot- cold identification they desire. The system demonstrates the feasibility of a block-level cloud tiering mechanism tailored to XFS that will allow for effective usage of both tiers of storage for large-scale enterprises. The rest of this paper is organized as follows: In Section II, relevant literature referencing intelligent cloud tiering systems and the functionalities of XFS is discussed. In Section III, the concept along with the system architecture is described. In Section IV, certain results and advantages of the concept are discussed. In Section V, a discussion of future research plans is provided. In Section VI, this paper is concluded.

2. LITERATURE SURVEY

This section presents an overview of the different methodologies used by various cloud tiering supports and a review of XFS. To understand cloud tiering better, multiple different researches proposing different methodologies for cold-block identification and tiering have been reviewed. The differences can be better understood by comparing these researches on certain parameters, which are detailed below:

- *Cold Block Identification:* The Novel Automated Cloud Storage Tiering System proposed in [1] predicts the data temperature by learning from past data access pat- terns and user behaviors, applying a stacked ensemble machine learning model. The prediction engine utilizes various input features such as file access frequency, size, type, user information, etc. If a file is estimated to be accessed in the next month, it will continue to stay in the hot storage; otherwise, it'll be moved to cold storage. In FabricPool [2] when a block is written to the local tier, it is assigned a temperature value indicating that it is hot. Over time, a background cooling scan cools blocks, making hot blocks warm and eventually turning blocks cold if they have not been read. Assuming no activity, a block becomes cold based on the time set by the user. [3] uses Deep Reinforcement Learning to dynamically classify data into different storage tiers based on its access patterns and cost considerations. [4] recommends a rule-based classification scheme that utilizes IF-THEN rules for predicting the storage tier for each object based on specific criteria such as access frequency, data size, and age of the stored object.
- *Type of Tiering:* [1] implements File-level tiering while [2] is an example of Block-level Tiering. In [2] when enough 4KB blocks from the same volume have been collected, they are concatenated into a 4MB object and moved to the cloud tier. [3] and [4] refer to data as objects and perform tiering accordingly.
- *System:* [1] and [4] are supposed to be deployed by cloud service providers. [2] works on ONTAP, NetApp's proprietary operating system used in storage disk arrays.
- *Special Features:* In [1], the authors evaluate their system by considering Feasibility, Reliability, and Quality of Experience (QoE). Their system comprises of the Data Content Manager, the Hot-Cold Prediction Engine, and the subsequently generated file priority list. The system allocates data to the appropriate storage tier without manual intervention. [2] introduces special conditions to ensure that the local tier is optimally utilized by tiering to the cloud tier only if the local tier is >50% full to prevent underutilization. It also has write-back prevention for when the local tier is heavily utilized. [3]'s use of Reinforcement Learning ensures that the agent learns optimal tiering policies through trial and error, maximizing long-term rewards (cost reduction) over time. [4] defines 4 tiers (Premium, Hot, Cold, Archive) instead of 2 which enables more effective allocation of data to appropriate storage tiers based on their priority scores and ensures optimal cost management while meeting data access and availability needs.

XFS is a high-performance 64-bit journaling file system renowned for its capability to handle large volumes of data efficiently. It maintains low performance degradation even when processing numerous files and nodes. XFS offers a variable- length block size mechanism, adjustable to meet system needs. XFS excels in parallel input/output (I/O) operations due to its design, which utilizes allocation groups—subdivisions of physical volumes. This design enables exceptional scalability, accommodating up to 110 threads, and optimizing file system bandwidth and file size when spanning multiple physical storage devices. Each allocation group manages its own nodes and free space, allowing multiple processes and threads to operate concurrently, thereby enhancing overall performance.

[5] IOCTL (Input/Output Control) commands in XFS are used to perform various control operations on files and file systems, beyond the capabilities of standard system calls. These commands allow for direct interaction with the file system to manage aspects like allocation, querying status, or performing administrative tasks. One notable IOCTL command in XFS is the "punch hole" command, which deallocates space within a file without altering the file's logical size. This operation is useful for freeing up unused space in sparse files, optimizing storage usage, and enhancing performance by reducing the amount of physical storage required. [6]

3. SYSTEM ARCHITECTURE AND METHODOLOGY

In this section, the proof of concept and the system architecture and methodology used is elaborated upon. The concept is based on block-level cloud tiering. Figure 1 shows the important components of the system. These components are further detailed below:

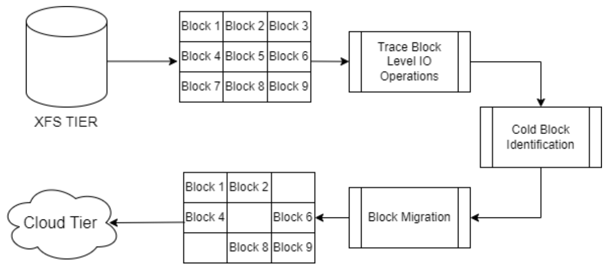


Figure 1 System Architecture

3.1 XFS Tier

The XFS tier is the designated local or primary storage, storing numerous files. Further, a file in XFS consists of many blocks. Hence, blocks in the context of file systems represent fixedsized units of data storage used to organize and manage files on storage media. XFS, specifically allows file systems to be created with block sizes ranging between 512 bytes and 64 KB, depending on the user's usage. The byte representation of the blocks also depends on the system command being used.

3.2 Tracing Block-Level IO Operations

The output of a system command called btrace provides detailed insights into block-level I/O events occurring within the system. Each entry in the output corresponds to a specific I/O operation, including read and write operations, along with relevant metadata such as the process ID, file descriptor, block number, and operation type. The logs are preprocessed to compute the Frequency of Access, Recency of Access, Mean Time Between Access, and the Standard Deviation of Access Time for each block. This data is then used for cold-block identification.

3.3 Cold Block Identification

Artificial Intelligence models provide a way to use historical data to predict future access patterns. Deep Learning Models, Neural Networks and Stacked Ensemble Models have been known to be precise with their predictions. As described in Section 1, the identification could also be based on certain time limits such as weekly or monthly. With the system, the aim was to create an automatic system that did not require manual intervention, hence the use of machine learning. Furthermore, the dataset used was unlabeled, prompting the use of the K-means clustering algorithm for creating two clusters: Hot Blocks and Cold Blocks. Figure 2 demonstrates the methodology for generating the list of Hot and Cold blocks. The features used for K-Means algorithm and their importance is as follows:

• Frequency: Frequency refers to the number of times a block has been accessed within a certain period. It indicates the level of activity or usage of a block. A higher frequency suggests that the block is accessed more frequently, indicating its importance or relevance in the system, and vice versa.

• Recency: Recency measures the time elapsed since the last access to a block. It reflects the temporal aspect of block access patterns. Blocks that have been accessed recently are more likely to be relevant or active compared to those accessed a long time ago.

• Mean Time Between Access (MTBA): MTBA represents the average time interval between consecutive accesses to a block. It provides insights into the average frequency or regularity of block access over time. A low MTBA indicates that blocks are accessed frequently, with shorter intervals between accesses, suggesting high activity or demand. Conversely, a high MTBA implies longer intervals between accesses, indicating lower activity or less demand for the block.

• Standard Deviation in Access Time: Standard deviation measures the dispersion or variability of access times for a block. It quantifies the extent to which access times deviate from the mean access time. A higher standard deviation indicates greater variability in access times, suggesting inconsistent or irregular access patterns.

Roman in which these guidelines have been set. The goal is to have a 9-point text, as you see here. Please use sans-serif or non-proportional fonts only for special purposes, such as distinguishing source code text. If Times Roman is not available, try the font named Computer Modern Roman. On a Macintosh, use the font named Times. Right margins should be justified, not ragged.

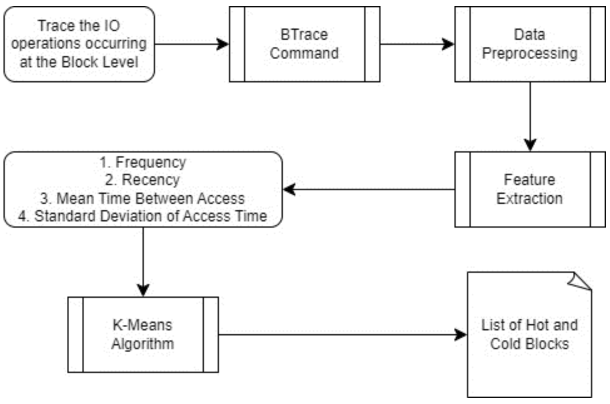

Trace the IO operations occurring at the Block Level
BTrace Command
Data Preprocessing
Feature Extraction
1. Frequency
2. Recency
3. Mean Time Between Access
4. Standard Deviation of Access Time
K-Means Algorithm
List of Hot and Cold Blocks

Figure 2 Tracing I/O Operations and Identifying Hot-Cold Blocks

3.4 Block Migration

The Block Migration is a four-step process. As the punch- hole IOCTL command requires a file path and offset value to deallocate the blocks. Hence, the first step involves mapping the block numbers of the cold blocks to the respective file path and calculating the offset. The next step is to store the block number, file name and offset in a table to ensure that local tier has access to information about the blocks that have been migrated to the cloud tier so as to perform read-write operations. The third step involves iteratively uploading the cold blocks to the cloud tier. The final step is performing the punch-hole operation and deallocating space in the local tier. The "punch hole" operation essentially reclaims space without having to rewrite the entire file. The third and fourth step occur simultaneously: every block that has been successfully uploaded to the cloud tier is immediately deallocated in the local tier.

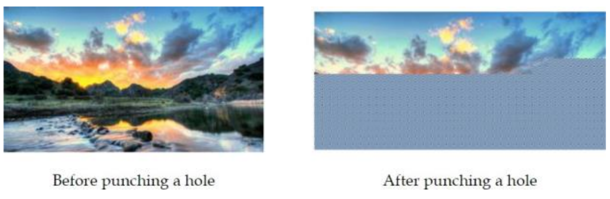
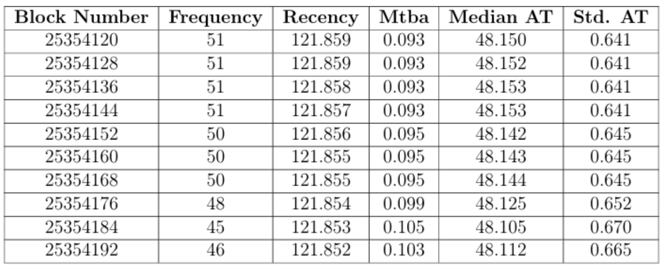


Figure 3 Demonstration of Punch-Hole Operation



Block Number	Frequency	Recency	Mtba	Median AT	Std. AT
25354120	51	121.859	0.093	48.150	0.641
25354128	51	121.859	0.093	48.152	0.641
25354136	51	121.858	0.093	48.153	0.641
25354144	51	121.857	0.093	48.153	0.641
25354152	50	121.856	0.095	48.142	0.645
25354160	50	121.855	0.095	48.143	0.645
25354168	50	121.855	0.095	48.144	0.645
25354176	48	121.854	0.099	48.125	0.652
25354184	45	121.853	0.105	48.105	0.670
25354192	46	121.852	0.103	48.112	0.665

Figure 4 Sample of Processed Block Data

4. RESULTS AND DISCUSSION

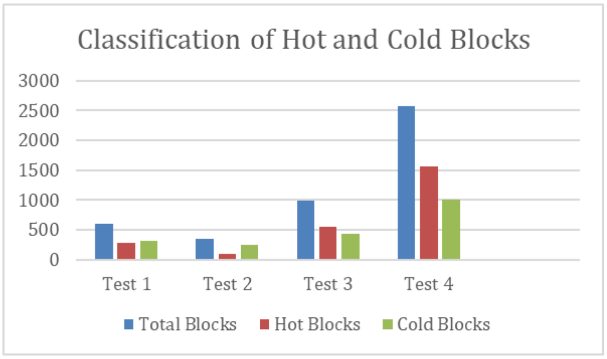


Figure 5 Testing of Hot and Cold Block Classification

As big data becomes increasingly prevalent, the utilization of the XFS file system rises due to its clear advantages in managing substantial data loads. Further, the need for Cloud Tiering is also significantly rising due to the reliability, cost effectiveness and versatility of cloud storage. The Input/Output Commands (IOCTL) along with the journaling mechanism offered by XFS lends itself to the implementation of block-level cloud tiering. Block-level tiering offers the advantage of faster data retrieval.

To evaluate the system, a dataset simulating organizational usage through random block access was generated. After preprocessing, K-means clustering was applied. As can be understood from the above graph, K-means clustering resulted in dynamic identification of cold blocks: 52.6% in test 1, 71.1% in test 2, 43.8% in test 3, and 38.9% in test 4. These cold blocks were then migrated to the cloud tier, freeing up space in the primary tier. Adjusting the number of clusters in the model allows for further refinement of the cloud tiering process.

The proposed system has a modular architecture which allows for different users to plug-in different components in the system as per their requirements without affecting the functionality of the rest of the system. This implies that different users could use different log tracing methods, cold- block identification algorithms, or block migration commands depending on their usage of the file system and cloud tier. Another advantage of the solution is that the system is independent of the operating system. The system can operate on any OS that has XFS mounted on it. Furthermore, the Cloud Service Provider are not required to provide any additional infrastructure to be compatible with the system.

Overall, the solution provides comprehensive proof that the concept of block-level tiering can be successfully and efficiently implemented on XFS, providing a cost-effective method of handling big data.

5. FUTURE SCOPE

This project aspired to provide a Proof-of-Concept model for implementing cloud tiering in XFS. Therefore, the scope was limited to a basic modularized implementation using Punch-hole IOCTL commands. However, in the future every component in the architecture can be refined to be better suited to the needs of big data enterprises.

For this solution, we had to generate our own dataset of IO commands and later use a clustering algorithm for cold block identification due to the absence of a dataset. In the future, a genuine and labelled dataset of Hot and Cold data can be generated using user input from big data enterprises. A genuine dataset will allow us to compute metrics such as precision of predictions and write-back costs. This labelled dataset can then be used to train Neural Networks or Classification algorithms that produce more accurate results than clustering algorithms. Moreover, the identification algorithm can be modified to either give more control to users to choose how and when tiering happens, or to make the system entirely automatic, depending on the requirements of the organization using the system. Further modifications can be made to the algorithm to ensure that the local tier is neither underutilized nor over-utilized.

Another interesting possibility is increasing the number of tiers in the cloud storage. Differentiating between two or more cloud tiers will better capture the nuances of the usage patterns of data, and make the system more cost effective. However, it would lead to an increase in the access time of the data depending on the type of cloud selected.

6. CONCLUSION

In this paper, a proof of concept for implementing block-level cloud tiering in XFS file system has been demonstrated. The system leverages the advanced capabilities of XFS, including its journaling and IOCTL command features, to effectively manage and optimize data storage across local and cloud tiers. By utilizing a machine learning algorithm for cold-block identification, efficient data placement based on access patterns, enhancing performance and cost-efficiency has been ensured. The modular architecture allows for customization and adaptability to different user requirements and operating environments.

The findings indicate that block-level tiering on XFS can significantly improve storage management for large-scale enterprises, offering faster data retrieval and reducing primary storage costs without sacrificing performance. The ability to seamlessly migrate data between tiers, combined with the robust scalability of XFS, positions this solution as a viable strategy for handling the growing demands of big data.

Future research can build upon this foundation by refining the components of the system, incorporating genuine datasets for more accurate predictions, and exploring multi-tier cloud storage strategies. These advancements will further enhance the utility and effectiveness of block-level tiering, providing a comprehensive, cost-effective solution for modern data storage challenges.

7. ACKNOWLEDGEMENTS

We thank Veritas Technologies LLC for the idea to conduct novel research in this domain. We also express our gratitude towards Mr. Sumit Dighe (Veritas Technologies LLC) for providing his technical guidance during the course of our project

8. REFERENCES

[1] Y.-F. Hsu, R. Irie, S. Murata, and M. Matsuoka, “A novel automated cloud storage tiering system through hot-cold data classification,” 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), 2018.

[2] J. Lantz, “FabricPool best practices TR-4598 NetApp.” https://www.netapp.com/pdf.html?item=/media/17239-tr-4598.pdf, 2024

[3] M. Liu, L. Pan and S. Liu, “RLTiering: A Cost-Driven Auto-Tiering System for Two-Tier Cloud Storage Using Deep Reinforcement Learning,” in IEEE Transactions on Parallel and Distributed Systems, vol. 34, no. 2, pp. 501-518, 1 Feb. 2023

[4] A. Q. Khan, N. Nikolov, M. Matskin, R. Prodan, C. Bussler, D. Roman, A. Soylu, “Towards Cloud Storage Tier Optimization with Rule-Based Classification” in 10th IFIP WG 6.12 European Conference, ESOCC 2023, October 24–25, 2023, Proceedings.

[5] Z. Wang, “Research of data storage mode and recovery method based on XFS file system,” in 7th IEEE International Conference on Software Engineering and Service Science (ICSESS), Beijing, pp. 369-372, 2016.

[6] J. Corbet, “XFS: the big storage file system,” LWN.net, Nov. 10, 2010.

[7] ”Block-Level Tiering vs File-Level Tiering,” Komprise, 2019.

[8] A. Raghavan, A. Chandra, and J. B. Weissman, “Tiera,” Proceedings of the 15th International Middleware Conference on- Middleware ’14, 2014.

[9] N. J. Mutkawoa, “Debugging Disk Issues with blktrace, blkparse, btrace and btt in Linux Environment,” Tunnelix, May 2024.