CrossFS: Improving Cross-Domain File System Performance with CRDT-Based Metadata Synchronization

QIWEN KE, *School of Informatics, Xiamen University*, China
YINA LV, *School of Informatics, Xiamen University*, China
ZHIHAO ZHANG, *School of Informatics, Xiamen University*, China
ZHIRONG SHEN, *School of Informatics, Xiamen University*, China
YUE YU, *Peng Cheng Laboratory*, China
HAILIANG CHEN, *NUDT*, China
ZHENLONG SONG, *NUDT*, China
XINBIAO GAN, *NUDT*, China
JIAXIN LI, *NUDT*, China
DONGSHENG LI, *NUDT*, China
XIN YAO, *Huawei Theory Lab*, China
MEILING WANG, *Huawei Theory Lab*, China
YIMING ZHANG, [1]*Shanghai Jiao Tong University*, [2]*Peng Cheng Laboratory*, China

Modern data-intensive applications increasingly demand efficient and scalable file systems that can operate across distributed and cross-domain environments. However, existing file systems are inefficient in metadata management, synchronization efficiency, and system scalability under high-concurrency and metadata-intensive workloads in cross-domain environments. To address these challenges, this paper introduces CrossFS (CFS), a cross-domain distributed file system that enhances consistency guarantees and metadata indexing. Specifically, CFS leverages conflict-free replicated data types (CRDTs) to synchronize metadata, achieving strong eventual consistency with minimal synchronization overhead, even across network partitions. Furthermore, CFS employs a Hybrid Tree indexing structure, tailored for distributed environments, which optimizes metadata operations by reducing query latency by up to 33.4% and write amplification by 30.7%. Additionally, CFS achieves adaptive caching strategies and a hybrid synchronization model that effectively balances consistency latency with data availability. Extensive evaluations show that CFS outperforms CephFS and GlusterFS, achieving up to 33.9% higher metadata throughput, 36% lower latency, and 42% better data operation efficiency.

This work is supported by the National Key Research and Development Program of China (grant no. 2022YFB4500300) and the National Natural Science Foundation of China (grant no. 62441220 and 62302514).

Qiwen Ke, Yina Lv, and Zhihao Zhang are co-primary authors.

Yiming Zhang is the corresponding author.

Authors' Contact Information: Qiwen Ke, kerneld@qq.com, *School of Informatics, Xiamen University*, Xiamen, Fujian, China; Yina Lv, *School of Informatics, Xiamen University*, Xiamen, Fujian, China, elainelv95@gmail.com; Zhihao Zhang, *School of Informatics, Xiamen University*, Xiamen, Fujian, China, zhihaoz@stu.xmu.edu.cn; Zhirong Shen, *School of Informatics, Xiamen University*, Xiamen, Fujian, China, shenzr@xmu.edu.cn; Yue Yu, *Peng Cheng Laboratory*, Shenzhen, China, yuy@pcl.ac.cn; Hailiang Chen, *NUDT*, Changsha, China, chenhailiang_15@163.com; Zhenlong Song, *NUDT*, Changsha, China, songzhl@sina.com; Xinbiao Gan, *NUDT*, Changsha, China, xinbiaogan@163.com; Jiaxin Li, *NUDT*, Changsha, China, licyh@nudt.edu.cn; Dongsheng Li, *NUDT*, Changsha, China, dsli@nudt.edu.cn; Xin Yao, *Huawei Theory Lab*, Hong Kong, China, yao.xin1@huawei.com; Meiling Wang, *Huawei Theory Lab*, Shenzhen, China, wangmeiling17@huawei.com; Yiming Zhang, [1]*Shanghai Jiao Tong University*, [2]*Peng Cheng Laboratory* , Shanghai, China, sdiris@gmail.com.

ACM 1557-735X/2018/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

CCS Concepts: • **Software and its engineering → Software development methods**.

Additional Key Words and Phrases: Distributed File Systems, Metadata Synchronization, Conflict-Free Replicated Data Types (CRDTs), FUSE

ACM Reference Format:
Qiwen Ke, Yina Lv, Zhihao Zhang, Zhirong Shen, Yue Yu, Hailiang Chen, Zhenlong Song, Xinbiao Gan, Jiaxin Li, Dongsheng Li, Xin Yao, Meiling Wang, and Yiming Zhang. 2018. CrossFS: Improving Cross-Domain File System Performance with CRDT-Based Metadata Synchronization. *J. ACM* 37, 4, Article 111 (August 2018), 25 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Traditional local file systems, such as Ext4 [41], XFS [39], and Btrfs [35], have been proven highly effective in handling large files and providing fast access to data stored on local disks. However, as enterprise operations increasingly adopt distributed architectures that span geographical boundaries, traditional local file systems frequently encounter substantial performance degradation [10]. This is because traditional local file systems are not inherently designed to efficiently handle large volumes of small files, while metadata-intensive workloads and frequent access to small files are common in cross-domain distributed environments [7], resulting in metadata overhead and performance penalties. To better manage the resources under distributed architectures, distributed file systems like HDFS [16], GFS [12], CephFS [18], and GlusterFS [5] have been adopted to enhance data sharing and access capabilities. However, these systems often face challenges in maintaining data consistency and effectively managing metadata across different network partitions [15]. The CAP theorem asserts that in distributed systems, especially those prone to network partitions, it is impossible to simultaneously achieve perfect consistency, availability, and partition tolerance [25]. This dilemma becomes particularly acute in scenarios involving cross-domain active-active replication, where maintaining partition tolerance is critical. Furthermore, the high latency and potential network partitions inherent in such settings can adversely affect system availability and user experience, complicating the implementation of common consistency protocols [2, 24, 30].

To address these challenges, we introduce CrossFS (CFS), a novel cross-domain file system that utilizes conflict-free replicated data types (CRDTs) [33] to ensure strong eventual consistency (SEC) while maintaining high availability and partition tolerance. CRDTs are applied to resolve conflicts in distributed environments without the need for complex synchronization protocols, thus making them ideal for deployment in cross-domain file systems [36]. By organizing metadata into a sparse table and managing it through a log-structured merge tree (LSM-tree) [6], CFS significantly reduces metadata overhead and enhances access efficiency. To address the write amplification and latency issues inherent in using standard RocksDB [11] for metadata-intensive workloads, CFS further optimizes configurations of RocksDB alongside its use of the FUSE [34] and rest_rpc [32] frameworks. Fine-tuned settings for batch writing and asynchronous commits significantly improve write efficiency and concurrent metadata modifications, which ensure robust data integrity and faster system recovery after failures.

Based on these technologies, CFS provides a scalable and efficient solution for cross-domain data management, effectively addressing the demands of modern, geographically dispersed applications. The main contributions of this paper are threefold: first, the use of CRDTs for conflict-free synchronization; second, the Hybrid Tree optimization for high-performance metadata querying; and third, optimized RocksDB configurations that boost write performance and reduce write amplification. Furthermore, the system leverages FUSE and rest_rpc to enable efficient cross-domain metadata synchronization and communication. Evaluations show that CFS achieves significant performance gains over existing systems, with up to 33.9% higher metadata throughput and 33.4% lower query latency compared to state-of-the-art distributed file systems like CephFS and GlusterFS [31]. Furthermore, CFS reduces write amplification by 30.7% and improves data operation efficiency by 42%, showcasing its ability to handle diverse workloads and meet the demands of metadata-intensive scenarios.

The remainder of this paper is organized as follows. Section 2 provides background on the evolution of distributed file systems, metadata management challenges, and the role of CRDTs in ensuring consistency. Section 3 presents the design and implementation of CFS, focusing on its architecture and key technical innovations. Section 4 evaluates the performance of CFS through comprehensive experiments, comparing it to existing local and distributed file systems. Section 5 reviews related work and highlights the unique contributions of CFS. Finally, Section 6 concludes the paper, summarizing key contributions and outlining directions for future research.

2 Background and Motivation

The rapid evolution of distributed computing environments, driven by the proliferation of data-intensive applications and the increasing need for real-time analytics, facilitates the development of sophisticated file systems that can efficiently manage data across distributed networks [14]. Traditional local file systems like Ext4, XFS, and Btrfs are optimized for single-node operations and are not inherently designed to handle the complexities of distributed computing, such as network latency, data consistency across nodes, and fault tolerance [10, 27, 40]. To understand the motivations behind the design of modern distributed file systems like CrossFS, it is crucial to explore the evolution of distributed storage, the challenges in metadata management, and the recent advancements in consistency and synchronization protocols.

2.1 Evolution of Distributed File Systems

The inception of distributed file systems (DFS) marked a significant transition from localized to geographically dispersed data storage solutions [12, 16, 18]. This evolution was driven by the growing data needs of large enterprises and the increasing complexity of application workflows that demanded high availability and scalability. Early distributed file systems, such as the Google File System (GFS) and the Hadoop Distributed File System (HDFS), were designed to meet the challenges of large-scale data storage and parallel data processing. These systems divide large files into smaller blocks and distributed them across multiple nodes to achieve scalability, fault tolerance, and parallelism in data access.

GFS and HDFS laid the foundation for future DFS architectures by implementing distributed metadata management and replication strategies to ensure data availability and reliability [10]. However, as these systems scaled, managing the metadata of millions or even billions of files became an increasingly significant challenge. Later systems, such as CephFS and GlusterFS, attempted to address these challenges by decentralizing metadata management and optimizing data access through more sophisticated caching and replication mechanisms. CephFS, for example, introduced the concept of a dynamic metadata cluster that could grow or shrink based on workload, thus allowing better scalability and avoiding metadata bottlenecks. Similarly, GlusterFS used an elastic hashing mechanism to distribute metadata across nodes, providing improved load balancing.

While these advancements improved scalability and availability, they often came at the cost of increased system complexity and a heavier reliance on network communication, leading to challenges in maintaining consistency and reducing latency [10]. Metadata operations, such as opening, renaming, or deleting files, became the bottleneck, particularly when workloads involved numerous small files or highly dynamic access patterns. As a result, the need for a more robust solution that could efficiently handle metadata management while maintaining strong consistency emerged as a key research area in distributed storage systems.

2.2 Key Challenges in Cross-Domain File Systems

In this paper, "cross-domain" refers to distributed file systems that span multiple administrative domains or geographically dispersed sites. While traditional DFSs and cross-domain DFSs share key challenges such as metadata scalability, consistency, and fault tolerance, the cross-domain setting introduces further complexities. Firstly, administrative heterogeneity means nodes are managed by different organizations with varying policies, authentication domains, and trust boundaries, which complicates coordination and security. Secondly, network diversity and latency arise because cross-domain deployments typically rely on wide-area networks (WANs), incurring higher latency, lower bandwidth, and increased partition risk compared to intra-datacenter networks. Finally, autonomous policy conflicts arise when domains independently define access controls, replication strategies, or update schedules, leading to more frequent metadata divergence and complex conflict resolution.

To provide efficient, scalable, and reliable data and metadata management for distributed file systems and modern cross-domain environments, the following main challenges should be addressed.

Challenge 1: Scalability of Metadata Management. In distributed systems, metadata operations are the primary source of performance bottlenecks, especially as the system scales to billions of files or directories. Traditional centralized metadata management, as used in systems like HDFS with its single NameNode, imposes significant scalability constraints and creates single points of failure. While decentralized metadata management, as implemented in CephFS and GlusterFS, mitigates some of these issues, it introduces high coordination costs and potential bottlenecks during frequent updates or conflicts. These challenges are further exacerbated in cross-domain systems, where metadata must be synchronized across geographically dispersed nodes. This synchronization increases the overhead of maintaining consistency and reducing system responsiveness.

Challenge 2: Balancing Consistency, Availability, and Performance. The CAP theorem highlights the inherent trade-offs in distributed systems, where achieving consistency, availability, and partition tolerance simultaneously is impossible. Cross-domain environments, characterized by frequent network partitions and high-latency interconnects, make these trade-offs even more pronounced. Maintaining strong consistency often incurs significant latency due to global coordination requirements, whereas eventual consistency introduces challenges in resolving conflicts and ensuring data integrity. A key challenge is designing a lightweight yet robust consistency mechanism that balances these trade-offs while maintaining low-latency operation and high availability.

Challenge 3: High-Concurrency Metadata Query Efficiency. Modern applications, such as real-time analytics, AI training, and large-scale simulations, demand efficient metadata access under high-concurrency workloads. Existing indexing solutions, such as the LSM-tree in RocksDB, struggle with high-frequency metadata updates and range queries due to compaction overhead and limited query optimization. Furthermore, traditional systems often fail to adapt to the mixed workloads seen in real-world scenarios, where frequent updates and read-heavy operations coexist. Optimizing metadata query performance under these conditions is critical for ensuring the scalability and usability of cross-domain file systems.

Additional Considerations: Small File Overheads and Fault Tolerance. Beyond the key challenges, small file operations and fault tolerance represent significant secondary considerations in distributed systems. Small files generate disproportionately high metadata overheads compared to large files, leading to degraded performance, particularly in scenarios involving frequent creation, deletion, and modification. Fault tolerance, essential in cross-domain settings, demands efficient recovery mechanisms that minimize data loss and downtime, particularly during network partitions or node failures. These considerations must be addressed holistically in the design of modern distributed file systems.

2.3 Advancements in Consistency and Synchronization

Achieving consistency in a distributed system is challenging due to the need to balance availability and partition tolerance, as described by the CAP theorem [22]. Traditional approaches to achieving consistency in distributed file systems rely on protocols like Two-Phase Commit (2PC)[2], Paxos[24], and Raft [30]. Among them, 2PC is a classic protocol used to achieve distributed consistency by coordinating commit or rollback decisions across participating nodes. While effective in ensuring atomicity, 2PC suffers from significant drawbacks, including the potential for indefinite blocking if a coordinator fails. Paxos and Raft are consensus algorithms designed to provide fault-tolerant consistency in distributed systems. However, the strong consistency provided by these protocols comes at the cost of significant latency, a result of the multiple communication rounds needed to reach consensus. This high overhead renders them impractical for systems that prioritize availability and responsiveness, especially across network partitions.

To address these limitations, conflict-free replicated data types (CRDTs) have emerged as a promising alternative for managing consistency in distributed systems [17, 42]. CRDTs are data structures that allow concurrent updates from multiple nodes to be merged in a way that ensures a consistent final state without the need for coordination. This property makes CRDTs particularly suitable for distributed file systems where network partitions are common, and availability is a priority. By using CRDTs, systems can achieve strong eventual consistency (SEC), ensuring that all replicas converge to the same state once all updates have been propagated.

In the context of metadata management, CRDTs can be used to handle operations such as file creation, deletion, and attribute modification without requiring complex locking mechanisms. For example, a G-Counter CRDT can be used to track the number of updates to a file, while an MV-Register CRDT can manage conflicting versions of metadata by retaining all versions until a resolution can be applied. This approach not only simplifies conflict resolution but also improves system availability by allowing updates to proceed independently at different nodes.

2.4 Implications for Modern Distributed File Systems

Recent studies in distributed systems have brought new opportunities and challenges for the design of large-scale file systems. The adoption of CRDT-based synchronization, LSM-tree-inspired metadata indexing, and hybrid storage architectures has reshaped the landscape of metadata management, consistency, and performance optimization [28]. These technologies allow distributed file systems to move beyond the limitations of centralized coordination and simple key-value indexing, enabling them to cope with high concurrency, geo-distribution, and frequent network partitions.

However, effectively integrating these technologies demands a carefully balanced architecture that addresses the trade-offs among scalability, availability, and data consistency, particularly in cross-domain environments. For example, while LSM-tree structures excel in write-intensive scenarios, their compaction overhead can degrade metadata query performance under specific access patterns. Conversely, although CRDTs provide conflict-free merging and high availability, they require configuration across different metadata types to achieve optimal results.

In designing CFS, we draw inspiration from these state-of-the-art techniques, combining CRDT-based metadata synchronization, a customized Hybrid Tree index for efficient range and versioned queries, and adaptive caching to accommodate diverse workloads and dynamic access patterns. The integration of these components allows CFS to achieve scalable and robust data management across geographically dispersed sites. In the following, we discuss the specific design choices and implementation strategies used in CFS in detail, highlighting how these technologies are orchestrated to meet the requirements of modern cross-domain distributed file systems.

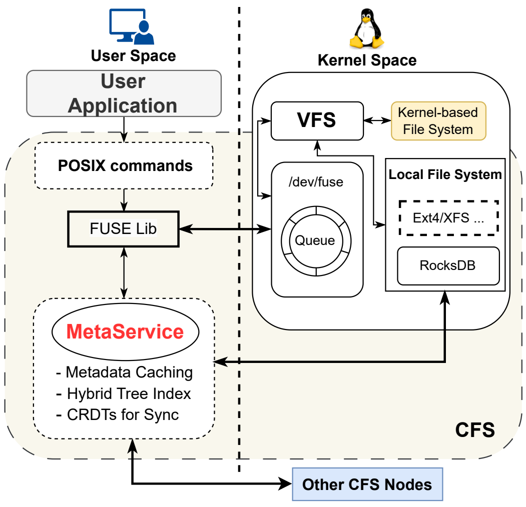


Fig. 1. CFS Architecture: User application file system calls are routed to CFS via the FUSE kernel module, while metadata storage and merging strategies are governed by the MetaService component.

3 Design & Implementation

In this section, we propose CFS, a cross-domain file system that ensures robust data consistency, low-latency access, and advanced metadata management. CFS consists of CRDTs, Hybrid Tree (HT) optimization, `rest_rpc`, and enhanced RocksDB integration. Specifically, CRDTs ensures strong eventual consistency and fault-tolerant data synchronization across nodes, while Hybrid Tree optimization improves metadata query efficiency and reduces write amplification, particularly in metadata-heavy workloads. The `rest_rpc` framework provides efficient, low-latency communication for synchronization. Enhanced RocksDB integration optimizes metadata storage and retrieval, ensuring high performance with low overhead. To achieve the above implementations, there are several challenges, including mitigating high-latency issues in metadata operations across network partitions, ensuring robust and consistent metadata updates through advanced version control and conflict resolution, and optimizations for both metadata-heavy and data-intensive workloads.

3.1 Architecture of CFS

Figure 1 shows the architecture of CFS, which follows a modular design to provide a seamless interface for cross-domain data management while optimizing performance and consistency. Specifically, CFS integrates traditional file system capabilities with modern data synchronization through a combination of the FUSE kernel module, MetaService, and a user-level storage integration model, optimizing for metadata-intensive workloads and dispersed storage environments. Moreover, the CFS framework transcends basic cross-domain synchronization by incorporating several innovations, such as Range Tree, Hybrid Tree optimization, and data structure enhancements to minimize latencies, improve bandwidth utilization, and ensure optimal metadata retrieval.

The following subsections provide an in-depth description of each major module:

- **User Application Layer**: This layer is built atop the FUSE framework to provide a seamless interface for user operations, while ensuring efficient metadata synchronization.
- **MetaService Layer**: This is responsible for managing metadata, ensuring consistency, and resolving conflicts. It introduces the Hybrid Tree for efficient metadata querying and CRDT-based synchronization for fault-tolerant consistency across nodes.

- **Local File System Layer**: This manages local data storage and file I/O operations, with optimizations for cross-domain metadata handling.

3.2 User Application Layer

Traditional user application layers in distributed file systems face several challenges in cross-domain environments. Existing FUSE-based implementations suffer from high metadata operation latency due to frequent user-space and kernel-space transitions. Additionally, traditional synchronization mechanisms rely on fixed periodic updates, which lack flexibility and may lead to metadata inconsistencies under network partitions.

To address these challenges, CFS adopts a multi-layered architecture and tailored mechanisms: (i) For scalable and efficient metadata management, we employ a Hybrid Tree index, optimized for metadata range queries and rapid versioned updates; (ii) To balance consistency and availability, CRDT-based synchronization is adopted, enabling strong eventual consistency with low overhead; (iii) For high performance, a hybrid tree index and caching further accelerate metadata queries and reduce synchronization costs. Concretely, the user application layer in CFS is built atop the FUSE user-level file system infrastructure. It ensures compatibility with existing POSIX-based file operations, including commands such as `touch`, `mkdir`, `rmdir`, `rm`, `mv`, and others. These commands are extended for cross-domain scenarios with added consistency and synchronization guarantees.

3.2.1 Enhanced Metadata Synchronization via FUSE API. To address metadata consistency challenges in distributed, cross-domain environments, we have extended the traditional FUSE methods by introducing custom handlers. Modifications to the `sync`, `syncdir`, `Makenode`, `Makedir`, and `Rmnode` handlers classify metadata during operations, enabling efficient synchronization and ensuring cross-domain consistency.

Manual and Periodic Synchronization Strategies. In CFS, administrators can manually trigger synchronization through the `sync` and `syncdir` commands, ensuring that metadata changes are not lost during network partitions or node disconnections. Once communication is restored, these operations allow for immediate synchronization. Additionally, the system automatically performs periodic synchronization at configurable intervals (default: one minute), maintaining consistency across nodes without requiring user intervention.

Optimized FUSE Performance for Metadata Management. To mitigate performance overhead in traditional FUSE-based implementations, CFS introduces metadata caching and a hybrid consistency model. The caching mechanism reduces latency by storing recently accessed metadata, minimizing delays in read-intensive operations, especially for small files and directories. The hybrid consistency model enables CFS to adapt between Strong Eventual Consistency (SEC) and immediate synchronization, based on workload characteristics and user-defined policies. These optimizations significantly reduce latency and improve overall user experience.

Caching for Metadata Efficiency. The caching strategy within the User Application Layer accelerates frequent metadata-intensive operations by reducing the need to query the full metadata database in MetaService. This is particularly beneficial for repeated operations on small files, which contribute significantly to overhead in distributed systems.

3.2.2 User-Initiated Consistency Mechanisms. User-initiated consistency mechanisms offer greater flexibility in synchronization management. The user application layer provides APIs for administrators to define directory-specific policies. For example, critical directories, such as those storing financial records, can enforce strong consistency, while less critical ones can use SEC for improved performance.

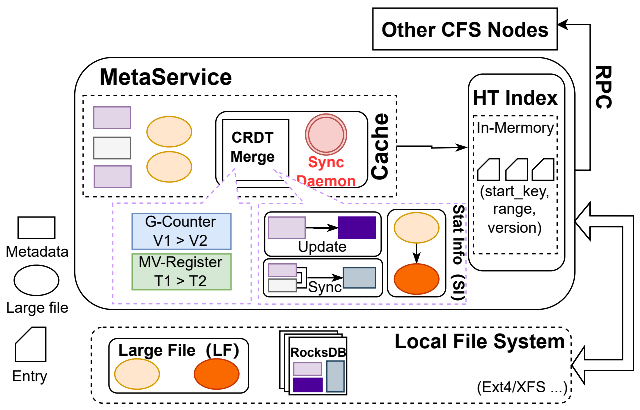


Fig. 2. The MetaService Architecture.

3.3 MetaService Layer

The MetaService layer is the core of CFS, responsible for managing metadata efficiently and ensuring consistency across distributed nodes. This layer integrates advanced indexing structures, conflict resolution mechanisms, and synchronization strategies. Figure 2 shows the MetaService architecture, including two components: (i) the Stat Info (SI) Module, which manages standard metadata operations including directory hierarchies and file attributes, and (ii) the Large File (LF) Module, optimized for large objects such as write-ahead logs and SSTables. The MetaService integrates the Hybrid Tree for efficient indexing, CRDT-based synchronization for conflict resolution, and enhanced caching mechanisms for accelerated metadata access.

3.3.1 Hybrid Tree (HT) Storage Model. HT is built on the Range Tree (RT) structure in xMeta [7], optimized from the indexing mechanism used in Ursa [26] block store to accommodate diverse storage workloads. While inspired by Ursa's composite-key concept, the HT has been specifically redesigned for distributed file system metadata. Its composite keys encode not only ranges but also semantic contexts such as namespace paths, entry types, and versions, which enable richer and more flexible queries. The design is tightly integrated with CRDT-based versioning to support concurrent and distributed metadata updates, and includes optimizations for scalable range queries under high-throughput, mixed workloads. Furthermore, unlike Ursa's fine-grained per-chunk mapping, HT operates across entire filesystem namespaces, supporting complex operations such as directory listing, attribute updates, and bulk traversal.

In the MetaService Layer, HT is designed to address the inefficiencies of traditional LSM-tree operations under write-intensive workloads and range queries. By leveraging composite keys that encapsulate contextual details such as start key and range, HT achieves efficient metadata queries and reduces write amplification. The workflow of HT is described in Algorithm 1.

Memory and Persistent Components. HT organizes metadata into two main storage components, similar to RocksDB, but introduces key optimizations tailored for metadata-intensive workloads:

- **Memory Component (MemTable)**: This stores recent updates in memory for fast insertions and lookups. Unlike RocksDB, HT prioritizes metadata-aware optimizations by using composite keys (<`start_key, range, version`>) to facilitate efficient range queries and minimize lookup overhead.
- **Persistent Component (SSTable)**: This represents the flushed data from MemTable to disk in a sorted structure. HT differs from RocksDB by employing a modified compaction strategy, which selectively merges overlapping key ranges to minimize write amplification.

Algorithm 1: Hybrid Tree Metadata Handling

```
Input: Metadata operation op (query or update)
Output: Query result or updated metadata
1  if op is update then
2      Parse to key-value $\langle k, v \rangle$;
3      Generate composite key $K = \langle \text{prefix}(k), \text{range}(k), \text{ver}(k) \rangle$;
4      Insert $(K, v)$ into MemTable;
5      if MemTable size > threshold then
6          Sort MemTable entries and flush as new SSTable;
7          Trigger compaction for overlapping SSTables;
8  else if op is query then
9      Search MemTable for key/range;
10     if not found then
11         Search corresponding SSTables;
12     return result
```

Comparison with Traditional LSM-trees. Unlike traditional LSM-trees that employ flat key-value mappings and prioritize write optimization for general storage, the HT in CFS is specifically designed for distributed file system metadata. The main differences include its composite key design, which encodes namespace path, range, and version to enable efficient range queries and multi-version concurrency management; fine-grained versioning with version/timestamp fields that natively support CRDT-based merge semantics, which is not available in vanilla LSM-tree designs; adaptive compaction strategies that minimize write amplification and prioritize hot subtrees according to file system access patterns; and deep integration with filesystem semantics such as directory listing, bulk traversal, and multi-attribute queries, moving beyond LSM-tree's generic key-value focus. In conclusion, these enhancements enable HT to efficiently support scalable, concurrent metadata management and fast query processing in cross-domain file systems.

3.3.2 CRDT-Based Synchronization. To ensure metadata consistency across distributed nodes, CFS employs CRDTs, which achieve strong eventual consistency (SEC) without relying on centralized locking. As detailed in Table 1, different CRDT types are strategically mapped to distinct metadata semantics. State-based set CRDTs handle directory structures by enabling idempotent merging of concurrent additions and removals of files or subdirectories. For file attributes such as permissions and ownership, operation-based Multi-Value Registers (MV-Register) support concurrent updates and resolve conflicts via version numbers or latest timestamps. Access counters (e.g., open/read/download counts) are managed using G-Counter CRDTs, which merge distributed counts without loss. Access control policies are also implemented with MV-Registers, applying a "maximum privilege" rule during merge to maintain secure and consistent permissions under concurrent modifications. To balance performance with consistency, CFS applies state-based synchronization for infrequently updated metadata such as directory trees, while operation-based synchronization is reserved for high-frequency or high-churn metadata like file rename operations.

The core synchronization protocol utilizes two CRDT types: the G-Counter, which tracks metadata update counts to prevent loss of updates, and the MV-Register, which maintains version information for conflict resolution by favoring the highest version or latest timestamp.

Table 1. CRDT type mapping for CFS metadata

Metadata Type	CRDT Type	Mathematical Model
Directory structure	State-based Set	$(S, \sqcup)$
File attributes	MV-Register	$\{op_1, op_2, ...\}$
Access counters	G-Counter	$\sum_{i=1}^{n} c_i$
Permissions	MV-Register	$\max(v_1, v_2)$

Algorithm 2: CRDT-based Metadata Synchronization

```
Input: Local updates buffer $B$, interval $T$, node set $C$
Output: Converged metadata across $C$
while system running do
    Every interval $T$:
        Package local updates as $P$;
        Broadcast $P$ to all nodes in $C$; receive $\{P_i\}$;
        foreach metadata entry k with conflicts do
            Select value by: highest version → latest timestamp → CRDT merge;
            Apply resolved entry locally;
        Clear $B$;
```

Algorithm 2 shows the synchronization workflow, which is implemented using the rest_rpc framework. These mechanisms ensure reliable conflict resolution and efficient propagation of metadata changes, even under network partition scenarios.

3.3.3 Automated Synchronization via rest_rpc. The MetaService leverages the rest_rpc framework to automate metadata synchronization between nodes. Synchronization is bidirectional, allowing both local and remote metadata updates to be exchanged efficiently. The synchronization interval (T) is user-configurable, balancing consistency requirements and performance overhead. This interval is set to one minute by default.

3.3.4 Journaling and Caching for Metadata Operations. To enhance reliability and performance, the MetaService Layer incorporates journaling for logging metadata operations to facilitate rapid system recovery, alongside adaptive caching and prefetching mechanisms that analyze access patterns and leverage locality characteristics to reduce query latency and optimize resource utilization. Prefetching dynamically adjusts metadata retrieval based on locality characteristics, ensuring seamless performance under high workloads. These features collectively enable the MetaService to handle metadata-intensive operations with high throughput and low latency, meeting the demanding requirements of modern distributed environments.

3.3.5 Integration of Metadata Synchronization and Consistency Mechanisms. To optimize data operations across different geographical and administrative domains, CFS integrates a metadata synchronization and consistency architecture that combines CRDT-based metadata merging, Hybrid Tree indexing, and the `rest_rpc` framework for lightweight synchronization. MetaService ensures metadata consistency by dynamically adjusting synchronization intervals based on workload conditions. By leveraging adaptive conflict resolution and efficient metadata indexing, CFS ensures that metadata queries and updates remain efficient even in highly distributed environments.

The combination of these techniques allows CFS to handle diverse operations efficiently, from small random writes to large sequential reads, ensuring system scalability, reliability, and high throughput in geographically distributed environments. Additionally, the `rest_rpc` framework enables efficient metadata synchronization between multiple file system nodes, minimizing communication overhead and ensuring strong eventual consistency across domains.

While CFS supports full-replication as a fallback mechanism to guarantee consistency and availability in highly collaborative environments, it primarily adopts a demand-driven strategy. Both data and metadata are only replicated to nodes that require shared access. Furthermore, metadata synchronization leverages CRDT-based mechanisms to support incremental state propagation: only modified or newly created metadata is synchronized among participating nodes, significantly reducing bandwidth and storage overhead in typical usage scenarios. This allows CFS to maintain efficient coordination while mitigating scalability bottlenecks associated with naive full-replication.

3.3.6 Metadata Operation Example. To illustrate the concrete implementation of CFS, we describe the end-to-end workflow for a typical file creation (`touch`) operation and a range metadata query. File Creation (`touch`) Workflow:

Step 1: User Layer Invocation. The application issues a `touch` request. The User Layer (built atop FUSE) checks its local cache for the target path and forwards the request to the MetaService if the file is absent.

Step 2: Metadata Insertion. The MetaService translates the request into a composite key (namespace path, entry type, version) and creates a new metadata entry in the HT.

Step 3: CRDT State Update. File attributes and counters are encapsulated as MV-Register and G-Counter CRDTs, respectively, and prepared for synchronization.

Step 4: Synchronization. The update buffer is synchronized with remote nodes using operation-based CRDT propagation. Concurrent creations are merged following CRDT semantics.

Step 5: Persistence. Metadata is batched and flushed to RocksDB. The HT key structure enables efficient storage and minimizes write amplification by compacting only affected ranges.

Range Metadata Query (e.g., Directory Listing):

Step 1: Request Initiation. A directory listing request triggers a range query over the namespace.

Step 2: Hybrid Tree Search. The Hybrid Tree's composite key enables efficient prefix or range searches, retrieving all relevant entries without unnecessary scans.

Step 3: Aggregation and Return. Results are returned to the user layer, with hot results optionally cached.

These concrete operation flows demonstrate how CFS leverages its architectural features, particularly the Hybrid Tree index and CRDT-based synchronization, to enable efficient, conflict-free, and scalable metadata management in cross-domain environments.

4 Evaluation and Discussion

4.1 Experimental Setup

To evaluate the performance of CFS, we deploy a 6-node cluster simulating a real-world distributed and cross-domain environment. Each node is equipped with dual Intel Xeon Gold 5218R CPUs, 128GB RAM, 2TB NVMe SSD, and 40GbE network interface with RoCEv2 support for low-latency communication. All nodes run Ubuntu 22.04 LTS to ensure environmental consistency across the cluster. CephFS (v15.2.9) and GlusterFS (v9.3.2) are deployed on the same hardware nodes, using the default recommended settings. For each file system (including CFS), client threads are evenly distributed across all nodes, and there is no overlap in working sets unless otherwise specified. This ensures all systems are evaluated under identical conditions for direct performance comparison. For metadata synchronization, CFS adopts rest_rpc, with RocksDB as the metadata persistence backend. Benchmarking tools include FIO [3], MDTest [4], and YCSB [9], with Prometheus and Grafana used for monitoring. Each experiment was repeated ten times and the average value was taken as the final result to avoid experimental errors. The experiments aim to evaluate not only traditional metrics like IOPS, bandwidth, and latency, but also more nuanced metrics like consistency overhead, metadata throughput, and system response under failure conditions.

To evaluate the effectiveness of CFS, we compare it against two widely used distributed file systems: CephFS and GlusterFS. All three systems run on the same cluster nodes with identical software and workload configurations. CephFS is known for its high availability but higher latency under heavy synchronization. GlusterFS exhibits lower synchronization overhead but offers weaker consistency during network partitions. These provide robust baselines for assessing the performance of CFS under varying synchronization conditions.

4.2 Metadata Management Performance Evaluation

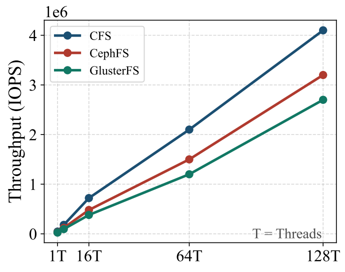
Efficient metadata management is crucial for distributed file systems, as metadata-intensive workloads often dominate system performance in large-scale environments. To evaluate the metadata handling capabilities of CFS, we conduct extensive tests using YCSB with customized workloads. YCSB workloads are mapped to file system operations as follows: each YCSB key corresponds to a file path within a hierarchical directory tree, and each operation (read, update, insert, delete) is translated into a corresponding file system operation (e.g., file open/read, attribute modification, file creation/deletion). CFS is compared with CephFS and GlusterFS, focusing on throughput, latency, and metadata transaction rates under various workloads and concurrency levels. All systems use Ext4 as the underlying local file system to ensure consistent test conditions.

Workload Configuration: We collect throughput (IOPS), tail latency (99th percentile), and transactions per second (TPS). IOPS refers to the number of data operations (read/write) completed per second, while TPS specifically denotes the number of file system metadata transactions (e.g., create, delete, rename) processed per second. The experimental configuration is as follows.

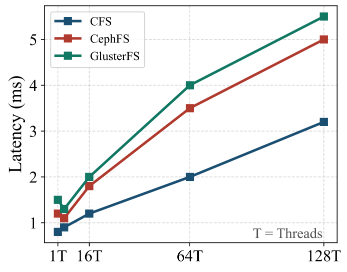
- **Workloads:** The YCSB tool is used to generate three types of workloads:
  - **Read-Heavy:** 80% reads, 20% updates.
  - **Write-Heavy:** 50% reads, 50% updates.
  - **Mixed:** 50% reads, 30% updates, and 20% inserts.
- **Concurrency Levels:** The tests are conducted with thread counts of 1, 4, 16, 64, and 128 to simulate varying levels of parallelism.
- **Dataset:** 10 million key-value pairs, each 1KB in size.

4.2.1 Read-Heavy Workload Results. Figure 3 shows the performance comparison for read-heavy workloads, where 80% of operations are metadata reads and 20% are updates. From the results, CFS achieves significant performance benefits over CephFS and GlusterFS in all evaluated metrics.

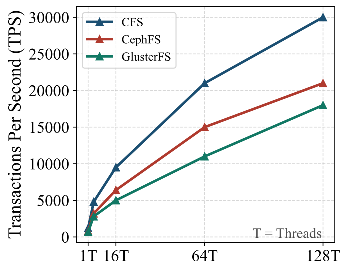
Specifically, CFS achieves throughput improvement of up to 33.9% over CephFS and 57.9% over GlusterFS when running 128 threads, as shown in Figure 3(a). This performance gain is largely attributable to the Hybrid Tree's optimized range query mechanism, which minimizes redundant metadata lookups. The latency at 99th percentile reflects stable performance, which is reduced by up to 36% for CFS, as shown in Figure 3(b). This is achieved through intelligent caching and prefetching strategies that effectively reduce disk I/O operations and mitigate network delays. Moreover, CFS can achieve 42% higher TPS when running 128 threads, providing high concurrency capability, as shown in Figure 3(c).



(a) Throughput (IOPS).

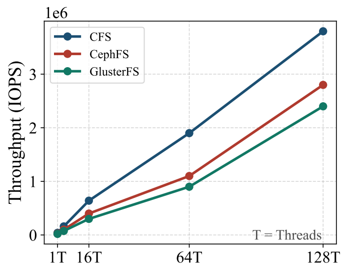


(b) Latency (99th Percentile).

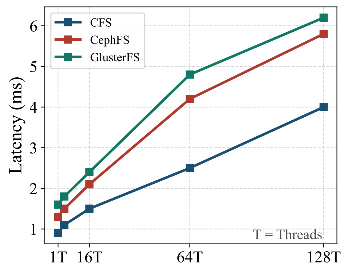


(c) TPS.

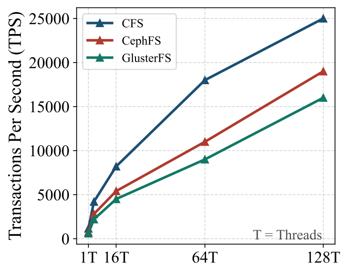
Fig. 3. Performance comparison for read-heavy workloads across different thread counts.



(a) Throughput (IOPS).



(b) Latency (99th Percentile).



(c) TPS.

Fig. 4. Performance comparison for write-heavy workloads across different thread counts.

4.2.2 *Write-Heavy Workload Results.*

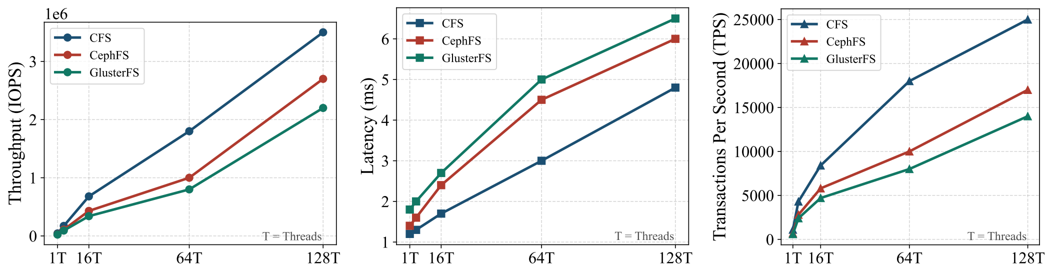
To evaluate the impact of frequent writes, we further test the write-heavy workloads that consist of 50% metadata updates and 50% reads. Figure 4 shows the performance comparison by varying the number of threads.

From the results, the throughput of CFS consistently outperforms CephFS across all concurrency levels, with the advantage most prominent under high concurrency. At 128 threads, CFS achieves approximately 35.7% higher throughput than CephFS. This demonstrates the superior scalability of CFS enabled by its optimized compaction strategy and reduced write amplification. In terms of latency, CFS reduces 99th percentile latency by up to 31.03% compared to CephFS at peak concurrency (128T), benefiting from efficient metadata batching and lower I/O contention. The TPS of CFS also increases by 30.97% over CephFS at peak concurrency. Therefore, the design of CFS achieves high scalability and low real-time responsiveness.

4.2.3 *Mixed Workload Results.*

In real-world scenarios, the workload often includes reads, updates, and inserts. To better evaluate the performance of CFS, we construct the mixed workload, consisting of 50% reads, 30% updates, and 20% inserts, as shown in Figure 5.

Throughput for CFS surpasses CephFS by 34.5% at 128 threads, driven by the Hybrid Tree's adaptive indexing, which optimizes metadata retrieval for varying access patterns. Latency improvements are equally significant, with CFS reducing 99th percentile latencies by 20%-26.2% compared to both CephFS and GlusterFS under high concurrency. TPS results demonstrate a 46.3% improvement over CephFS, further highlighting CFS 's ability to balance diverse workloads efficiently. The combination of adaptive metadata prefetching and intelligent caching ensures consistent performance across varying thread counts, making CFS suitable for complex operational environments.



(a) Throughput (IOPS). (b) Latency (99th Percentile). (c) TPS.

Fig. 5. Performance comparison for mixed workloads across different thread counts.

4.2.4 Discussions. The evaluation confirms that CFS consistently outperforms CephFS and GlusterFS across various workloads, particularly under high-concurrency conditions. These improvements are largely due to the efficient HT structure, which enhances metadata query efficiency. HT achieves significantly lower query latency, particularly for range queries, by optimizing metadata retrieval through its composite key structure. This allows CFS to handle sequential and random queries with fewer disk accesses compared to LSM-trees, resulting in reduced query latency. Additionally, HT's dynamic compaction strategy minimizes write amplification, achieving up to 30% lower write amplification than traditional LSM-trees. While both CephFS and GlusterFS also employ compaction, HT's selective merging of overlapping key ranges during compaction significantly reduces redundant writes, which is especially beneficial under mixed workloads with frequent updates. Throughput results show that CFS maintains high performance across different thread counts, demonstrating its scalability and efficiency.

The combination of efficient metadata indexing, reduced write amplification, and high throughput makes CFS a robust solution for metadata-intensive workloads in distributed file systems, addressing both performance and scalability challenges.

4.3 Hybrid Tree and Range Tree Performance Evaluation

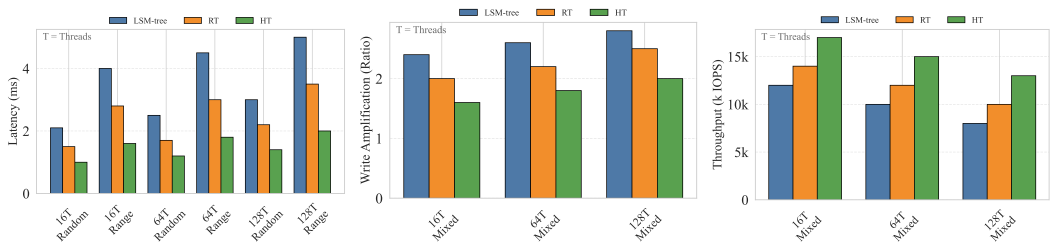
The Hybrid Tree (HT) in CFS is a key innovation aimed at optimizing metadata indexing and query processing. By extending the capabilities of traditional LSM-trees and addressing the limitations of Range Tree (RT) structures, HT provides enhanced performance for diverse workloads. This section evaluates the effectiveness of HT compared to RT and the standard LSM-tree used in RocksDB.

4.3.1 Experimental Methodology. To evaluate the performance of HT, we conduct comprehensive benchmarking using metadata-intensive workloads. Each workload is tested under thread counts of 16, 64, and 128 to reflect varying concurrency levels. The evaluation compared HT against the traditional LSM-tree and RT across the following metrics:

- **Query Latency:** Time taken to retrieve metadata for sequential and random queries.
- **Write Amplification:** Ratio of total disk writes to logical writes, reflecting indexing and compaction efficiency.
- **Throughput (IOPS):** Number of indexing operations (writes and queries) performed per second.

The experiments use the YCSB benchmark suite, simulating various workloads:

- `Random Queries:` Simulating directory lookups in distributed file systems.
- `Range Queries:` Modeling metadata scans for tasks such as backup or recovery operations.
- `Mixed Workloads:` Combining queries and updates to emulate real-world scenarios.



(a) Query Latency (ms). (b) Write Amplification (Ratio). (c) Throughput (IOPS).

Fig. 6. Performance comparison of Hybrid Tree (HT), Range Tree (RT), and LSM-tree across different workloads and thread counts.

4.3.2 Results and Analysis. Figure 6 presents the performance metrics for HT, RT, and LSM-tree across the evaluated workloads.

Query Latency. As shown in Figure 6(a), HT significantly reduces query latency compared to RT and LSM-tree. For range queries, HT achieves up to 2.5x lower latency than LSM-tree. This improvement comes from HT's composite key structure, which allows for more efficient metadata retrieval with fewer disk accesses. The optimized range indexing mechanism also improves random query latency by reducing the number of required key lookups and leveraging in-memory caching.

Write Amplification. Figure 6(b) shows the write amplification comparison. The dynamic compaction strategy of HT minimizes redundant writes caused by overlapping key ranges, achieving up to 30.7% lower write amplification compared to LSM-tree. While RT also reduces write amplification relative to LSM-tree, its static range design results in higher overhead during mixed workloads with frequent updates.

Throughput (IOPS). As shown in Figure 6(c), HT outperforms RT and LSM-tree in throughput across all thread counts, demonstrating its ability to scale efficiently with increasing concurrency. At 128 threads, HT sustains 13,000 IOPS, compared to 10,000 IOPS for RT and 8,000 IOPS for LSM-tree. This improvement highlights HT's scalability, as it consistently handles increasing concurrency without significant performance degradation. The higher throughput is attributed to HT's efficient indexing and compaction strategies, which maintain performance even under heavy workloads.

4.3.3 Conclusion. The evaluation confirms that Hybrid Tree addresses key limitations of earlier indexing structures like RT and LSM-tree, providing superior performance for CrossFS's metadata-intensive use cases. Key improvements include:

- **Efficient Metadata Retrieval:** HT's composite key structure reduces query latency by optimizing range and random lookups.
- **Reduced Resource Overhead:** Dynamic compaction minimizes write amplification, enabling efficient storage utilization.
- **Scalability:** HT maintains high throughput across varying concurrency levels, outperforming RT and LSM-tree in mixed workloads.

These results demonstrate the suitability of HT for modern distributed file systems, addressing both performance and scalability challenges in metadata management.

4.4 Cross-Domain Consistency Strategies Impact Analysis

Cross-domain consistency is critical for CFS, enabling reliable operations across geographically distributed environments. To evaluate the impact of CFS's consistency strategies, we conducted comprehensive experiments under varying network conditions, employing manual, periodic, and hybrid synchronization approaches. The evaluation aims to identify trade-offs between latency, data availability, and synchronization overhead.

4.4.1 Experimental Methodology. The experiments are designed to simulate real-world scenarios, including normal network conditions, network partitions, and high-latency environments. Key evaluation metrics included:

- **Consistency Latency:** Time required to synchronize metadata across all cluster nodes.
- **Data Availability:** Percentage of time during which data remained consistent and accessible under network disruptions. Data availability is measured as the proportion of benchmark time during which requested files remain accessible and consistent despite simulated network partitions or server failures. This is quantified by periodically polling for successful access and verifying consistency at each time interval, and reporting the fraction of time all replicas can access up-to-date data.
- **Synchronization Overhead:** Additional CPU and memory usage caused by synchronization activities.

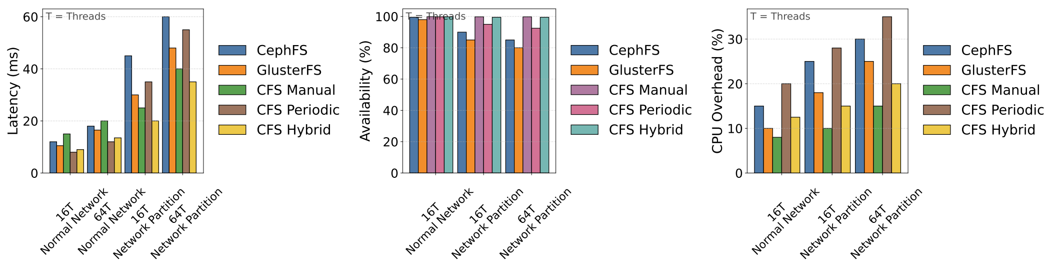
Since the experimental setup, workload configurations, and comparison baselines are already explained in Section 4.1, we will focus here on the specific results and analysis of cross-domain consistency strategies.

4.4.2 Results and Analysis. Figure 7 summarizes the results across all evaluated workloads and network scenarios.

Consistency Latency. As shown in Figure 7(a), periodic synchronization provided lower latency during normal network conditions, as it continuously propagated updates. However, manual synchronization results in higher latency because updates are only propagated when triggered by user actions, leading to delays in metadata synchronization. The hybrid approach achieved the best balance by dynamically adjusting synchronization intervals. This allows CFS to maintain a lower latency under normal network conditions while still ensuring consistency in network partitions. During network partitions, the hybrid approach reduces latency by 42%-27.1% compared to CephFS and GlusterFS by optimizing the frequency of synchronization operations and resolving conflicts more efficiently.

Data Availability. Figure 7(b) illustrates data availability. Manual synchronization maintains high availability (99.5%) during network partitions, as updates were carefully controlled. Periodic synchronization shows a slight decrease in availability during high-latency scenarios due to delayed conflict resolution. The hybrid approach achieves 99.5% availability across all scenarios, due to its combination of timely synchronization during low-latency periods and adaptive conflict resolution under network partitions. By dynamically adjusting the synchronization intervals and resolving conflicts efficiently, the hybrid strategy minimizes data unavailability, even under fluctuating network conditions. This adaptability ensures that CFS maintains high data availability while balancing the trade-offs between consistency and performance, even in challenging environments.

Synchronization Overhead. Synchronization overhead quantifies the additional system resources (CPU, memory, and network bandwidth) consumed by background metadata synchronization processes, beyond the baseline workload. This is evaluated by comparing system resource usage



(a) Consistency Latency (ms). (b) Data Availability (%). (c) Synchronization Overhead.

Fig. 7. Performance comparison of cross-domain consistency strategies across different network scenarios.

during periods of synchronization to that during idle or steady-state operation. Figure 7(c) reports the relative increase in resource consumption, highlighting the cost of frequent metadata exchange and merge operations. While manual synchronization can reduce the overhead by only updating when explicitly triggered, this comes at the cost of real-time consistency, leading to delayed updates. In contrast, the hybrid approach optimizes the trade-off by adjusting synchronization intervals based on system load and network conditions. This dynamic adjustment ensures that the system maintains a low overhead while still ensuring consistency, resulting in a 16.7%-33.4% reduction in synchronization overhead compared to CephFS under high concurrency workloads. By efficiently balancing synchronization frequency and resource usage, the hybrid approach provides better performance, especially when the system experiences varying workloads.

The evaluation demonstrates that CFS 's hybrid synchronization strategy offers significant advantages, including superior consistency latency in challenging network conditions, outperforming CephFS and GlusterFS by up to 42%. High data availability (99.5%) across diverse scenarios, ensuring reliable cross-domain operations while maintaining minimal synchronization overhead, achieved through optimized intervals and adaptive conflict resolution mechanisms. These findings validate the effectiveness of CFS in maintaining cross-domain consistency, making it a robust solution for distributed file systems in modern environments.

4.5 Distributed File System Performance

In this section, we use FIO to evaluate CFS in comparison to CephFS and GlusterFS under realistic cross-domain workloads. Each FIO run uses 16, 64, and 128 threads, 4 KB and 1 MB block sizes, and a total data set of 200 GB per node. This range of workloads stresses both small-file and large-block I/O paths.

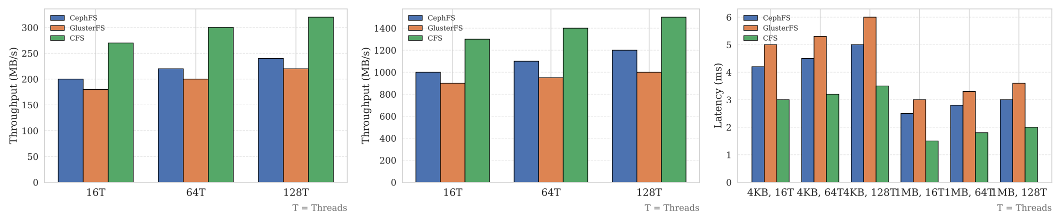
4.5.1 Sequential I/O Performance. Figure 8 shows the sequential I/O performance of CFS, CephFS, and GlusterFS across small (4KB) and large (1MB) block sizes.

Throughput: For 4KB sequential I/O, CFS achieves a 34.8% higher throughput compared to CephFS and 50.3% compared to GlusterFS. For large block sizes, the throughput improvement reaches up to 30%, attributed to efficient data-path handling and caching strategies.

Latency: The optimized Hybrid Tree structure and metadata prefetching allow CFS to maintain significantly lower latency (28.6%-40%) compared to CephFS and GlusterFS.

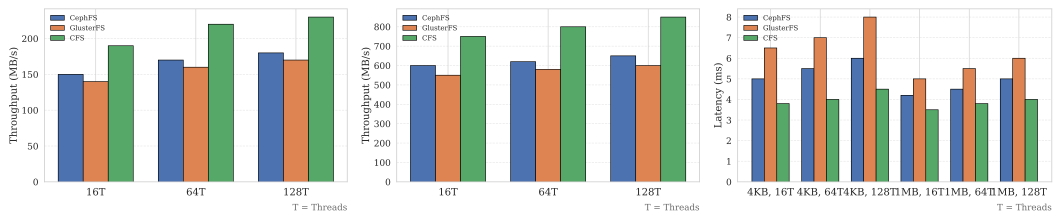
4.5.2 Random I/O Performance. Figure 9 shows the random I/O results.

Throughput: CFS achieves a 26.7%-35.7% improvement over CephFS and GlusterFS across block sizes, with 4KB workloads showing the highest gains due to efficient in-memory operations.



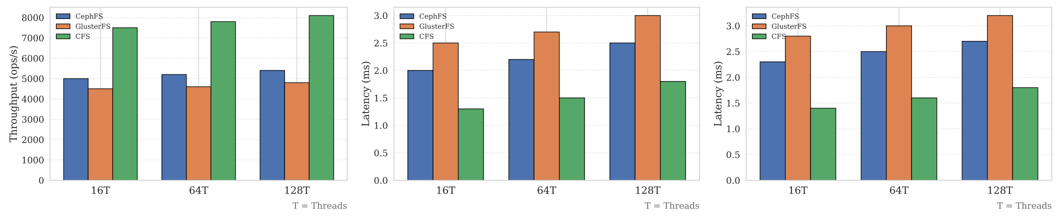
(a) 4KB Throughput. (b) 1MB Throughput. (c) Latency.

Fig. 8. Sequential I/O performance comparison.



(a) 4KB Throughput. (b) 1MB Throughput. (c) Latency.

Fig. 9. Random I/O performance comparison.



(a) File Creation. (b) Directory Scan Latency. (c) Overall Metadata Latency.

Fig. 10. Metadata operation performance comparison.

Latency: The optimized I/O path and Hybrid Tree indexing reduce read/write latency by up to 25%-43.8% compared to CephFS and GlusterFS under 128 threads, maintaining superior performance under high-concurrency random workloads.

4.5.3 Metadata Performance. Metadata operations, including file creation, deletion, and directory scans, are evaluated using MDTest [4] as shown in Figure 10. MDTest is executed concurrently on every node in the cluster with 16, 64, and 128 threads. For each test, we measure metadata performance with two file sizes, 4 KB and 1 MB, corresponding to small-file and large-file workloads. The test hierarchy comprises directories with a depth of 5, each containing 1,000 files and 10 subdirectories, for a total of 100,000 files. All experiments use POSIX semantics with synchronous operations to guarantee consistency. Across all thread counts, CFS achieves a 46.3%-65.4% higher file creation throughput and reduces directory scan latency and overall metadata latency by 48%-64.4% compared to CephFS and GlusterFS.

CFS achieves disproportionately higher throughput improvements for large I/O operations due to two factors. First, its hybrid tree index reduces lookup latency and metadata amplification, especially for large files where metadata access is localized and easily cached. Metadata queries are served from cache or fast index structures, and this synergy explains why metadata caching—although typically associated with small file access—can also significantly accelerate large I/O by shortening transaction setup and commit time.

4.6 Metadata Synchronization and Conflict Resolution

In this section, we evaluate the performance of the CRDT-based synchronization mechanism and analyze the efficiency of metadata conflict resolution in CFS. The focus is on synchronization throughput, conflict resolution latency, and consistency convergence time under concurrent workloads.

4.6.1 Evaluation of CRDT-Based Synchronization. We designed metadata-intensive workloads simulating frequent concurrent updates across distributed nodes. Each node generated conflicting metadata updates that required synchronization and conflict resolution. The following metrics were measured:

- **Synchronization Throughput (TPS)**: Number of metadata updates synchronized per second across nodes.
- **Conflict Resolution Latency**: Conflict resolution latency is defined as the elapsed time between the arrival (or detection) of concurrent conflicting metadata updates and the completion of their merging into a single consistent state. In our evaluation, we trigger conflicting updates to the same file or directory attribute from multiple nodes, then measure the time from update submission until the system successfully reconciles the conflict and all replicas reflect the resolved state. This includes network propagation delay, conflict detection, and CRDT-based merge execution.
- **Consistency Convergence Time**: Consistency convergence time is measured as the interval from the initial submission of a metadata or data update to the point when all nodes in the system have fully synchronized and reflect the updated state. This metric captures both synchronization delay and any protocol-imposed lag (e.g., periodic anti-entropy rounds). In our experiments, convergence time is determined by issuing an update at one node and repeatedly polling all other nodes until they all return the latest value.

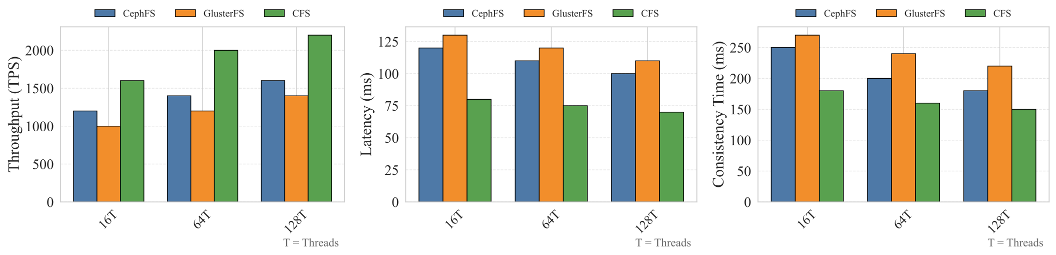
We compared CFS against CephFS and GlusterFS. Figure 11 presents the results, which are summarized below.

Synchronization Throughput: CFS achieves up to 2,200 TPS under 128 threads, outperforming CephFS (1,600 TPS) and GlusterFS (1,400 TPS). The lightweight CRDT-based synchronization eliminates the need for complex locking, improving throughput.

Conflict Resolution Latency: CFS demonstrates a 31.8% lower conflict resolution latency compared to CephFS and 37.5% lower than GlusterFS under 64 threads. The use of versioned metadata and CRDT merging semantics ensures rapid resolution with minimal overhead.

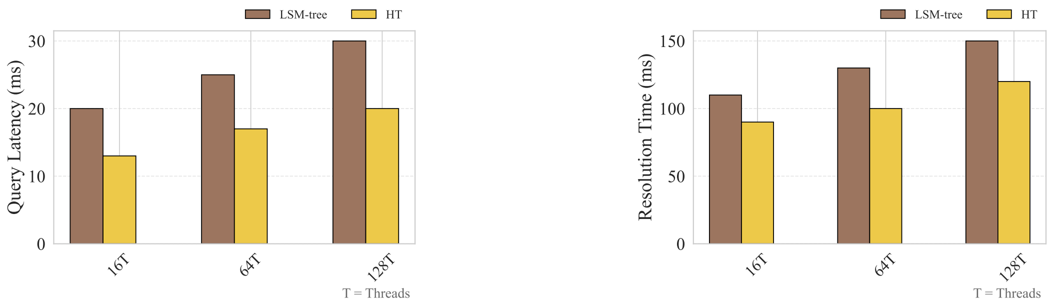
Consistency Convergence Time: CFS converges to a consistent state within 150 ms, significantly faster than CephFS (180 ms) and GlusterFS (220 ms).

4.6.2 Conflict Resolution Efficiency with Hybrid Tree. We further evaluated the efficiency of the HT during conflict resolution by analyzing concurrent metadata updates (inserts, deletes, renames) under high load, as shown in Figure 12. As the indexing and retrieval mechanisms directly impact conflict resolution, we focused on query latency and resolution time. The LSM-tree baseline refers



(a) Metadata Sync Throughput. (b) Conflict Resolution Latency. (c) Consistency Convergence Time.

Fig. 11. Performance of CRDT-based metadata synchronization and conflict resolution in CFS, compared with CephFS and GlusterFS.



(a) Metadata Query Latency. (b) Conflict Resolution Time.

Fig. 12. Impact between HT and LSM-tree based CFS on metadata conflict resolution efficiency.

to a CFS variant where all metadata indexing is performed using a standard LSM-tree structure, without HT's composite key or adaptive optimizations.

- **Query Latency**: HT reduces query latency by 33.4% compared to traditional LSM-tree, particularly for range queries under 128 threads. This improvement comes from HT's range-based composite key design, which minimizes disk I/O during conflict resolution.
- **Conflict Resolution Time**: HT accelerates metadata conflict resolution by up to 20% compared to LSM-tree under 128 threads, as it enables efficient merging of metadata entries without unnecessary disk operations.

The results confirm that HT optimizations significantly enhance the efficiency of metadata conflict resolution, making CFS more suitable for metadata-intensive distributed workloads.

4.6.3 Discussion. The experimental findings demonstrate that CFS achieves significant performance gains in metadata synchronization and conflict resolution:

- CRDT-based synchronization ensures high throughput and low latency while eliminating the need for locking mechanisms.
- Hybrid Tree enhances query and resolution efficiency, accelerating conflict resolution and improving system responsiveness.

These results highlight CFS's ability to handle metadata-intensive workloads in geographically distributed environments with superior consistency guarantees and minimal overhead.

5 Related Work

The field of distributed file systems has seen significant advancements over the years, particularly in the domains of local and distributed metadata management, consistency protocols, and conflict resolution strategies. This section reviews the current state-of-the-art in these areas and highlights the limitations addressed by CFS.

5.1 Distributed File Systems

Distributed file systems like HDFS, GFS, CephFS, and GlusterFS and Lustre [21] have been designed to support large-scale data storage across multiple nodes. They enhance scalability and availability but face persistent challenges in metadata management and synchronization.

HDFS and GFS: Systems like HDFS and GFS employ a centralized metadata server (NameNode or Master) to manage file system operations. While this architecture simplifies implementation, it also introduces a single point of failure and creates performance bottlenecks during high metadata operation loads [13, 38]. Such limitations are partially addressed in CephFS and GlusterFS by decentralizing metadata, but issues with latency and consistency persist.

CephFS, GlusterFS, and Lustre: These distributed file systems distribute both data and metadata across nodes to improve fault tolerance and scalability [5, 44]. These systems typically employ data replication or erasure coding for high availability and durability in the data plane. However, metadata consistency remains a challenge due to network partitions and synchronization overhead across distributed nodes. CFS builds upon these systems by using CRDTs for metadata synchronization, ensuring consistency without compromising availability. Furthermore, the hybrid storage model in CrossFS improves both metadata retrieval speed and system responsiveness. While considerable efforts have been made to advance DFS scalability and fault tolerance, efficient metadata management persists as a critical gap, especially in cross-domain settings [8]. CFS addresses this issue by implementing a decentralized metadata architecture coupled with CRDTs, enabling improved scalability, reduced latency, and more efficient synchronization across distributed nodes.

Data plane availability: Unlike CephFS and Lustre, which provide native data replication or erasure coding for fault tolerance, the current implementation of CFS relies on the underlying storage backend (local filesystem) for data durability. Integrating advanced data redundancy mechanisms is a promising direction for future work to achieve end-to-end availability.

5.2 Conflict Resolution in Distributed File Systems

Conflict resolution in distributed file systems is a critical area of study, particularly for ensuring data consistency across different nodes without sacrificing availability. Traditional methods such as two-phase commit (2PC), Paxos, and Raft introduce considerable complexity and latency, especially in distributed environments with high node counts [23, 29].

CRDT-Based Approaches: Recent work has explored the use of CRDTs to manage metadata conflicts in distributed settings [19, 37]. CRDTs allow distributed systems to achieve strong eventual consistency (SEC) without requiring coordination among nodes for every operation. Ahmed et al. [1] and Kleppmann et al. [19] have demonstrated the use of add-wins and move-wins CRDTs to automate conflict resolution for file operations.

Comparison to ElmerFS: ElmerFS is a geo-replicated file system that employs CRDTs for all core structures to ensure strong eventual consistency and high availability. ElmerFS focuses on correctness of conflict resolution (e.g., name conflicts, deletions, inode link tracking) and POSIX interface compliance in the face of extreme network partitioning and offline work. While ElmerFS provides intuitive conflict handling and legacy compatibility, it does not address scalable, high-performance metadata indexing or query optimization; its design sacrifices certain performance optimizations for clarity of convergence and correctness. In contrast, CrossFS leverages a Hybrid Tree structure for fast, scalable metadata access, integrates caching layers, and is evaluated under modern distributed workloads for throughput and latency. Nevertheless, CrossFS can benefit from ElmerFS's comprehensive conflict handling semantics in future extensions. Direct experimental comparison is limited by prototype maturity and codebase availability, but our analysis highlights the key differences and relative advantages. However, these studies are primarily limited to file-level operations, with less emphasis on the efficient handling of metadata conflicts or hierarchical data structures in cross-domain scenarios.

Different from the above works, CFS extends CRDTs to manage metadata at multiple levels, incorporating version control and hybrid conflict resolution to maintain data consistency across geographically distributed nodes. This approach significantly reduces user intervention and optimizes the merging process for complex directory structures.

5.3 Access Control Mechanisms in Distributed Systems

Access control is an essential aspect of distributed systems, ensuring that data is accessed and modified only by authorized users. Traditional distributed file systems often struggle to maintain adaptable access control policies, especially under high availability requirements.

Highly Available Access Control: Weber proposed highly available access control models for geo-distributed information systems using causal consistency and highly available transactions [43]. This work introduces a formal model that balances data modifications with access control constraints, relying on permission lattices to merge conflicting updates. CFS builds on this by incorporating metadata-level access control policies directly into its CRDT-based conflict resolution mechanism, thereby ensuring that access rights are preserved and conflicts are resolved in a way that aligns with the most restrictive permissions.

Move Operations and Access Control: Kleppmann et al. developed algorithms to handle move operations using CRDTs, treating operations like move, add, and delete as consistent transformations [20]. CFS adopts a similar approach but extends it by integrating these move operations into its hybrid storage structure, thereby enhancing both metadata conflict resolution and access control management in cross-domain environments.

5.4 Hybrid Storage and Metadata Management Approaches

Advancements in metadata management have also led to the adoption of hybrid storage solutions. Systems like RocksDB utilize log-structured merge trees (LSM-tree) to enhance write performance and efficiently manage high volumes of data.

LSM-tree based Metadata Management: RocksDB's LSM-tree architecture provides efficient mechanisms for storing and retrieving metadata, but suffers from write amplification, particularly in scenarios involving frequent range queries [11]. In CFS, the Hybrid Tree structure modifies the traditional LSM-tree by adding a two-level indexing mechanism optimized for both range and point queries. This hybrid indexing model improves metadata access speed and reduces the number of disk operations required for updates, thereby mitigating one of the major drawbacks of LSM-tree based systems.

Range and Hybrid Tree Mechanisms: Inspired by the Range Tree approach, which was initially designed for block storage scenarios, CFS introduces the Hybrid Tree for managing general-purpose metadata across distributed nodes [15]. Unlike Range Tree, Hybrid Tree is optimized for both sequential and random access patterns, ensuring that it effectively supports a wide range of file system operations. This innovation helps reduce latency and optimize resource utilization for distributed workloads.

Table 2. Comparison of Distributed File Systems

File System	Metadata Management	Consistency	Scalability	Conflict Resolution
HDFS	Centralized	Weak	High	Manual (2PC)
GFS	Centralized	Weak	Moderate	Manual (2PC)
CephFS	Decentralized	Strong	High	Manual/Semi-Automated
GlusterFS	Decentralized	Strong	Moderate	Manual
CrossFS	Hybrid	SEC (CRDTs)	High	Automated (CRDT-Based)

Prefetching and Caching Mechanisms: Prefetching and caching have also been areas of focus in recent distributed file systems, aiming to reduce access latencies for frequently accessed metadata. CFS employs adaptive metadata prefetching, which dynamically adjusts prefetching strategies based on observed access patterns, thereby reducing the time taken for metadata retrieval and enhancing overall system responsiveness. The intelligent caching mechanism implemented in CrossFS not only accelerates frequently occurring file operations but also ensures that metadata consistency is maintained across nodes, even during network disruptions.

5.5 Comparison and Summary

While existing distributed file systems and metadata management approaches have made significant strides in enhancing scalability, availability, and fault tolerance, there remain notable challenges in metadata synchronization, conflict resolution, and efficient access control across domains. Table 2 provides a comparison of major distributed file systems, highlighting their limitations in terms of metadata management, consistency, and scalability.

CFS distinguishes itself by integrating advanced metadata management techniques, CRDT-based conflict resolution, and a hybrid storage architecture that collectively address the limitations of traditional DFSs. By incorporating both manual and automated synchronization, adaptive caching, and an enhanced storage model, CFS provides a comprehensive solution for modern distributed data management needs, especially in cross-domain settings where both metadata consistency and system scalability are critical [22].

6 Conclusion

CFS is a cross-domain distributed file system that addresses the core challenges of metadata management, consistency, and scalability in geographically distributed environments. CFS includes three techniques: 1) conflict-free replicated data types for metadata consistency; 2) Hybrid Tree indexing structure for optimized query performance; and 3) adaptive caching and synchronization strategies. The design effectively addresses key distributed storage complexities, including small-file management, metadata consistency, and sustained throughput under diverse workload conditions. Evaluation results show that CFS outperforms existing distributed file systems like CephFS and GlusterFS. For metadata-intensive workloads, CFS reduces query latency by up to 33.4% and improves throughput by 33.9%. For data-intensive workloads, CFS achieves 26.7%-35.7% higher throughput and 25%-43.8% lower latency in random I/Os, demonstrating efficient handling of diverse block sizes. Furthermore, CFS combines hybrid synchronization with CRDT-based conflict resolution to ensure over 99.5% data availability in challenging network conditions with minimal overhead, thus demonstrating strong consistency and performance in cross-domain environments. In the future, CFS will focus on enhancing write performance by optimizing RocksDB's compaction mechanisms, implementing more dynamic metadata distribution strategies, and exploring application scenarios such as real-time analytics and machine learning pipelines. Additionally, efficient designs for metadata and data operations are an avenue for future exploration.

References

[1] Mehdi Ahmed-Nacer, Stéphane Martin, and Pascal Urso. 2012. File system on CRDT. *arXiv preprint arXiv:1207.5990* (2012).

[2] Yousef J Al-Houmaily and Panos K Chrysanthis. 2004. 1-2PC: the one-two phase atomic commit protocol. In *Proceedings of the 2004 ACM symposium on Applied computing*. 684–691.

[3] Jens Axboe. 2024. fio - Flexible I/O Tester. https://fio.readthedocs.io/. Accessed: 2024-12-29.

[4] MDTest benchmark. [n. d.]. https://docs.daos.io/v2.4/testing/ior/.

[5] Eric B Boyer, Matthew C Broomfield, and Terrell A Perrotti. 2012. *Glusterfs one storage server to rule them all.* Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[6] Hao Chen, Chaoyi Ruan, Cheng Li, Xiaosong Ma, and Yinlong Xu. 2021. {SpanDB}: A fast,{Cost-Effective}{LSM-tree} based {KV} store on hybrid storage. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. 17–32.

[7] Yan Chen, Qiwen Ke, Huiba Li, Yongwei Wu, and Yiming Zhang. 2024. xMeta: SSD-HDD-Hybrid Optimization for Metadata Maintenance of Cloud-scale Object Storage. *ACM Trans. Archit. Code Optim.* 21, 2, Article 40 (may 2024), 20 pages. https://doi.org/10.1145/3652606

[8] Yanling Chen, F. Ma, Y. Zhou, Z. Yan, Q. Liao, and Y. Jiang. 2025. Themis: Finding imbalance failures in distributed file systems via a load variance model. http://www.wingtecher.com/themes/WingTecherResearch/assets/papers/paper_from_25/themis_eurosys25.pdf

[9] Brian F. Cooper, Adam Silberstein, Erwin Tam, Ramakrishna Ramakrishnan, and Raghu Ramakrishnan. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC)*. ACM, 143–154. https://doi.org/10.1145/1807128.1807152

[10] Hao Dai, Yang Wang, Kenneth B Kent, Lingfang Zeng, and Chengzhong Xu. 2022. The state of the art of metadata managements in large-scale distributed file systems—scalability, performance and availability. *IEEE Transactions on Parallel and Distributed Systems* 33, 12 (2022), 3850–3869.

[11] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. Rocksdb: Evolution of development priorities in a key-value store serving large-scale applications. *ACM Transactions on Storage (TOS)* 17, 4 (2021), 1–32.

[12] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*. 29–43.

[13] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In *ACM SIGOPS Operating Systems Review*, Vol. 37. ACM, 29–43.

[14] Deepak Gupta and Rinkle Rani. 2019. A study of big data evolution and research challenges. *Journal of Information Science* 45, 3 (2019), 322–340. https://doi.org/10.1177/0165551518789880 arXiv:https://doi.org/10.1177/0165551518789880

[15] Yu Hua, Hong Jiang, Yifeng Zhu, Dan Feng, and Lei Tian. 2011. Semantic-aware metadata organization paradigm in next-generation file systems. *IEEE Transactions on Parallel and Distributed Systems* 23, 2 (2011), 337–344.

[16] Liu Jiang, Bing Li, and Meina Song. 2010. THE optimization of HDFS based on small files. In *2010 3Rd IEEE international conference on broadband network and multimedia technology (IC-BNMT)*. IEEE, 912–915.

[17] Karthik Kambatla, Giorgos Kollias, Vipin Kumar, and Ananth Grama. 2014. Trends in big data analytics. *Journal of parallel and distributed computing* 74, 7 (2014), 2561–2573.

[18] Hojun Kim et al. 2021. *Analyzing Causes of Metadata Service Overheads in Ceph File System.* Ph. D. Dissertation. DGIST.

[19] Martin Kleppmann and Alastair R Beresford. 2017. A conflict-free replicated JSON datatype. *ACM Transactions on Computer Systems (TOCS)* 35, 4 (2017), 1–34.

[20] Martin Kleppmann, Heidi Howard, and Alastair R Beresford. 2019. Byzantine Eventual Consistency and Authentication. *ACM Transactions on Computer Systems (TOCS)* 37, 1 (2019), 1–41.

[21] Petros Koutoupis. 2011. The lustre distributed filesystem. *Linux Journal* 2011, 210 (2011), 3.

[22] Pradeep Kumar and H Howie Huang. 2020. Graphone: A data store for real-time analytics on evolving graphs. *ACM Transactions on Storage (TOS)* 15, 4 (2020), 1–40.

[23] Leslie Lamport. 1998. The part-time parliament. *ACM Transactions on Computer Systems* 16, 2 (1998), 133–169.

[24] Leslie Lamport. 2006. Fast paxos. *Distributed Computing* 19 (2006), 79–103.

[25] Edward A Lee, Ravi Akella, Soroush Bateni, Shaokai Lin, Marten Lohstroh, and Christian Menard. 2023. Consistency vs. availability in distributed real-time systems. *arXiv preprint arXiv:2301.08906* (2023).

[26] Huiba Li, Yiming Zhang, Dongsheng Li, Zhiming Zhang, Shengyun Liu, Peng Huang, Zheng Qin, Kai Chen, and Yongqiang Xiong. 2019. Ursa: Hybrid block storage for cloud-scale virtual disks. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–17.

[27] Peter Macko and Jason Hennessey. 2022. Survey of distributed file system design choices. *ACM Transactions on Storage (TOS)* 18, 1 (2022), 1–34.

[28] Ovidiu-Cristian Marcu and Pascal Bouvry. 2024. *Big data stream processing.* Ph. D. Dissertation. University of Luxembourg.

[29] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm (Raft). In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 305–319.

[30] Diego Ongaro and John Ousterhout. 2015. The raft consensus algorithm. *Lecture Notes CS* 190 (2015), 2022.

[31] Akshita Parekh, Urvashi Karnani Gaur, and Vipul Garg. 2020. Analytical modelling of distributed file systems (GlusterFS and CephFS). In *Reliability, Safety and Hazard Assessment for Risk-Based Technologies: Proceedings of ICRESH 2019*. Springer, 213–222.

[32] easy to use rpc framework. QiCOSMOS Team. Modern C++(C++11), simple. [n. d.]. Retrieved from https://github.com/qicosmos/rest_rpc/. ([n. d.]).

[33] Pierre-Antoine Rault, Claudia-Lavinia Ignat, and Olivier Perrin. 2023. Access control based on CRDTs for collaborative distributed applications. In *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 1369–1376.

[34] The reference implementation of the Linux FUSE (Filesystem in Userspace) interface. [n. d.]. Retrieved from https://github.com/libfuse/libfuse. ([n. d.]).

[35] Ohad Rodeh, Josef Bacik, and Chris Mason. 2013. Btrfs: The Linux B-Tree Filesystem. In *ACM Transactions on Storage (TOS)*, Vol. 9. ACM, 9:1–9:32. https://doi.org/10.1145/2501620.2501623

[36] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-free replicated data types. In *Stabilization, Safety, and Security of Distributed Systems: 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings 13*. Springer, 386–400.

[37] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-free replicated data types. In *Proceedings of the 13th international conference on Principles and practice of distributed systems*. 386–400.

[38] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. IEEE, 1–10.

[39] Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto, and Geoff Peck. 1996. Scalability in the XFS File System.. In *USENIX Annual Technical Conference*, Vol. 15.

[40] Tran Doan Thanh, Subaji Mohan, Eunmi Choi, SangBum Kim, and Pilsung Kim. 2008. A taxonomy and survey on distributed file systems. In *2008 Fourth international conference on networked computing and advanced information management*, Vol. 1. IEEE, 144–149.

[41] Theodore Ts'o, Mingming Cao, Andreas Dilger, Alex Tomas, Dave Kleikamp, Eric Sandeen, and Sam Naghshineh. 2007. The New ext4 Filesystem: Current Status and Future Plans. In *Proceedings of the Linux Symposium*. Linux Foundation, 21–33. https://www.kernel.org/doc/ols/2007/ols2007v1-pages-21-33.pdf

[42] Romain Vaillant, Dimitrios Vasilas, Marc Shapiro, and Thuy Linh Nguyen. 2021. CRDTs for truly concurrent file systems. In *Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems*. 35–41.

[43] Samuel Weber, Amal Al-Akkad, and Jens Wegner. 2014. Causal consistency meets access control: Specifying and verifying policies for geo-distributed systems. In *Proceedings of the ACM Symposium on Cloud Computing*. 1–12.

[44] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. 2006. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*. 307–320.