Multidisciplinary Journal of Engineering, Technology and Sciences, Volume 1 Number 1, 2024
Online publication with Google Scholar indexing, Email: mjets85@gmail.com
Title: **A Comparative Analysis of EXT4 and NTFS: Performance, Security, and Reliability Features**
Authors: **Ojuawo Olutayo Oyewole & Jiboku Folahan Joseph**

A COMPARATIVE ANALYSIS OF EXT4 AND NTFS: PERFORMANCE, SECURITY, AND RELIABILITY FEATURES

[1] Ojuawo Olutayo Oyewole & [2] Jiboku Folahan Joseph
[1,2] Department of Computer Science, The Federal Polytechnic Ilaro, Nigeria,

Abstract
This paper presents a comprehensive comparative analysis of the EXT4 and NTFS file systems, focusing on their performance, security, and reliability features. EXT4 is a widely used file system for Linux-based systems, while NTFS is the default file system for Windows environments. Both file systems are designed to handle large storage capacities and provide robust solutions for different operating environments. The study contrasts the two in terms of read/write performance, journaling, data integrity, encryption, access control mechanisms, and their ability to recover from system crashes or power failures. This paper draws upon prior research and experimental analysis to highlight the strengths and limitations of both file systems under various use cases.
Keywords: *EXT4, NTFS, File systems, Performance, Security.*

1. Introduction

File systems are a fundamental component of any operating system, serving as the framework that organizes, stores, retrieves, and manages data on storage devices. Their role is particularly crucial in modern computing environments where data volumes continue to expand exponentially. From personal computing devices to large-scale enterprise systems, the efficiency, reliability, and security of the file system directly affect system performance and data integrity. File systems are integral to managing how data is written to disk, how files are organized, and how data can be accessed by users or applications. Moreover, with the proliferation of digital content, cloud computing, and big data, choosing the right file system has become an essential consideration for system administrators, data managers, and end-users alike (Silberschatz et al., 2018).

Two of the most widely used file systems today are EXT4 (Fourth Extended File System) and NTFS (New Technology File System). EXT4 is predominantly utilized in Linux-based operating systems, while NTFS is the default file system for Windows. Both file systems were designed to meet the demands of modern computing, including handling large volumes of data and providing mechanisms for maintaining data integrity and security. However, despite their shared goals, EXT4 and NTFS differ significantly in their architecture, optimization, and the specific use cases they serve.

EXT4 was introduced in 2008 as the successor to EXT3, which had been the standard file system for many Linux distributions. EXT4 brought several improvements, including increased support for large file systems, enhanced performance, and new features to reduce fragmentation and speed up file system checks. It is known for its robustness, speed, and backward compatibility with previous versions of the EXT family. EXT4 has become the default file system for many Linux distributions, from personal computing to enterprise servers, because of its scalability and strong performance (Mathur et al., 2007).

NTFS, developed by Microsoft and introduced with the Windows NT operating system in 1993, was designed as a modern file system to replace the older FAT (File Allocation Table) systems used in earlier versions of Windows. NTFS includes several advanced features such as journaling, file compression, encryption, and the ability to set detailed permissions through Access Control Lists (ACLs). These features make NTFS a versatile and powerful file system, especially in environments where data security, reliability, and management of large files are paramount (Sweeney, 1996). NTFS has become the de facto file system for Windows-based systems and remains widely used in both personal computing and enterprise environments.

Considering the growing demand for more reliable, secure, and efficient file systems, a comparative analysis of EXT4 and NTFS is essential for understanding the strengths and weaknesses of each file system. Performance, security, and reliability are the primary criteria by which file systems are evaluated, particularly in environments that require high uptime, data protection, and efficient data access. In terms of performance, factors like read/write speeds, handling of file fragmentation, and optimization for specific workloads play a crucial role. On the security front, features such as encryption, user authentication, and access control determine how well a file system can protect sensitive data. Reliability, often measured through features like journaling and recovery mechanisms, is key in ensuring data integrity, especially in the face of system crashes or unexpected shutdowns.

This paper explores these aspects in detail by comparing the performance, security, and reliability features of EXT4 and NTFS. While EXT4 is tailored for Linux environments and excels in providing a lightweight, efficient file system, NTFS is optimized for Windows and is known for its robust security and detailed permission settings. Through this comparative analysis, we aim to provide insights into which file system is better suited for various use cases and operational environments, whether for desktop use, enterprise-level servers, or specific workloads such as database management, cloud storage, or general computing.

2. EXT4 Overview

EXT4 (Fourth Extended File System) is the successor to the EXT3 file system, introduced in 2008 as part of the Linux kernel 2.6.28. Its development was driven by the need for a more scalable and efficient file system capable of handling the growing data requirements in both consumer and enterprise environments. While retaining backward compatibility with EXT3, EXT4 brings several new features and improvements that make it suitable for modern computing demands, particularly in the realm of high-capacity storage systems (Mathur et al., 2007).

2.1 Backward Compatibility and Transition from EXT3

One of the critical strengths of EXT4 is its backward compatibility with EXT3, allowing for a seamless transition for users and system administrators who wish to upgrade to EXT4 without losing access to existing data or needing to reformat their drives. The ability to mount an EXT3 file system as EXT4 and vice versa allows for flexibility in managing data across multiple systems. This backward compatibility also provides a risk-free option for those who want to benefit from some of the performance enhancements of EXT4 without committing to a complete migration.

The transition from EXT3 to EXT4 was not just an incremental update but a substantial improvement designed to address some of the performance limitations found in EXT3, particularly when dealing with large file sizes and large numbers of files. EXT4's ability to handle modern workloads efficiently makes it an appealing choice for systems that manage massive volumes of data, such as cloud storage servers, databases, and multimedia storage solutions (Ts'o, 2010).

2.2 Extents and Reduced Fragmentation

One of the most significant improvements in EXT4 is the introduction of *extents*, which replace the traditional block mapping used in EXT3. In EXT3, a file is mapped to disk blocks individually, which could lead to inefficient storage management and fragmentation, especially for large files. Extents in EXT4 are designed to group a range of contiguous blocks together, reducing fragmentation and improving file access times. Each extent can map up to 128 MB of contiguous space, significantly reducing the number of block pointers needed for large files (Mathur et al., 2007). This not only boosts performance but also helps prevent file fragmentation, a common issue in file systems handling large files or heavy workloads.

By reducing fragmentation, EXT4 ensures that data is stored more efficiently, leading to faster read and write operations. This is particularly beneficial in environments that deal with large files, such as multimedia editing or data analytics, where file system performance can have a direct impact on productivity.

2.3 Delayed Allocation and Improved Write Efficiency

Another notable feature of EXT4 is *delayed allocation*, which postpones the allocation of disk blocks until the data is written to disk. By delaying this process, EXT4 can optimize the placement of data on disk, reducing fragmentation and improving overall write efficiency. This feature works in conjunction with the extents mechanism to ensure that data is written in large, contiguous chunks, further enhancing performance (Ts'o, 2010).

Delayed allocation allows EXT4 to gather information about file size and disk availability before committing data to disk, which enables more intelligent and efficient data placement. As a result, the file system can reduce the need for subsequent readjustments or defragmentation, making EXT4 particularly suitable for workloads that involve frequent writing of temporary files or data that may change size frequently before being finalized, such as log files or data caches.

2.4 Scalability and Large File Support

EXT4 is designed with scalability in mind, making it well-suited for modern storage environments where file systems are expected to manage enormous amounts of data. EXT4 supports individual file sizes of up to 16 terabytes (TB) and a total file system size of up to 1 exabyte (EB), which is far beyond the capabilities of its predecessors and many other file systems (Mathur et al., 2007). This makes EXT4 a highly scalable solution for a wide range of applications, from personal computers with modest storage needs to enterprise servers handling vast datasets.

The large file size support is particularly important in today's era of high-definition media, virtual machine disk images, and big data analytics, where files can often exceed the terabyte range. EXT4's capacity to handle such large files without performance degradation or the need for specialized file systems ensures its relevance in high-demand environments. Moreover, as storage technology continues to evolve, EXT4 remains a future-proof solution due to its ability to manage vast amounts of data efficiently.

2.5 Fast File System Check (fsck)

File system checks (fsck) are essential for maintaining data integrity and recovering from system crashes or unexpected power failures. In older file systems, performing an fsck on large volumes could be a time-consuming process, often requiring hours or even days on large-scale storage systems. EXT4 addresses this issue with its *fast fsck* feature, which dramatically reduces the time required to perform file system checks (Ts'o, 2010).

EXT4 achieves this improvement by only scanning portions of the file system that have been modified since the last check, rather than scanning the entire file system. This selective scanning significantly reduces the time needed to verify the integrity of the file system, allowing for faster recovery and minimizing downtime. For mission-critical environments where uptime is a priority, such as enterprise data centers or cloud infrastructure, this feature is invaluable.

2.6 Journaling and Data Integrity

Like its predecessor, EXT4 is a journaling file system, meaning it keeps a record (journal) of changes to the file system before committing them to the disk. This ensures that the file system remains in a consistent state even in the event of a crash or power failure. However, EXT4 offers several enhancements over EXT3's journaling capabilities. By default, EXT4 journals metadata only, but it can be configured for full data journaling for environments where data integrity is critical (Mathur et al., 2007).

EXT4's journaling system is designed to strike a balance between performance and reliability. In most scenarios, journaling metadata is sufficient to maintain file system consistency while minimizing performance overhead. However, for applications that demand a higher level of data integrity such as

financial systems, databases, or systems with critical log file full journaling provides an extra layer of protection, ensuring that even data itself is written consistently in the event of a failure.

2.7 Improved Allocation Algorithms

EXT4 introduces several advanced allocation algorithms that improve the overall efficiency of the file system. One such algorithm is the *multi-block allocator*, which allows the file system to allocate multiple blocks in a single operation, reducing the overhead associated with individual block allocations. This improves performance, particularly when dealing with large files that require contiguous storage space. The allocator is designed to optimize the layout of files on disk, reducing fragmentation and improving access times (Ts'o, 2010).

Additionally, EXT4's *buddy system allocator* groups block into "buddies" of varying sizes, allowing the file system to allocate memory more efficiently and avoid the fragmentation issues commonly found in older allocation systems. These improvements ensure that EXT4 can handle a wide range of workloads, from small files accessed randomly to large files requiring sequential access, making it a versatile choice for different types of storage needs.

3. NTFS Overview

NTFS (New Technology File System) was developed by Microsoft in 1993, introduced with Windows NT 3.1, and remains the default file system for modern Windows operating systems, including Windows 10, 11, and Windows Server editions. NTFS was designed to be a robust, secure, and scalable file system, addressing the limitations of the earlier FAT (File Allocation Table) systems, which struggled to handle large file sizes and offered limited security features. NTFS introduced a new architecture, and a suite of advanced features aimed at enhancing performance, reliability, and data protection, making it well-suited for both consumer and enterprise environments (Sweeney, 1996).

3.1 Evolution and Design Goals

Before NTFS, Microsoft's earlier file systems, such as FAT16 and FAT32, had significant limitations in terms of scalability, security, and file management. FAT32, for instance, had a maximum file size limit of 4GB and a maximum partition size of 8TB, which was insufficient for modern applications and enterprise-level systems. In response to these challenges, Microsoft developed NTFS to support larger volumes and provide advanced features for data management and protection.

The primary design goals of NTFS were to improve performance, provide advanced security mechanisms, and support large file systems while maintaining compatibility with Windows-based applications. NTFS was built to accommodate the increasing complexity of file storage systems, with an emphasis on data integrity, recoverability, and flexibility. Its modular design and extensive feature set have ensured that NTFS remains a dominant file system for Windows-based platforms, even decades after its introduction (Carrier, 2005).

3.2 Journaling and Data Integrity

One of the key features of NTFS is its support for *journaling*, which significantly improves data integrity and recovery capabilities. Journaling refers to the process of keeping a log of changes before they are committed to the main file system. In the event of an unexpected system crash or power failure, the journal allows NTFS to quickly recover and restore the file system to a consistent state by replaying or discarding incomplete transactions (Sweeney, 1996).

NTFS uses a metadata-based journaling system, meaning that it primarily records changes to file system metadata, such as file names, locations, and permissions, rather than the actual data itself. This minimizes the performance overhead associated with journaling while still providing a high level of data integrity. In environments where data loss could result in significant disruptions, such as enterprise servers or financial

systems, NTFS's journaling feature is invaluable for maintaining system stability.

3.3 Security Features: Access Control Lists (ACLs) and Encryption

NTFS offers a comprehensive security model based on *Access Control Lists (ACLs)*, which allow administrators to define permissions for individual users or groups at a fine-grained level. This flexibility ensures that different users can have different levels of access to files and directories, enhancing data security in multi-user environments. ACLs allow for a wide range of permission settings, including read, write, execute, and delete privileges. This contrasts with the simpler permission models used in older file systems, such as FAT32, which lacked the ability to apply detailed user-specific permissions (Microsoft, 2020).

In addition to ACLs, NTFS supports *Encrypting File System (EFS)*, which allows individual files or folders to be encrypted at the file system level. EFS is tightly integrated with the Windows operating system, making it seamless for users to secure their data without requiring additional encryption software. Encryption keys are stored securely, and only authorized users can access the encrypted data, ensuring that sensitive information remains protected even if a device is compromised or stolen (Microsoft, 2020).
The combination of ACLs and EFS provides a robust security framework that makes NTFS an ideal file system for environments where data confidentiality and access control are critical. These features are particularly useful in enterprise environments, where sensitive data such as financial records, intellectual property, or personal information must be protected from unauthorized access.

3.4 Support for Large File Sizes and Volumes

NTFS was designed with scalability in mind, supporting both large file sizes and large storage volumes. The theoretical maximum file size supported by NTFS is 16 exabytes (EB), although practical implementations typically limit file sizes to 256 terabytes (TB) due to hardware and software constraints (Carrier, 2005). This makes NTFS well-suited for environments where large files, such as high-definition video, large databases, or virtual machine disk images, are common.
In addition to supporting large individual files, NTFS can manage very large partitions. Modern implementations of NTFS can support partitions up to 256TB, far exceeding the limitations of older file systems such as FAT32, which capped partition sizes at 32GB. This scalability makes NTFS a popular choice for enterprise-level storage solutions, where large amounts of data need to be stored and managed efficiently.

3.5 Compression and Deduplication Features

NTFS also offers file compression at the file system level, which can help optimize disk space usage. The built-in compression feature allows files to be stored in a compressed format without requiring separate compression tools. This is particularly beneficial in scenarios where storage space is at a premium, such as in data centers or systems with limited storage capacity. NTFS compression is transparent to users, meaning that files can be read and written normally even when compressed. The compression feature is especially useful for storing large amounts of text files, databases, or documents that compress well (Sweeney, 1996).
Furthermore, NTFS supports deduplication, a storage optimization technique that reduces the amount of redundant data stored on a disk. Deduplication scans the file system for duplicate data blocks and ensures that only one copy of the data is stored, with subsequent copies being replaced by references to the original. This feature is particularly useful in environments where large volumes of data are generated, such as backup systems or virtualized environments, as it can significantly reduce storage costs by minimizing the amount of disk space required (Xia et al., 2014).

3.6 Fault Tolerance and Self-Healing

NTFS includes several fault-tolerance mechanisms to ensure data reliability and system stability. One of the key features is its ability to detect and correct file system errors through a process known as *self-healing*. In the event of a disk error or corruption, NTFS can automatically repair damaged files or metadata without

requiring user intervention, ensuring that the file system remains in a consistent state (Microsoft, 2020). This is achieved through NTFS's use of the Master File Table (MFT), which keeps a detailed record of all files and directories on the disk.

The *Transaction Safe File System (TxF)* is another feature of NTFS that adds an extra layer of fault tolerance. TxF allows for atomic file operations, ensuring that file changes are either fully completed or rolled back if an error occurs. This is especially useful for applications that require consistent data writes, such as databases or financial systems, where partial updates or corruption could lead to significant data loss.

3.7 NTFS and Windows Integration

One of NTFS's major advantages is its deep integration with the Windows operating system. NTFS was specifically designed to work seamlessly with Windows, supporting all of the file system features required by modern Windows applications, including the Windows registry, system file management, and security features like user authentication and encryption. This tight integration ensures that NTFS is optimized for performance and reliability within the Windows environment, making it the default choice for all modern Windows installations (Carrier, 2005).
Additionally, NTFS supports several advanced features that enhance Windows' performance and usability, such as *Volume Shadow Copy Service (VSS)*, which allows for the creation of system snapshots. VSS enables users to recover previous versions of files or restore the system to a previous state without significant downtime. This feature is especially important in enterprise environments where continuous data protection is essential.

4. Performance Comparison

When comparing EXT4 and NTFS, one of the most critical aspects to evaluate is their performance, particularly in terms of read/write speeds and the way they handle file system fragmentation. Both file systems are designed for different operating environments EXT4 for Linux-based systems and NTFS for Windows which impacts how each performs in various scenarios. Performance differences can also arise due to differences in system architectures, workloads, and hardware. This section delves into the comparative performance of EXT4 and NTFS in key areas such as read/write efficiency, file system fragmentation, and overall system optimization.

4.1 Read/Write Performance

The read/write performance of a file system is a crucial factor in determining its efficiency in accessing and storing data. File systems manage the data flow between storage devices (such as hard drives or SSDs) and the operating system, and their read/write performance directly impacts system responsiveness, application performance, and user experience.

4.1.1 Sequential Read/Write Performance

Sequential read and write operations refer to reading or writing data in contiguous blocks. These operations are typically faster because the system can access data in a linear fashion without having to jump to different locations on the disk. Studies have demonstrated that EXT4 generally excels in sequential read/write performance, especially in Linux environments, as it is specifically optimized for the Linux kernel's I/O subsystems (Prabhakar et al., 2019).

One of the primary reasons for EXT4's superiority in sequential operations is its use of *extents* and *delayed allocation*. Extents allow the file system to allocate multiple contiguous blocks at once, reducing the overhead associated with mapping individual blocks. This results in fewer disks seeks, lower fragmentation, and improved throughput for large file transfers, which are common in sequential read/write workloads such as video rendering or database backups (Mathur et al., 2007).
Moreover, the delayed allocation mechanism in EXT4 optimizes write performance by deferring the

assignment of disk blocks until the data is ready to be written. This allows the system to make better decisions about data placement on the disk, further enhancing sequential write speeds by reducing fragmentation and clustering related data blocks together (Ts'o, 2010).

4.1.2 Random Read/Write Performance

In contrast to sequential operations, random read/write operations involve accessing or writing data in non-contiguous locations on the disk. This can be more demanding on the file system, as it requires more frequent disk seeks and metadata updates. NTFS has been shown to excel in random read/write performance, particularly in Windows environments, where its architecture is optimized for handling small, frequent updates to the file system, such as those generated by applications, databases, or operating system files (Sweeney, 1996).

NTFS benefits from its efficient handling of *metadata* through the Master File Table (MFT). The MFT centralizes metadata storage and allows NTFS to quickly locate files and update information such as permissions, attributes, and timestamps. This centralization reduces the overhead associated with random read/write operations, making NTFS well-suited for workloads that involve a mix of small, non-contiguous file operations, such as file servers, database applications, or system processes that require frequent updates to file metadata (Carrier, 2005).

NTFS also implements a journaling system that records changes to the file system in a transaction log before committing them to disk. This ensures that, even in the event of a system crash or power failure, the file system remains consistent and can recover quickly without requiring a full file system check. Although journaling can introduce a small performance overhead, especially in sequential write-heavy workloads, it enhances NTFS's random read/write efficiency by reducing the risk of data corruption and speeding up recovery times in failure scenarios (Microsoft, 2020).

4.2 File System Fragmentation

File system fragmentation occurs when a file is broken into non-contiguous blocks on the disk, which can significantly degrade performance, especially in random read/write operations. Both EXT4 and NTFS implement mechanisms to manage fragmentation, but their approaches and effectiveness differ.

4.2.1 EXT4 and Fragmentation Management

EXT4 addresses the issue of fragmentation through its use of *extents*. By grouping contiguous blocks into single units, extents help minimize fragmentation and ensure that files are stored in large, contiguous areas of the disk. This is particularly beneficial for sequential operations, as the file system can access data more efficiently when it is stored contiguously (Mathur et al., 2007). The use of extents also improves the overall scalability of EXT4, as it reduces the number of block pointers needed to map large files, further reducing the likelihood of fragmentation as files grow.

In addition to extents, the delayed allocation mechanism in EXT4 plays a crucial role in preventing fragmentation. By delaying the allocation of disk blocks until data is ready to be written, EXT4 can optimize the placement of data on disk, ensuring that related blocks are stored together. This not only improves performance but also reduces the need for defragmentation, as the file system naturally avoids creating fragmented files (Ts'o, 2010).

However, fragmentation can still occur in EXT4 under certain conditions, particularly in systems with high disk utilization or when many small files are frequently created, modified, or deleted. Fortunately, EXT4 includes a built-in tool called e4defrag, which allows users to manually defragment the file system, when necessary, though the need for defragmentation is much less frequent compared to older file systems like EXT2 and EXT3.

4.2.2 NTFS and Fragmentation Management

NTFS, while optimized for random read/write performance, is more prone to fragmentation compared to EXT4 due to its *block-based* architecture. In NTFS, files are divided into blocks of fixed sizes, and as files are modified or deleted, free space becomes fragmented, causing new files to be stored in non-contiguous locations. This can lead to performance degradation over time, particularly in systems with heavy file system activity or high fragmentation levels (Sweeney, 1996).

To mitigate the effects of fragmentation, NTFS includes a built-in *defragmentation tool* that can be run manually or scheduled to optimize file placement and reduce fragmentation. The defragmentation process rearranges fragmented files so that they occupy contiguous blocks on the disk, improving both read and write performance. Modern versions of Windows, starting with Windows 7, include automatic defragmentation features that run in the background, helping to maintain performance without requiring manual intervention (Microsoft, 2020).

Additionally, NTFS uses *Sparse Files*, a feature that allows the file system to allocate large files more efficiently by only storing non-empty portions of the file. Sparse files help reduce the amount of disk space consumed by large files and can reduce fragmentation by minimizing the number of non-contiguous blocks required to store them (Carrier, 2005).

4.3 Performance in Different Workload Scenarios

The performance of EXT4 and NTFS varies depending on the type of workload and the environment in which they are used. In general, EXT4 is more efficient in *Linux-based systems* and excels in workloads that involve large, contiguous data transfers, such as media production, backup systems, or data archiving. Its use of extents and delayed allocation allows it to handle large files with minimal fragmentation, making it ideal for systems with large sequential read/write workloads (Mathur et al., 2007).
NTFS, on the other hand, is optimized for *Windows-based systems* and performs well in environments that require a mix of random and sequential access, such as file servers, virtualized environments, or database systems. NTFS's efficient handling of metadata and its robust journaling system make it well-suited for workloads that involve frequent updates to small files or applications with complex file structures, such as system files or application logs (Carrier, 2005).

5. Security Comparison

Security is a paramount concern in file systems, particularly for organizations and users handling sensitive or personal data. File systems employ various security mechanisms to safeguard data from unauthorized access, ensure data integrity, and provide secure deletion. Both EXT4 and NTFS offer security features, but they differ in how they implement access control, encryption, and additional security measures. This section takes a look at the security capabilities of EXT4 and NTFS, focusing on access control, encryption, and additional security features that help prevent unauthorized access and data breaches.

5.1 Access Control

Access control mechanisms define who has permission to read, modify, or execute files within a system. A well-implemented access control system is crucial for protecting sensitive information, ensuring that only authorized users can interact with specific data. Both EXT4 and NTFS implement access control, but they differ significantly in their approaches and granularity.

5.1.1 NTFS Access Control

NTFS (New Technology File System) offers robust access control mechanisms through its implementation of Access Control Lists (ACLs). ACLs allow administrators to specify fine-grained permissions for individual files and directories, providing flexibility and control over data access. NTFS supports Discretionary Access Control (DAC) and System Access Control (SAC), enabling administrators to set permissions based on both user identity and system-level rules. Each file and folder in NTFS can have its

own ACL, which defines specific actions that different users or groups can perform, such as reading, writing, or executing (Microsoft, 2020).

This level of granularity is particularly beneficial in environments where multiple users require different levels of access to the same data. For example, in a corporate setting, ACLs allow administrators to ensure that only authorized personnel can modify critical files while granting read-only access to other users. NTFS's support for inheritance also simplifies the management of permissions by automatically applying parent folder permissions to child files and folders, reducing administrative overhead (Carrier, 2005).

5.1.2 EXT4 Access Control

EXT4, the default file system for many Linux distributions, relies on the traditional Unix-based permission model, which defines access control based on three classes of users: owner, group, and others. Each file and directory have three types of permissions (read, write, and execute) that can be assigned to these user classes. While this model is simple and effective for most use cases, it lacks the fine-grained control offered by NTFS's ACLs.

To address the limitations of the traditional Unix permission model, EXT4 supports Access Control Lists (ACLs) through the POSIX extension. POSIX ACLs allow administrators to assign additional permissions beyond the standard owner, group, and others model. This provides more flexibility in environments where more granular access control is required. However, even with POSIX ACLs, the EXT4 system is generally considered less flexible and sophisticated than NTFS when it comes to fine-tuning access control at the individual file or user level (Ts'o, 2010).

In environments where strict access control is necessary, such as enterprise settings with diverse user groups and hierarchical data structures, NTFS's ACLs are often seen as more suitable. On the other hand, EXT4's simpler permission model, enhanced by POSIX ACLs, remains effective for many Linux environments, particularly in scenarios where fine-grained control is not critical.

5.2 Encryption

Encryption is a critical feature for securing data at rest, ensuring that unauthorized users cannot access sensitive information even if they gain physical access to the storage device. Both EXT4 and NTFS offer encryption capabilities, but they differ in their implementation and integration with the operating system.

5.2.1 NTFS Encryption (Encrypting File System - EFS)

NTFS includes built-in encryption through the Encrypting File System (EFS), which provides file-level encryption for individual files and folders. EFS is deeply integrated with the Windows operating system, allowing users and administrators to easily encrypt and decrypt files without the need for third-party tools. EFS is designed to be user-friendly, with encryption occurring automatically in the background once enabled, and users can continue to interact with encrypted files just as they would with non-encrypted ones (Microsoft, 2020).

EFS works by using a combination of public key and symmetric key encryption. When a file is encrypted with EFS, the system generates a symmetric key to encrypt the file's data. This symmetric key is then encrypted with the user's public key and stored in the file's metadata. When the user accesses the file, their private key is used to decrypt the symmetric key, which in turn decrypts the file. This layered approach ensures that even if someone gains access to the encrypted file, they cannot decrypt it without the corresponding private key (Carrier, 2005).

One advantage of EFS is its seamless integration with other Windows security features, such as BitLocker, which provides full-disk encryption. Together, EFS and BitLocker offer comprehensive encryption solutions for both individual files and entire drives, making NTFS a robust choice for environments that

require strong encryption capabilities.

5.2.2 EXT4 Encryption

Unlike NTFS, EXT4 did not initially include built-in encryption. However, Linux users had access to third-party encryption tools like eCryptfs and dm-crypt to secure data on EXT4 file systems. These tools provide flexible encryption options, but their use requires additional configuration and setup compared to NTFS's native EFS (Ts'o, 2010).

In 2015, EXT4 introduced native encryption support, bringing it closer to NTFS in terms of security features. EXT4's native encryption is file-based, similar to NTFS's EFS, and allows users to encrypt individual directories. However, unlike NTFS, which integrates encryption into the operating system's file management features, EXT4's encryption requires more manual intervention and configuration, often involving command-line tools and kernel parameters (Ts'o, 2015).

While EXT4's encryption is secure, it is generally considered less user-friendly than NTFS's EFS, particularly for non-technical users. Additionally, EXT4 encryption is more limited in scope, lacking some of the advanced features found in NTFS, such as support for encrypting file metadata. This makes NTFS a more robust choice for users or organizations that require extensive encryption features, particularly in enterprise environments where ease of use and integration with other security tools are essential.

5.3 Additional Security Features

5.3.1 NTFS Security Features

Beyond ACLs and encryption, NTFS includes several additional security features that enhance its ability to protect data. One notable feature is its journaling capability, which helps protect against data corruption in the event of a system crash or power failure. The journal records changes to the file system in a transaction log before committing them to disk, ensuring that the system can recover to a consistent state even if a failure occurs (Microsoft, 2020).

NTFS also supports file compression, allowing users to reduce the size of files on the disk without needing third-party compression tools. While compression is not a security feature per se, it can help prevent data leakage by ensuring that compressed files are more difficult to manipulate or tamper with (Carrier, 2005).

5.3.2 EXT4 Security Features

EXT4 also supports journaling, which enhances data integrity by ensuring that the file system can recover from crashes or unexpected shutdowns. Like NTFS, EXT4's journaling capability ensures that file system changes are recorded before they are committed to disk, reducing the risk of data corruption.

One unique security feature of EXT4 is its support for secure deletion, which allows users to securely erase files by overwriting the file's data before deleting it. This ensures that deleted files cannot be recovered by forensic tools, providing an additional layer of security for sensitive data (Ts'o, 2010). NTFS, by contrast, does not include a built-in secure deletion mechanism, though third-party tools can be used to achieve the same result.

In terms of security, both EXT4 and NTFS offer robust features to protect data, but they differ in implementation and ease of use. NTFS provides more advanced access control mechanisms through its use of ACLs, allowing for granular permission settings at the individual file level. It also offers native encryption through EFS, which is tightly integrated with the Windows operating system and provides a user-friendly experience for encrypting and securing files. Additionally, NTFS's journaling capability enhances data integrity and recovery.

EXT4, on the other hand, relies on the traditional Linux permission model, with support for POSIX ACLs for more granular control. Its encryption options have improved with the introduction of native file-based encryption, though it still lacks the seamless integration and user-friendliness of NTFS's EFS. EXT4 also

includes unique security features such as secure deletion, which provides additional protection for sensitive data. Ultimately, the choice between EXT4 and NTFS for security depends on the specific requirements of the user or organization, with NTFS offering more advanced features for environments that demand high levels of security and ease of use.

6. Reliability Comparison

Reliability is a key factor when evaluating file systems, as it determines how well a system can preserve data integrity and recover from failures such as system crashes, power outages, or hardware malfunctions. Both EXT4 and NTFS have been designed with reliability in mind, incorporating mechanisms like journaling, error recovery, and data integrity checks. This section compares the reliability of these two file systems, focusing on their journaling techniques, recovery mechanisms, and other reliability-enhancing features.

6.1 Journaling

Journaling is a technique used by modern file systems to enhance reliability by keeping a log (journal) of changes that will be made to the file system. This log ensures that, in the event of a crash or sudden power loss, the system can restore consistency by replaying or discarding incomplete transactions.

6.1.1 EXT4 Journaling

EXT4, the fourth version of the Extended File System, implements a metadata-only journaling model by default, which logs changes to the file system's metadata before committing them to disk. This approach strikes a balance between performance and reliability, ensuring that critical file system structures (such as directories and inode tables) are always consistent without incurring a significant performance penalty. For users or environments that require even higher levels of reliability, EXT4 can be configured to use full data journaling, where both file data and metadata changes are logged before being written to disk (Mathur et al., 2007).

However, full data journaling comes with a performance trade-off. Logging file data increases the amount of information that must be written to the journal, leading to slower write operations. For many users, the default metadata-only journaling is sufficient, but in environments where data corruption could have severe consequences, the full journaling option provides an extra layer of protection.
Another advantage of EXT4's journaling system is its use of checksumming for journal entries, which helps detect corruption in the journal itself. If a journal entry is found to be corrupted, EXT4 can discard it, minimizing the risk of propagating errors during recovery. This feature enhances the overall reliability of the file system, especially in cases where the storage medium itself may be prone to errors.

6.1.2 NTFS Journaling

NTFS, developed by Microsoft, employs a more comprehensive journaling model that logs both metadata and file data changes. This full journaling approach ensures that the file system can recover not only its structure but also the content of files in the event of a failure. NTFS's journaling system, known as the NTFS Log File Service, creates a transaction log that records changes before they are committed to the file system. If a crash occurs, NTFS can use this log to roll back incomplete transactions, ensuring that the file system remains in a consistent state (Sweeney, 1996).

The primary advantage of NTFS's full journaling system is the higher level of protection it offers against data corruption. In environments where data integrity is paramount, such as financial systems or enterprise servers, NTFS's ability to journal both metadata and data provides a more reliable solution than EXT4's default metadata-only journaling.

However, this increased reliability comes at a cost: the overhead associated with journaling both metadata and file data can reduce performance, particularly in write-intensive workloads. In environments where

performance is critical, such as large-scale data processing or high-performance computing, this overhead can be a significant factor in choosing between file systems (Aoyama & Iwasaki, 2019). Nonetheless, NTFS's reliability benefits make it a preferred choice for systems where the cost of data corruption is high.

6.2 Recovery Mechanisms

Recovery mechanisms are essential for restoring a file system to a consistent state following a crash, power failure, or unexpected shutdown. Both EXT4 and NTFS employ advanced recovery techniques to minimize data loss and ensure data integrity.

6.2.1 EXT4 Recovery Mechanisms

One of the standout features of EXT4 is its fast file system check (fsck), which significantly reduces the time required to scan and repair the file system after an error. Traditional file system checks can be time-consuming, especially on large storage volumes, but EXT4's optimized fsck mechanism accelerates this process by focusing on areas of the file system that are likely to have encountered errors (Ts'o, 2010). This improvement is particularly valuable in scenarios where downtime must be minimized, such as in server environments or mission-critical systems.

In addition to its fsck improvements, EXT4 incorporates several features that enhance its recovery capabilities. One such feature is orphan inode management, which tracks inodes (file metadata) that are in the process of being deleted. If a system crash occurs while a file is being deleted, the orphan inode mechanism ensures that the file is properly cleaned up during the next boot, preventing the file system from entering an inconsistent state (Mathur et al., 2007).

EXT4's journaling system also plays a key role in recovery. After a crash, the file system replays the journal to restore any incomplete transactions, ensuring that the file system remains consistent. If the journal itself is corrupted, EXT4 can use checksums to detect the corruption and discard the faulty entries, further enhancing the file system's ability to recover from errors.

6.2.2 NTFS Recovery Mechanisms

NTFS's recovery mechanisms are built around its journaling system, which logs all file system changes before they are committed. In the event of a crash, NTFS uses this transaction log to replay or discard changes, ensuring that the file system remains in a consistent state (Microsoft, 2020). This approach makes NTFS particularly resilient to data corruption caused by unexpected shutdowns or hardware failures.

NTFS also supports a feature known as Transactional NTFS (TxF), which allows applications to perform file operations as atomic transactions. If an error occurs during the transaction, NTFS can roll back the changes, ensuring that the file system remains consistent. TxF is particularly useful in environments where data integrity is critical, such as financial systems or database servers, as it allows file operations to be completed in a manner that guarantees consistency even in the event of failure (Russinovich et al., 2012).

Another important aspect of NTFS's recovery capabilities is its self-healing feature, introduced in Windows Vista and later versions of the operating system. This feature allows NTFS to automatically detect and repair minor file system corruption without requiring a full file system check or user intervention. This reduces downtime and enhances the reliability of NTFS in environments where manual recovery procedures are not feasible.

6.3 Data Integrity and Error Detection

Both EXT4 and NTFS employ mechanisms to ensure data integrity, but they take different approaches to error detection and correction.

6.3.1 EXT4 Data Integrity Features

EXT4 ensures data integrity through a combination of journaling, checksumming, and file system checks. The use of checksums in EXT4's journal entries help detect corruption in the journal itself, ensuring that

only valid transactions are replayed during recovery. Additionally, EXT4's fsck tool can detect and repair file system inconsistencies, further enhancing data integrity (Ts'o, 2010).

One limitation of EXT4, however, is its lack of built-in checksumming for file data. While the journal is protected by checksums, file data is not, meaning that data corruption could go undetected unless additional tools, such as RAID or file system integrity checkers, are used. This makes EXT4 less reliable than NTFS in environments where data corruption is a primary concern.

6.3.2 NTFS Data Integrity Features

NTFS takes a more comprehensive approach to data integrity by incorporating checksumming for both file system metadata and file data. This ensures that any corruption, whether in the file system structure or the data itself, can be detected and corrected. NTFS's ability to verify the integrity of file data is particularly valuable in environments where data integrity is critical, such as financial institutions or healthcare systems (Microsoft, 2020).

NTFS also supports the use of redundant storage solutions, such as RAID, to further enhance data integrity. By combining NTFS's built-in error detection with RAID's redundancy, organizations can ensure that their data remains intact even in the event of hardware failures or data corruption. In terms of reliability, both EXT4 and NTFS offer robust features to ensure data integrity and recover from crashes, but they differ in their approaches. EXT4's fast file system check (fsck), orphan inode management, and checksummed journaling provide effective recovery mechanisms while maintaining a balance between performance and reliability. However, its default metadata-only journaling and lack of built-in file data checksumming make it less comprehensive than NTFS in terms of data integrity protection.

NTFS, with its full journaling model, transactional file operations, and self-healing features, offers a higher level of reliability, particularly in environments where data corruption is unacceptable. While its additional overhead can reduce performance in write-heavy workloads, NTFS's robust recovery and data integrity features make it a preferred choice for enterprise systems where reliability is paramount.

7 Conclusion

Considering each file system is made for a different purpose, the decision between EXT4 and NTFS is based on the needs of the system and environment. EXT4 is best suited for Linux and open-source environments, offering reliable performance and scalability for handling large amounts of data. It is especially useful in servers, cloud storage, and systems that prioritize processing data in a sequential manner. EXT4's straightforward design ensures good performance and basic security features, which are often enough for Linux-based systems. On the other hand, NTFS is ideal for Windows environments where advanced security and compatibility are key priorities. NTFS supports features like Access Control Lists (ACLs), Encrypting File System (EFS), and advanced journaling, which protect data and ensure its integrity. These capabilities make NTFS a popular choice for enterprise systems and corporate environments handling sensitive or critical data. Additionally, NTFS works seamlessly with Windows, making it a practical choice for desktop systems where random read/write performance is more important than sequential data processing. EXT4 is a strong option for Linux systems that need scalability and performance, while NTFS is better suited for Windows systems that require advanced security and reliability. The decision should be based on the operating system in use, the specific performance needs of the application, and the level of security and reliability required for the system. Both file systems are reliable and effective, but their strengths align with different use cases.

References

Aoyama, W., & Iwasaki, H. (2019). Dajfs: A new file system with per-directory adaptive journaling. *Journal of Information Processing*, *27*, 369-377. Carrier, B. (2005). *File system forensic analysis*. Addison-Wesley.

Mathur, A., Cao, M., Bhattacharya, S., Dilger, A., Tomas, M., & Vivier, L. (2007). The new ext4 filesystem: Current status and future plans. *Linux Symposium*, 193-200.

Microsoft. (2020). *NTFS technical reference*. Microsoft Documentation. https://docs.microsoft.com/en-us/windows-server/storage/file-server/ntfs-overview

Prabhakar, K., Singla, A., & Sharma, R. (2019). Performance evaluation of file systems on SSD and HDD in a Linux environment. *Journal of Systems Architecture*, *93*, 58-69. https://doi.org/10.1016/j.sysarc.2018.12.001

Prabhakar, M., Kumar, R., & Das, A. (2019). Comparative performance analysis of file systems: NTFS, FAT, EXT, XFS, BTRFS. *International Journal of Scientific Research in Computer Science and Engineering*, *7*(2), 15-22. https://doi.org/10.26438/ijscse/v7i2.1522

Russinovich, M., Solomon, D., & Ionescu, A. (2012). *Windows internals*. Microsoft Press.

Sweeney, D. (1996). NTFS design and implementation. *Microsoft Developer Network Journal*, *1*(2), 14-25.

Ts'o, T. (2010). *Ext4 disk layout*. Linux Kernel Documentation. https://ext4.wiki.kernel.org

Ts'o, T. (2015). *Native encryption in EXT4*. Linux Kernel Documentation.

Xia, W., Jiang, H., Feng, D., & Tian, L. (2014, March). Combining deduplication and delta compression to achieve low-overhead data reduction on backup datasets. In *2014 Data Compression Conference* (pp. 203-212). IEEE.