computers

Article

High-Performance Computing Storage Performance and Design Patterns—Btrfs and ZFS Performance for Different Use Cases

Vedran Dakic [1,*], Mario Kovac [2,*] and Igor Videc [1]

1 Department of Operating Systems, Algebra University, 10000 Zagreb, Croatia; igor.videc@algebra.hr
2 Faculty of Electrical Engineering and Computing, University of Zagreb, 10000 Zagreb, Croatia
* Correspondence: vedran.dakic@algebra.hr (V.D.); mario.kovac@fer.hr (M.K.)

Abstract: Filesystems are essential components in contemporary computer systems that organize and manage data. Their performance is crucial in various applications, from web servers to data storage systems. This paper helps to pick the suitable filesystem by comparing btrfs with ZFS by considering multiple situations and applications, ranging from sequential and random performance in the most common use cases to extreme use cases like high-performance computing (HPC). It showcases each option's benefits and drawbacks, considering different usage scenarios. The performance of btrfs and ZFS will be evaluated through rigorous testing. They will assess their capabilities in handling huge files, managing numerous small files, and the speed of data read and write across varied usage levels. The analysis indicates no definitive answer; the selection of the optimal filesystem is contingent upon individual data-access requirements.

Keywords: btrfs; ZFS; Linux; filesystem; performance; HPC

check for updates

Citation: Dakic, V.; Kovac, M.; Videc, I. High-Performance Computing Storage Performance and Design Patterns—Btrfs and ZFS Performance for Different Use Cases. *Computers* **2024**, *13*, 139. https://doi.org/10.3390/computers13060139

Academic Editor: Paolo Bellavista

Received: 3 May 2024
Revised: 20 May 2024
Accepted: 22 May 2024
Published: 3 June 2024

1. Introduction

Efficient and reliable information systems rely on excellent data management. Given the rapid increase in data volumes, there is an urgent requirement for sophisticated filesystems that can effectively manage, store, and protect data. Linux, a dominant operating system in server and cloud environments, provides support for various filesystems, including ext4, XFS, btrfs, and ZFS. Btrfs (B-tree file system) and ZFS (zettabyte file system) are known for their advanced capabilities, including efficient management of enormous data volumes and the capacity to recover from errors automatically. These characteristics make them essential for constructing dependable, expandable, and secure information infrastructures, garnering substantial attention from IT professionals and academics. Also, the flexibility to expand and shrink the storage capacity is one of the essential elements that should be considered [1]. There are other requirements, including that the storage must execute many other CPU-intensive tasks, such as compression, data deduplication, checksum calculations, snapshots, and data cloning [2].

Although btrfs and ZFS are acknowledged for their groundbreaking features, comprehensive comparisons that examine their performance in various usage circumstances are needed. Real-world data management encompasses intricate activities, such as state imaging, replication, and dynamic resource allocation, necessitating a comprehensive comprehension of how multiple filesystems accomplish these duties. Storage can employ a physical resource like multiple virtual resources or pool several physical resources together to execute as a virtual resource [3]. In that regard, this research aims to provide an impartial analysis that will assist in comprehending the merits and drawbacks of each system. This will enable people to make well-informed decisions when choosing a filesystem for specific applications.

The purpose of this study is to achieve multiple objectives. The primary aim is to comprehensively examine the btrfs and ZFS filesystems, emphasizing their prominent characteristics. It also aims to test performance on both platforms across different usage scenarios, most importantly, for a test high-performance computing (HPC) application that we selected for the test. The study tries to determine how one system may surpass the other and investigate how the configuration and features impact overall performance. Ultimately, this paper will combine the findings to provide practical advice and suggestions for incorporating these filesystems into critical IT infrastructures. The main contribution of this study is its capacity to guide the characteristics and performance of btrfs and ZFS filesystems. This information is the foundation for making well-informed decisions while developing and overseeing filesystems. Furthermore, the findings of this study can be utilized as a foundation for forthcoming research and advancement in the field of data administration and filesystems.

2. Problem Status

This paper's topic is a part of our large-scale research into large-scale virtualized, cloud, and HPC environments. Over the past ten years, while working with these environments, we concluded that choosing a storage system (technology) and filesystem running on top of it is one of the most important design decisions you can make in these prominent environments. There is a multitude of reasons for this conclusion:

- Performance optimization: It is an entirely different scenario if we use storage for storing users' files (file server) versus virtualization, versus cloud, and versus HPC. The filesystem plays a critical role, as it determines how fast we can read from and write to it based on the scenario requirements;
- Scalability and flexibility: As much as file servers do not necessarily have to be able to scale quickly, the same is not valid for virtualization, cloud, and HPC—these scenarios need to be easily scalable—both for growing datasets and increased computational loads;
- Data integrity and reliability: If we lose a file or two on a general file server, it usually is not the end of the world for the company. On the other hand, if we lose a file in HPC environments, that corrupts research and results. That is why the filesystem has to be a building block that ensures that no corruption happens during the store or retrieve processes, especially when thinking about design for failures that will occur (it is not a question of "if"; it is a question of "when"). We also need to consider that different filesystems treat inevitable failures differently. For example, ext4 and XFS react differently to fsync failures (after the fsync system call has been initiated, it is expected that the application's buffer is going to be flushed to the storage device—if this does not complete successfully, we have a fsync failure) [4];
- Advanced features: Snapshots, deduplication, caching, and similar features are paramount when dealing with large-scale data storage, especially in cloud and HPC environments. These features can have a significant impact on storage efficiency and data-management tasks;
- Overhead management: In large-scale environments like cloud and HPC, we must use all available resources efficiently. We do not want to waste unnecessary computational power on storage I/O or storage I/O bottlenecks when discussing storage.

If the storage system does not meet these requirements, the long-term effects are going to create problems in terms of:

- Performance degradation: The wrong filesystem and storage choice will create I/O bottlenecks, which will, in turn, slow down our computation and reduce system efficiency;
- Scalability problems: If we cannot scale our storage, large-scale environments like cloud and HPC are going to hinder our growth;
- Data corruption and loss: A filesystem that is not robust enough for our data-storage requirements will eventually lead to data corruption or loss, especially if we do not plan for failures or crashes. Having in mind that data integrity in scientific research is crucial, this might lead to severe problems concerning projects and credibility;

- Inability to efficiently operate storage: If we cannot use snapshots while running virtual machines, that is a significant operational problem. If we cannot use data deduplication, there is a substantial chance that we are wasting available storage space.

As a result, we determined that choosing the proper filesystem, especially in HPC, is one of the most critical topics when designing an HPC environment. This is why we skipped some other common types of filesystems (for example, ext4, FAT, and NTFS) in terms of testing, as it would be impractical, at times irresponsible, and downright counterproductive to use these filesystems for large virtualized, cloud, or HPC environments. We explore these key considerations and slowly build our test suite out—from some less intensive scenarios to running an HPC app. This approach should give us a good overview of how to design our storage environment properly so that we do not end up in situations where we have to re-design, as this is a very costly exercise.

3. Overview of Currently Used Filesystems

Developing filesystems is a continuous endeavor to provide technologies for more effective data administration and storage. During the initial stages of computing, filesystems were very uncomplicated, specifically created to manage tiny data collections within systems. Nevertheless, with the progression of technology and the exponential growth of data volumes, it became clear that there was a requirement for more robust filesystems. The development of early filesystems, such as FAT (file allocation table) and NTFS (new technology file system) aimed to manage data efficiently. FAT was initially launched with MS-DOS and later chosen as the standard for Windows operating systems, serving as the basis for organizing and retrieving data. If we are running a Windows-only environment, NTFS is the best choice of available filesystems [5]. Considering that most virtualized, cloud, and HPC environments are not Windows based, we must survey open-source filesystems for those scenarios.

3.1. File Allocation Table (FAT) Filesystems

The file allocation table (FAT) filesystem, created by Microsoft (Albuquerque, NM, USA) in 1977, is one of the most uncomplicated and widely utilized filesystems across various computing platforms. It has several versions, including FAT12, FAT16, FAT32, and exFAT, each with distinct features and limitations. The FAT filesystem's architecture is straightforward, comprising the boot sector, the FAT region, the root directory, and the data region. The boot sector contains essential metadata about the filesystem, such as type, size, and layout. The FAT region includes one or more copies of the file allocation table, an array of entries mapping the clusters (the minor allocable units of disk space) used by files. The root directory stores entries for files and directories, while the data region holds the actual file and directory contents.

FAT has four different types:

- FAT12: It uses 12-bit entries and supports volumes up to 32 MB in size, with cluster sizes ranging from 512 bytes to 4 KB. It is still used in floppy disks;
- FAT16: it uses 16-bit entries and supports volumes up to 4 GB, with cluster sizes ranging from 2 KB to 32 KB;
- FAT32: it uses 32-bit entries (28 used) and supports volumes up to 2 TB and files up to 4 GB in size;
- exFAT: made for flash drives and supports a max file size of 128 PB.

In terms of advantages, the FAT stack of filesystems is very portable, easy to implement, and, therefore, compatible. It can be used with Windows, Linux, and Mac computers. However, it has no security features, is heavily prone to fragmentation, and lacks journaling capabilities, something that NTFS solved in the Microsoft-based world.

3.2. NTFS

The new technology file system (NTFS), created by Microsoft, is widely used in Windows operating systems due to its strong performance and extensive features. It was introduced in 1993 as a collaboration with IBM [6]. One of NTFS's primary benefits is its ability to support large file sizes and disk volumes, making it ideal for modern storage needs, as data continues to expand. NTFS effectively manages disk space using advanced data structures like the master file table (MFT), which facilitates quick access and efficient file management [6]. Additionally, NTFS enhances security through Access Control Lists (ACLs), allowing administrators to set detailed permissions for individual files and directories [6]. A notable feature of NTFS is its journaling capability, which improves reliability and data recovery after system crashes. By maintaining a log of changes, NTFS ensures data integrity and speeds up recovery from unexpected shutdowns, thereby minimizing data loss. This feature is precious in environments where data integrity is paramount. NTFS also supports file compression, which saves disk space by reducing file sizes without needing third-party software. Its support for symbolic links, hard links, and mount points adds flexibility in managing complex directory structures. However, NTFS has its disadvantages. One major drawback is its limited compatibility. NTFS is optimized for Windows and lacks native support in many non-Windows systems, such as various Linux distributions and macOS [6]. This can be problematic in mixed computing environments where cross-platform file sharing is required. While third-party tools and drivers can provide some level of compatibility, they often do not perform as well as native support. Performance issues can also arise with NTFS under certain conditions. Although it performs well, NTFS can suffer from fragmentation over time, where files are broken into pieces and scattered across the disk. While NTFS includes a defragmentation utility, maintaining optimal performance requires regular maintenance, which can be cumbersome in large-scale environments. The overhead from features like ACLs and journaling can also lead to higher CPU and memory usage compared to simpler filesystems, impacting overall system performance in resource-constrained settings. In performance comparisons, such as a study evaluating EXT4 and NTFS on SSDs, NTFS often falls behind EXT4 in various benchmarks [7]. This performance gap highlights NTFS's limitations in high-performance scenarios, especially with the intensive read–write operations typical in modern applications. Additionally, NTFS's handling of fsync failures—which ensure data are physically written to disk—can lead to data corruption and loss, posing significant risks to data integrity [4]. This issue underscores the need to understand NTFS's specific limitations when using it in critical environments. In forensic investigations, NTFS's detailed metadata can be advantageous for data recovery and analysis. However, the complexity of NTFS structures can also make forensic analysis more challenging compared to simpler filesystems like FAT32 [6]. NTFS is a robust and feature-rich filesystem suitable for various use cases, particularly within Windows environments, where it's often used as a part of DFS (Distributed File System), a vital component of any cloud-scale data processing middleware [8]. Its strengths in security, reliability, and support for large files and volumes are balanced by its compatibility issues, potential performance overhead, and the need for regular maintenance to prevent fragmentation. The decision to use NTFS should be based on the specific requirements and constraints of the deployment environment.

Both FAT and NTFS filesystems are widely available on Windows operating systems. Still, some subtle differences exist when using them with other operating systems, such as, for example, Linux (and macOS, but this is a bit less relevant for our paper). NTFS filesystems are not supported in most Linux distributions out of the box, especially when writing on them, and they require additional modules to be installed and configured.

We choose not to use NTFS in our tests for various reasons. To use a filesystem like NTFS in a virtualized, cloud, or HPC environment, we would have to create a file share, most probably based on SMB (server message block) as NFS (network file system) performance on the Windows Server is very bad. Then, we would have to mount this SMB storage, which will add another layer of abstraction that will, in turn, have additional performance penalties without even considering the high availability. We would have to create a SOFS (scale-out file server) or storage spaces directly to make it highly available, or, more realistically, a file server failover cluster with CSV (cluster shared volume). This significantly impacts how we design our environments and day-to-day operations, as these features have very complicated setups and do not necessarily offer the level of performance we are after. Considering our requirements, we selected btrfs, ZFS, and XFS as part of the test suite.

3.3. Ext4 Filesystem

The evolution of filesystem development followed a distinct path in Unix and Linux. The ext (extended file system) and its subsequent versions, ext2, ext3, and ext4, brought about notable advancements by enhancing efficiency, enabling support for larger files and systems and adding new capabilities, such as journaling and extensible metadata. The latest filesystem from the ext family is ext4, a widely used filesystem in the Linux environment, developed to resolve its predecessor's capacity and scalability issues [9]. It has a bulk of new features when compared to its older versions:

- Extents: Ext4 uses extents instead of the traditional block mapping scheme in ext2 and ext3 [10]. An extent is a contiguous block of storage, which enhances performance and reduces fragmentation by enabling more efficient management of large files;
- Backward compatibility: Ext4 maintains backward compatibility with ext3 and ext2, allowing seamless upgrades. An ext3 filesystem can be mounted as ext4 without reformatting, preserving data and minimizing downtime [11];
- Delayed allocation: this technique defers the allocation of disk blocks until data are written to disk, optimizing file layout, reducing fragmentation, and improving write performance;
- Journaling: Like ext3, ext4 supports journaling, which logs changes before they are committed to the filesystem, enhancing data integrity and aiding in quick crash recovery. Ext4's journaling is more efficient and includes checksums to ensure journal integrity;
- Support for larger files and filesystems: ext4 can handle files up to 16 TB and volumes up to 1 EB, making it suitable for applications requiring substantial storage capacities;
- Online defragmentation: ext4 supports online defragmentation, permitting users to defragment the filesystem while it is in use, maintaining optimal performance without system downtime.

Having our requirements in mind, ext4 has some apparent flaws. Let us list some of them:

- Lack of advanced features: Ext4 lacks advanced capabilities in newer filesystems like BTRFS and ZFS. It does not natively support data deduplication, compression, or integrated volume management, which can be essential for modern storage solutions;
- Scalability limitations: Although ext4 supports large file sizes (up to 16 TB) and volumes (up to 1 EB), it may not scale as efficiently as newer filesystems designed for today's storage demands. This can be a limitation in environments requiring extensive scalability and dynamic storage management;
- No native snapshot support: Ext4 does not offer native support for snapshots, which are crucial for creating point-in-time copies of the filesystem. This feature is available in filesystems like ZFS and BTRFS, making them more suitable for frequent backup and recovery scenarios;
- Fragmentation: Despite having mechanisms to reduce fragmentation, ext4 can still become fragmented over time, especially with heavy usage or in environments with numerous small files. Fragmentation can degrade performance, requiring periodic defragmentation to maintain efficiency, as ext4 does not actively use faster areas freed by deleting files, which can result in a significant performance decline [12];
- Limited self-healing: Unlike filesystems like ZFS, ext4 does not have built-in self-healing capabilities to detect and repair data corruption automatically. This lack means ext4 relies more on manual intervention and external tools for maintaining data integrity;

- No built-in RAID: Ext4 lacks native RAID (redundant array of independent disks) support, requiring users to rely on external software or hardware RAID solutions to implement redundancy and improve performance. ZFS, for example, includes built-in RAID functionalities;
- Journaling overhead: While journaling enhances data integrity and recovery, it introduces performance overhead. This can be a drawback for write-intensive applications compared to non-journaled filesystems or those with more optimized journaling mechanisms;
- Management complexity: Managing ext4 can be more complex, particularly in large volumes or advanced configurations. This complexity can increase administrative overhead compared to more integrated solutions, like those offered by ZFS.

There are other problems with ext4 for the scenarios that we are covering in this paper. Ext4 lacks a lot of features that we are after in this paper, including no caching, no direct data deduplication, and pooling implemented via LVM (logical volume manager), which means that LVM needs to be used from the start (otherwise, we have to re-format the disk). This is why we do not want to use ext4 to store our virtual machines in virtualized environments, cloud environments, or, even worse, HPC environments.

In the future, filesystems in Linux contexts will face new issues, including cloud scalability, data security in a more sensitive cyber landscape, and effective administration of the massive data generated by IoT devices and huge data centers. Further integration of technology, such as artificial intelligence and machine learning, might provide more opportunities for automating and optimizing data-management processes.

Linux also supports various more sophisticated filesystems in addition to the ext filesystems, which are specifically designed to cater to contemporary computer systems' unique requirements. With the increasing demand for handling significant amounts of data and fulfilling more intricate security and reliability standards, sophisticated filesystems such as XFS, btrfs, and ZFS were developed. These systems were specifically engineered to tackle the complexities of contemporary computing environments. Both have sophisticated functionalities, such as integrated volume management, dynamic expansion, snapshots, and self-recovery, making them robust solutions for data management in challenging circumstances.

3.4. XFS

XFS, created by Silicon Graphics, Inc. (SGI, Mountain View, California, USA) in 1993, is a filesystem known for its strong performance, ability to handle massive amounts of data, and efficient operation, especially in large-scale settings. XFS is a 64-bit filesystem that can handle huge files and volumes. It supports filesystems up to eight exabytes (EB) and individual files up to 8 EB, making it well-suited for enterprise-level storage requirements. The system employs an extent-based allocation approach to optimize speed and minimize fragmentation by organizing files into contiguous blocks [9]; B+ trees index file information and directory entries, enabling fast access to files and directories. XFS differs from standard filesystems in that it dynamically allocates inodes as required rather than statically allocating them at creation time. This approach prevents shortages of inodes and provides increased flexibility. The journaling method records changes before finalizing them in the filesystem, guaranteeing data integrity and facilitating rapid recovery from system crashes. XFS is a massively scalable filesystem [13] designed to efficiently handle many input/output activities simultaneously, especially when dealing with small files [11]. It incorporates advanced allocation techniques, such as delayed allocation, which delays the allocation of disk blocks until data are written [10]. This helps optimize the arrangement of files and enhances writing performance. It effectively manages sparse files by avoiding the allocation of disk space for blocks that contain only zeros, resulting in space savings and improved speed. In addition, XFS offers the capability of conducting online defragmentation and scaling. This means administrators can defragment and expand the filesystem without unmounting it, resulting in more flexibility and reduced downtime.

Although XFS offers notable advantages, it also has various restrictions. The filesystem does not possess sophisticated capabilities in more recent filesystems like BTRFS and ZFS, such as integrated data deduplication and compression. XFS lacks built-in snapshot functionality, crucial for producing filesystem copies at certain times. As a result, it may not be the most suitable choice for regular backup and recovery requirements. In addition, it does not have integrated RAID capabilities, which means additional RAID solutions are necessary to achieve redundancy and improve performance. Administering XFS can be intricate, particularly in extensive or extremely dynamic settings, resulting in a higher administrative burden than more unified systems such as ZFS. Although XFS has superior fragmentation management compared to other filesystems, it is nevertheless susceptible to fragmentation over time, especially when subjected to heavy usage or many small files. This can lead to a decline in performance and require defragmentation. Unlike ZFS, XFS lacks inherent tools for automated identification and rectification of data corruption; instead, it depends more heavily on user intervention. The delayed allocation function, although enhancing performance, might occasionally result in unforeseen data loss in the event of system crashes or power shortages. While XFS may be able to accommodate sparse files, its performance in managing them may not be on par with more recent filesystems that are specifically designed to optimize for modern usage patterns. XFS is a robust filesystem well-suited for large-scale environments due to its high performance, scalability, and efficient metadata handling. However, it may not be the best choice for specific high-demand environments that require advanced features, snapshot capabilities, RAID support, and data integrity checking. In such cases, filesystems like BTRFS and ZFS, which offer more extensive features, are more suitable.

3.5. Btrfs

Btrfs, often known as "Butter FS" or "B-tree FS", is a contemporary filesystem designed to meet the increasing demand for data management in Linux environments. Its history and development demonstrate the endeavor to overcome current limitations and offer the sophisticated capabilities required by contemporary computer systems. Btrfs is a modern copy-on-write filesystem primarily used in the Linux operating system [14].

Btrfs was initiated in 2007 by Chris Mason, who was employed at Oracle Corporation then. The primary objective behind the development of btrfs was to design a filesystem that could meet the increasing demands for data storage by providing enhanced performance, improved scalability, and sophisticated functionalities, such as state snapshots and integrated multi-disk management. Btrfs was developed to address the constraints of current filesystems, particularly their ability to scale and adapt.

The development of btrfs was driven by several crucial aims from the beginning. One of the main goals was to enhance scalability, allowing for managing enormous amounts of data and the development of capacity as needed. Furthermore, there has been a notable emphasis on developing sophisticated data-management capabilities. This encompasses using snapshots, cloning, and integrated volume management, giving users increased flexibility and control over their data. The primary objective was to guarantee enhanced dependability and the ability to recuperate autonomously. Btrfs ensures strong data integrity by implementing data-corruption detection and automatic correction mechanisms. Furthermore, transparent data compression has been incorporated, decreasing the overall storage capacity needed and enhancing the efficiency of btrfs in disk utilization.

Btrfs garnered swift attention from the Linux community due to its intriguing features and potential. Throughout the years, efforts have been directed towards enhancing stability, performance, and functionality, leading to integrating btrfs into the primary selection of the Linux kernel in 2009 (Linux kernel 2.6.29). Since its inception, btrfs has undergone continuous development, making it one of the most sophisticated filesystems accessible in a Linux context.

Btrfs was jointly developed, with contributions from multiple developers and corporations, including Oracle, Red Hat, Facebook, and others. The extensive backing has ensured the ongoing expansion and development of btrfs, effectively meeting the requirements of users and organizations.

Btrfs architecture primarily relies on B-tree (balanced tree) structures for storing metadata and some data types. B-trees facilitate efficient operations, such as reading, writing, and data search, in extensive filesystems due to their capacity to uphold a balanced multi-level tree structure. The presence of this structure is crucial for the optimal functioning of btrfs, since it facilitates rapid indexing and data retrieval. Here is an example of the offset and size fields in the btrfs item that indicates where in the leaf the data can be found:

As shown in Figure 1, the btrfs block header includes a checksum for the block content, filesystem UUID for the block owner, and its block number. Btrfs employs the copy-on-write (CoW) technique, which guarantees that when altering data, modified blocks are initially duplicated rather than directly modifying the originals. This method enhances data security and integrity by eliminating partial updates and destruction. Additionally, it enables advanced functionalities like state snapshots and cloning without considerably increasing the required storage space.

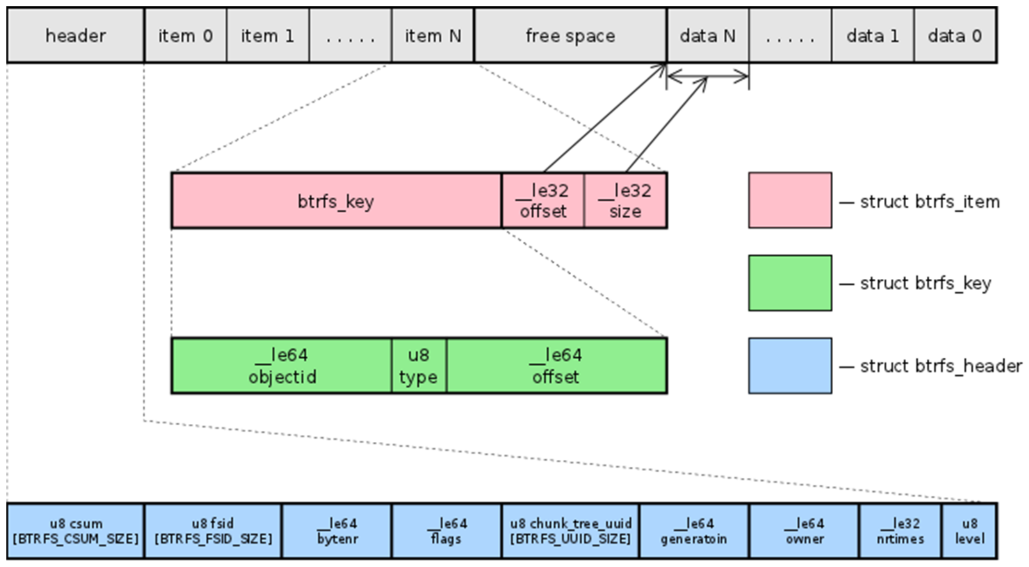

Figure 1. Btrfs architecture, Btrfs design, https://btrfs.readthedocs.io/en/latest/dev/dev-btrfs-design.html, accessed on 2 May 2024.

Btrfs provides built-in volume management and RAID capabilities, allowing users to create and manage numerous data volumes within a single filesystem. The device supports multiple RAID configurations, including RAID 0, 1, 10, 5*, and 6*, allowing users to manage redundancy and performance effectively.

Btrfs utilizes checksums for metadata and data, enabling it to identify and automatically rectify data corruption, guaranteeing a superior level of data integrity. This functionality and RAID support offer a resilient technique for safeguarding data against hardware failures and corruption.

Btrfs enables seamless data compression on a disk, facilitating optimal storage-capacity utilization. Compression can be implemented on the entire filesystem or specific files and directories, allowing the users to enhance storage efficiency.

One distinctive aspect of btrfs is its ability to create subvolumes, conceptually distinct sections inside the same filesystem that may be mounted and managed separately. This allows for sophisticated space and access rule management and streamlined maintenance of backups and snapshots.

The btrfs filesystem is architecturally designed to offer exceptional performance, adaptability, and dependability for contemporary computer systems. Btrfs is an advanced filesystem incorporating B-trees, CoW, integrated volume management, and many capabilities like snapshots, cloning, self-recovery, and transparent compression. These advancements enable btrfs to handle modern computing and data-management requirements effectively.

3.6. ZFS

ZFS, often known as the zettabyte filesystem, is a robust and scalable data-management solution that delivers exceptional performance. Sun Microsystems founded it, and it is currently a division of the Oracle Corporation. The software commenced development in 2001, and the initial open-source iteration was launched in 2005 as a component of the OpenSolaris operating system. The project's objective was to develop a filesystem that could address the issues of scalability, data integrity, and ease of management in the filesystems of that era.

ZFS was designed to achieve extensive scalability, allowing it to handle enormous volumes of data with the ability to scale permits systems to expand seemingly without limitations. ZFS is not only scalable, but it also prioritizes data integrity as a crucial aspect. ZFS guarantees data accuracy by implementing thorough integrity checks and automatic error correction procedures, which remain effective despite physical disk damage or other potential error sources. Furthermore, ZFS is characterized by its inherent capability for effortless administration. The system incorporates sophisticated features, such as dynamic partitioning and space management, significantly streamlining administration and maintenance tasks. ZFS enables system administrators to efficiently manage resources and store them more effectively by utilizing straightforward commands and automating daily operations.

As shown in Figure 2, ZFS architecture is built on top of the idea of storage pooling - a disk or a partition provides capacity pooled to the overall ZFS capacity. ZFS rapidly gained popularity due to its pioneering features and resilient architecture. After Oracle acquired Sun Microsystems, Oracle sustained ZFS's progress. On the other hand, the open-source iteration of ZFS, known as OpenZFS, continued to develop and improve with the help of the open-source community. OpenZFS is the foundation for ZFS implementations on several platforms, such as Linux and FreeBSD. The architecture of the ZFS filesystem serves as the basis for its advanced features and high performance, employing various new technological solutions.

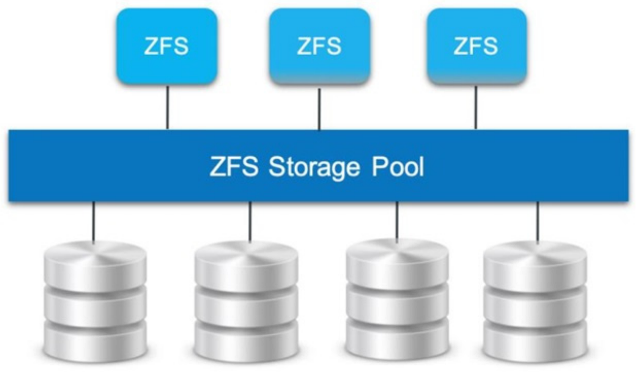

Figure 2. ZFS architecture, Architectural Overview of the Oracle ZFS Storage Appliance, Oracle White Paper, August 2019, https://www.oracle.com/technetwork/server-storage/sun-unified-storage/documentation/o14-001-architecture-overview-zfsa-2099942.pdf, accessed on 2 May 2024.

ZFS integrates filesystem and volume-management (LVM) capabilities into a unified and intricate solution. Using an integrated approach, ZFS efficiently manages storage and data, eliminating the need for distinct layers and tools. ZFS is characterized by its crucial attribute of having a 128-bit address space, equivalent to two raised to the power of 128. ZFS's extensive scalability guarantees that it can accommodate future storage requirements without imposing restrictions on the size of filesystems or individual files.

ZFS employs the copy-on-write method, which avoids directly overwriting the original data when making changes. Alternatively, the data are initially stored in a different location, and once the process is successfully finished, the original data are modified.

ZFS stores a distinct checksum for each block of data, which is saved independently from the data itself. When reading data, ZFS performs an automated checksum verification, guaranteeing its integrity and accuracy. If an error is detected, ZFS can automatically restore the data by utilizing redundant copies. In ZFS, the primary storage unit is a "storage pool" or zpool [15], instead of conventional partitioning and space allocation methods. Users can augment the pool by including additional drives, and ZFS autonomously regulates the storage capacity within the pool, facilitating effortless capacity extension with the addition of new disks. The devices added to the pool are immediately available for use/storage, which occurs transparently to the user [16]. There are also QoS features in ZFS. For example, we can change the configuration of ZFS to prioritize application I/O during RAID recovery, which can mitigate the performance degradation of a declustered RAID [17].

Thanks to its copy-on-write technique, ZFS enables the creation of snapshots and clones without substantial additional storage usage. Snapshots are immutable, whereas copied data are editable, providing efficient choices for backup, archiving, and testing purposes. ZFS enables block-level deduplication and compression, resulting in space efficiency through the storing of distinct data copies and real-time data compression. These qualities are helpful in contexts with a significant amount of repetitive data.

ZFS introduces RAID-Z, an enhanced iteration of conventional RAID setups that addresses some RAID limitations, such as the "RAID hole" issue. RAID-Z provides a significant level of redundancy and ensures data integrity while maintaining optimal performance.

ZFS allows for the transmission and reception of snapshots between ZFS pools, even across remote systems. This capability enhances data replication efficiency and simplifies backup and restore procedures.

ZFS utilizes the ZFS intent log (ZIL) to enhance the efficiency of transaction workloads. It offers a secure means of storing unconfirmed transactions in case of system failure, enabling fast recovery and minimal data loss.

ZFS's architecture embodies outstanding stability, scalability, and efficiency in data management. It sets a high standard for filesystems by combining filesystem and volume management, along with advanced features like CoW, data integrity, snapshots, deduplication, and RAID-Z. It provides a solid foundation for various applications, from data centers to cloud infrastructure and multimedia services. Furthermore, because of its architecture, the added flexibility and security might be more critical in your environment than pure performance [18]. Regarding data protection, data resilvering in ZFS is reactive; data and parity blocks are read, regenerated, and stored after failures are detected [19]. In ZFS terms, resilvering is copying data and calculating its parity between the hard drive of interest and another in a RAID group when such a hard drive has been replaced [20].

4. Comparison of Key Features between btrfs, ZFS, and XFS

The critical characteristics of filesystems can be classified based on various criteria, including performance, reliability, scalability, and data management. Let us systematically evaluate criteria and categorize btrfs, ZFS, and XFS accordingly.

4.1. Performance

When evaluating the performance of btrfs, ZFS, and XFS filesystems, it is crucial to consider multiple factors influencing speed and efficiency. These factors include handling large files, recording and reading speed, and the effects of advanced features like deduplication, compression, and snapshot management.

4.2. Working with Large Files

Because of their copy-on-write architecture, btrfs and XFS are highly versatile and can efficiently manage large files. However, copy-on-write can lead to fragmentation, especially when dealing with intensive workloads, ultimately harming performance over time. ZFS, which also utilizes copy-on-write (CoW), excels in managing large files due to its use of RAID-Z and superior data-management algorithms, with an additional layer of read-and-write caching technologies at its disposal. ZFS has a significant advantage in efficiently managing large volumes of data because of its broad scalability.

4.3. Burn and Read Speed

Btrfs demonstrates excellent performance in various situations, particularly on SSDs, due to its implementation of TRIM and its speedy copy-on-write (CoW) technique. Dynamic space management enables the customization of resource allocation, leading to enhanced recording and reading speed under specific circumstances. ZFS is designed to achieve optimal performance in challenging environments, offering exceptional write and read speeds through its integrated cache (ARC and L2ARC) and log (ZIL) support. ZFS enhances data access with advanced techniques such as prefetching. XFS, known for its robustness and scalability, excels in large-scale environments with its extent-based allocation scheme that reduces fragmentation and improves performance. Its use of B+ trees for indexing metadata allows for rapid access to files and directories, and its journaling mechanism ensures data integrity and quick recovery from crashes. XFS also supports dynamic inode allocation, which prevents inode shortages and offers greater flexibility. Furthermore, XFS's ability to handle parallel I/O operations and its support for online defragmentation and resizing provide additional performance benefits and management flexibility.

4.4. Impact of Deduplication and Compression

Btrfs offers the capability of transparent compression, which can enhance recording performance in certain situations by lowering the volume of data that needs to be written to the disk. Nevertheless, Btrfs lacks the capability for filesystem-level deduplication, which can provide a constraint in environments with substantial redundant data. ZFS provides the capability of deduplication and compression, substantially reducing storage space and improving efficiency. Nevertheless, deduplication in ZFS can be demanding regarding resources and necessitates a substantial amount of RAM to achieve ideal efficiency, impacting the overall system performance in some setups. XFS, on the other hand, does not support native deduplication or compression. However, it compensates with its advanced extent-based allocation scheme, which minimizes fragmentation and optimizes performance. XFS also excels in environments requiring robust scalability and high performance, especially with large files and volumes. Its journaling mechanism ensures data integrity and swift recovery from crashes, and it supports dynamic inode allocation, preventing inode shortages and enhancing flexibility. Additionally, XFS offers features like online defragmentation, resizing, and deduplication but does not support compression.

5. Experimental Setup and Study Methodology

Our testing setup consists of multiple HP ProLiant DL380 Gen10 servers (24-core Xeon CPU, 256 GB of memory). We specifically selected 2U servers, as they offer the best price–performance ratio regarding the potential to add future expansions, such as hard disk or SSD (solid state drive) storage or PCI Express cards. Regarding the storage subsystem, we used a stack of 10.000 rpm Seagate Savio 900 GB disks. The OS disk was on a separate 240 GB SATA SSD.

FIO (flexible IO tester) was used for many of these tests, and configuration files were created for all test scenarios. Here is an excerpt from the FIO configuration file:

[global]
ioengine = libaio; it uses the libaio IO engine for asynchronous input/output.
direct = 1; bypassing the cache of the operating system for I/O operations.

```
size = 10 G; total file size for testing.
directory = /test; the directory where the test files will be created.
runtime = 600; the duration of each job in seconds.
time_based; allows testing based on time instead of file size.
group_reporting; it aggregates reports for all jobs into one.
cpus_allowed_policy = split; divide available CPUs evenly between jobs
[job_1_seq_read]
size = 100 G
numjobs = 1
rw = read; sets the action type to sequential reading.
bs = 1 M; the block size for I/O operations is 1 megabyte.
name = seq_read; job name for identification in the reports.
stonewall; do not start a new job until it is completed.
[job_2_seq_write]
size = 100 G
numjobs = 1
rw = write; sets the action type to sequential writing.
bs = 1 M; the block size for I/O operations is 1 megabyte.
name = seq_write; job name for identification in the reports.
stonewall; do not start a new job until it is completed.
.......
[job_6_db_server]
size = 100 G
rw = randrw; random read and write for database-server load simulation
rwmixread = 50; reading and writing ratio set at 50%
bs = 8 k–64 k; block size range from 8 KiB to 64 KiB for simulation of typical database operations
iodepth = 128; increased queue depth to simulate a large number of parallel queries
name = db_server
numjobs = 1;
stonewall; do not start a new job until it is completed.
```

Fio runs with the previously created configuration file using the following command:

```
fio --eta = always --output = /home/ividec/fio_test_name.txt fio_test.ini
```

Regarding ZFS settings for caching, we had two scenarios—one with and one without. In ZFS, caching can be turned off via simple commands:

```
zfs set primarycache = none poolname
```

or

```
zfs set secondarycache = none pool name
```

Sequential reading refers to the uninterrupted retrieval of data from a disk, typically used for loading huge files or streaming videos. This method of accessing data becomes essential when accessing huge files. Using FIO tools, it was determined that ZFS, when equipped with the integrated cache, exhibits notably superior performance, even while handling extensive data sets. ZFS is well-suited for applications that need fast access to huge files, reducing waiting times and improving overall efficiency.

6. Performance Results for Most Common Use Cases

We divided our tests into six categories: sequential read–write tests, random read–write tests, mixed-workload tests, web-server workload tests, database workload tests, and HPC workload tests. Let us start with sequential read–write tests for bandwidth, IOPS (input/output operations per second), and CPU usage.

6.1. Sequential Performance Tests

Let us first discuss sequential performance tests for bandwidth, IOPS, and CPU usage to see if we can establish some patterns:

The term "sequential" usually refers to when a process retrieves data from storage, typically used for loading huge files like streaming videos, etc. In these tests, as shown in Figure 3, it is visible that ZFS is miles ahead in reading performance while being in the top three in terms of writing performance. Unlike other filesystems, performance can be improved using more memory and faster/lower latency NVMe (non-volatile memory express) SSDs. XFS and btrfs have excellent performance in writing data but are nowhere near the read performance of ZFS. This is especially relevant when discussing that RAIDZ topologies are the most common ZFS use cases, as they offer excellent performance while being sensible with raw capacity. Therefore, if we are looking for the best solution for a heavily read-prone use case with large files, ZFS is a clear winner.

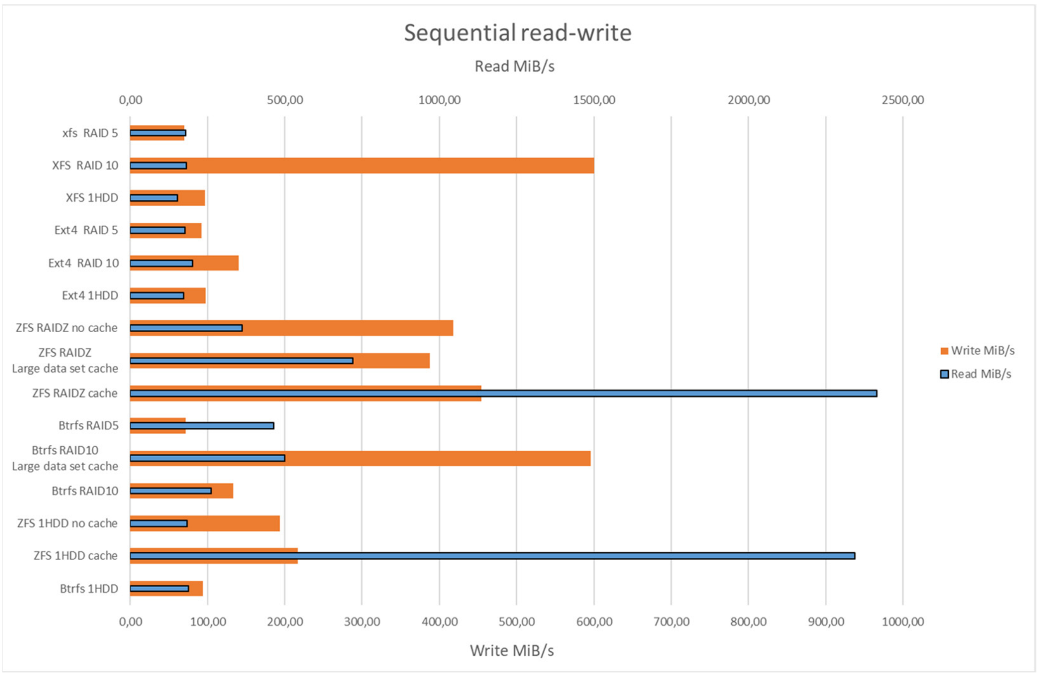

Figure 3. Sequential read–write results for bandwidth.

The next round of tests is related to sequential IOPS measurement for both read and write operations. Let us see what our performance results look like.

Again, as expected, ZFS is far ahead in IOPS tests for read operations and again in the top three for write operations, as Figure 4. clearly shows. XFS and btrfs are again putting up a good fight but still cannot come close to ZFS in read operations. Let us now move on to CPU usage tests.

There are benefits to using btrfs and XFS as storage solutions regarding CPU usage. But that comes at a price, and Figure 5. explains. As seen in the previous two figures, both are much slower than ZFS and have no room to increase their performance, while the ZFS scores could still be improved.

6.2. Random Performance Tests

Random access is a scenario in which data on a disk is accessed non-sequentially. It is present in numerous applications where rapid retrieval of small files can significantly impact performance. So, let us now discuss random performance tests for bandwidth, IOPS, and CPU usage to see if we can learn more about the patterns we noticed while doing the sequential tests.

As shown in Figure 6, ZFS demonstrates outstanding performance regarding available bandwidth for random reads and writes. This further highlights the importance of effective caching methodology when searching for top-level storage performance. Even more impressive is that it displays the same read and write performance.

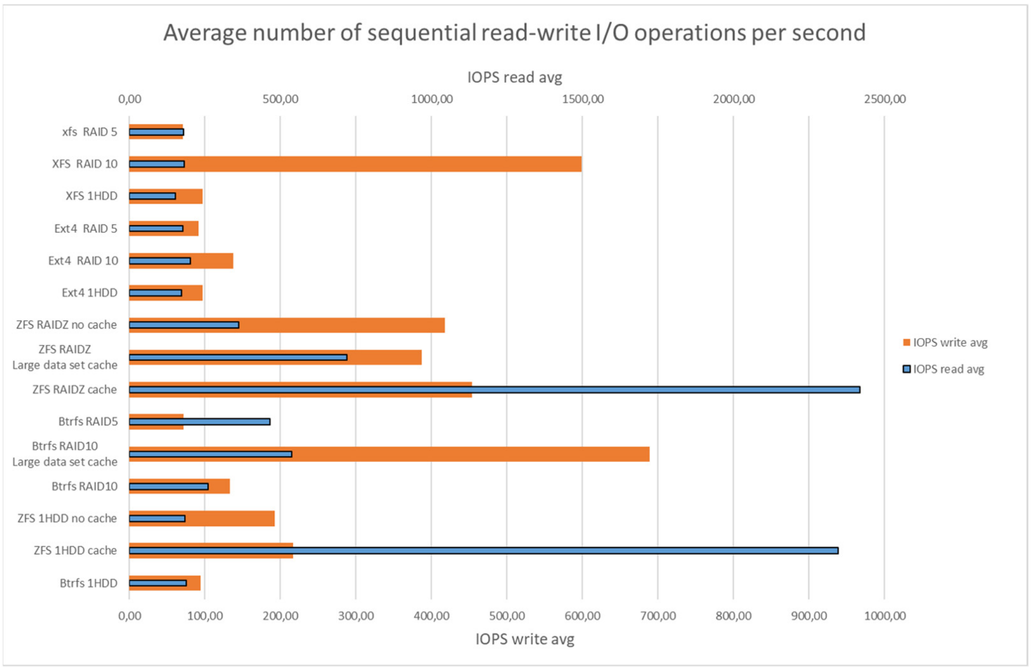

Figure 4. Sequential write patterns for bandwidth and IOPS—read, write, and CPU usage.

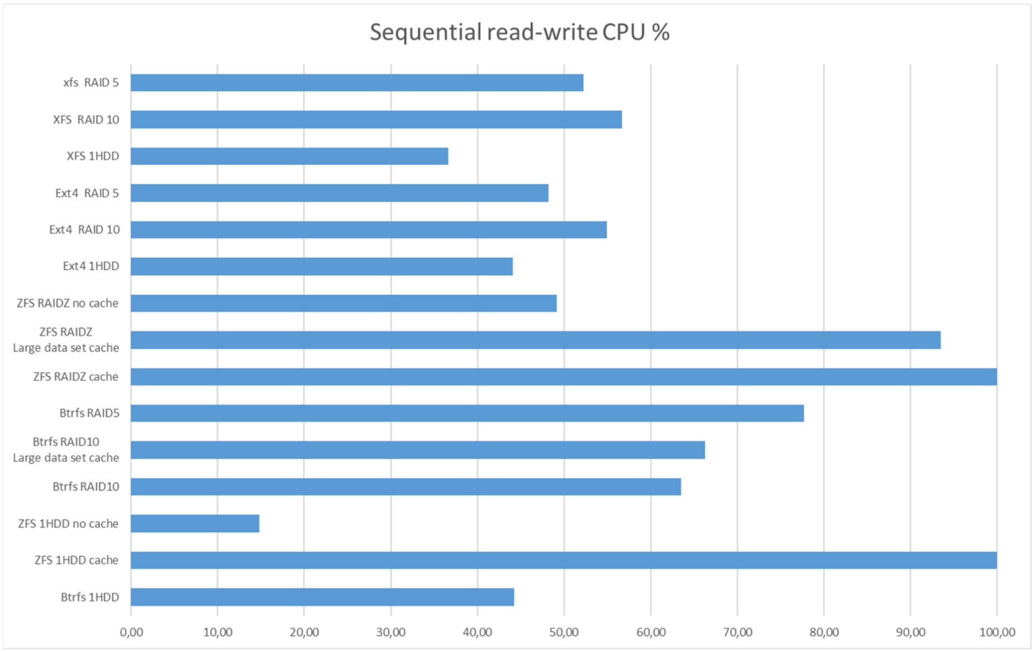

Figure 5. CPU usage for sequential tests.

The next test we need to conduct is to check the IOPS read and write averages for random read and write workloads.

Again, ZFS completely dominates the test, posting five times better scores than the first real competitor, as shown in Figure 7.

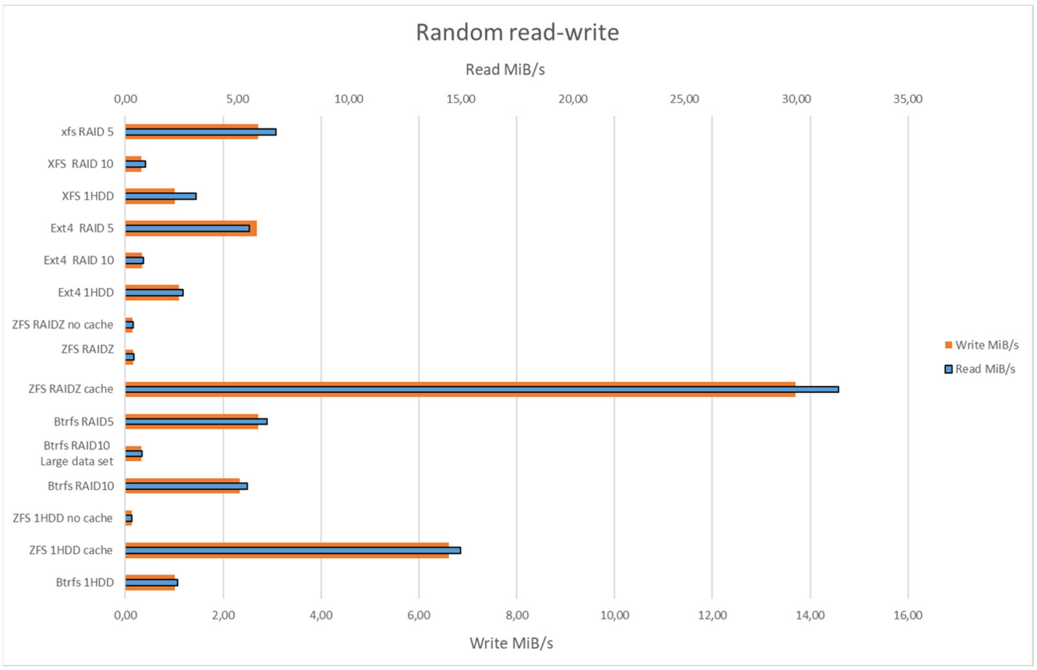

Figure 6. Random read–write bandwidth results.

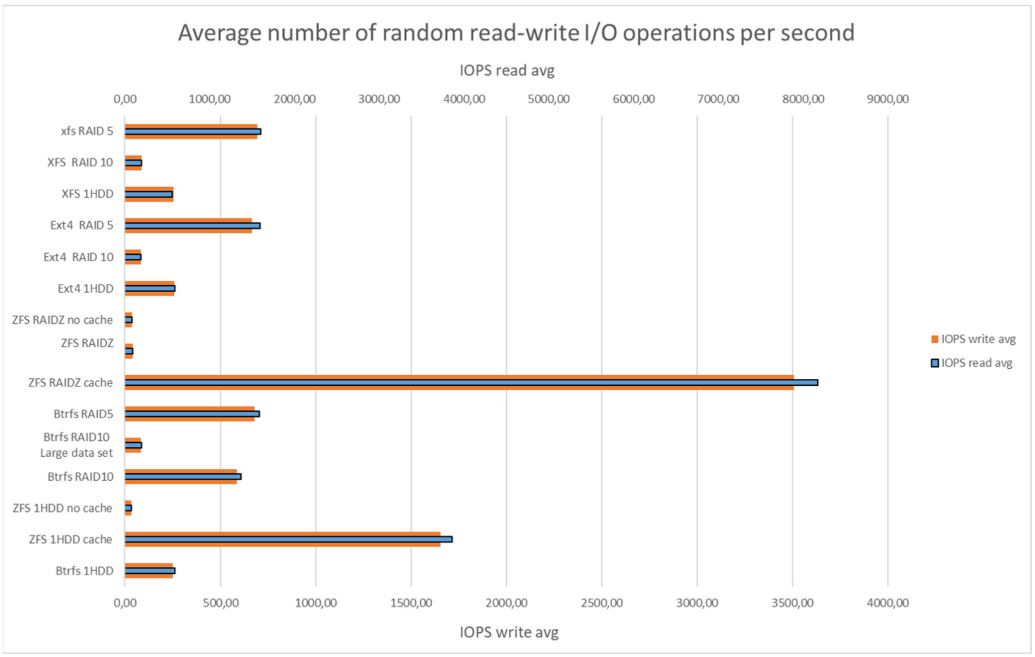

Figure 7. Random read–write IOPS results.

The next round of tests relates to CPU usage when dealing with random workloads, so let us check those results now.

As we can see on Figure 8, we have a similar situation to the sequential scenario. While the CPU usage is higher, so is the performance. This is partly because RAIDZ uses an erasure coding methodology (RAIDZ is very similar to RAID level 5).

Let us see what happens when we test our filesystems using a mixed-load performance test.

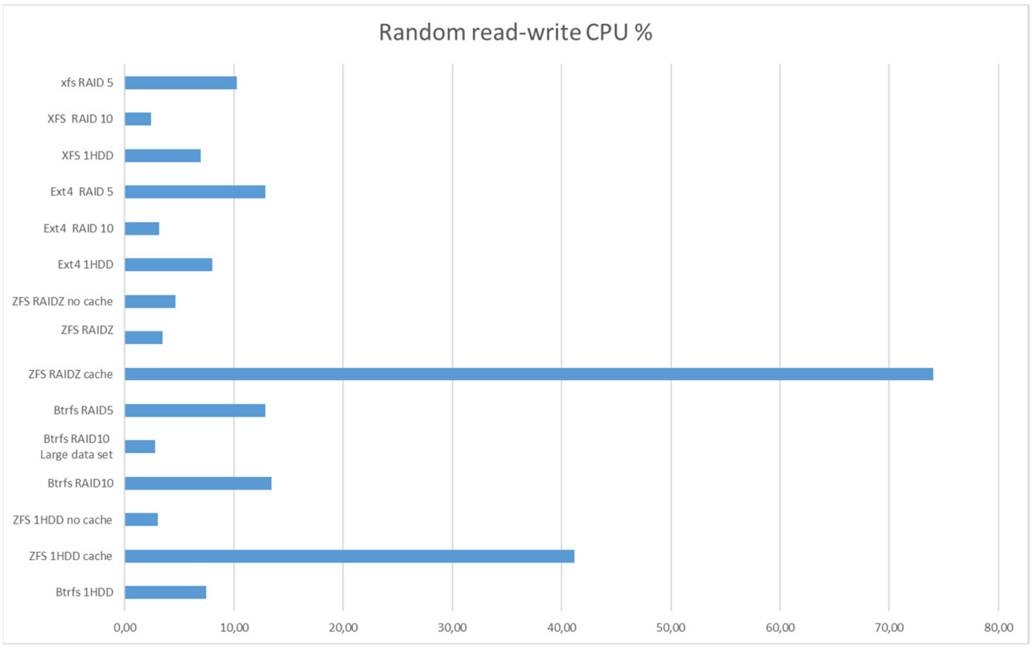

Figure 8. CPU usage for random workload.

6.3. *Mixed-Workloads Performance Tests*

A mixed load is a complex procedure that combines sequential and random read–write operations. It is particularly challenging as it assesses a filesystem's capacity to handle various tasks efficiently. Let us first discuss the results for the bandwidth for our mixed-workload performance test:

Figure 9 shows that ZFS is a clear winner, as this scenario combines sequential and random workloads, which proved to be ZFS's strong suit. It is surprising how vast this gap is, as we still see a massive gap between ZFS and its first competitor.

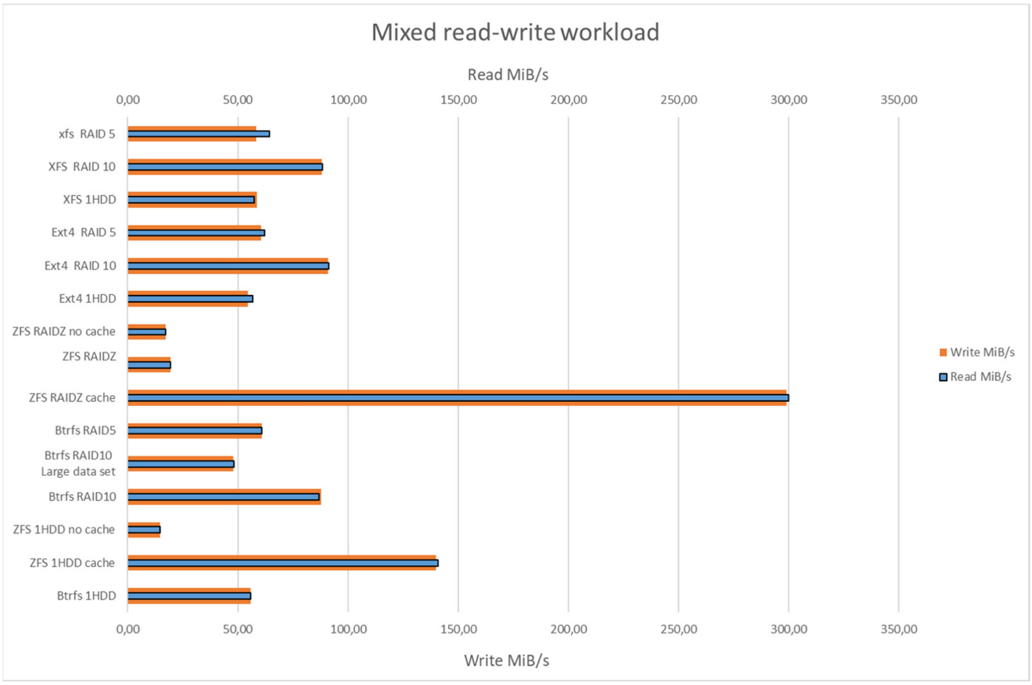

Figure 9. Mixed-workload bandwidth results.

Now let us see the results for the mixed-workload IOPS test.

As shown on Figure 10, ZFS stays ahead of the pack by a considerable margin. Let us check the CPU usage test now.

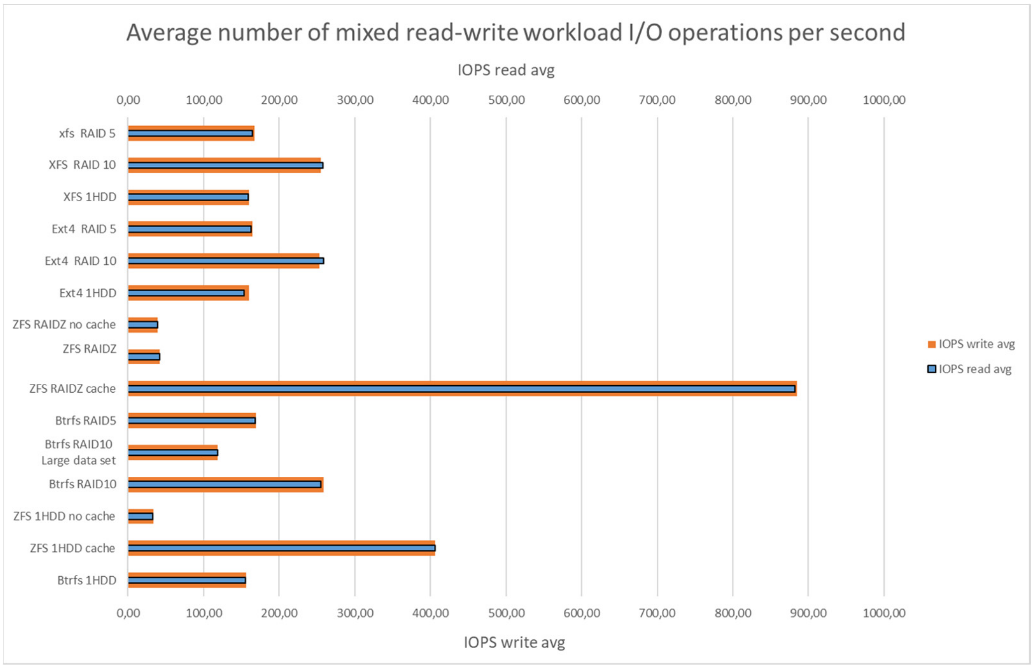

Figure 10. Mixed-workload IOPS performance test.

We still see the same pattern in Figure 11; ZFS keeps using more CPU, which is expected but provides significantly better performance. We look forward to web-server workload tests to see if something changes, as clear patterns emerge from our tests.

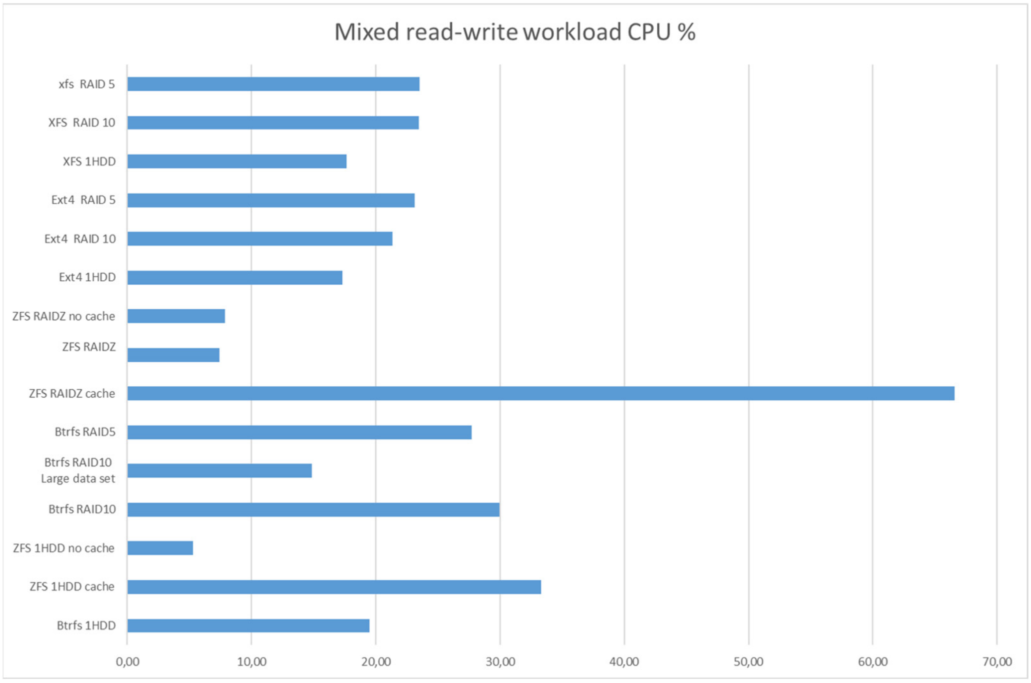

Figure 11. CPU usage for mixed-workload test.

6.4. Web Performance Tests

Web-server load refers to obtaining data that involves multiple concurrent requests for small files. In our scenario, we have set the test to simulate reading and writing to a

10 GB file in 120 s runs or a 100 GB file for a 600 s run with an 80/20 read–write ratio and depth of 64. It is a critical aspect of filesystems in online hosting setups. Let us first check the bandwidth test for our web-server workload.

ZFS still keeps way ahead, although XFS jumped in performance and was close in this scenario, which is visible in Figure 12. Still, it is not good enough to overtake ZFS's performance.

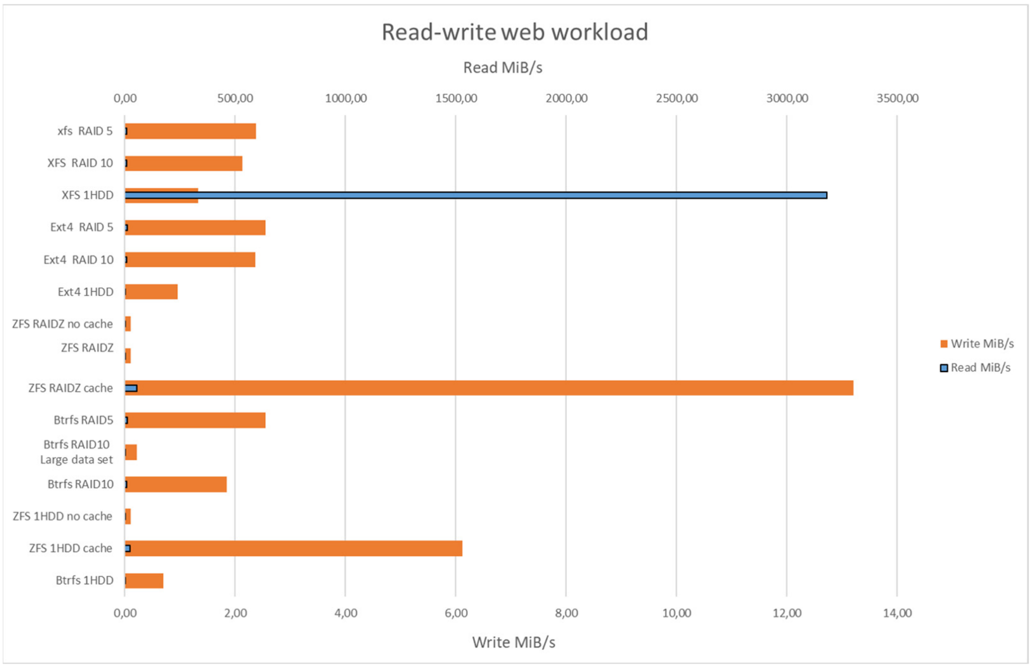

Figure 12. Web workload bandwidth test results.

Let us now check the IOPS test for the web workload test.

Figure 13 shows that in the web-server IOPS test, ZFS is even further ahead than the rest of our tested filesystems, maintaining a healthy 6x plus performance lead. Let us check the CPU usage scores.

When equipped with an improved cache, ZFS delivers markedly superior performance in all tests, barring this one, guaranteeing swift response even during periods of heavy usage. Figure 14. clearly shows this dominance. These findings indicate that ZFS is the most suitable option for web servers that need rapid response times. If we are worried about CPU usage, then btrfs and ext4 seem like a viable option.

The next round of tests concerns database performance, as we expand the tasks we require our storage to perform. Databases are notoriously sensitive to write latency and performance jitters. Let us check which filesystem performs best.

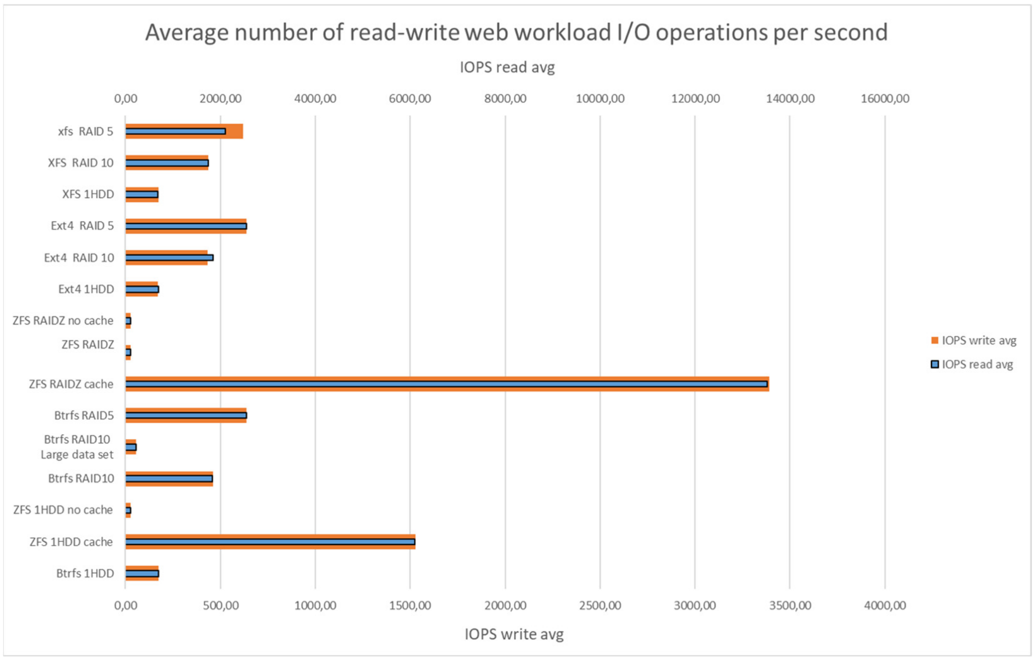

Figure 13. Web-server performance test IOPS scores.

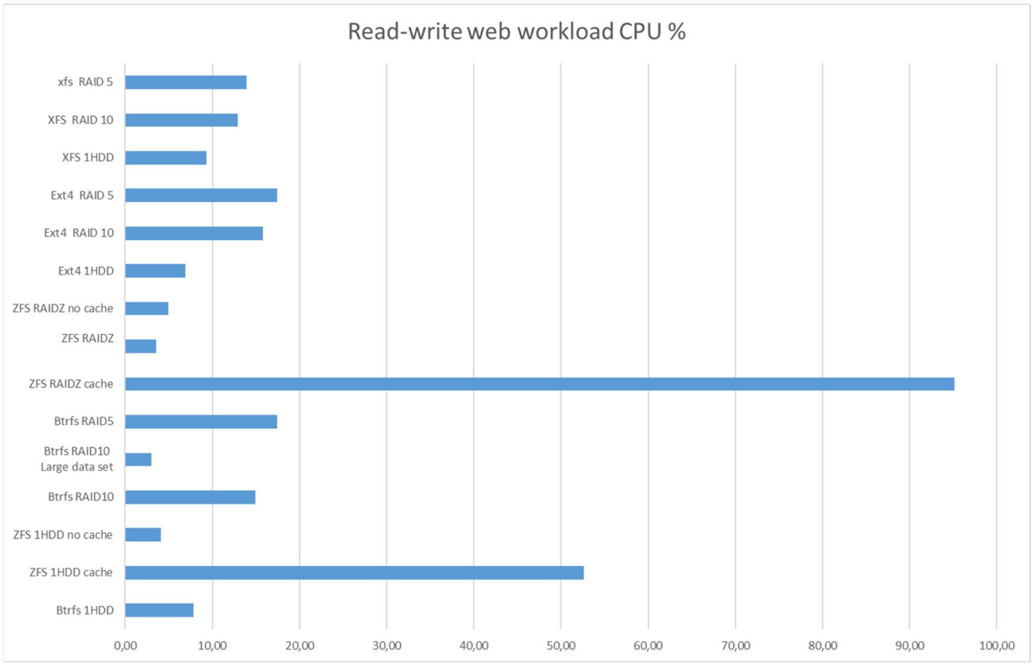

Figure 14. Web performance results in terms of CPU usage.

6.5. DB Performance Tests

The database-server load is characteristic of conventional database servers, where rapid access and data processing are crucial. Let us start with the usual bandwidth test.

As we can conclude from Figure 15, ZFS continues to have the measure of the field, performing roughly twice as fast in terms of read and write bandwidth. Let us check if the same story continues with IOPS measurement.

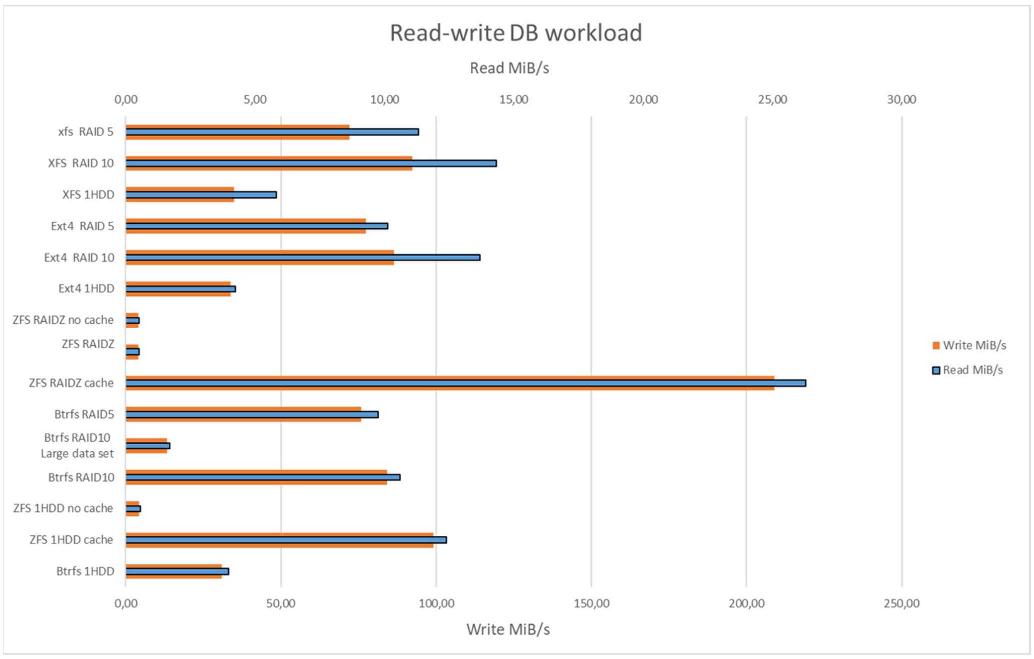

Figure 15. Database bandwidth test.

As Figure 16 shows, ZFS continues to have a significant advantage over any other filesystem. Before we move on to the most critical test related to HPC app performance, let us briefly check the CPU usage characteristics for the DB workload.

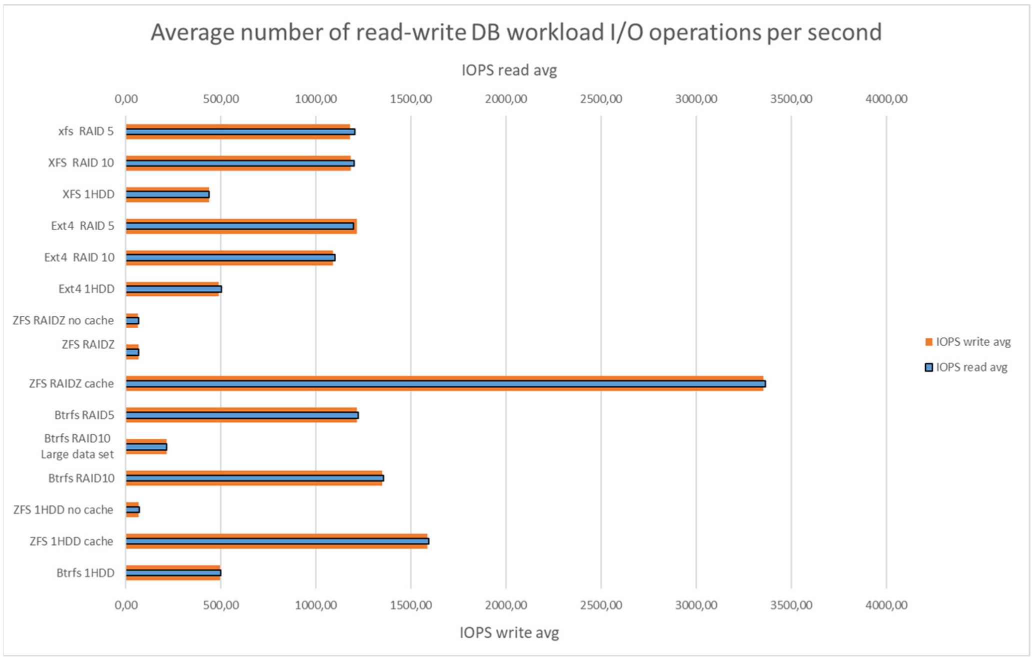

Figure 16. Database performance IOPS test.

ZFS is still the most CPU hungry but still delivers the best performance, which Figure 17. clearly shows. This is just a continuation of the trend we noticed at our tests' start.

Before we move on to the final round of tests, let us summarize what we have found so far. ZFS was a clear winner in all the tests, sometimes 2× and sometimes up to 6× ahead of the competition. If our article's premise is correct, we suspect something similar should happen with the HPC app performance test. Let us check that scenario now.

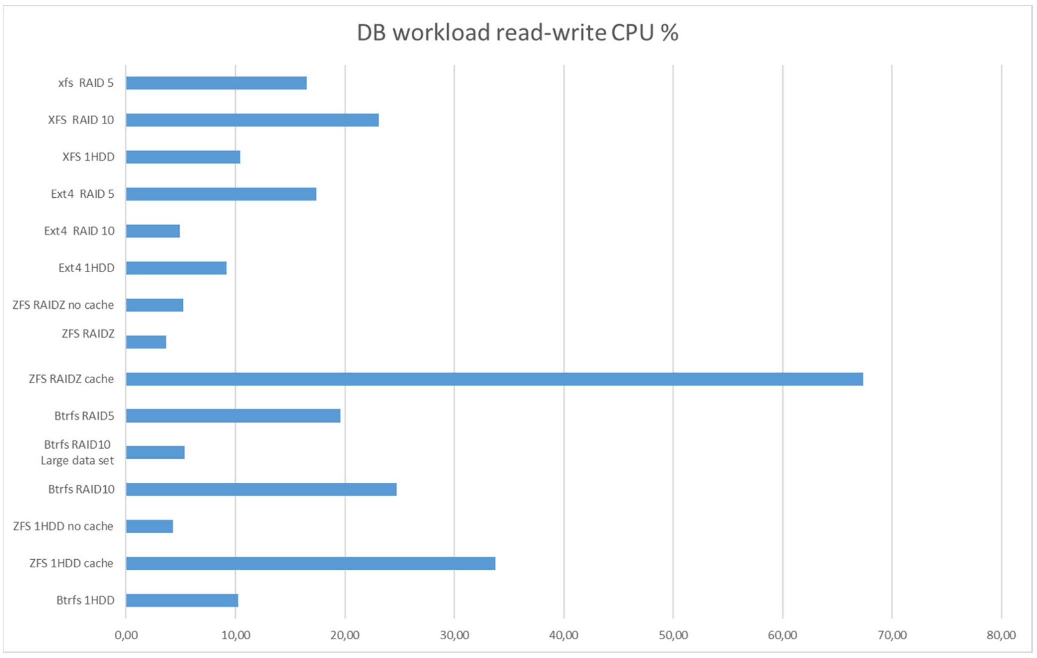

Figure 17. CPU usage for DB performance test.

6.6. HPC Performance Tests

In HPC, the efficiency of each system component is critical for achieving optimal performance. This is particularly true for storage systems within Kubernetes-managed environments. This part of our paper provides an in-depth analysis of storage performance across different filesystems, such as btrfs and ZFS, in the context of a Kubernetes cluster running an HPC application.

The setup includes a Kubernetes cluster spread across 16 physical nodes. Each node contributes to substantial computing resources, collectively providing 256 physical cores and 1 TB of memory. Such a configuration is essential for supporting the application's intensive computational demands and using parallel processing frameworks like OpenMP and MPI. In terms of storage, the data are read and written by using the same storage setup as in the previous tests discussed earlier in this paper.

In this Kubernetes environment, the horizontal pod autoscaling (HPA) feature dynamically scales the number of containers based on computational demand, with some spare computing power reserved for regular OS and Kubernetes operations. The system is configured to scale up to 16 replicas of custom-configured pods, each containing segments of the HPC application. This dynamic scaling is crucial for adapting to the varying load and ensuring efficient resource utilization throughout the performance measurement process. Here are the results for our tests using the same testing parameters as earlier in the paper.

The analysis demonstrates a hierarchy in filesystem performance, with ZFS leading, followed by btrfs and XFS in different configurations, as shown in Figure 18. Let us now check what happens when we conduct a round of IOPS tests for this workload.

ZFS is again setting the standard for how a filesystem should perform by a 2x margin or more. Looking at Figure 19, it's very clear why ZFS would be a great filesystem choice for applications that produce a lot of IOPS.

The last round of tests is related to the HPC app workload CPU usage test. Let us check what our tests tell us.

This is quite surprising; suddenly, the CPU usage characteristics of our filesystems are much closer, especially for more advanced configurations. But, again, as shown on Figure 20, ZFS uses more CPU, and XFS is very near (and we do not mean that positively). But, the performance difference between ZFS and everything else justifies the difference.

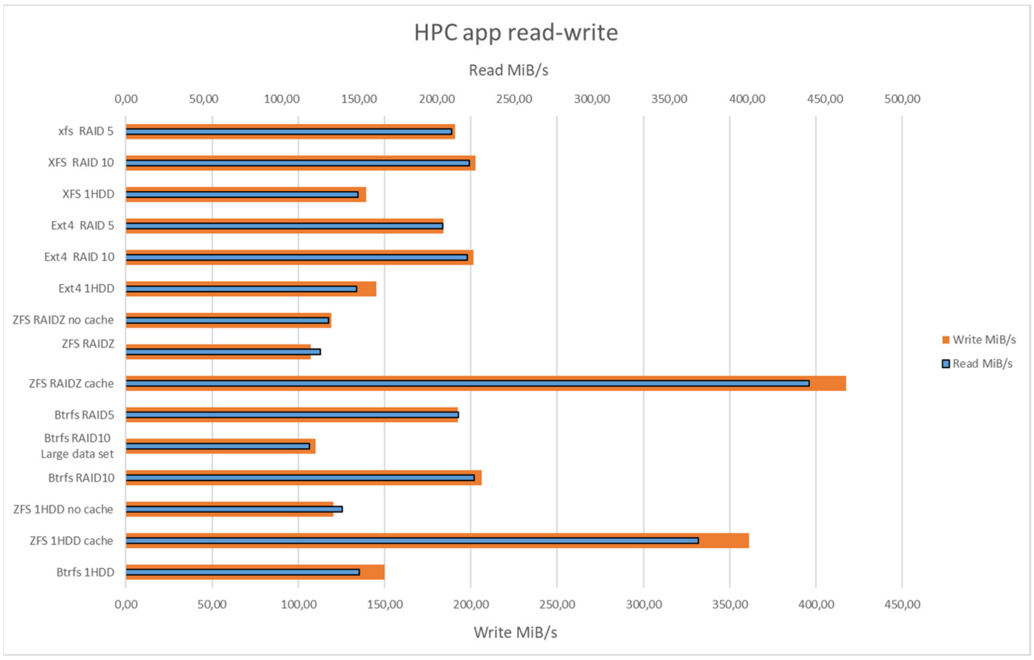

Figure 18. HPC application bandwidth performance measurement across all storage configurations.

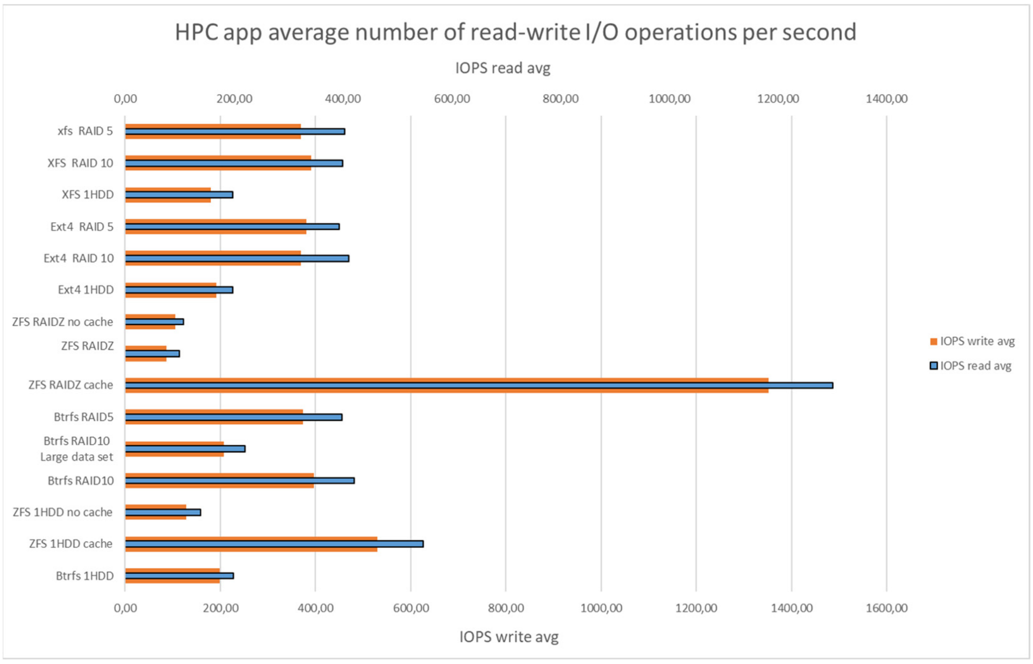

Figure 19. HPC application IOPS performance across filesystems.

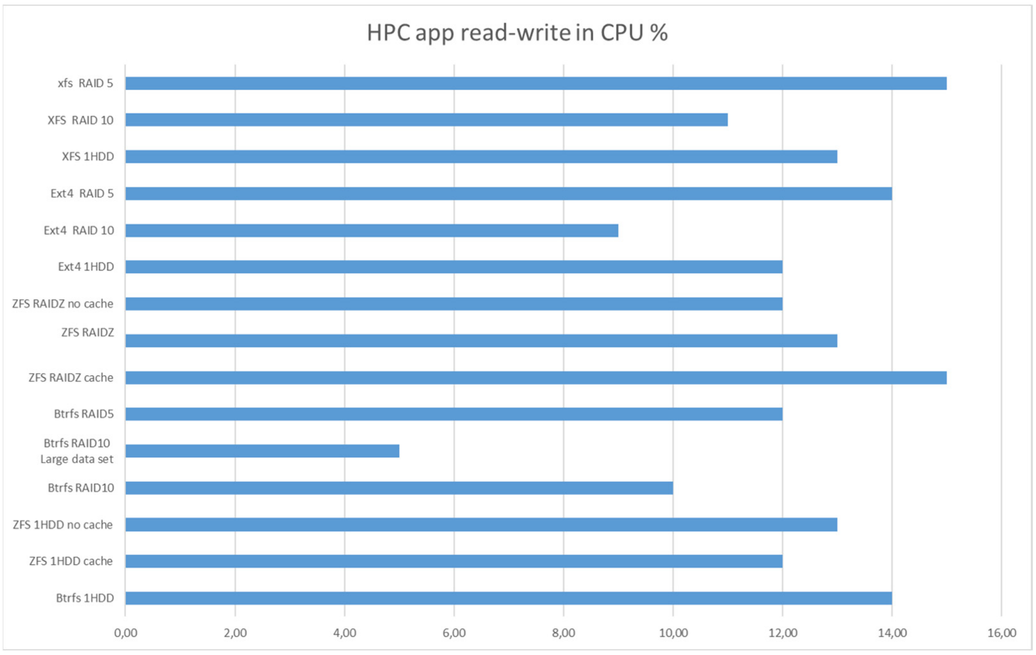

Figure 20. CPU usage for HPC app benchmark.

7. Which Filesystem to Use for HPC Application Managed by Kubernetes

The filesystem selection should be meticulously matched with the needs and prerequisites of the applications it caters to. This research provides a foundation for making educated choices by emphasizing each filesystem's significant performance attributes, benefits, and constraints regarding performance and functionality. It offers users and system administrators the essential information to determine the most suitable option for their requirements.

In Kubernetes deployments that leverage HPC applications, selecting the appropriate filesystem is pivotal for optimizing data throughput, processing speed, and latency, depending on the application. The scalability and robustness of ZFS make it an exceptional choice for applications that demand peak performance and require reliability and data integrity. These qualities are vital in fields that depend on precise scientific calculations and data analysis, where accuracy is paramount.

As mentioned, choosing the proper filesystem for HPC environments should consider data integrity, performance, and scalability factors. For systems like the one described here, ZFS offers superior performance and reliability, making it the preferred choice for demanding HPC applications. Continuous performance monitoring and adjustments, informed by evolving workloads and Kubernetes enhancements, are crucial for maintaining an optimized computational environment. This sort of analysis can have a long-term impact on how we design HPC environments, especially considering that we can build different types of available open-source storage systems for free and scale to incredible capacities while maintaining high storage performance. And if we want to go more towards SDS (software-defined storage), the most viable solution seems to be using Ceph, as it enables us to implement distributed, highly available storage that can be used to host object, file, and block-based storage devices on top of it. We need to be aware that the nature of Ceph + ZFS means that both storage subsystems will use CoW, which is not the best approach in terms of performance, and because network-based replication will always be the primary bottleneck in terms of latency. Depending on the architecture and how Ceph is used, parts of the latency problem can be solved by employing faster network standards and RDMA (remote direct memory access) technologies like ROCE (RDMA over converged ethernet) and by using offload technologies like DPDK (Intel's Data Plane Development Kit). At this point, RDMA is a community-supported feature, while DPDK support should become available when the next Ceph release comes out based on the multi-core Crimson project. Furthermore, other work has shown that there are alternative approaches for combining thread scheduling mechanisms and run-time CPU load monitoring that can improve the I/O performance dynamically [21]. There are some known problems with the interoperability of ZFS with Linux services, such as NSCD (name server caching daemon) [22]. Also, over time, multiple advancements were made in the scientific community regarding data integrity, for example, a concept of flexible end-to-end data integrity [23]. Overall, operational gains can be achieved using that type of architecture, but we will also have different design challenges if we go the Ceph route.

Let us discuss operational gains first:

- Ceph allows us to have an all-in-one solution covering block-based, file-based, and object-based storage, some of which might be required for specific workloads;
- Ceph is easily scalable, offers excellent performance when configured correctly, and can be used cheaply because it works with regular commodity hardware;
- Ceph offers a wide range of options for high availability and fault tolerance, which might be required for many scenarios where resilience is a significant design factor.

In terms of design challenges that might have huge implications on how we design HPC environments:

- Using Ceph might also require different calculations when capacity planning, as having multiple copies of data or objects requires more disks, servers, or both;
- This might, in turn, significantly affect the overall HPC data-center design, as it is an entirely different design scenario when storage high availability and fault tolerance need to be a part of the design equation.

One of the most important conclusions of our paper is related to the design aspect of an HPC data center from a storage perspective when using Ceph-based infrastructure to conduct ZFS vs. ZFS natively. Because Ceph-based infrastructure has the replication capability, which inherently means higher availability and much better data resilience, the data ingestion process for our applications does not have to be a two-stage process, leading to quite a bit less infrastructure size and cost. On the other hand, with ZFS, we do have to pay attention to the fact that we do not have as much availability and resilience, and there will be situations where that might come into question. It is not impossible to conduct ZFS in highly available situations, but it will mostly be burdened by many license costs and quite a bit more configuration. This scenario's initial operational and cost overhead might make it worthwhile to investigate Ceph-based storage infrastructure, albeit with a bit more added latency and performance penalty. We will delve into this in future research, as there will be many additional parameters to tune in this scenario, a process that is not necessarily well documented or researched.

8. Discussion

The filesystem selection should be meticulously matched with the needs and prerequisites of the applications it caters to. This research provides a foundation for making educated choices by emphasizing each filesystem's significant performance attributes, benefits, and constraints regarding performance and functionality. It offers users and system administrators the essential information to determine the most suitable option for their requirements.

Based on the testing, ZFS demonstrated superior performance in multiple tests when configured with sufficient cache memory. Simultaneously, btrfs showed its capacity to offer similar functionalities while consuming substantially fewer resources. ZFS is notable for its high reliability and innovative features, including built-in compression and RAIDZ. Although btrfs is still in development and may need to be more stable, it presents a promising option due to its lower resource consumption. But with the cache capabilities of ZFS, the performance increase that comes with adding RAM, adding faster NVMe SSDs for L2ARC (basically second-level random read cache), and ZIL (synchronous write cache) offer an excellent possibility for a huge performance increase.

9. Conclusions

ZFS has several benefits, including its long-lasting nature, strong resilience, and extensive sophisticated features. Nevertheless, these advantages are accompanied by increased resource utilization. Btrfs exhibits reduced resource use yet encounters challenges related to stability and long-term advancement.

When evaluating their utilization, ZFS is a superior option for high-criticality systems where the utmost durability and robustness are necessary. Btrfs is a favorable choice for systems that are not as important yet prioritize the efficient use of resources.

This research guides selecting between btrfs and ZFS, emphasizing the importance of comprehending their distinct characteristics and constraints to meet the diverse requirements of the IT environment. Choosing a filesystem necessitates a well-rounded evaluation of technical and operational factors to maximize performance and efficiency. This study affirms the significance of conducting a thorough assessment and customization of technical solutions to meet individual requirements, confirming that making the appropriate selection can significantly enhance the success and reliability of IT systems. It also puts the onus on the correct design process, a part of the overall process that has often been conducted the wrong way, which, in turn, means that we do not fully exploit the capabilities of our infrastructure. We will delve into this topic in future research, as we see a real possibility for significant changes to the design of future HPC data centers on the horizon.

Author Contributions: Conceptualization, V.D. and M.K.; methodology, V.D. and M.K.; software, V.D.; validation, I.V.; formal analysis, I.V.; investigation, I.V.; resources, V.D.; data curation, V.D.; writing—original draft preparation, V.D. and M.K.; writing—review and editing, V.D.; visualization, I.V.; supervision, M.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The raw data of the experiments can be requested from the authors.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ab Karim, M.B.; Luke, J.-Y.; Wong, M.-T.; Sian, P.-Y.; Hong, O. Ext4, XFS, BtrFS and ZFS Linux File Systems on RADOS Block Devices (RBD): I/O Performance, Flexibility and Ease of Use Comparisons. In Proceedings of the 2016 IEEE Conference on Open Systems (ICOS), Langkawi Island, Malaysia, 10–12 October 2016.
2. Gurjar, D.; Kumbhar, S.S. File I/O Performance Analysis of ZFS & BTRFS over iSCSI on a Storage Pool of Flash Drives. In Proceedings of the 2019 International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 17–19 July 2019.
3. Gurjar, D.; Kumbhar, S.S. A Review on Performance Analysis of ZFS & BTRFS. In Proceedings of the 2019 International Conference on Communication and Signal Processing (ICCSP), Melmaruvathur, India, 4–6 April 2019.
4. Rebello, A.; Patel, Y.; Alagappan, R.; Arpaci-Dusseau, A.C.; Arpaci-Dusseau, R.H. Can Applications Recover from fsync Failures? *ACM Trans. Storage (TOS)* **2021**, *17*, 1–30. [CrossRef]
5. Reddy, B.I. Comparative Analysis of Distributed File Systems. *Int. J. Res. Pap. Publ.* **2021**, *9*, 20–26. [CrossRef]
6. Hermon, R.; Singh, U.; Singh, B. NTFS: Introduction and Analysis from Forensics Point of View. In Proceedings of the 2023 International Conference for Advancement in Technology (ICONAT), Goa, India, 24–26 January 2023.
7. Sterniczuk, B. Comparison of EXT4 and NTFS Filesystem Performance. *JCSI* **2022**, *25*, 297–300. [CrossRef]
8. Wu, Y.; Ye, F.; Chen, K.; Zheng, W. Modeling of Distributed File Systems for Practical Performance Analysis. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 156–166. [CrossRef]
9. Đorđević, B.; Timčenko, V.; Maček, N.; Bogdanoski, M. Performance Modeling of File System Pairs in Hypervisor-Based Virtual Environment Applied on KVM Hypervisor Case Study. *J. Comput. Forensic Sci.* **2022**, *1*, 55–75. [CrossRef]
10. Ramadan, A. A Dockers Storage Performance Evaluation: Impact of Backing File Systems. *J. Intell. Syst. Internet Things* **2021**, *3*, 8–17. [CrossRef]
11. Dordevic, B.; Timcenko, V. A performance comparison of Linux filesystems in hypervisor type 2 virtualized environment. In Proceedings of the INFOTEH-JAHORINA, Sarajevo, Bosnia and Herzegovina, 22 March 2017; Volume 17, pp. 630–634.

12. Nakagami, M.; Fortes, J.A.B.; Yamaguchi, S. Performance Improvement of Hadoop Ext4-Based Disk I/O. In Proceedings of the 2020 Eighth International Symposium on Computing and Networking (CANDAR), Naha, Japan, 24–27 November 2020.
13. Pesic, D.; Djordjevic, B.; Timcenko, V. Competition of Virtualized Ext4, Xfs and Btrfs Filesystems under Type-2 Hypervisor. In Proceedings of the 2016 24th Telecommunications Forum (TELFOR), Belgrade, Serbia, 22–26 November 2016.
14. Hilgert, J.-N.; Lambertz, M.; Yang, S. Forensic Analysis of Multiple Device BTRFS Configurations Using The Sleuth Kit. *Digit. Investig.* **2018**, *26*, S21–S29. [CrossRef]
15. Vaidya, M.; Deshpande, S. Comparative Analysis of Various Distributed File Systems & Performance Evaluation Using Map Reduce Implementation. In Proceedings of the 2016 International Conference on Recent Advances and Innovations in Engineering (ICRAIE), Jaipur, India, 23–25 December 2016.
16. Beebe, N.L.; Stacy, S.D.; Stuckey, D. Digital Forensic Implications of ZFS. *Digit. Investig.* **2009**, *6*, S99–S107. [CrossRef]
17. Qiao, Z.; Liang, S.; Chen, H.-B.; Fu, S.; Settlemyer, B. Exploring Declustered Software RAID for Enhanced Reliability and Recovery Performance in HPC Storage Systems. In Proceedings of the 2019 38th Symposium on Reliable Distributed Systems (SRDS), Lyon, France, 1–4 October 2019.
18. Pugh, C.; Carrol, T.; Henderson, P. Ensuring High Availability and Recoverability of Acquired Data. In Proceedings of the 2011 IEEE/NPSS 24th Symposium on Fusion Engineering, Chicago, IL, USA, 26–30 June 2011.
19. Qiao, Z.; Hochstetler, J.; Liang, S.; Fu, S.; Chen, H.; Settlemyer, B. Incorporate Proactive Data Protection in ZFS Towards Reliable Storage Systems. In Proceedings of the 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), Athens, Greece, 12–15 August 2018.
20. Widianto, E.D.; Prasetijo, A.B.; Ghufroni, A. On the Implementation of ZFS (Zettabyte File System) Storage System. In Proceedings of the 2016 3rd International Conference on Information Technology, Computer and Electrical Engineering (ICITACEE), Semarang, Indonesia, 19–20 October 2016.
21. Bang, J.; Kim, C.; Byun, E.-K.; Sung, H.; Lee, J.; Eom, H. Accelerating I/O Performance of ZFS-Based Lustre File System in HPC Environment. *J. Supercomput.* **2022**, *79*, 7665–7691. [CrossRef]
22. Pugh, C.; Henderson, P.; Silber, K.; Carroll, T.; Ying, K. Utilizing Zfs for the Storage of Acquired Data. In Proceedings of the 2009 23rd IEEE/NPSS Symposium on Fusion Engineering, San Diego, CA, USA, 1–5 June 2009.
23. Zhang, Y.; Myers, D.S.; Arpaci-Dusseau, A.C.; Arpaci-Dusseau, R.H. Zettabyte Reliability with Flexible End-to-End Data Integrity. In Proceedings of the 2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST), Long Beach, CA, USA, 6–10 May 2013.