Performance Evaluation of Ext4, XFS, and Btrfs Filesystems

Jianqiu Wu
School of Data Science
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
225040310@link.cuhk.edu.cn

Aobo Guo
School of Data Science
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
225040143@link.cuhk.edu.cn

***Abstract*—This paper evaluates the performance of three Linux filesystems: Ext4, XFS, and Btrfs. Using fio, sysbench, and dd, we measured sequential and random input and output behavior under baseline, video-editing, mail-server, direct sequential-copy, and mount-option tuning workloads. The fio baseline and dd sanity checks used direct input and output, whereas the sysbench mail-server approximation used buffered input and output with frequent synchronization operations. The results show that Ext4 achieved the highest random-write performance among the baseline tests, XFS achieved the highest sequential-write throughput and random-read IOPS, and Btrfs provided advanced copy-on-write functionality at a measurable random-write cost in this environment. Mount-option tuning produced workload-dependent effects; therefore, crash-safety-reducing options should not be treated as production recommendations based on these measurements.**

***Index Terms*—filesystem, Ext4, XFS, Btrfs, benchmark, I/O operations per second (IOPS), latency, performance evaluation**

I. Introduction

A. Background and Motivation

Filesystem performance is a critical factor in determining overall system efficiency, directly impacting application responsiveness, throughput, and user experience. As data-intensive applications continue to proliferate, the choice of filesystem becomes increasingly important for system administrators and developers alike. Linux offers multiple filesystem options, each designed with different trade-offs between performance, reliability, and features. Understanding these trade-offs through systematic benchmarking is essential for making informed deployment decisions.

Ext4, XFS, and Btrfs represent three distinct design philosophies in Linux filesystem development [1]–[5]. Ext4, as the successor to Ext3, emphasizes backward compatibility and stability. XFS, originally developed by Silicon Graphics, Inc. (SGI), focuses on high-performance concurrent operations. Btrfs introduces modern features such as Copy-on-Write (CoW) and snapshots but with different performance characteristics. This study aims to provide quantitative performance comparisons across these filesystems under representative workload patterns.

B. Project Objectives

This experiment was designed to achieve the following objectives:

1) Establish baseline performance metrics, including throughput and input/output (I/O) operations per second (IOPS), for Ext4, XFS, and Btrfs across sequential read, sequential write, random read, and random write workloads.
2) Evaluate filesystem performance under video-editing scenarios characterized by large-file sequential I/O with high concurrency.
3) Assess filesystem behavior under mail-server workloads featuring small-file random synchronous writes and metadata operations.
4) Investigate the performance impact of common mount-option tuning strategies including `noatime`, `data=writeback`, and `barrier=0`.

C. Report Structure

The remainder of this paper is organized as follows. Section II provides background information on the evaluated filesystems and benchmarking tools. Section III describes the experimental methodology, including the test environment, workload design, and tuning parameters. Section IV presents the experimental results. Section V discusses performance trade-offs and methodology considerations. Section VI concludes with key findings and recommendations.

II. Background

A. Filesystem Overview

1) Ext4: The Stable Workhorse: Ext4 (Fourth Extended Filesystem) is the successor to Ext3 and has been the default filesystem for many Linux distributions. Key characteristics include the following.

Journaling: Ext4 maintains a journal that records metadata operations before they are committed, enabling fast recovery after system crashes.

Extent Allocation: Unlike the traditional block mapping scheme, Ext4 uses extents—contiguous ranges of blocks—which significantly improves large-file performance and reduces metadata overhead.

Backward Compatibility: Ext4 maintains compatibility with Ext2 and Ext3, allowing migration from older filesystems.

Direct I/O Path: Ext4 provides a relatively straightforward overwrite path for random writes, which can be advantageous for certain workload types.

2) XFS: High-Performance Enterprise Choice: XFS was originally developed by Silicon Graphics for the IRIX operating system and later ported to Linux. Its architectural highlights include the following.

Allocation Groups (AGs): XFS divides the filesystem into multiple allocation groups, each operating independently. This design enables parallel operations, as different threads can work in different AGs with reduced contention.

B+ Tree Metadata: XFS uses B+ trees extensively for free-space management and inode indexing, providing efficient lookup even for large filesystems with many files.

Delayed Allocation: XFS employs delayed allocation, which allows the filesystem to aggregate small writes and optimize disk placement before committing data to media.

Scalability: XFS is designed for large filesystems and high-concurrency workloads, making it common in enterprise storage environments.

3) Btrfs: The Modern Copy-on-Write Filesystem: Btrfs (B-tree file system) represents a modern approach to filesystem design, introducing several features.

Copy-on-Write (CoW): When modifying existing data, Btrfs writes the new data to free space and updates pointers, keeping the original data intact until the operation completes. This enables efficient snapshots and zero-copy cloning.

Data Integrity: Btrfs stores checksums for both data and metadata, enabling detection and, under suitable redundancy configurations, correction of silent data corruption.

Flexible RAID: Btrfs provides native support for several Redundant Array of Independent Disks (RAID) configurations with integrated redundancy management.

Performance Implications: The CoW mechanism can introduce write amplification for random writes, as each modification may require updates to multiple B-tree nodes along the write path.

B. Benchmark Tools

1) `fio`: Flexible I/O Tester: `fio` is a versatile and widely used I/O benchmarking tool that supports numerous I/O engines and workload patterns [6]. Key features include:

- support for synchronous and asynchronous I/O engines, including `libaio`, `io_uring`, and `sync`;
- configurable block sizes, queue depths, and job counts;
- JavaScript Object Notation (JSON) output for automated parsing;
- percentile latency reporting, including P50, P95, and P99;
- direct-I/O support for bypassing the page cache.

2) `sysbench`: `sysbench` is a multi-threaded benchmark tool that includes file-I/O testing capabilities [7]. For our mail-server approximation, we used:

- small-block random read/write operations;
- configurable `fsync` frequency for durability testing;
- latency and throughput reporting;
- multi-threaded concurrent access simulation.

3) `dd`: Direct Sequential Copy Sanity Check: `dd` is a Unix utility for copying and converting data. Although it is not a comprehensive benchmark tool, it provides:

- simple sequential I/O testing;
- direct-I/O capability via flags;
- a baseline sanity check for filesystem throughput;
- low-overhead measurement suitable for quick validation.

III. Experimental Methodology

A. Test Environment

The experiments were conducted on a virtualized Ubuntu 22.04 environment with the specifications listed in Table I.

TABLE I
TEST ENVIRONMENT USED FOR ALL LOOPBACK-BASED FILESYSTEM BENCHMARKS.

Component	Specification
Operating System	Ubuntu 22.04 (Linux 6.8.0)
CPU Cores	4 virtual cores
Memory	8 GB RAM
Disk Configuration	3 separate loopback images (10 GB each)
Filesystems	Ext4, XFS, Btrfs
`fio` Version	3.28
`sysbench` Version	1.0.20

B. Disk Configuration

Three separate loopback image files were created and formatted with different filesystems, as shown in Table II.

TABLE II
LOOPBACK IMAGE AND FILESYSTEM ASSIGNMENT USED IN ALL EXPERIMENTS.

Device	Filesystem	Mount Point
`/dev/loop0`	Ext4	`/mnt/ext4_test`
`/dev/loop1`	XFS	`/mnt/xfs_test`
`/dev/loop2`	Btrfs	`/mnt/btrfs_test`

Each filesystem was mounted with default options for baseline testing and remounted with specific options for tuning experiments.

C. `fio` Baseline Tests

The baseline `fio` tests evaluated four fundamental I/O patterns, summarized in Table III.

TABLE III
BASELINE `fio` WORKLOAD CONFIGURATIONS; EACH WORKLOAD RAN FOR 30 S WITH GROUP REPORTING ENABLED.

Workload	Pattern	Block Size	I/O Depth	Direct I/O
`seq_read`	Seq. Read	1 MB	16	Yes
`seq_write`	Seq. Write	1 MB	16	Yes
`rand_read`	Rand. Read	4 KB	16	Yes
`rand_write`	Rand. Write	4 KB	16	Yes

Each workload was executed for 30 s with a time-based constraint and group reporting enabled. When repeated runs were available, the table reports the representative aggregate values and the figures show the corresponding run-level latency behavior. The 50th, 95th, and 99th percentile latencies (P50, P95, and P99) were recorded for each test.

D. Video-Editor Scenario

The video-editor scenario simulated a workload characterized by large-file sequential I/O with high concurrency:

- **Block Size:** 1 MB
- **I/O Depth:** 32
- **Number of Jobs:** 4 concurrent jobs
- **Runtime:** 45 s
- **Test Size:** 4 GB per job

This configuration models a multi-track video-editing workflow where multiple streams are read or written simultaneously.

E. Mail-Server Scenario

The mail-server approximation used `sysbench` with parameters scaled for the virtualized environment:

- **File Count:** 10,000 files
- **Total Size:** 40,000 KB
- **Block Size:** 4 KB
- **Test Mode:** random read/write (`rndrw`)
- **`fsync` Frequency:** `--file-fsync-freq=1`, which forces `fsync` after every write
- **Threads:** 4
- **Runtime:** 45 s

The `--file-fsync-freq=1` parameter forces a synchronous flush after each write, testing filesystem behavior under heavy metadata and durability pressure.

F. `dd` Sanity Check

The `dd` sanity test was conducted with `bs=1M`, `count=1024`, and `oflag=direct` for writes, and with `iflag=direct` for reads, transferring approximately 1 GB per direction on each filesystem.

G. Tuning Experiments

Three categories of tuning were evaluated:

1) **`noatime`:** eliminates access-time updates on file reads, reducing unnecessary metadata writes.
2) **`data=writeback` (Ext4 only):** relaxes ordering constraints on data blocks relative to metadata.
3) **`barrier=0` (Ext4 only):** disables write barriers for potential performance gains at the cost of crash safety.

The XFS `nobarrier` option has been deprecated in modern kernel versions in favor of hardware-level Force Unit Access (FUA) and cache-flush support, and was therefore not tested.

IV. Results

A. Baseline Performance

Table IV presents the complete baseline performance results across all filesystems and workloads. Fig. 1, Fig. 2, and Fig. 3 visualize the same data set from three complementary perspectives.

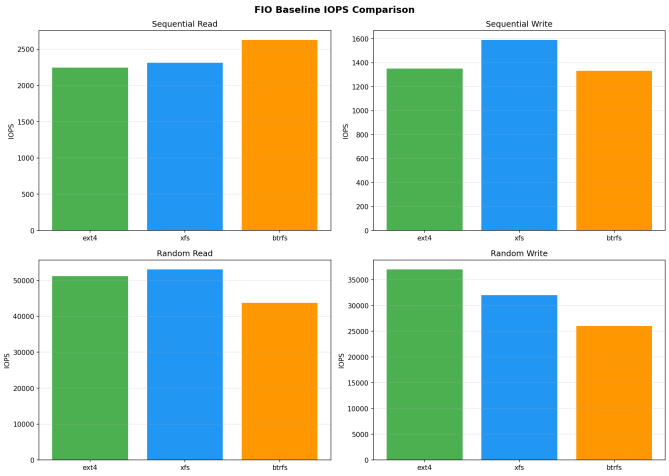

Fig. 1. IOPS comparison of Ext4, XFS, and Btrfs across the four baseline workloads; higher values are better.

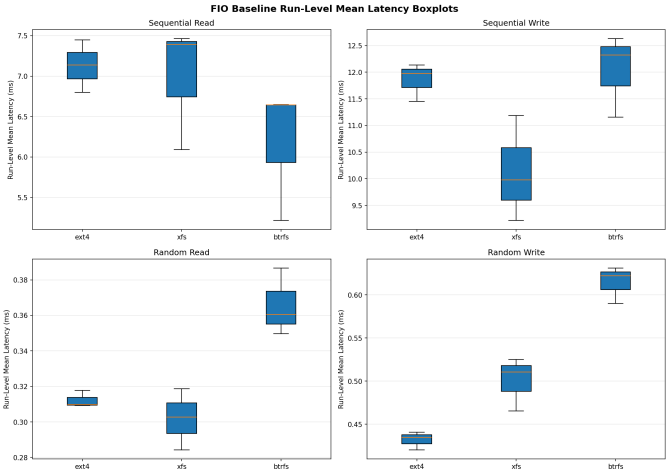

Fig. 2. Run-level mean-latency distribution in milliseconds for each filesystem–workload pair in the baseline `fio` tests; lower values are better.

Key observations from baseline results:

- *Random-write performance:* Ext4 achieved the highest random-write IOPS (36,996), followed by XFS (32,011) and Btrfs (26,024). The 30% performance gap between Ext4 and Btrfs, using Ext4 as the denominator, is consistent with the additional write-path overhead expected from Btrfs's CoW mechanism.
- *Random-read performance:* XFS led random-read operations with 53,021 IOPS, slightly ahead of Ext4 (51,154 IOPS) and Btrfs (43,754 IOPS).

TABLE IV
BASELINE FIO PERFORMANCE RESULTS UNDER DIRECT I/O, QUEUE DEPTH 16, AND 30 S RUNTIME; HIGHER IOPS/THROUGHPUT AND LOWER LATENCY ARE BETTER.

FS	Workload	IOPS	MB/s	Avg Lat. (ms)	P50 (ms)	P95 (ms)	P99 (ms)
Ext4	seq_read	2,247	2,247	7.13	6.63	10.64	16.34
Ext4	seq_write	1,350	1,350	11.85	11.51	17.35	23.29
Ext4	rand_read	51,154	200	0.31	0.23	0.63	1.32
Ext4	rand_write	36,996	145	0.43	0.33	0.90	1.37
XFS	seq_read	2,311	2,311	6.98	6.78	9.45	13.17
XFS	seq_write	1,589	1,589	10.13	9.85	15.77	21.80
XFS	rand_read	53,021	207	0.30	0.23	0.59	1.03
XFS	rand_write	32,011	125	0.50	0.31	1.05	3.45
Btrfs	seq_read	2,626	2,626	6.17	5.78	9.01	15.34
Btrfs	seq_write	1,333	1,333	12.04	11.25	18.13	24.86
Btrfs	rand_read	43,754	171	0.37	0.28	0.73	1.42
Btrfs	rand_write	26,024	102	0.61	0.50	1.21	1.92

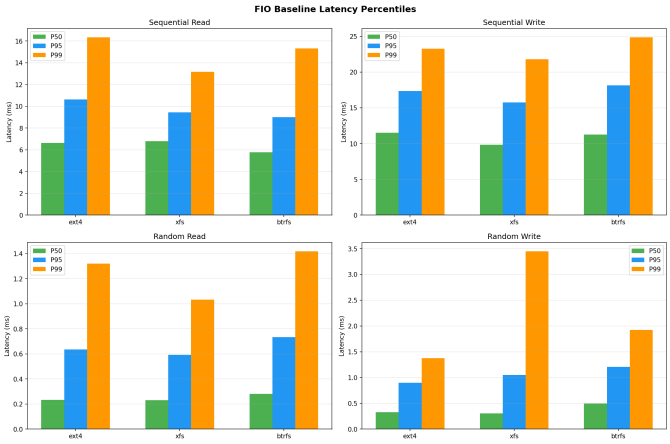

Fig. 3. P50, P95, and P99 latency percentiles for the four baseline workloads; lower values are better.

- *Sequential-read performance:* Btrfs delivered the highest single-job sequential-read throughput (2,626 MB/s), outperforming XFS (2,311 MB/s) and Ext4 (2,247 MB/s); however, this advantage did not persist under the multi-job video-editor scenario discussed in Section IV-B.
- *Sequential-write performance:* XFS demonstrated the highest sequential-write throughput at 1,589 MB/s, outperforming Ext4 (1,350 MB/s) and Btrfs (1,333 MB/s).
- *Latency characteristics:* All filesystems showed sub-millisecond median latencies for random-I/O operations, with XFS exhibiting the lowest P99 latency for random reads (1.03 ms).

B. Video-Editor Scenario Results

Table V shows the performance results for the video-editor scenario with four concurrent jobs and 1 MB block size.

Analysis:

- *Read performance:* Ext4 achieved the highest sequential-read throughput at 2,070 MB/s, approximately 34% higher than XFS (1,545 MB/s) and 35% higher than Btrfs (1,527 MB/s). The reversal relative to the single-job baseline should be interpreted cautiously because concurrency, loopback storage, and caching behavior may all affect the observed ordering.
- *Write performance:* All three filesystems demonstrated similar sequential-write performance in the 1,197–1,244 MB/s range, suggesting that the loopback layer may be a dominant bottleneck for write throughput in this environment.
- *Latency impact:* The high I/O depth and multiple concurrent jobs resulted in elevated latencies compared with baseline tests, with P99 latencies reaching approximately 100–180 ms.

TABLE V
VIDEO-EDITOR SCENARIO PERFORMANCE UNDER FIO: FOUR JOBS, 1 MB BLOCK SIZE, QUEUE DEPTH 32, AND 45 S RUNTIME.

FS	Workload	IOPS	MB/s	Avg Lat. (ms)	P50 (ms)	P99 (ms)
Ext4	seq_read	2,070	2,070	62.24	57.23	109.58
Ext4	seq_write	1,197	1,197	107.00	103.63	173.02
XFS	seq_read	1,545	1,545	83.01	74.62	167.42
XFS	seq_write	1,239	1,239	103.30	97.34	155.54
Btrfs	seq_read	1,527	1,527	86.91	83.62	141.03
Btrfs	seq_write	1,244	1,244	102.97	96.64	180.70

C. Mail-Server Scenario Results

The mail-server scenario with sysbench revealed distinct behavior patterns across filesystems, as presented in Table VI.

Analysis:

- *Ext4 and XFS comparison:* Ext4 and XFS demonstrated stable and predictable behavior with approximately 16,000 events/s and sub-millisecond average latency.
- *Btrfs anomaly:* Btrfs produced anomalous metrics with zero-reported latency values and extremely high event and fsync counts. These values should be treated as a

TABLE VI
MAIL-SERVER SCENARIO RESULTS UNDER SYSBENCH: RANDOM READ/WRITE, FSYNC AFTER EVERY WRITE, FOUR THREADS, AND 45 S RUNTIME.

FS	Events/s	Reads/s	Writes/s	Fsyncs/s	Avg Lat. (ms)	P95 (ms)
Ext4	16,391	2.47	1.65	17,223	0.23	0.44
XFS	16,092	2.40	1.61	16,937	0.24	0.46
Btrfs	2,237,751[a]	336	224	2,238,046[a]	0.00[a]	0.00[a]

[a] Anomalous Btrfs counters; excluded from comparative performance ranking.

measurement anomaly rather than evidence of genuinely superior performance; confirming the cause would require additional tracing, remount verification, and repeated measurements.

- *Conclusion:* For mail-server workloads with frequent synchronous operations, Ext4 and XFS provide more interpretable performance results than Btrfs in the present environment.

D. `dd` Sequential I/O Results

Table VII reports the direct sequential copy sanity-check results.

TABLE VII
DD SEQUENTIAL-I/O SANITY CHECK USING BS=1M, COUNT=1024, DIRECT I/O, AND APPROXIMATELY 1 GB TRANSFER PER DIRECTION.

FS	Direction	Throughput (GB/s)	User (s)	Sys (s)
Ext4	Write	1.10	0.00	0.11
Ext4	Read	1.70	0.00	0.05
XFS	Write	2.10	0.00	0.10
XFS	Read	1.50	0.00	0.06
Btrfs	Write	2.20	0.00	0.14
Btrfs	Read	1.30	0.00	0.05

The `dd` results serve as a sanity check, confirming that all filesystems achieve greater than 1 GB/s throughput for sequential operations. Quantitative comparisons should rely primarily on `fio` results, as `dd`'s single-threaded synchronous model differs from `fio`'s more comprehensive testing methodology.

E. Tuning Experiment Results

Fig. 4 and Table VIII summarize the impact of the three tuning profiles.

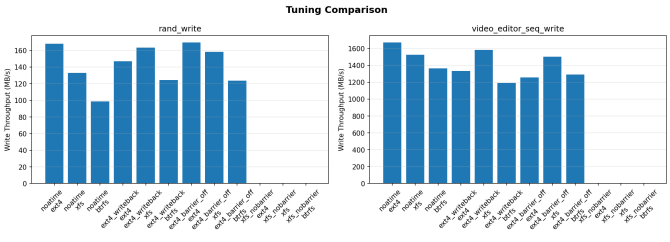

Fig. 4. Random-write and video-editor sequential-write performance under `noatime`, `data=writeback`, and `barrier=0` tuning profiles.

Analysis:

TABLE VIII
TUNING EXPERIMENT RESULTS REPORTING WRITE THROUGHPUT AND AVERAGE LATENCY; UNSAFE EXT4 PROFILES ARE INCLUDED ONLY AS CONTROLLED EXPERIMENTS.

Profile	FS	Workload	MB/s	Avg Lat. (ms)
`noatime`	Ext4	`rand_write`	168.23	0.37
`noatime`	Ext4	`video_editor_write`	1,671	77.57
`noatime`	XFS	`rand_write`	133.04	0.47
`noatime`	XFS	`video_editor_write`	1,525	83.99
`noatime`	Btrfs	`rand_write`	98.67	0.63
`noatime`	Btrfs	`video_editor_write`	1,364	93.92
`ext4_writeback`	Ext4	`rand_write`	147.18	0.42
`ext4_writeback`	Ext4	`video_editor_write`	1,333	100.41
`ext4_barrier_off`	Ext4	`rand_write`	169.49	0.37
`ext4_barrier_off`	Ext4	`video_editor_write`	1,255	101.89

- *`noatime` impact:* Enabling `noatime` produced workload-dependent effects. Ext4 random-write throughput improved from 145 MB/s to 168.23 MB/s, while Btrfs random-write throughput slightly decreased in this single-run measurement; therefore, this result should be interpreted as workload-specific rather than a general causal guarantee.
- *Writeback mode:* Ext4's `data=writeback` mode did not provide a clear random-write improvement in this measurement; statistical confirmation would require repeated trials.
- *`barrier=0` impact:* Disabling barriers provided approximately 17% random-write improvement for Ext4 in this measurement but at a significant crash-safety cost.
- *XFS `nobarrier`:* This option has been deprecated in modern Linux kernels and was not tested.

V. DISCUSSION

A. Performance Trade-offs Analysis

1) Ext4 Random-Write Advantage: Ext4's higher measured random-write performance in this environment (36,996 IOPS) compared with XFS (32,011 IOPS) and Btrfs (26,024 IOPS) is plausibly related to its comparatively direct overwrite path. When modifying existing data blocks, Ext4 can update them in place without requiring the same CoW allocation and tree-update path used by Btrfs. This characteristic suggests that Ext4 may be a strong candidate for similar write-intensive workloads, subject to confirmation on the target storage stack.

2) XFS Parallelism Benefits: XFS's allocation-group architecture enables parallel operations. Multiple threads can operate on different AGs with reduced contention, which is consistent with XFS's strength in sequential-write and random-read operations in our measurements. The B+ tree based metadata indexing may also contribute to efficient space allocation and lookup.

3) Btrfs CoW Performance Cost: Btrfs's CoW mechanism provides important features such as snapshots and data-integrity verification, but it is expected to introduce additional overhead for random writes. Each write operation may require allocating new space, writing data, and updating multiple B-tree nodes along the path. In our measurements, Btrfs random-

write IOPS were approximately 30% lower than Ext4 when Ext4 is used as the denominator. However, for sequential-write workloads that allocate fresh extents, Btrfs performed comparably with the other filesystems.

B. Methodology Considerations

1) Direct I/O vs. Buffered I/O: Our `fio` baseline tests employed direct I/O (`direct=1`) to bypass the page cache and measure filesystem behavior with reduced kernel-buffering effects. However, many real-world applications rely on buffered I/O, where the filesystem leverages the page cache. The `sysbench` mail-server tests used buffered I/O, which partly explains why their performance characteristics differ from the direct-I/O baseline tests.

The comparison between direct-I/O and buffered-I/O results should therefore be made with awareness of these different caching semantics: direct-I/O results are more representative of storage-path behavior under cache bypass, while buffered-I/O results better reflect typical application behavior.

2) Virtualized Environment Limitations: All experiments were conducted in a virtualized cloud environment with loopback-based storage. This introduces several limitations:

- Storage is constrained by the host system's resources and may not represent bare-metal performance.
- Loopback images add an indirection layer that may affect I/O latency measurements.
- Virtualized environments may have different I/O scheduling characteristics from physical hosts.
- CPU and memory resources may be shared with other virtual machines on the host.

Results should therefore be interpreted as relative comparisons within this test environment rather than absolute performance predictions for production bare-metal deployments.

C. Limitations

1) *Limited repetition:* Owing to resource constraints, some scenario and tuning tests were executed only once rather than repeated multiple times. This limits the statistical confidence of the corresponding conclusions and prevents formal significance testing.
2) *Workload scaling:* Mail-server tests were scaled down from ideal parameters to fit the virtualized environment, potentially affecting the representativeness of the results.
3) *Block-device diversity:* Tests were conducted using loopback images, which may exhibit different characteristics from physical storage devices such as solid-state drives (SSDs), Non-Volatile Memory Express (NVMe) drives, and hard disk drives (HDDs).
4) *Version specificity:* Results may vary across different kernel versions and filesystem-tool versions.
5) *Scope limitation:* This study focuses on performance metrics and does not comprehensively evaluate reliability, data integrity, or administrative ease of use.

VI. Conclusion

This benchmarking study of Ext4, XFS, and Btrfs in a virtualized loopback-based environment yields the following key findings:

1) *Ext4 is strongest for random writes in the baseline test:* Ext4 achieved 36,996 IOPS for random writes, compared with 32,011 IOPS for XFS and 26,024 IOPS for Btrfs. This corresponds to a 30% lower Btrfs result when Ext4 is used as the denominator, or a 42% Ext4 advantage when Btrfs is used as the denominator.
2) *XFS is strongest for sequential writes and random reads in the baseline test:* XFS achieved the highest sequential-write throughput (1,589 MB/s) and random-read IOPS (53,021), which is consistent with its allocation-group architecture and metadata design.
3) *Btrfs trades random-write speed for advanced features:* Btrfs provides CoW-based functionality, snapshots, and checksumming, but its baseline random-write result was lower than those of Ext4 and XFS in this setup.
4) *Mount-option tuning is workload-dependent:* `noatime` improved some write workloads, but the gains were not uniform across all filesystems and workloads. Such tuning should therefore be validated on the target workload rather than assumed universally beneficial.
5) *Safety-critical options are not production recommendations:* Options such as `barrier=0` and `data=writeback` were included only to illustrate performance–safety trade-offs. The observed gains in this experiment do not justify weakening crash-consistency guarantees in production systems.

Within this test environment, these results suggest Ext4 for random-write-heavy workloads, XFS for sequential I/O and large-scale concurrent operations, and Btrfs when snapshots, checksums, and self-healing features are more important than peak random-write performance. These recommendations are environment-specific and should be revalidated on bare-metal storage before production deployment.

Acknowledgment

This work was completed as part of the Operating System Course. The authors thank the course instructor and teaching assistants for their guidance throughout this experiment and fellow classmates for discussions on benchmarking methodology.

References

[1] The Linux Kernel Documentation, "ext4 data structures and algorithms." [Online]. Available: https://docs.kernel.org/filesystems/ext4/. Accessed: May 4, 2026.
[2] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, "The new ext4 filesystem: current status and future plans," in *Proc. Linux Symp.*, Ottawa, ON, Canada, Jun. 2007, pp. 21–34. [Online]. Available: https://www.kernel.org/doc/ols/2007/ols2007v2-pages-21-34.pdf
[3] The Linux Kernel Documentation, "XFS filesystem documentation." [Online]. Available: https://docs.kernel.org/filesystems/xfs/index.html. Accessed: May 4, 2026.
[4] BTRFS Documentation, "Welcome to BTRFS documentation!" [Online]. Available: https://btrfs.readthedocs.io/en/latest/. Accessed: May 4, 2026.

[5] O. Rodeh, J. Bacik, and C. Mason, “BTRFS: The Linux B-tree filesystem,” *ACM Trans. Storage*, vol. 9, no. 3, Art. no. 9, pp. 1–32, Aug. 2013, doi: 10.1145/2501620.2501623.
[6] J. Axboe, “fio—Flexible I/O Tester,” Git repository. [Online]. Available: https://github.com/axboe/fio. Accessed: May 4, 2026.
[7] A. Kopytov, “sysbench: scriptable database and system performance benchmark,” Git repository. [Online]. Available: https://github.com/akopytov/sysbench. Accessed: May 4, 2026.