File Systems' Impact on SSD Performance and Energy Efficiency

Sarah Massie, Stephanie Sithu, Jacob Higdon, Bryan Harris, and Nihat Altiparmak
Department of Computer Science and Engineering
University of Louisville, USA
{sarah.massie,stephanie.sithu,jacob.higdon,bryan.harris.1,nihat.altiparmak}@louisville.edu

2025 IEEE International Conference on Cloud Computing Technology and Science (CloudCom) | 979-8-3315-6634-0/25/$31.00 ©2025 IEEE | DOI: 10.1109/CLOUDCOM67567.2025.11331581

***Abstract*—The rapid expansion of digital services, particularly generative AI and cloud computing, has significantly increased the energy demands of data centers. This growth poses serious threats to both the economy and the environment by straining power grids, driving up electricity costs, increasing greenhouse gas emissions, and exacerbating water scarcity. While energy-aware computing practices have emerged as a promising solution, the role of operating system components, especially file systems remains underexplored. In this paper, we investigate the performance and energy efficiency implications of three widely used Linux file systems: *ext4*, *xfs*, and *f2fs*. Using a high-performance NVMe SSD, a power meter, and a diverse set of synthetic and real-world IO workloads, we conduct a systematic evaluation across varying request sizes and concurrency levels. Our results reveal that *f2fs* excels in low-intensity random write workloads due to its log-structured design, while *xfs* consistently delivers the best energy efficiency under high-concurrency scenarios. We highlight how file system design choices can significantly influence system-wide energy consumption and advocate for energy-aware system software design as a critical step toward sustainable cloud infrastructure.**

***Index Terms*—storage and file systems, SSD, IO performance, energy efficiency**

I. Introduction

Everyday activities of regular Internet users, such as streaming video, engaging with social media, using generative AI tools, and accessing various cloud applications place a significant load on cloud data centers. In addition, organizations across industry and academia increasingly rely on cloud-based or on-premises data centers to drive breakthrough innovations. However, this reliance comes at a substantial environmental and economic cost due to the high energy consumption of data centers, especially with the recent surge in AI adoption and the growing scale of infrastructure. In 2022, data centers consumed approximately 460 TWh of electricity (equivalent to around $46 billion), and this figure is projected to double by 2026 [1]. To contextualize, this level of energy usage rivals the annual consumption of large industrialized nations such as Germany or France [2].

Operating systems play a critical role in the efficient management of computing resources, directly influencing both performance and energy efficiency. There are currently over 8,000 data centers worldwide, with roughly one-third located in the United States [1]. Each data center typically houses thousands of physical machines, each running at least one operating system. In cloud environments, multiple operating systems often run concurrently on a single physical machine via virtualization. Consequently, even a modest percentage reduction in power consumption at the operating system level can lead to significant global energy savings, which is an essential step toward decarbonization.

As hardware technology evolves, operating systems must adapt to leverage these advancements while maintaining efficiency. One notable hardware shift in recent years has been the transition from traditional Hard Disk Drives (HDDs) to flash-based Solid-State Drives (SSDs). SSDs have gained widespread adoption due to declining costs and increased capacities, driven by improvements in flash technology and internal drive architecture. Unlike HDDs, which rely on mechanical components for data access and are limited by seek time and rotational latency, SSDs are fully electronic and offer faster access times. Moreover, NVMe SSDs support parallel request handling through internal parallelism and advanced controllers, enabling significantly higher IO performance, provided the system software is optimized accordingly. In this context, the operating system, particularly its file system component, is crucial for harnessing the full potential of SSDs.

File systems are responsible for organizing, managing, and providing access to data on storage devices such as HDDs and SSDs. They structure data hierarchically through directories and files, manage file-to-storage mappings via allocation strategies, and track available space using free space management techniques. Additionally, file systems handle operations such as file creation, deletion, access, and modification, facilitating seamless interaction between users, applications, and data. Advanced features may include permission management, access control, error detection and correction, caching, encryption, defragmentation, and backup/restore capabilities. The design and implementation of a file system directly affect a system's IO performance and energy efficiency [3].

In this paper, we investigate the IO performance and energy implications of three widely used Linux file systems: *ext4* [4], *xfs* [5], and *f2fs* [6]. *ext4*, the successor to the traditional Linux file system, offers journaling and enhancements in capacity and performance, making it the default in many distributions such as Debian and Ubuntu. *xfs* is optimized for high performance, parallelism, and scalability, and is the default in Red Hat Enterprise Linux (RHEL) and Rocky Linux. *f2fs*, or flash-friendly file system, is specifically designed for SSDs, employing a log-structured architecture to align with the characteristics of flash storage.

Authorized licensed use limited to: University of Louisville. Downloaded on February 23,2026 at 20:36:44 UTC from IEEE Xplore. Restrictions apply.

A key concept in energy-efficient computing is energy proportionality, which posits that energy consumption should scale proportionally with performance [7]. In other words, higher energy usage should yield correspondingly higher performance. However, energy considerations are often overlooked in system software design, where performance traditionally takes precedence. Our empirical evaluation using a modern flash SSD, a power meter, diverse IO workloads, and the latest stable Linux kernel (6.16) measures IO performance per unit of energy, such as bytes transferred per joule. By integrating performance and energy into a unified metric, we aim to contribute to the broader field of energy-aware computing. Our findings offer valuable insights for both the development of future file systems and the optimization of existing ones, supporting efforts to reduce the energy footprint of data centers and promote sustainable technology practices.

II. Background and Related Work

This section first describes the design choices of the *ext4*, *xfs*, and *f2fs* file systems, followed by a review of related work.

A. File System Design Choices

A file system is a method and data structure used by an operating system to manage and organize files on a storage device. It provides mechanisms to store, retrieve, and update data, ensuring both data integrity and efficient access. Design choices in file systems can significantly affect system performance and energy efficiency [3]. This study focuses on three widely used file systems in Linux environments: *ext4*, *xfs*, and *f2fs*. Below, we summarize their key design characteristics.

***ext4* [4]** is the fourth extended file system developed for Linux in the late 2000s, offering substantial improvements over its predecessors, *ext3* and *ext2*. Like *ext3*, *ext4* supports journaling, which enhances reliability by maintaining a log of changes before committing them to the main file system, particularly useful during system crashes or power failures. *ext4* also supports larger volumes and files, surpassing the limitations of earlier versions. Performance enhancements include extent-based file allocation (allocating contiguous blocks), delayed allocation (efficient grouping of block allocations), and faster file system checks by skipping unallocated block groups and unused inode table sections. Due to its balance of performance, stability, and reliability, *ext4* is the default file system in many Linux distributions, including Debian and Ubuntu.

***xfs* [5]** originated at Silicon Graphics in the early 1990s and was later ported to Linux. Renowned for its robustness and scalability, *xfs* is optimized for handling large files and parallel IO operations through its use of allocation groups. It efficiently handles concurrent IO from multiple applications, making it well-suited for multi-processor systems and high IO workloads. This parallelism is particularly beneficial for SSDs, which are designed for concurrent IO. Like *ext4*, *xfs* includes extent-based allocation and delayed allocation, along with delayed logging and advanced metadata operations. Its ability to manage large data volumes and deliver fast access makes it a popular choice for high-performance computing and big data applications. It is the default file system in Red Hat Enterprise Linux (RHEL) and Rocky Linux.

***f2fs* [6] (flash-friendly file system)** was developed by Samsung in the early 2010s specifically for NAND flash-based storage devices such as SSDs. *f2fs* employs a log-structured file system architecture [8], writing all modifications sequentially in a log-like format (out-of-place writes). This design improves SSD performance and longevity by minimizing write amplification, a phenomenon where a single write operation triggers multiple internal writes. Since sequential writes are generally faster on SSDs, log-structured designs can improve random write performance by converting them into sequential operations. However, this approach also generates frequent obsolete blocks that must be reclaimed via a periodic garbage collection process, separate from the SSD's internal garbage collection. Additional features of *f2fs* include multi-head logging for separating hot and cold data, and adaptive logging for efficient block allocation.

B. Related Work

Benchmarking is essential for evaluating system performance and energy efficiency. Numerous prior studies have examined the performance and energy impact of file systems. Early research primarily focused on HDDs and older file system generations [3], [9]–[11], highlighting the importance of data localization to reduce disk head movement [9]–[11], and advocating for metrics such as storage performance per joule to better assess energy efficiency [3], which we also adopt in this study.

Subsequent work shifted focus to SSDs, analyzing performance and reliability [12]–[16]. These studies explored file system scalability [12], [13], the impact on write amplification [14], IO performance using SATA SSDs [15], and reliability in the context of SSD failures [16]. While some online forums and benchmarking websites have shared performance data [17]–[20], a systematic and rigorous analysis remains lacking. Benchmarking file and storage systems is particularly challenging due to the complexity of modern storage stacks, which include caching layers, kernel interactions, and concurrent kernel tasks [21]. Thus, a transparent and methodical benchmarking approach using diverse workloads is essential to capture the full spectrum of system behavior [22].

Beyond limited studies in mobile and embedded contexts [23], the energy efficiency of file systems on SSDs has not been thoroughly investigated in desktop and server environments. Moreover, the Linux kernel evolves rapidly, and recent changes can significantly affect performance and energy efficiency, necessitating frequent benchmarking. Innovations in the storage subsystem, such as widespread adoption of faster PCIe generations and NVMe SSDs, further motivate a reevaluation of file system performance under modern conditions. Our study addresses this gap by systematically evaluating the implications of file system design choices on the IO performance and energy efficiency using recent hardware and kernel versions.

Authorized licensed use limited to: University of Louisville. Downloaded on February 23,2026 at 20:36:44 UTC from IEEE Xplore. Restrictions apply.

Finally, distributed file systems have gained prominence in cloud and scientific computing environments due to their scalability and fault tolerance. Prior work has evaluated the performance of distributed file systems such as Ceph [24], GlusterFS [25], and Lustre [26] in FUSE-based environments, showing that performance varies significantly with data characteristics and workload patterns [27]. Their findings underscore the importance of aligning file system features with specific scientific data analysis needs to optimize throughput and reliability. Our work is orthogonal to this line of research, as distributed file systems typically run atop local file systems.

III. Implications of File System Design Choices

This section explores the impact of file system design decisions on IO performance and energy efficiency.

A. Experimental Setup, Methodology, and Metrics

Our test storage device is a flash-based 2 TB Samsung 990 PRO PCIe 4.0 NVMe SSD with a heatsink, connected to a PCIe 4.0-compatible Dell Precision T3600 workstation. The system is equipped with a 12th-generation Intel i7-12700 CPU (3.6/4.8 GHz, 4E/8P cores, 20 threads), 16 GB RAM, and a 1 TB SK Hynix PC801 NVMe SSD used exclusively for the operating system (not for IO testing). We used Ubuntu 24.04 LTS with the latest stable Linux kernel version 6.16. To measure energy consumption, we employed an Onset HOBO power meter [28], which continuously records the system's power usage to embedded memory, thereby eliminating any software overhead on the test system.

We developed an in-house automation script to systematically execute a series of tests across the evaluated file systems: *ext4*, *xfs*, and *f2fs*. For each file system, the process includes periodic SSD preconditioning (using `fstrim` and `blkdiscard` as appropriate), formatting and mounting the file system, executing various IO workloads, and unmounting the file system. This process is repeated for each file system, and the entire experiment is conducted 10 times. We report the average and standard deviation across these 10 runs.

For all file systems, we used the `noatime` mount option to disable last access time updates. To minimize the effects of caching, we either performed direct IO (via *fio*) or ensured that dataset sizes significantly exceeded the available RAM (*rocksdb*). We evaluate file system performance and energy efficiency using three primary metrics:

IO Performance: Depending on the workload, IO performance is measured in terms of bandwidth (bytes/sec) or throughput (IOPS or ops/sec).

System Power Consumption: Power consumption of the entire system is recorded every second in watts using the HOBO power meter, which is connected to the system's sole power supply. To ensure accurate timestamp synchronization, both the test system and the power meter were synchronized using the same time server.

Energy Efficiency: Energy efficiency is calculated as the ratio of IO performance to system power consumption as follows:

$$\text{Energy Efficiency} = \frac{\text{IO Performance (bytes/sec or IOPS)}}{\text{System Power Consumption (watts)}} \quad (1)$$

This metric provides a clear indication of how effectively a file system translates power usage into IO performance. Measuring overall system power consumption is essential, as high-performance IO workloads place significant demands not only on the storage device but also on system software and other hardware components, particularly the CPU and RAM, due to frequent IO submission, scheduling, and completion operations [29]–[32]. Since one watt equals one joule per second, the equation above directly translates to bytes/joule or IO/joule, representing the amount of IO performance achieved per unit of energy consumed. Higher values indicate better energy efficiency, making this a valuable metric for reducing the energy footprint of data centers.

B. Workloads

We evaluate IO performance and energy efficiency using the following storage IO workloads:

***fio* [33]:** We used the flexible IO tester (*fio*) version 3.36 to generate microbenchmarks. The *io_uring* interface was selected due to its popularity, high-performance characteristics, and support for asynchronous IO [34]. The IO depth, which defines the number of outstanding IO requests issued to the storage device, was set to 64 based on the internal parallelism capabilities of our test SSD. All tests used direct IO to bypass the page cache, ensuring that all IO requests were sent directly to the storage device. Each test included a 30-second measurement window followed by a 30-second idle period to allow the system to cool down.

We evaluated five types of IO operations on a 100 GB file: random reads, sequential reads, random writes, sequential writes, and a random mixed workload consisting of 50% random reads and 50% random writes. Additionally, we used two IO sizes: 4 KB and 128 KB. To simulate both parallel IO and concurrent multi-tenant behavior common in cloud environments, we ran experiments with one, four, and eight processes issuing IO requests from multiple cores in parallel.

***RocksDB [35] with YCSB [36]*:** To complement synthetic benchmarks, we also evaluated real-world application workloads. While *fio* is useful for testing system IO limits, realistic application patterns are better captured using actual software. For this purpose, we used the RocksDB key-value store, which is optimized for low-latency lookups and is well-suited for flash SSDs. We paired RocksDB with the Yahoo Cloud Serving Benchmark (YCSB), a widely adopted framework for evaluating the performance of cloud data-serving systems such as BigTable [37] and Cassandra [38].

For benchmarking, we initialized a RocksDB database on our test SSD with 12 million records, each containing a single 4 KB field. This setup occupied approximately 50 GB initially and grew over time as certain workloads inserted additional records. YCSB includes six realistic workloads that simulate common cloud service patterns:

Authorized licensed use limited to: University of Louisville. Downloaded on February 23,2026 at 20:36:44 UTC from IEEE Xplore. Restrictions apply.

- **Workload A – Update-Heavy Workload:** Issues a mix of 50% reads and 50% writes, where updates do not require reading the original record. This simulates session store behavior, recording recent user actions.
- **Workload B – Read-Mostly Workload:** Issues 95% reads and 5% writes, simulating scenarios like photo tagging, where most operations are reads.
- **Workload C – Read-Only Workload:** Issues 100% reads, emulating a user profile cache where profiles are constructed elsewhere (e.g., in Hadoop).
- **Workload D – Read-Latest Workload:** Inserts new records and assumes the most recent ones are the most popular. This simulates user status updates where users seek the latest information.
- **Workload E – Short Range Queries:** Performs short-range scans instead of individual record accesses. This simulates threaded conversations, where posts are clustered by thread ID. Like Workload D, it also inserts new records.
- **Workload F – Read-Modify-Write:** Reads a record and then modifies it, simulating user databases where records are frequently updated.

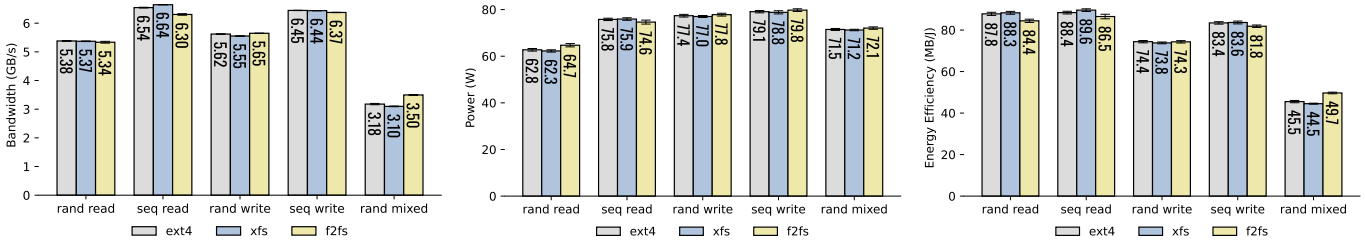
As with *fio*, we ran *rocksdb-ycsb* workloads using one, four, and eight threads to evaluate concurrent access behavior.

C. Experimental Results

Figures 1, 2, and 3 present the (a) IO performance, (b) power consumption, and (c) energy efficiency of *fio* workloads using 128 KB request sizes, under one-, four-, and eight-process scenarios, respectively. Results for 4 KB request sizes are shown in Figures 4, 5, and 6. The numbers on the bars represent the mean values across 10 runs, and the error bars indicate the standard deviation.

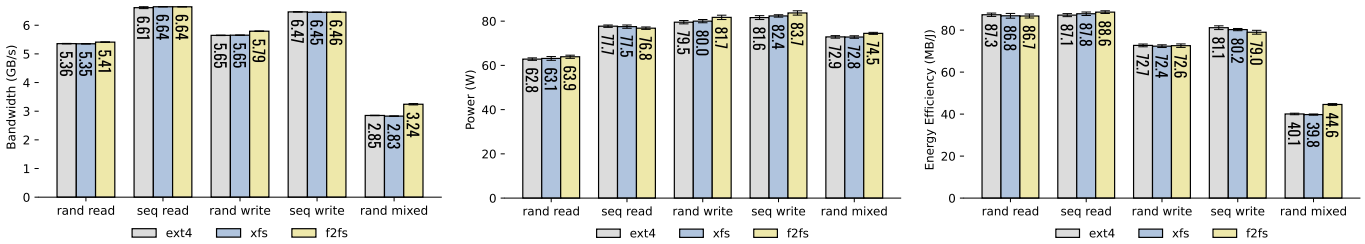
Observation 1. *f2fs achieves the best IO performance for random write and random mixed workloads in low IO intensity (IOPS) scenarios.*

Applications issuing large IO requests (e.g., 128 KB) can saturate the available storage bandwidth with relatively low IO intensities (IOPS), as seen in Figures 1, 2, and 3. In these experiments, the maximum throughput achieved is approximately 54 K IOPS, which is relatively low for a device capable of providing over 1 M IOPS but still sufficient to provide 6.6 GB/s bandwidth (close to device saturation) due to large request sizes. Such low IOPS levels exert less pressure on the file system and system software responsible for IO submission, scheduling, and completion. In these 128 KB experiments, *f2fs* performs comparably to other file systems for sequential writes



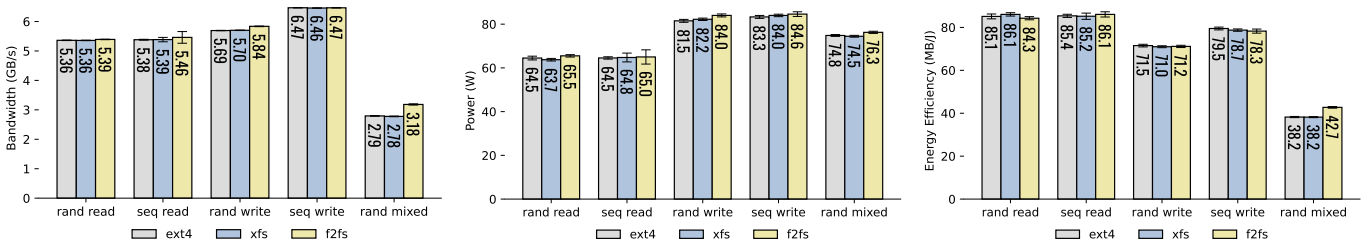
(a) IO Performance (GB/s) (b) Power Consumption (watts) (c) Energy Efficiency (MB/joule)

Fig. 1: *fio* Benchmark, 128 KB IO Requests, 1 Process



(a) IO Performance (GB/s) (b) Power Consumption (watts) (c) Energy Efficiency (MB/joule)

Fig. 2: *fio* Benchmark, 128 KB IO Requests, 4 Processes



(a) IO Performance (GB/s) (b) Power Consumption (watts) (c) Energy Efficiency (MB/joule)

Fig. 3: *fio* Benchmark, 128 KB IO Requests, 8 Processes

Authorized licensed use limited to: University of Louisville. Downloaded on February 23,2026 at 20:36:44 UTC from IEEE Xplore. Restrictions apply.

and outperforms them in random write and random mixed scenarios.

Observation 2. *f2fs achieves the best energy efficiency for random mixed workloads in low IO intensity (IOPS) scenarios.*

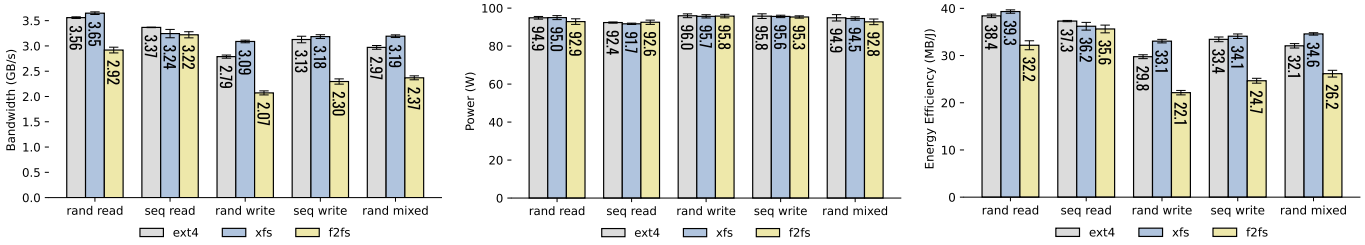
f2fs delivers superior IO performance in random write and random mixed scenarios due to its log-structured design, which efficiently converts random writes into sequential ones, an advantage for flash SSDs under low IOPS conditions. Although this conversion introduces additional processing overhead, resulting in higher power consumption, the performance gains in random mixed workloads outweigh the energy cost, making *f2fs* the most energy-efficient file system in these cases. However, in the 100% random write scenario, the extra energy consumption does not yield a significant performance benefit, resulting in lower energy efficiency compared to other file systems.

Observation 3. *f2fs suffers from write performance in high IO intensity (IOPS) scenarios, especially as IO concurrency increases.*

Modern flash SSDs with high internal parallelism and PCIe 4.0+ support can handle millions of small IO operations per second. Applications issuing small requests (e.g., 4 KB) place significantly higher stress on the file system, allowing us to evaluate scalability. As shown in Figures 4, 5, and 6, *f2fs* consistently exhibits the lowest write performance across random write, sequential write, and random mixed scenarios. This is likely due to lock contention in shared data structures managing its log. Under high concurrency, such as in the four- and eight-process cases, these structures become bottlenecks. Even in the single-process case (Fig. 4), *f2fs* struggles to maintain performance under high IOPS. In Fig. 6, both *ext4* and *xfs* scale to 1.62 M IOPS for sequential writes, achieving 6.18 GB/s bandwidth and nearly saturating the device. In contrast, *f2fs* remains below 3 GB/s, less than half the performance of the others, due to its inability to scale under high IO intensity and concurrency.

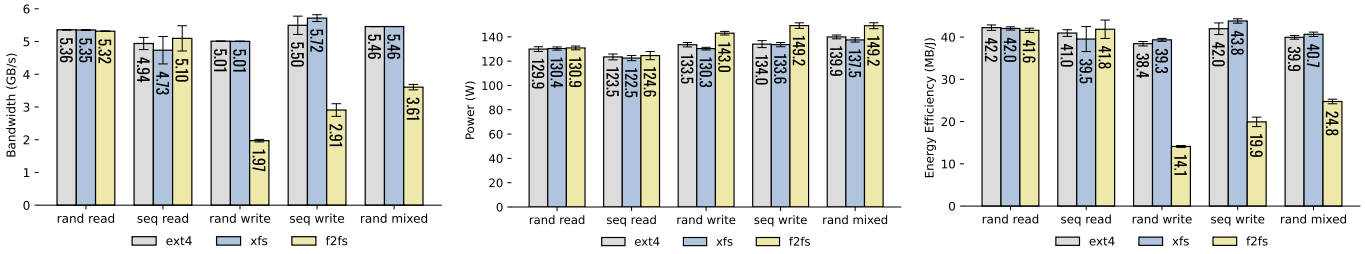
Observation 4. *Overall, xfs is the best option for energy efficiency.*

For read operations, all three file systems generally show comparable IO performance and energy efficiency. However, when considering all request sizes (4 KB and 128 KB), request types (read, write, mixed), and access patterns (random,



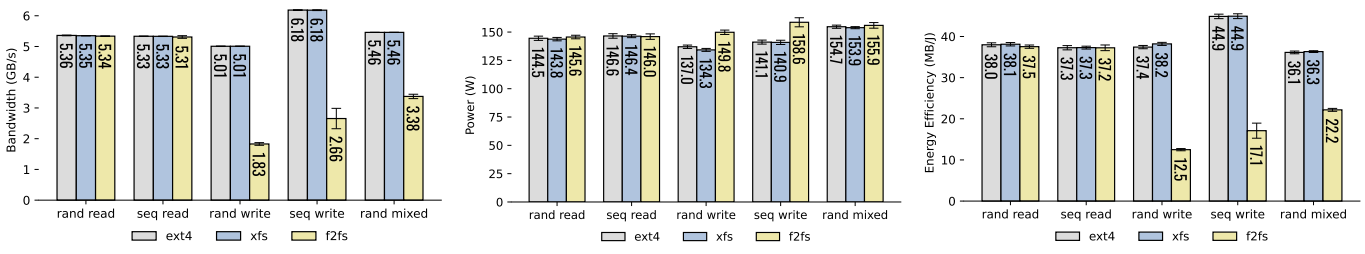
(a) IO Performance (GB/s) (b) Power Consumption (watts) (c) Energy Efficiency (MB/joule)

Fig. 4: *fio* Benchmark, 4 KB IO Requests, 1 Process



(a) IO Performance (GB/s) (b) Power Consumption (watts) (c) Energy Efficiency (MB/joule)

Fig. 5: *fio* Benchmark, 4 KB IO Requests, 4 Processes



(a) IO Performance (GB/s) (b) Power Consumption (watts) (c) Energy Efficiency (MB/joule)

Fig. 6: *fio* Benchmark, 4 KB IO Requests, 8 Processes

Authorized licensed use limited to: University of Louisville. Downloaded on February 23,2026 at 20:36:44 UTC from IEEE Xplore. Restrictions apply.

sequential) tested with *fio*, *xfs* emerges as the most energy-efficient option, achieving the highest energy efficiency scores in more than half of the experiments. This holds true even when its IO performance is similar to that of *ext4* in some cases. The superior energy efficiency of *xfs* is likely due to its high-performance parallel IO design, which handles high IOPS without incurring additional power consumption. Further discussion is provided on this in Section IV.

Figures 7, 8, and 9 show the (a) IO performance, (b) power consumption, and (c) energy efficiency of *rocksdb-ycsb* application workloads under one-, four-, and eight-thread scenarios, respectively. As in the *fio* experiments, the bar values represent the mean of 10 runs, and the error bars indicate the standard deviation.

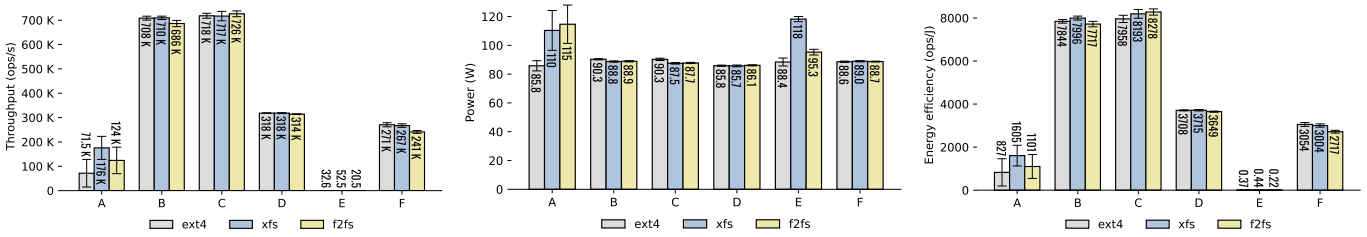
Since all *rocksdb-ycsb* workloads involving writes (A, B, E, and F) operate at high IOPS levels, Observations 1 and 2 cannot be confirmed using these workloads, as they pertain to low IO intensity scenarios. This highlights the importance of microbenchmarks, which allow comprehensive analysis across a wide range of IO behaviors, despite being synthetic. In contrast, macrobenchmarks such as *rocksdb-ycsb* reflect real-world application behavior but offer less flexibility in workload characteristics. However, *rocksdb-ycsb* workloads do support Observations 3 and 4. In high-intensity workloads involving writes (A, B, E, and F), *f2fs* continues to underperform in general (Obs. 3). For read workloads (C and D), all file systems perform similarly, consistent with *fio* results. Overall, *xfs* remains the most energy-efficient file system across all *rocksdb-ycsb* workloads (Obs. 4).

IV. Discussion

Our experimental results reveal several important insights into the performance and energy efficiency characteristics of modern Linux file systems when used with flash-based NVMe SSDs under varying IO workloads. These findings have implications for both system software design and practical deployment decisions in cloud and data center environments.

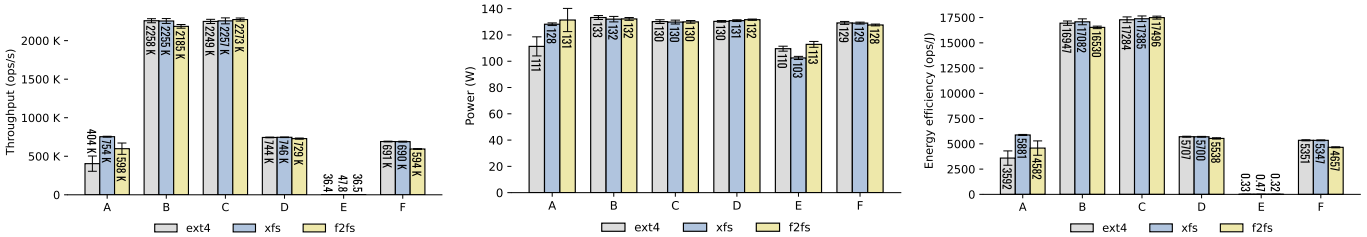
A. Microbenchmarks vs. Macrobenchmarks

The combination of synthetic microbenchmarks and real-world macrobenchmarks provides a more comprehensive understanding of file system behavior. Our findings show that while microbenchmarks are essential for fine-grained control over IO patterns, enabling us to isolate specific performance and energy characteristics as well as stress-testing scalability, macrobenchmarks are equally important for validating those findings in real-world contexts. For example, the poor scalability of *f2fs* under high-concurrency write workloads observed in *fio* is also evident in *rocksdb-ycsb*.



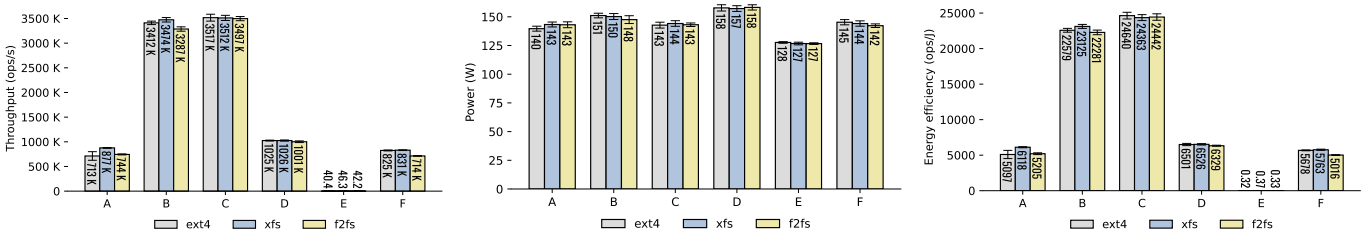
(a) IO Performance (ops/s) (b) Power Consumption (watts) (c) Energy Efficiency (ops/joule)

Fig. 7: *rocksdb-ycsb* Benchmark, 1 Thread



(a) IO Performance (ops/s) (b) Power Consumption (watts) (c) Energy Efficiency (ops/joule)

Fig. 8: *rocksdb-ycsb* Benchmark, 4 Threads



(a) IO Performance (ops/s) (b) Power Consumption (watts) (c) Energy Efficiency (ops/joule)

Fig. 9: *rocksdb-ycsb* Benchmark, 8 Threads

Authorized licensed use limited to: University of Louisville. Downloaded on February 23,2026 at 20:36:44 UTC from IEEE Xplore. Restrictions apply.

B. Impact of IO Intensity and Concurrency

One of the key takeaways from our study is the significant influence of IO intensity (IOPS) and concurrency on file system behavior. Under low IO intensity scenarios with large request sizes (e.g., 128 KB), *f2fs* demonstrates superior performance for random write and random mixed workloads. This is primarily due to its log-structured design, which efficiently converts random writes into sequential ones, an operation well-suited to the characteristics of flash SSDs. However, this advantage diminishes under high IO intensity scenarios, particularly with small request sizes (e.g., 4 KB) and increased concurrency. In these cases, *f2fs* suffers from performance degradation, likely due to lock contention in its internal data structures. This bottleneck becomes more pronounced as the number of concurrent processes increases, highlighting a scalability limitation in its current implementation.

C. Implications for Cloud and Edge Environments

The insights from this study are particularly relevant for cloud and edge computing environments, where storage systems must balance performance, scalability, and energy efficiency. In multi-tenant cloud environments, where concurrent access to shared storage is common, file systems like *xfs* that scale well under high concurrency and maintain energy efficiency are advantageous. On the other hand, in edge computing scenarios where workloads may be less concurrent and energy constraints are more stringent, *f2fs* may offer benefits, especially for write-heavy applications with low IO intensity. These trade-offs highlight the need for workload-aware file system selection in modern computing environments.

D. Energy Efficiency as a First-Class Metric

Our results underscore the importance of considering energy efficiency, not just raw performance, when evaluating file systems for cloud-scale deployments. While *f2fs* achieves high performance in certain scenarios, its energy efficiency varies depending on workload characteristics. In contrast, *xfs* consistently delivers strong energy efficiency across a wide range of workloads, even when its performance is comparable to *ext4*. This suggests that *xfs*'s design, which emphasizes parallelism and scalability, allows it to handle high IO loads without incurring significant additional power consumption. As energy costs and sustainability concerns continue to grow in importance, especially in large-scale data centers, such characteristics make *xfs* a compelling choice for general-purpose deployments.

The primary design feature behind *xfs*'s high-performance parallel IO capability is its use of allocation groups, which divide the file system's storage space into independent, self-contained units. This design enables concurrent operations on different units in parallel. The independence of allocation groups allows *xfs* to effectively leverage multi-core processors and high-performance flash SSDs with internal parallel IO capabilities. The minimum size of an allocation group is 16 MB, and the maximum is 1 TB. In our experiments, the file system was configured with four allocation groups, based on the default `mkfs.xfs` settings in Linux, similar to the default configurations used for the other file systems tested. For increased parallelism, the number of allocation groups can be configured to match the number of CPU cores in the system. In *xfs*'s design, individual files can span multiple allocation groups, and during file allocation, *xfs* may favor less busy allocation groups to distribute the load. For very large and rapidly growing files, *xfs* strategically places new extents in less-contended allocation groups to maximize parallel IO.

The use of allocation groups exemplifies an energy-aware system software design, one that improves IO performance while maintaining proportionally lower energy consumption. Such optimizations are achievable when system software developers carefully consider the architectural characteristics of the hardware they interact with and align their designs accordingly. For example, legacy file system optimizations tailored for HDD mechanics may not only fail to benefit SSDs but may also negatively impact energy efficiency. To reduce the energy consumption of data centers, system software developers should adopt an energy-aware mindset, evaluating the energy implications of their design choices relative to the performance benefits achieved, akin to a profit-loss analysis. Given the significant potential for energy savings at the operating system level, energy-aware components such as file systems are essential for mitigating the environmental and economic impact of data centers. These optimizations can help achieve sustainability goals without compromising performance or the societal benefits enabled by cloud computing infrastructure.

E. Limitations and Future Work

While our study provides valuable insights, it also has several limitations. First, we evaluated only a single SSD model. Future work could extend this analysis to include a broader range of storage configurations, such as RAID arrays of SSDs, to assess scalability and redundancy impacts. Our benchmarking methodology combined synthetic microbenchmarks and real-world macrobenchmarks; however, future work can also incorporate Filebench [39] to evaluate metadata-intensive operations. Beyond performance and energy efficiency, the impact of file systems on SSD longevity remains an important area for exploration. Finally, deeper analysis of *f2fs* under high-concurrency workloads, such as profiling lock contention, memory usage, and internal queueing behavior, could inform targeted optimizations or inspire new file system designs that better balance performance, scalability, and energy efficiency. It is important to note that this study focuses solely on the performance and energy implications of file systems. Further investigation is needed in scenarios where other factors, such as data reliability or security are primary concerns.

V. Conclusion

This paper investigates the impact of file system design choices on IO performance and energy efficiency. Our evaluation shows that while log-structured designs can improve SSD performance in low-intensity write scenarios, they may lead to significant performance degradation under highly concurrent,

Authorized licensed use limited to: University of Louisville. Downloaded on February 23,2026 at 20:36:44 UTC from IEEE Xplore. Restrictions apply.

write-intensive workloads if not carefully implemented. This is a critical concern as modern NVMe SSDs can exert substantial pressure on the shared data structures of log-structured file systems. Our findings also demonstrate that energy-aware file system designs are feasible, offering performance improvements without incurring substantial energy costs.

REFERENCES

[1] "Electricity 2024 - analysis and forecast to 2026," https://www.iea.org/reports/electricity-2024, Jan 2024, International Energy Agency.

[2] "Electricity consumption of countries," https://www.cia.gov/the-world-factbook/field/electricity/country-comparison, 2020 est., CIA World Factbook.

[3] P. Sehgal, V. Tarasov, and E. Zadok, "Evaluating performance and energy in file system server workloads," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, ser. FAST'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 19–19. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855511.1855530

[4] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, and L. Vivier, "The new ext4 filesystem: current status and future plans," in *Proceedings of the Linux symposium*, vol. 2, 2007, pp. 21–33.

[5] A. Sweeney, D. Doucette, W. Hu, C. Anderson, M. Nishimoto, and G. Peck, "Scalability in the xfs file system," in *Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '96. Berkeley, CA, USA: USENIX Association, 1996, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1268299.1268300

[6] C. Lee, D. Sim, J.-Y. Hwang, and S. Cho, "F2fs: A new file system for flash storage," in *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, ser. FAST'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 273–286. [Online]. Available: http://dl.acm.org/citation.cfm?id=2750482.2750503

[7] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, 2007.

[8] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Trans. Comput. Syst.*, vol. 10, no. 1, p. 26–52, feb 1992. [Online]. Available: https://doi.org/10.1145/146941.146943

[9] H. Huang, W. Hung, and K. G. Shin, "Fs2: dynamic data replication in free disk space for improving disk performance and energy consumption," *SIGOPS Oper. Syst. Rev.*, vol. 39, no. 5, p. 263–276, Oct. 2005. [Online]. Available: https://doi.org/10.1145/1095809.1095836

[10] L. Dong, "High performance energy efficient file storage system," Ph.D. dissertation, University of Nebraska-Lincoln, 2006. [Online]. Available: https://digitalcommons.unl.edu/dissertations/AAI3216420

[11] D. Essary and A. Amer, "Predictive data grouping: Defining the bounds of energy and latency reduction through predictive data grouping and replication," *ACM Trans. Storage*, vol. 4, no. 1, May 2008. [Online]. Available: https://doi.org/10.1145/1353452.1353454

[12] S. Shaikh, "Billion-files file systems (bffs): A comparison," arXiv preprint arXiv:2408.01805, 2024. [Online]. Available: https://arxiv.org/pdf/2408.01805

[13] C. Min, S. Kashyap, S. Maass, W. Kang, and T. Kim, "Understanding manycore scalability of file systems," in *Proc. USENIX Annual Technical Conference (ATC)*, 2016. [Online]. Available: https://www.usenix.org/system/files/conference/atc16/atc16_paper-min.pdf

[14] J. Mohan, R. Kadekodi, and V. Chidambaram, "Analyzing io amplification in Linux file systems," *arXiv preprint arXiv:1707.08514*, 2017. [Online]. Available: https://ar5iv.labs.arxiv.org/html/1707.08514

[15] J. Kljajić, N. Bogdanović, M. Nankovski, M. Tončev, and B. Djordjević, "Performance analysis of 64-bit ext4, xfs and btrfs filesystems on the solid-state disk technology," in *INFOTEH-JAHORINA*, 2016. [Online]. Available: https://picture.iczhiku.com/resource/paper/SHIekZuUaeOqrVbb.pdf

[16] S. Jaffer, S. Maneas, A. Hwang, and B. Schroeder, "Evaluating file system reliability on solid state drives," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA: USENIX Association, Jul. 2019, pp. 783–798. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/jaffer

[17] R. Bravenboer, "Xfs vs ext4 performance," https://unix.stackexchange.com/questions/525613/xfs-vs-ext4-performance, June 2019, accessed: 2025-09-04.

[18] M. Kassab and M. Taha, "Linux filesystems: Ext4, btrfs, xfs, zfs and more," https://www.networkworld.com/article/3631604/linux-filesystems-ext4-btrfs-xfs-zfs-and-more.html, January 2025, accessed: 2025-09-04.

[19] M. Larabel, "Bcachefs, btrfs, ext4, f2fs & xfs file-system performance on crucial t705 pcie 5.0 nvme ssd," https://www.phoronix.com/review/linux-615-filesystems/6, May 2025, accessed: 2025-09-04.

[20] P. S. Blog, "Xfs vs. ext4: Which linux file system is better?" https://blog.purestorage.com/purely-educational/xfs-vs-ext4-which-linux-file-system-is-better/, January 2024, accessed: 2025-09-04.

[21] V. Tarasov, S. Bhanage, E. Zadok, and M. Seltzer, "Benchmarking file system benchmarking: It {* IS*} rocket science," in *13th Workshop on Hot Topics in Operating Systems (HotOS XIII)*, 2011.

[22] A. Traeger, E. Zadok, N. Joukov, and C. P. Wright, "A nine year study of file system and storage benchmarking," *ACM Transactions on Storage (TOS)*, vol. 4, no. 2, pp. 1–56, 2008.

[23] J. Mohan, D. Purohith, M. Halpern, V. Chidambaram, and V. J. Reddi, "Storage on your smartphone uses more energy than you think," in *Proc. USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2017. [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2021/10/hotstorage17-paper-mohan.pdf

[24] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320. [Online]. Available: http://dl.acm.org/citation.cfm?id=1298455.1298485

[25] "Gluster filesystem," https://github.com/gluster/glusterfs.

[26] "Lustre file system," http://lustre.opensfs.org/.

[27] J.-Y. Lee, M.-H. Kim, S. A. R. Shah, S.-U. Ahn, H. Yoon, and S.-Y. Noh, "Performance evaluations of distributed file systems for scientific big data in fuse environment," *Electronics*, vol. 10, no. 12, p. 1471, 2021.

[28] *HOBO® Plug Load Logger (UX120-018) Manual*, Onset Computer Corporation, http://www.onsetcomp.com/files/manual_pdfs/17838-E%20MAN-UX120-018.pdf.

[29] B. Harris and N. Altiparmak, "Ultra-low latency ssds' impact on overall energy efficiency," in *12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '20)*. USENIX Association, July 2020.

[30] S. Sundar, W. Simpson, J. Higdon, C. Whitaker, B. Harris, and N. Altiparmak, "Energy implications of io interface design choices," in *15th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '23)*, July 2023.

[31] C. Whitaker, S. Sundar, B. Harris, and N. Altiparmak, "Do we still need io schedulers for low-latency disks?" in *15th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '23)*, July 2023.

[32] B. Harris and N. Altiparmak, "When poll is more energy efficient than interrupt," in *14th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '22)*, July 2022.

[33] J. Axboe, "Fio - flexible i/o tester," https://github.com/axboe/fio, May 2025.

[34] J. Axboe, "Efficient io through io_uring," https://kernel.dk/io_uring.pdf, Oct 2019.

[35] "Rocksdb: A persistent key-value store for fast storage environments. facebook." https://rocksdb.org/.

[36] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807152

[37] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: a distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 15–15.

[38] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1773912.1773922

[39] V. Tarasov, E. Zadok, and S. Shepler, "Filebench: A flexible framework for file system benchmarking," *;login: The USENIX Magazine*, vol. 41, no. 1, pp. 6–12, March 2016.

Authorized licensed use limited to: University of Louisville. Downloaded on February 23,2026 at 20:36:44 UTC from IEEE Xplore. Restrictions apply.