BTRFS vs Ext4: Filesystems Impact on Energy Consumption of eMMC-Based Embedded Computers

Brenda Jacomelli
Institute of Mathematics and Computer Science (ICMC)
University of São Paulo (USP)
São Carlos, São Paulo, Brazil
brendajacomelli@usp.br

Sarita Mazzini Bruschi
Institute of Mathematics and Computer Science (ICMC)
University of São Paulo (USP)
São Carlos, São Paulo, Brazil
sarita@icmc.usp.br

***Abstract*—Advancements in microelectronics have expanded embedded systems' capacities, with a significant number now incorporating processing units supporting Linux-based distributions. In energy-constrained environments, selecting an energy-efficient filesystem for embedded devices is imperative, yet many performance evaluations focus solely on their time efficiency. To overcome these limitations, this paper proposes a performance evaluation on eMMC-based embedded computers to understand the impact of different filesystems on their energy consumption. We generated a basic Linux image and copied it to the eMMC storage twice. For each copy, we formatted the root filesystem partition with different filesystems (EXT4 and BTRFS) and conducted file operation benchmarks. Measurements of mean power during each benchmark execution were performed by an external microcontrolled device. From the results, it can be inferred that EXT4 is the best overall option for power saving when executing file operations.**

***Index Terms*—embedded, filesystems, energy consumption, Linux**

I. Introduction

Embedded Systems are computing systems designed, in terms of hardware and software, to perform a specific function. These systems are present in multiple essential industries, such as the medical equipment industry, the agricultural machinery industry, and others. Due to the heterogeneity of the projects that require embedded solutions, it is very common to see several proposed architectures. Some of them have a micro-processing unit (MPU) capable of supporting conventional operating systems, such as Linux-based distributions.

Despite the improving capacity that embedded devices achieved in the last years, it is important to note that they still possess limited computing resources compared to conventional computers. Besides, Internet of Things (IoT) technologies and other implementations often demand deployment in environments with constrained energy resources [1]. This underscores the importance of designing embedded systems that prioritize better time efficiency and energy consumption for optimal performance. When it comes to software design of Embedded Linux systems one important aspect to investigate is the I/O (input/output) performance of filesystems and its broader impact on the entire system.

Therefore, this paper proposes a performance evaluation of filesystems on eMMC-based embedded computers to understand the impact of different filesystems on their energy consumption. The evaluation focuses on essential file operations, such as sequential and random reading/writing, with throughput and energy consumption as key metrics. The chosen filesystems for this experiment are EXT4 and BTRFS, that are open-source and supported by the Linux kernel.

II. Related Works

In [2], the authors benchmarked the JFFS2, UBIFS, and YAFFS flash filesystems on an Armadeus APF27 embedded board. They evaluated (un)mount times, file operations, and compression impact. YAFFS excelled in file searches, UBIFS in file creation and mounting, and JFFS2 in file deletion. Compression reduced data size by up to 40% in JFFS2 and UBIFS but had little effect on YAFFS.

The study in [3] analyzed EXT4, NTFS, and BTRFS on a workstation with Solid State Drive (SSD) and Hard Disk Drive (HDD) using the Fio tool to assess sequential and random read/write operations under different Linux I/O schedulers. EXT4 consistently outperformed BTRFS on all SSD workloads.

In [4], EXT2, EXT3, ReiserFS, and XFS were evaluated for performance and energy consumption on a CentOS 5.3 server. Results showed that no single configuration fits all workloads, but tuning mount and format options improved energy efficiency by 5–149% and 6–136%, respectively.

Our study differs in focusing on Embedded Linux Computers with Embedded Multimedia Cards (eMMCs) as the main secondary storage device, which rarely compose the scenario of filesystems' evaluation. Besides, it not only considers the time efficiency of the file operations but also the energy consumption associated to their execution.

III. Flash Memories

Concerning embedded computers, it is crucial to consider secondary storage devices that are not only non-volatile but also boast low energy consumption, robustness, and compactness. Nowadays, this characteristics are typically achieved with the use of **NAND Flash Memories**. These devices store large amounts of data in small chips, operate on a single power supply, and have no moving parts. They can also be written and erased through software and usually outperform HDDs in speed. However, their write/erase mechanism limits robustness and data integrity: memory is divided into erase blocks, and changing a single bit from 0 to 1 requires erasing an entire block. This block-based write-cycle defines flash lifetime, and a suboptimal read/write algorithm can significantly accelerate their failure [5].

Wear Leveling, for example, is one of the most crucial mechanisms for increasing the NAND flash lifetime. This technique ensures that erase and write operations are distributed across all memory blocks as equally as possible, preventing the failure of certain cells from affecting the overall memory function. Additionally, having mechanisms to detect and correct errors, identify bad blocks, and prevent data loss during power failures is essential. Consequently, there are two approaches: using raw NAND Flash memories and relying on the OS and device drivers to handle these problems, or utilizing managed NAND. Managed Flash memories combine memory chips with a microcontroller that runs a firmware known as Flash Translation Layer (FTL), which is responsible for managing NAND access problems, facilitating communication with the entire system, and allowing the use of conventional block filesystems.

Both **Secure Digital Cards (SD Cards)** and **eMMCs** are managed NAND flash devices widely used in embedded microprocessor-based systems. Their hardware architectures are similar, typically employing a 4-wire **Serial Peripheral Interface (SPI)** for system access. The main difference lies in packaging: eMMCs are soldered directly to the board, while SD Cards use a removable connector. Memory quality depends not only on hardware but also on the firmware algorithms controlling access, which directly affect lifespan. This paper focuses on **eMMC-based embedded computers** due to their prevalence in embedded, mobile, and infotainment systems.

IV. Block Filesystems

Given that the read/write mechanism of flash memories differs from another types of secondary memory devices, implement it requires a bit more care from the OS. Therefore, not only the device driver must be suitable for this type of memory, but also the filesystem, due to its importance in file operation syscalls and permanent data safety. So, filesystems dedicated to flash devices can be called Flash Filesystems, some examples are JFFS2, YAFFS2 and UBIFS, and they consider important aspects such as write-cycle and wear leveling. Alternatively, we have Block Filesystems, that are appropriate for non-flash devices or, simply, block devices. They are conventional filesystems, utilized by HDDs and managed flash devices. Despite SSDs, SD Cards and eMMCs being flash-based, their controller allows the use of Block Filesystems since it handles and abstracts all the read/write cycles. The following subsections will delve into two Block Filesystems that will be utilized in this paper's performance evaluation.

A. EXT4

The EXT4, also known as the Fourth Extended Filesystem is, as indicated by its name, the fourth major rewrite of the Extended Filesystem (EXT), that has been the most used filesystem on Linux devices since 1992. Just like its predecessor, the EXT3, the fourth version is a journaled filesystem. The necessity of updating the third version of EXT emerged mainly by the fact that at the time that EXT4 was released, in kernel version v2.6.1 on 2008, the EXT3 was limited to filesystems of about 16 TB and enterprise workloads disks were already approaching this limit [6].

The fourth version of EXT not only upgraded the filesystem limit to 1EB with 4 KB block size, but also implemented a lot of new features that increased its file operation performance compared to its predecessor [7], such as **journal checksumming**, **discard/TRIM**, **fast fsck**, **delayed allocation** [8].

Besides all that, the EXT4 implementation enhances the compatibility with the EXT3, i.e EXT3 can be mounted as EXT4 without reformatting or reinstalling your OS and software environment. Popular Linux distributions, like Debian and Ubuntu, uses EXT4 as default.

B. BTRFS

The B-tree filesystem, better known as BTRFS, is a copy on write (COW) filesystem designed for Linux. Unlike journaled filesystems, it addresses crash-consistency problems by always allocating a new location for modified blocks. In other words, when a block is modified, BTRFS allocates a new location on disk, makes the modifications, writes it to the new location, and then frees the old location. The old location is only freed at the end of the process, ensuring that this filesystem always maintains its consistency [9].

As suggested by its name, BTRFS stores data using the b-tree data structure. Its structure is composed of nodes, leaves, and one or more levels. In each node, there is the disk location of the next-level nodes or leaves, and the actual data is stored in the leaves. One advantage of this implementation over other filesystems is that the inode of a file can be allocated next to the contents of that file. This significantly improves performance for operations with small files, as memory fragmentation can be avoided by allocating inodes in empty spaces [9]. Besides the important features that BTRFS shares with EXT4, such as **delayed allocation** and **TRIM/Discard**, it has implemented particular ones like **snapshot** and **checksumming** [10].

The Fedora distribution uses BTRFS as its default filesystem.

V. Materials

A. Embedded Computer

We conducted our performance evaluation on the embedded computer composed by the System on Module (SoM) Colibri i.MX6ULL and the carrier board Aster from Toradex manufacturer. This system is equipped with a NXP i.MX6ULL CPU (ARM architecture, one core and 900 MHz frequency clock) and a 1 GB DDR3L 16 bits RAM. The secondary storage is a 4 GB eMMC, the MX52LM04A11 chip manufactured by Macronix.

B. Energy Meter

The energy meter device is composed of the STM32 Black Pill board and the INA219 power monitor module. The communication between the module and the board is done by the I^2C interface, through the data and the clock wires, SDA and SCL respectively. The STM32 Black Pill is a board based on the STM32F411CEU6 microcontroller from STMicroeletronics. It features an Arm Cortex-M4 CPU, 512KB of flash, 128KB of RAM, multiple interfaces,and offers good cost-benefit, which makes it suitable for many applications.

The INA219 component from Texas Instruments is a bidirectional current sensor that can be used to monitor the power of buses up to 26V. An external resistor called a shunt resistor is used for its operation. One of its terminals is connected to the current supply and to the IN+ terminal of the INA219 and the load and IN- terminal are connected to the other.

The voltage between the terminals of the shunt resistor is known as the shunt voltage. Similarly, the voltage measured between IN- and ground is referred to as the bus voltage [11]. In this project, we are using a module based on the INA219 with a shunt resistor of $100\,m\Omega$. It has a I^2C interface and measures currents up to $3.2\,A$, in the positive and negative directions, with a resolution of $0.8\,mA$.

VI. Methods

A. File Operations

We used Fio [12], a flexible synthetic benchmark tool, to assess the I/O performance of EXT4 and BTRFS. Both sequential and random read and write operations were tested. To enable energy consumption measurement, we modified the tool with a patch to notify the Energy Meter before and after each test, through a UART interface. Fio was configured with the following parameters: a single thread, 500MB I/O size and the block size (the amount of data transferred in each I/O operation) ranging from 16KB to 64KB. We generated a basic Linux image with two partitions (boot and root filesystem) and copied it to the eMMC storage with the root filesystem of interest. Each operation's throughput benchmark was performed only once per filesystem, so the results represent single-run measurements. The Linux image that ran on the Embedded Computer was generated using the Yocto Project, with the kernel version 6.1.42. Both filesystems, ext4 and BTRFS, were mounted using the kernel's default parameters, without additional mount options.

B. Energy Consumption

In order to measure the power delivered by the Aster power supply, which provides $5\,V/3\,A$, we needed to rip the VCC wire, connect the IN+ terminal of the INA219 in the supply side and the IN- terminal in the connector side.

The energy meter firmware recorded the instant power every millisecond and the noise from current measurements was mitigated by a 10-position moving average filter. All the measured data was accessible through another UART interface. The initiation or termination of an I/O benchmarking are communicated to the meter and forwarded to the user. The device firmware was developed using the Zephyr RTOS.

Using all power samples collected during the operation, it was possible to estimate the average power and subsequently the energy consumed to perform a file operation for each Mebibyte (MiB) transferred:

$$E = \int_{t_1}^{t_2} p(t), dt \approx \bar{P} \cdot (t_2 - t_1) \approx \sum_{n=1}^{N} P_n \cdot \Delta t \quad [J] \tag{1}$$

Given that:

- P_n is the instantaneous power at sample n
- Δt is the time interval considered to calculate the energy
- N is the total number of samples

Since the benchmark returns the throughput (T_{hrp}) in Mebibytes per second (MiB/s), it is possible to compute the energy per MiB by evaluating the time required to process one MiB:

$$E_{MiB} = \sum_{n=1}^{N} P_n \cdot \frac{1}{T_{hrp}} \quad \left[\frac{J}{\text{MiB}}\right] \tag{2}$$

VII. Results and Discussion

As shown in Tables I and II, EXT4 surpasses BTRFS in throughput for all tested file operations and block sizes, processing at least 6% more data per second. For sequential and random reads, EXT4 reaches up to 86% higher throughput. This difference results from BTRFS's internal structure, which includes COW behavior, metadata duplication, and checksumming. While these mechanisms enhance data safety, they add latency and reduce throughput, especially in workloads with frequent small writes such as logging.

Both filesystems showed higher read than write performance, as expected for eMMC-based systems, since eMMCs are controlled NAND flash devices requiring block erasure before writing. Larger block sizes improved read performance for both, reducing I/O overhead and enhancing efficiency, particularly in sequential operations.

As seen in Tables III and IV, mean power remained stable across block sizes, with low standard deviation. Power depends mainly on operation type and filesystem mechanisms rather than block size, which affects execution time but not the energy consumed per second. BTRFS consistently showed higher mean power than EXT4 in all operations.

According to Tables V and VI, derived as described in Subsection VI-B, BTRFS also consumed more energy per 1 MiB transferred in all operations. Its higher mean power leads to greater total energy usage, confirming EXT4's superior efficiency in both power and energy consumption.

Although these results come from eMMC-based hardware under specific workloads and block sizes, and may not represent all scenarios, they indicate that EXT4 is better suited for energy-constrained systems. BTRFS, despite lower efficiency, provides valuable features such as snapshotting, checksumming, and enhanced data integrity. Therefore, the choice between EXT4 and BTRFS should balance energy efficiency and the reliability features required by each application.

TABLE I
FILE OPERATION THROUGHPUT FOR EXT4 (MiB/s)

Block size	**ext4**			
	read	**write**	**random read**	**random write**
16K	44.8	12.5	14.3	9.9
32K	44.3	12.4	20.0	10.9
64K	43.4	12.5	25.3	12.0

TABLE II
FILE OPERATION THROUGHPUT FOR BTRFS (MiB/s)

Block size	**BTRFS**			
	read	**write**	**random read**	**random write**
16K	24.0	11.8	10.4	8.5
32K	37.6	11.5	13.8	8.2
64K	41.00	11.3	15.6	10.9

TABLE III
MEAN POWER FOR EXT4 FILESYSTEM OPERATIONS (WATTS)

Block size	**ext4**			
	read	**write**	**random read**	**random write**
16K	1.26±0.6	1.15±0.06	1.20±0.01	1.15±0.07
32K	1.26±0.07	1.15±0.07	1.21±0.02	1.15±0.07
64K	1.26±0.07	1.15±0.07	1.20±0.03	1.15±0.07

Values are expressed as mean ± standard deviation over the samples.

TABLE IV
MEAN POWER FOR BTRFS FILESYSTEM OPERATIONS (WATTS)

Block size	**BTRFS**			
	read	**write**	**random read**	**random write**
16K	1.37±0.9	1.19±0.07	1.23±0.02	1.23±0.10
32K	1.40±0.08	1.19±0.08	1.24±0.06	1.21±0.11
64K	1.39±0.10	1.18±0.07	1.22±0.06	1.21±0.09

Values are expressed as mean ± standard deviation over the samples.

VIII. CONCLUSION

In this paper, we presented key data on the impact of different root filesystems (EXT4 and BTRFS) on the energy consumption of eMMC-based embedded Linux systems. EXT4 consistently outperforms BTRFS in throughput, power, and energy consumption across various block sizes and file operations. Although BTRFS provides advanced features such as data integrity and snapshot capabilities, these come with additional performance and energy overheads that may limit its suitability in energy-constrained environments. Therefore, EXT4 is the recommended filesystem for embedded systems using controlled NAND flash storage when energy efficiency is critical. However, there is no one-size-fits-all solution. This study focuses on a specific use case and aims to assist developers and engineers in selecting the most suitable filesystem for their particular needs. Future work could explore optimizations for BTRFS, consider data safety mechanisms, evaluate other filesystem candidates, and include repeated benchmark runs to enable more robust statistical analysis.

TABLE V
ENERGY CONSUMPTION PER MIB TRANSFERRED FOR EXT4 (mJ/MiB)

Block size	**ext4**			
	read	**write**	**random read**	**random write**
16K	28.12	92.00	83.92	116.16
32K	28.44	92.74	60.50	105.50
64K	29.03	92.00	47.43	95.83

TABLE VI
ENERGY CONSUMPTION PER MIB TRANSFERRED FOR BTRFS (mJ/MiB)

Block size	**BTRFS**			
	read	**write**	**random read**	**random write**
16K	57.08	100.85	118.27	144.03
32K	37.23	103.48	89.86	147.15
64K	33.90	105.31	78.21	111.01

REFERENCES

[1] C. Guo, S. Ci, Y. Zhou, and Y. Yang, "A survey of energy consumption measurement in embedded systems," *IEEE Access*, vol. 9, pp. 60516–60530, 2021.

[2] P. Olivier, J. Boukhobza, and E. Senn, "On benchmarking embedded linux flash filesystems," *ACM SIGBED Rev.*, vol. 9, no. 2, pp. 43–47, 2012.

[3] G. Rakshith *et al.*, "Performance analysis of secondary storage media through filesystems benchmarking," in *Proc. 3rd Int. Conf. Trends in Electronics and Informatics (ICOEI)*, 2019, pp. 1222–1226.

[4] P. Sehgal, V. Tarasov, and E. Zadok, "Evaluating performance and energy in filesystem server workloads," in *Proc. USENIX Conf. File and Storage Technol. (FAST)*, 2010, pp. 253–266.

[5] Christopher Hallinan. *Embedded Linux Primer: A Practical Real-World Approach*. 2nd edition, Prentice Hall Press, USA, 2010. ISBN: 0137017839.

[6] Avantika Mathur, Mingming Cao, Suparna Bhattacharya, Andreas Dilger, Alex Tomas, and Laurent Vivier. The new ext4 filesystem: current status and future plans. In *Proceedings of the Linux symposium*, volume 2, pages 21–33, 2007.

[7] Mingming Cao, Suparna Bhattacharya, and Ted Ts'o. Ext4: The Next Generation of Ext2/3 Filesystem. In *LSF*, 2007.

[8] *Ext4 (and Ext2/Ext3) Wiki*, 2019. Available at: https://ext4.wiki.kernel.org/index.php/Ext4_Howto#EXT4_features. Accessed: 2024-01-19.

[9] Josef Bacik. Btrfs: the Swiss army knife of storage. *USENIX Login*, 37:7–15, 2012.

[10] *BTRFS documentation* Available at: https://btrfs.readthedocs.io/en/latest/. [Accessed: Jun. 02, 2025].

[11] Texas Instruments. *INA219 Zerø-Drift, Bidirectional Current/Power Monitor With I2C Interface*. Technical Report SBOS448G, March 2015. Rev. 7.

[12] *fio – Flexible I/O tester* Available at: https://fio.readthedocs.io/en/latest/. [Accessed: Jun. 02, 2025].