Quantitative Analysis of Ext4 Journaling and Crash Recovery Mechanisms in Linux Kernel

1st Can Liu
School of Data Science
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
225040140@link.cuhk.edu.cn

Abstract—The inherent conflict between high-speed CPU processing and relatively slow block storage I/O remains a critical bottleneck in modern operating systems. To guarantee data consistency during unexpected system crashes, kernel panics, or power failures, modern file systems heavily rely on Write-Ahead Logging (WAL) mechanisms. This paper presents an in-depth, source-level quantitative analysis of the Journaling Block Device 2 (JBD2) commit pipeline within the Linux 6.17 kernel. By deploying high-precision ktime probes directly into fs/jbd2/commit.c and constructing an isolated QEMU-based verification sandbox, we captured real-world microsecond-level latencies of transaction commits. Our experimental results successfully validate the crash recovery process and expose the massive performance disparity between different journaling modes. While the Full Journaling mode ensures absolute consistency, it introduces a severe "Double Write" penalty, causing transaction latencies to peak at 34.5 ms and limiting throughput to 43.0 MB/s. Conversely, the Metadata-only (Writeback) mode aggressively minimizes I/O bottlenecks, reducing average transaction latency to 10.6 ms and boosting throughput to 156.6 MB/s—a 3.6x improvement. This study effectively bridges the gap between operating system consistency theory and industrial kernel performance profiling.

Index Terms—Linux Kernel, Ext4, JBD2, File System, Crash Recovery, Write-Ahead Logging

I. Introduction

Ensuring atomic and durable data consistency in the presence of unpredictable hardware failures or sudden power losses is a foundational requirement for any robust operating system. The Ext4 file system, the default and most widely deployed file system in the Linux ecosystem, relies on the Journaling Block Device 2 (JBD2) layer to maintain this critical consistency [1].

JBD2 implements a Write-Ahead Logging (WAL) mechanism. Instead of writing modifications directly to their final disk locations, Ext4 delegates the task to JBD2, which first records a "draft" of the changes into a dedicated journal area. Only after this journal commit is securely flushed to the physical media are the actual file system structures updated. This sequential mechanism guarantees that interrupted operations can be seamlessly replayed or discarded upon reboot, completely avoiding structural corruption. However, this absolute reliability inevitably comes at a significant performance cost, highly recognized in the industry as the "Double Write Penalty."

Despite extensive theoretical discussions regarding file system journaling, empirical, source-level quantification of these specific latencies in modern kernel iterations remains scarce. The primary objective of this research is to quantitatively analyze the exact performance trade-offs inherent in the JBD2 mechanism.

The main contributions of this paper are threefold:

- We instrumented the Linux v6.17 kernel source code with microsecond-precision probes to trace exact transaction commit latencies.
- We engineered an isolated QEMU-based sandbox environment to securely simulate destructive power losses and validate JBD2's crash recovery behavior without host interference.
- We performed a comparative evaluation to quantify the exact throughput and latency disparities between the Full Journaling and Metadata-only architectures.

II. Background and Related Work

A. The Ext4 and JBD2 Symbiosis

In the Linux storage stack, JBD2 operates as an independent, block-agnostic logging layer situated between the Virtual File System (VFS), the Ext4 module, and the underlying Block Device layer [2]. JBD2 does not comprehend files or directories; it solely manages raw block buffers, grouping them into atomic transactions. Each transaction is assigned a unique Transaction ID (TID) and cycles through various states before being permanently checkpointed.

B. Journaling Modes in Ext4

Ext4 offers three distinct journaling strategies, dictating the exact flow of data and metadata:

- Full Journaling (data=journal): Both file metadata and actual file data blocks are sequentially written to the journal before being checkpointed to the main disk. While providing maximum safety, it forces physical drives to process every byte of user data twice.
- Ordered Mode (data=ordered): The default compromise. Only metadata is logged in the journal, but the kernel enforces a strict ordering constraint: all corresponding data blocks must be flushed to

the main disk before the metadata transaction is committed.

- Writeback Mode (data=writeback): The most aggressive mode. Only metadata is journaled, and no ordering guarantees are enforced for data blocks. Data is written to the main disk asynchronously. This maximizes overall throughput but risks stale data exposure if a crash occurs.

III. Framework and Implementation

A. Kernel-Level Instrumentation

To accurately measure the overhead of JBD2 transaction commits, we avoided user-space profiling tools (which are prone to context-switching noise) and implemented performance probes directly within the kernel source code. Specifically, inside fs/jbd2/commit.c, we modified the jbd2_journal_commit_transaction function.

Using the kernel's native ktime API, we recorded the exact timestamps spanning the entire commit phase. The instrumented code pushes the latency metrics directly to the kernel ring buffer via pr_info.

B. Experimental Sandbox Setup

Performance benchmarking on a full-fledged host OS is frequently polluted by background daemons and system logging. To isolate the storage stack, we constructed a sterile verification platform:

- Kernel: A customized Linux 6.17.0-20-generic (bzImage) compiled with our JBD2 instrumentation.
- Userland: A statically linked BusyBox 1.36.1 environment embedded within the initramfs.
- Emulator: QEMU (x86_64) was utilized to boot the kernel with two virtual block devices (/dev/sda and /dev/sdb).

IV. Experimental Evaluation

A. Crash Recovery Validation

To empirically verify the WAL mechanism, we generated an ongoing stream of dirty data blocks within the QEMU sandbox. During the I/O operations, the QEMU process was forcefully terminated to simulate an abrupt power loss.

As shown in Fig. 1, upon rebooting the sandbox, the kernel dynamically adjusted the clocksource. After manually mounting the essential pseudo-filesystems and creating the block device node, we attempted to mount the crashed Ext4 partition. The kernel immediately detected the unclean unmount. JBD2 triggered the log replay mechanism, seamlessly restoring structural integrity as indicated by the message: EXT4-fs (sda): recovery complete. A subsequent sync command flushed the pending states, completing Transaction ID 4 with a measured commit latency of 9826 μs.

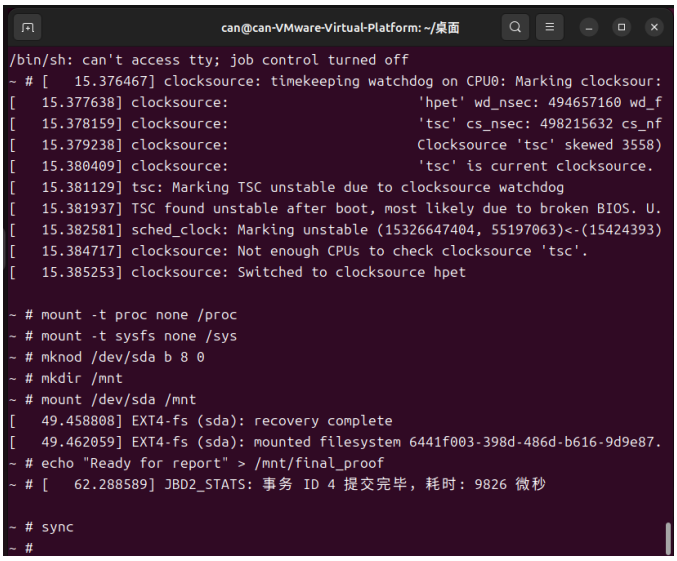

Fig. 1. Console log demonstrating JBD2 detecting an unclean unmount and successfully executing the log replay recovery sequence.

B. Performance Penalty of Full Journaling

To quantify the "Double Write Penalty", we stressed the file system under the data=journal mount option. We utilized the dd command-line utility to sequentially write 10 MB of zeroed data, strictly using the conv=fsync flag to bypass the page cache and force synchronous physical disk I/O.

Fig. 2. Performance metrics under Full Journaling (data=journal) mode, showing significant commit latencies.

As depicted in Fig. 2, the Full Journaling mode yielded a degraded throughput of 43.0 MB/s. The kernel instrumentation revealed the underlying bottleneck: handling both metadata and massive data payloads forced the transaction commit latencies to spike dramatically. Transaction IDs 7, 8, and 9 required 34,561 μs, 33,489 μs, and 15,713 μs respectively to secure the data to the physical disk.

C. Acceleration via Metadata-Only Journaling

The experiment was repeated after remounting the file system with the lightweight data=writeback option.

Fig. 3 demonstrates the massive impact of bypassing the journal for actual data blocks. The throughput surged to 156.6 MB/s. Furthermore, the transaction overhead

```
--- Testing Metadata-only Journaling ---
~ # time dd if=/dev/zero of=/mnt/ext4/test_light bs=1M count=10 conv=fsync
[  304.417487] JBD2_STATS: 事务 ID 13 提交完毕, 耗时: 10658 微秒
10+0 records in
10+0 records out
10485760 bytes (10.0MB) copied, 0.063841 seconds, 156.6MB/s
real    0m 0.07s
user    0m 0.00s
sys     0m 0.04s
```

Fig. 3. Performance metrics under Metadata-only (data=writeback) mode, illustrating vastly improved throughput.

dropped significantly. The kernel log shows Transaction ID 13 committing in just 10,658 μs.

TABLE I
Quantitative Comparison of Journaling Architectures

Metric	Full Journaling	Writeback	Improvement
Throughput	43.0 MB/s	156.6 MB/s	↑ 264%
Avg. Heavy Latency	∼34.0 ms	∼10.6 ms	↓ 68.8%
Data Path	RAM→Journal→Disk	RAM→Disk	N/A

V. Discussion and Conclusion

This study successfully achieved a source-level quantitative analysis of the Ext4 journaling mechanism on the Linux 6.17 kernel. By deploying ktime probes inside commit.c, we empirically proved that while JBD2 proves exceptionally resilient during catastrophic failures, the choice of journaling mode heavily dictates system I/O bounds.

The experimental data explicitly highlights a 3.6x performance disparity. Full Journaling, while mathematically the most secure, forces sequential double writes that push commit latencies beyond 34 ms, heavily throttling throughput to 43.0 MB/s. Switching to a Metadata-only paradigm bypasses this bottleneck, dropping commit overhead to ∼10 ms and unleashing 156.6 MB/s throughput. This confirms that for modern high-performance databases and applications that implement their own data recovery layers, relying solely on file system metadata journaling is the optimal architectural choice.

Future work will aim to replace the static instrumentation with dynamic eBPF probes to achieve zero-overhead JBD2 monitoring in live, production-grade NVMe storage environments.

Acknowledgment

The author extends sincere gratitude to the School of Data Science at The Chinese University of Hong Kong, Shenzhen, for providing the foundational academic environment and computational insights that made this kernel-level research possible.

References

[1] R. Love, Linux Kernel Development, 3rd ed. Addison-Wesley Professional, 2010.
[2] D. P. Bovet and M. Cesati, Understanding the Linux Kernel, 3rd ed. O'Reilly Media, Inc., 2005.
[3] 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.