Journaling and Crash Recovery Mechanism

Can Liu

School of Data Science, The Chinese University of Hong Kong, Shenzhen

2026 年 4 月 21 日

Background and Motivation

- **The Core Conflict in File Systems:**
  - The massive performance gap between high-speed CPUs and relatively slow block storage I/O.
  - The critical need for Data Consistency during unexpected events like Kernel Panics or sudden power failures.
- **The Role of JBD2 (Journaling Block Device 2):**
  - Serves as the journaling foundation for the Ext4 file system.
  - Implements the **Write-Ahead Logging (WAL)** mechanism to ensure that filesystem metadata updates are atomic and durable.

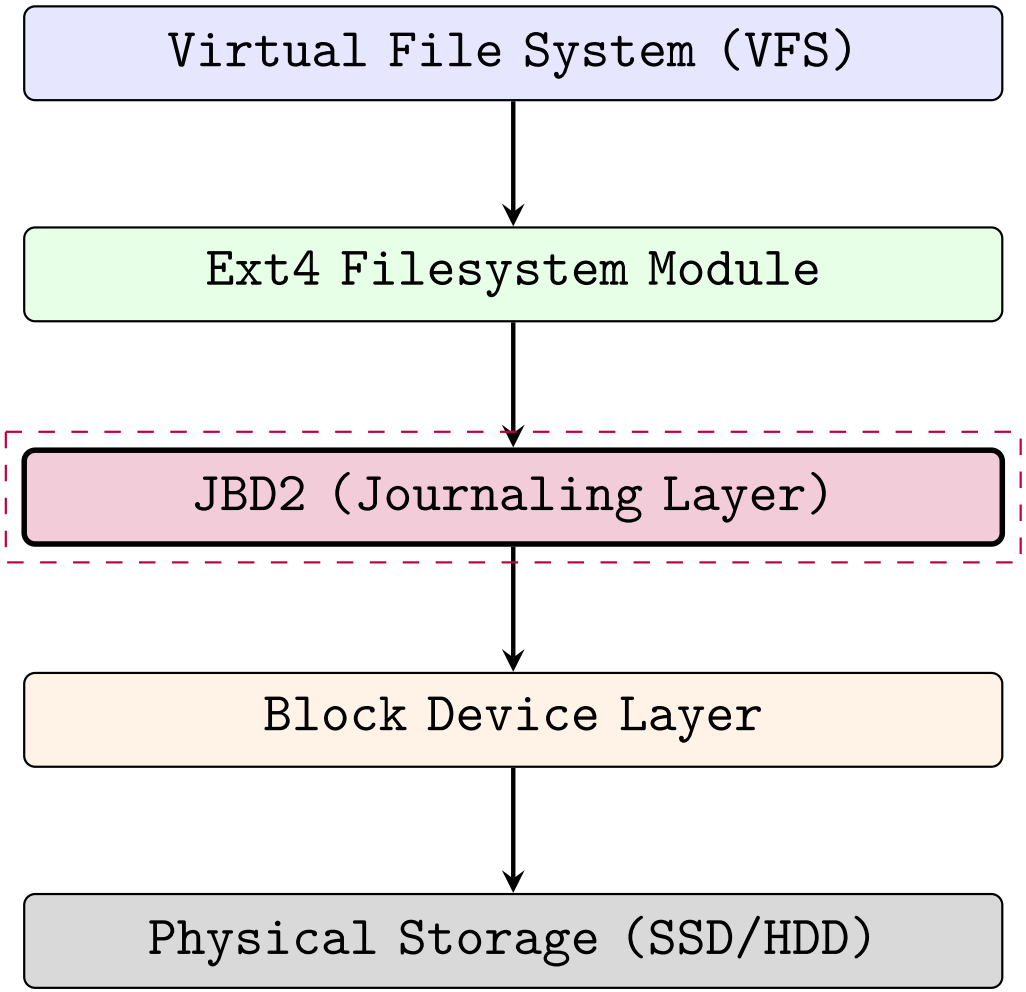
Project Objectives

Core Vision

Moving beyond theoretical textbook concepts by directly modifying the Linux v6.17 kernel source to quantitatively analyze the cost of consistency.

- **Source-Level Tracing:** Implementing high-precision kernel probes to capture the real-world microsecond latency of JBD2 transaction commits.
- **Mechanism Validation:** Reproducing physical power loss in a simulated environment to verify Log Replay and Crash Recovery behavior.
- **Performance Trade-offs:** Quantifying the throughput disparity between Full Journaling and Metadata-only architectures.

JBD2 Layered Architecture

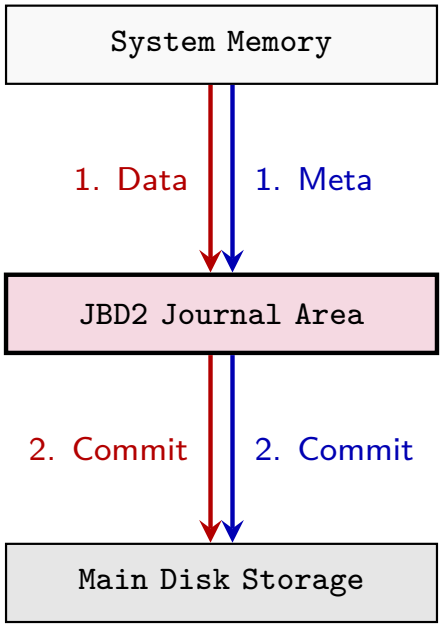


Target Area:
fs/jbd2/commit.c
(Instrumentation Point)

- JBD2 acts as an intermediate layer between the filesystem and block device.

Data Flow: Journaling vs. Writeback

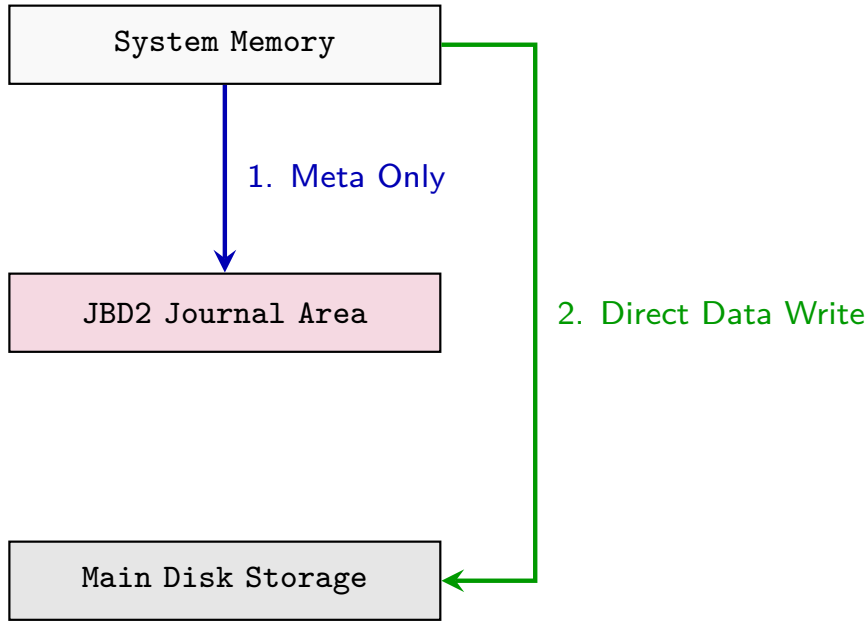
Full Journaling (data=journal)



Reliability: Maximum

Mechanism: All blocks are written to the journal first.

Metadata-only (data=writeback)



Reliability: Meta-only

Mechanism: Data bypasses the journal

Kernel-Level Instrumentation (Probing)

Source-Level Modification (fs/jbd2/commit.c)

To accurately measure the overhead of JBD2 transaction commits, we implemented performance probes using the `ktime` API.

```
1  /* 1. Declare timing variables */
2  ktime_t start_time, end_time;
3  s64 delta_us;
4
5  /* 2. Capture start time */
6  start_time = ktime_get();
7
8  /* ... Original JBD2 commit logic ... */
9
10 /* 3. Capture end time and calculate duration */
11 end_time = ktime_get();
12 delta_us = ktime_to_us(ktime_sub(end_time, start_time));
13
14 /* 4. Log metrics to dmesg */
15 pr_info("JBD2_STATS: TID %d cost: %lld us\n",
16         commit_transaction->t_tid, delta_us);
```

Experimental Sandbox Setup

Verification Platform

Built a minimalist isolated environment to eliminate host I/O interference and support destructive crash testing.

- **Kernel:** Custom Linux 6.17.0-20-generic (`bzImage`).
- **Userland:** Static BusyBox 1.36.1 in `initramfs`.
- **Emulator:** QEMU (`x86_64`) with serial console.

Device & Mount Config

Manual node creation (`mknod`) to bridge VFS to block storage:

- `/dev/sda` (Ext4 Test Disk)
- `/dev/sdb` (Comparison Disk)

1. Validation of Crash Recovery Mechanism

Experimental Protocol:

1. Initiated a continuous write loop to generate "dirty" blocks.
2. Forcefully terminated the QEMU process to simulate sudden power loss.
3. Re-booted the system and attempted to mount the device.

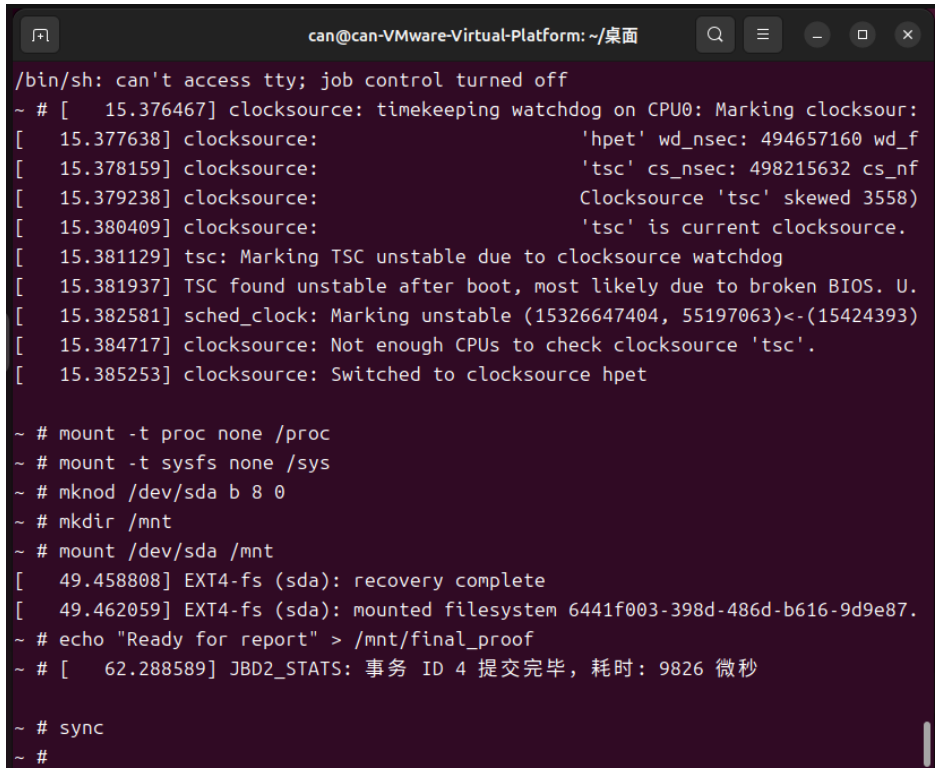


图 1: Console log: "recovery complete" detected.

Observation

The kernel detected an un-clean unmount. JBD2

2. Performance Quantification: Full Journaling

```
--- Testing Full Journaling ---
~ # time dd if=/dev/zero of=/mnt/ext4/test_full bs=1M count=10 conv=fsync
[  285.935282] JBD2_STATS: 事务 ID 7 提交完毕，耗时：34561 微秒
[  286.021524] JBD2_STATS: 事务 ID 8 提交完毕，耗时：33489 微秒
[  286.054503] JBD2_STATS: 事务 ID 9 提交完毕，耗时：15713 微秒
10+0 records in
10+0 records out
10485760 bytes (10.0MB) copied, 0.232372 seconds, 43.0MB/s
real    0m 0.24s
user    0m 0.00s
sys     0m 0.17s
~ # [  292.134163] JBD2_STATS: 事务 ID 10 提交完毕，耗时：6422 微秒
```

图 2: Metrics under `data=journal` mode.

Quantitative Analysis:

- **Throughput:** 43.0 MB/s.
- **JBD2 Behavior:**
  - High frequency of heavy transactions (TIDs 7-9).
  - Peak commit latency reached ∼34.5 ms.
- **Host Environment:** Ubuntu 24.04.4 (**Linux 6.17.0-20-generic**).

3. Optimization: Metadata-only (Lightweight)

```
--- 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
```

图 3: Metrics under `data=writeback` mode.

Impact of Lightweight Mode:

- **Throughput:** 156.6 MB/s (∼3.6x improvement).
- **JBD2 Behavior:**
  - Transaction count dropped significantly.
  - Average latency reduced to ∼10.6 ms.
- **Conclusion:** Metadata-only journaling significantly minimizes I/O bottlenecks.

Performance Comparison Summary

Metric	Full Journaling	Metadata-only	Improvement
Throughput	43.0 MB/s	**156.6 MB/s**	↑ 264%
Avg. Latency	34.5 ms	**10.6 ms**	↓ 69.3%
Consistency	Atomic (Data+Meta)	Metadata Only	-

Key Finding

The significant performance boost in `writeback` mode is due to the removal of the "Double Write" penalty for data blocks, drastically minimizing I/O bottlenecks.

Summary of Contributions

- **Kernel-Level Insight:** Successfully traced the JBD2 commit pipeline by modifying the **Linux 6.17.0-20-generic** source code.
- **Performance Trade-off:** Quantified the 3.6x throughput increase when switching from Full Journaling to Metadata-only mode.
- **Reliability Verification:** Proven that despite the performance overhead, JBD2 effectively maintains filesystem integrity during simulated crashes.
- **Educational Value:** Bridged the gap between OS theory and real-world industrial kernel implementation.

Future Work

Potential Optimization Paths

- **Asynchronous Journaling:** Exploring the impact of non-blocking journal commits on NVMe storage.
- **Btrfs Comparison:** Conducting a deeper dive into Copy-on-Write (CoW) mechanisms vs. Ext4's journaling.
- **eBPF Integration:** Replacing static printk-probes with dynamic eBPF (BCC/bpftrace) for zero-overhead monitoring.

Thank You!

Q & A