Operating Systems Project: Topic 14

File System Performance Characterization (Ext4 vs. XFS vs. Btrfs)

Liangsen Wang

224040364@link.cuhk.edu.cn

2026.2.3

Outline

Outline

The Mission: Topic 14 Requirements

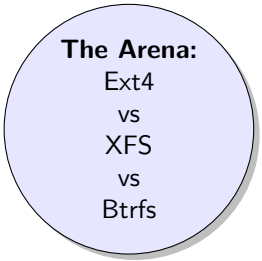
Objective: Evaluate and compare the performance of Linux's major file systems under various workloads.

Requirement 1: Workload Generation

- **Tools:** Use industry-standard tools: `fio`, `sysbench`, and `dd`.
- **Scenarios:** Generate **Sequential** and **Random** Read/Write patterns.

Requirement 2: Measurement

- **Metrics:** Measure **Throughput (MB/s)**, **IOPS**, and **Latency**.
- **Resource:** Analyze **CPU Usage** (User vs System) during tests.



Advanced Options

Option A: Parameter Tuning

- **Mount Options:** Tune parameters like `barrier=0`, `noatime`, or `data=writeback`.
- **Goal:** Quantify the performance gain vs. safety trade-off.

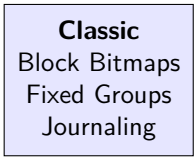
Option B: Custom Workloads

- **Scenario 1:** "The Mail Server" (Many tiny files, random sync writes).
- **Scenario 2:** "The Video Editor" (Large sequential reads/writes).
- **Analysis:** Which FS wins in which scenario?

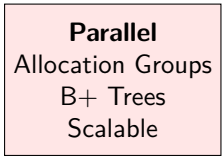
Outline

Theory 1: The Contenders (Architecture Overview)

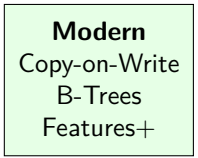
Why do they perform differently? It's all about data structures.

Ext4 

The "Generalist"

XFS 

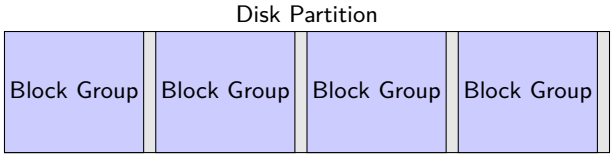
The "High-Perf"

Btrfs 

The "Feature-Rich"

Theory 2: Ext4 Mechanics (The Robust Standard)

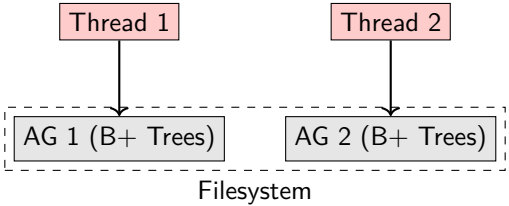
Structure: Block Groups.



Pros: Proven stability, low CPU usage.
Cons: Locking contention in Block Groups limits parallelism.

Theory 3: XFS Mechanics (The Parallel Beast)

Structure: Allocation Groups (AGs) are independent.

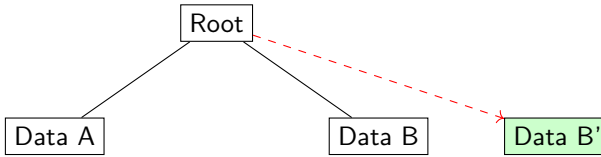


Key Advantage: Parallel I/O. Thread 1 writes to AG1 while Thread 2 writes to AG2. **No Locking Conflict.**

Theory 4: Btrfs Mechanics (Copy-on-Write)

Structure: Everything is a B-Tree (CoW).

- **Write:** Never overwrite in place.
- **1.** Write new data to free space.
- **2.** Update tree pointers up to root.
- **3.** Atomic Switch.



Trade-off: High CPU cost (checksums) and fragmentation, but great features (Snapshots).

Outline

Methodology 1: FIO (Flexible I/O Tester)

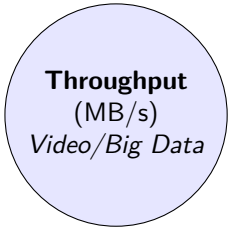
Don't rely on `dd`. Use `fio` for scientific measurement.

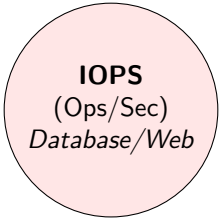
Anatomy of an FIO Job

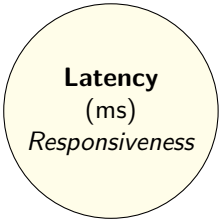
```
fio --name=seq_write --ioengine=libaio --rw=write --bs=1M --numjobs=4 --size=10G
--direct=1
```

- **ioengine:** How we talk to kernel (sync, libaio, io_uring).
- **rw:** Pattern (read, write, randread, randwrite).
- **bs:** Block Size (4k for random, 1M for sequential).
- **direct:** Bypass Page Cache (Measure Disk) vs Buffered (Measure Kernel).

Methodology 2: Key Metrics



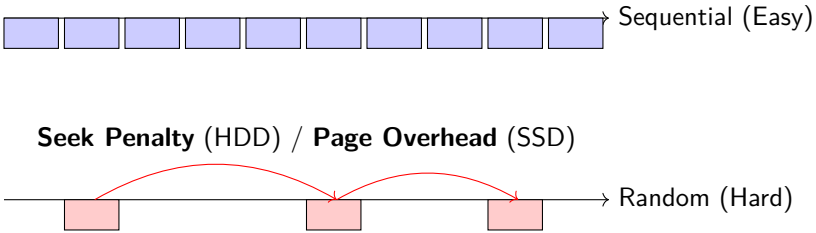




Note: Throughput = IOPS $\times$ Block Size.

Outline

Workload 1: Sequential vs Random



Hypothesis: XFS should win Sequential (Scalability). Ext4 might handle Random better on smaller systems.

Workload 2: Metadata Stress (The Mail Server)

Creating 1 million 1KB files.

Bottleneck:

- It's not data writing.
- It's **Journaling** and **B-Tree Updates**.
- Creating Inodes, updating Directories.

Expectation: Btrfs might struggle here due to CoW metadata overhead.

Roadmap

1. **Setup:** Create 3 partitions (10GB each). Format with Ext4, XFS, Btrfs.
2. **Baseline:** Run `fio` with default settings (Seq Write, Rand Read).
3. **Stress:** Run "Small File" test (use `sysbench fileio`).
4. **Tuning:** Remount with `noatime, barrier=0` and re-run.
5. **Report:** Plot Bar Charts (IOPS) and Box Plots (Latency).

Resources

Tools:

- `man fio`: The bible of I/O benchmarking.
- `iostat -x 1`: Monitor disk utilization live.

References:

- *"Filesystem benchmarking utility using FIO"*.
- Kernel Documentation.