CJFS: Concurrent Journaling for Better Scalability

21st USENIX Conference on File and Storage Technologies

Joontaek Oh; Seung Won Yoo; Hojin Nam; Changwoo Min and Youjip Won

Paper Notes

By JeongHa Lee

Abstract

In this paper, we propose CJFS, Concurrent Journaling Filesystem. CJFS extends EXT4 and addresses the fundamental limitations of the EXT4 journaling design, which are the main cause of the poor scalability of EXT4. The heavy-weight EXT4 journal suffers from two limitations. First, the journal commit is a strictly serial activity. Second, the journal commit uses the original page cache entry, not the copy of it, and subsequently any access to the in-flight page cache entry is blocked. To address these limitations, we propose four techniques, namely Dual Thread Journaling, Multi-version Shadow Paging, Opportunistic Coalescing, and Compound Flush. With Dual Thread design, CJFS can commit a transaction before the preceding journal commit finishes. With Multi-version Shadow Paging, CJFS can be free from the transaction conflict even though there can exist multiple committing transactions. With Opportunistic Coalescing, CJFS can mitigate the transaction lock-up overhead in journal commit so that it can increase the coalescing degree – i.e., the number of system calls associated with a single transaction – of a running transaction. With Compound Flush, CJFS minimizes the number of flush calls. CJFS improves the throughput by 81%, 68% and 125% in filebench varmail, dbench, and OLTP-Insert on MySQL, respectively, against EXT4 by removing the transaction conflict and lock-up overhead.

Problem Statement and Research Objectives

Journaling in EXT4

- **Block granularity physical logging**: leaves the EXT4 journaling under frequent transaction conflict and subsequently under scalability failure.
  - **Running transaction**: EXT4 maintains a set of page cache entries that need to be logged to the disk for journaling.
  - syscall updates the filesystem metadata ➔ acquires a lock(e.g., directory mutex) and obtains the journal handle ➔ the application modifies page cache entries ➔ inserts the updated page cache entries to the running transaction ➔ commits the running transaction either periodically or by the explicit fsync()
  - **Transaction conflict**: The application that needs to update the filesystem state is blocked if the associated page cache entry is being committed to the disk.
- **Serial journal commit**
  - EXT4 allocates a separate thread for journal commit, JBD thread. JBD thread can commit the following journal transaction only after the preceding journal transaction becomes durable.

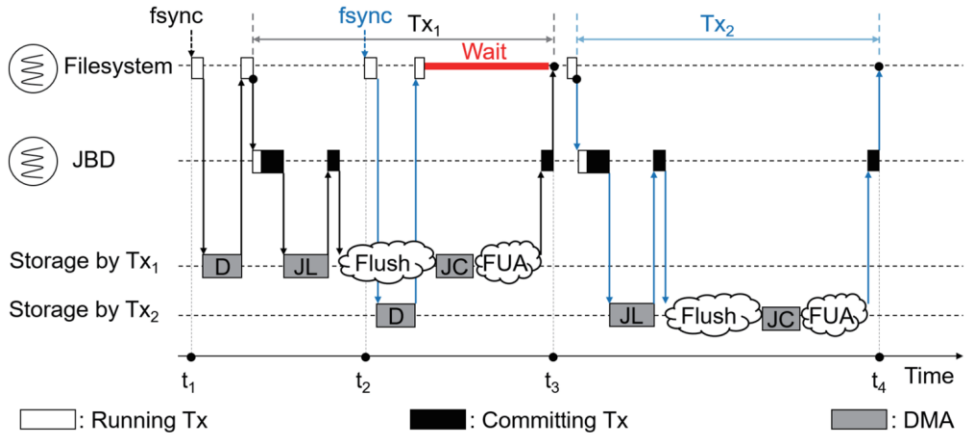


Figure 1: `fsync()` in EXT4

Problem Statement and Research Objectives

EXT4 journaling phases

1. **Coalescing in Running Transaction**: the application can modify the metadata and insert the associated page cache entry to the running transaction. (EXT4: compound transaction = group commit)
2. **Transaction Lock-Up in Running Transaction**: When the JBD thread needs to commit the running transaction, it stops issuing the journal handle to prohibit the new file operation from modifing the running transaction. Once all ongoing file operations with a handle finish, JBD thread changes the transaction state from running to committing.
   - In Fig. 2, modification of P4 is blocked.
3. **Shadow Paging in Committing Transaction**: When there occurs transaction conflict when the JBD thread prepares the page cache entries for DMA transfer, JBD thread creates the shadow copy of the conflict page and uses it for DMA transfer. It allows only one shadow page.
   - In Fig. 2, modifying P3 in Tx1 creates shadow page P'3, and the original P3 is added to new transaction Tx2.
4. **DMA in Committing Transaction**: When the log block of the committing transaction is transferred to the storage (DMA), the host establishes an exclusive lock on the associated page cache entry.
   - In Fig. 2, during DMA phase
     - An attempt to modify page P2 will be blocked.
     - Because P'3 (shadow page) is being transferred, the file operation which modifies P3 is not blocked and modifies the original page cache entry, P3.
     - An attempt to modify a page cache entry, P5 (not in commit) will successfully add the page to the new running transaction, Tx2.

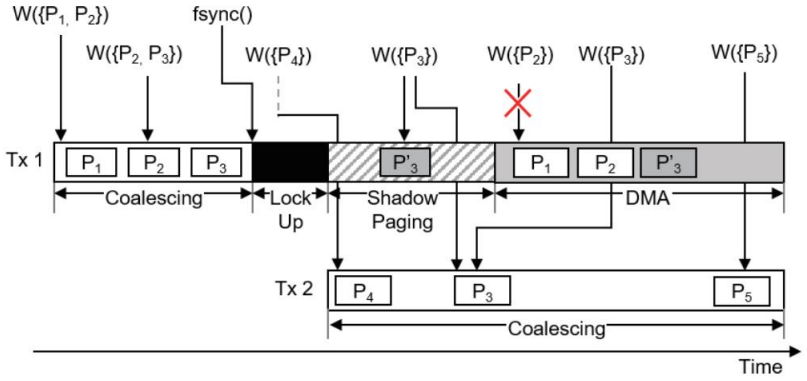


Figure 2: Dissection of EXT4 journaling phases

Problem Statement and Research Objectives

- EXT4 journaling suffers from two critical drawbacks; serial commit and committing the original page cache entry.
- Concurrency Control in EXT4 Filesystem Journaling
  - **The compound journaling** commits multiple file operations with a single journal commit.
  - **The shadow paging** allows the file operation and the journal commit operation to proceed in parallel while they share the same page cache entry.
- Existing Solutions to Scale Journaling

Filesystems	Concurrent Transactions		Multi-Threaded Commit
	Per-core	Per-region	
Z-journal [17]	◯	◯	
SpanFS [15]		◯	
IceFS [24]		◯	
MQFS [23]		◯	
BarrierFS [49]			△
XFS [45]			◯
iJournaling [32]	◯		
ScaleFS [6]	◯		

Table 1: Categories of existing scalable filesystems

Proposed Method

- **Dual Thread Journaling**: The journal commit operation is separated into two separate tasks; transferring the log blocks to the disk and making them durable. A separate thread is allocated for each operation. With Dual Thread Journaling, CJFS can commit a transaction while the preceding journal commit is still in progress.
- **Multi-Version Shadow Paging**: CJFS adopts multi-version shadow paging to resolve the transaction conflict. With multi-version shadow paging, CJFS uses the "copy" of the updated page cache entry in journal commit, allowing the transaction to proceed without being affected by transaction conflicts.

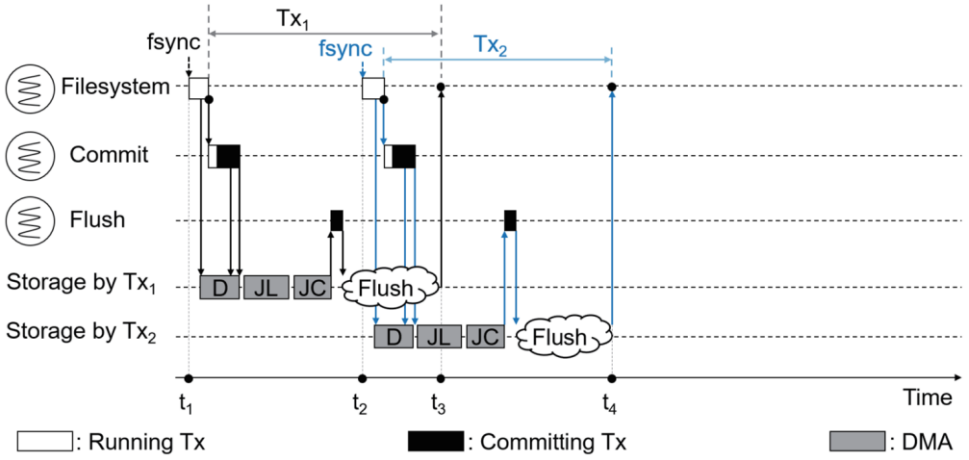


Figure 9: Concurrent Transaction Commit in Dual Thread Journaling. CJFS performs $\texttt{Tx}_1$'s flush phase and $\texttt{Tx}_2$ commit phase concurrently

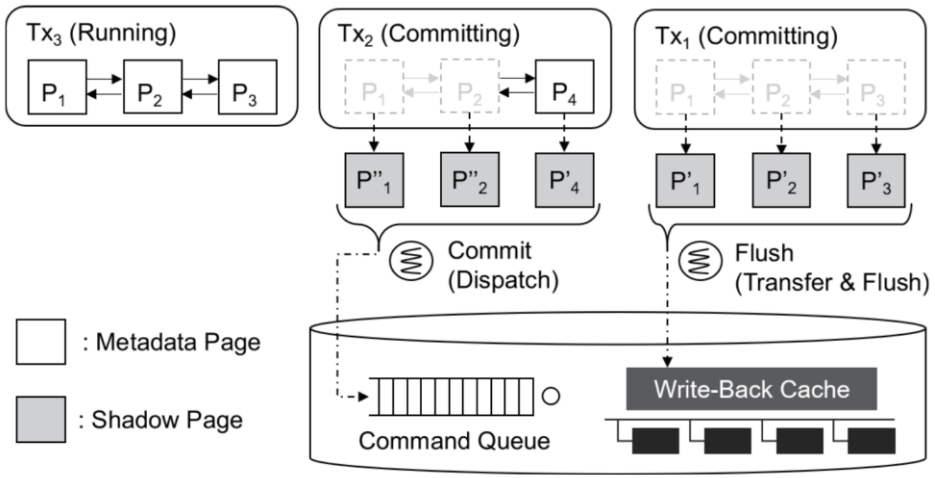


Figure 10: Multi-Version Shadow Paging

Proposed Method

- **Opportunistic Transaction Coalescing**: CJFS adopts opportunistic coalescing to mitigate transaction lock-up overhead. To increase the compound degree of the journal transaction, The running transaction is released from the LOCKED state when it with a committing transactions.
- **Compound Flush**: CJFS creates a large number of flush commands since it creates multiple committing transactions in-flight, with each committing transaction issuing its own flush to make its journal transaction durable. ➔ To relieve the overhead of servicing the flush commands, CJFS combines consecutive flushes from the concurrent transactions into a single flush. Compound flush significantly reduces the latency of the individual fsync() calls.

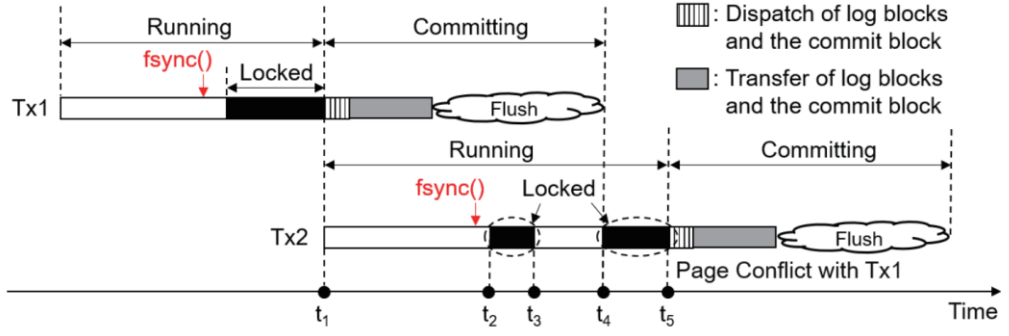


Figure 11: Illustration of Opportunistic Coalescing

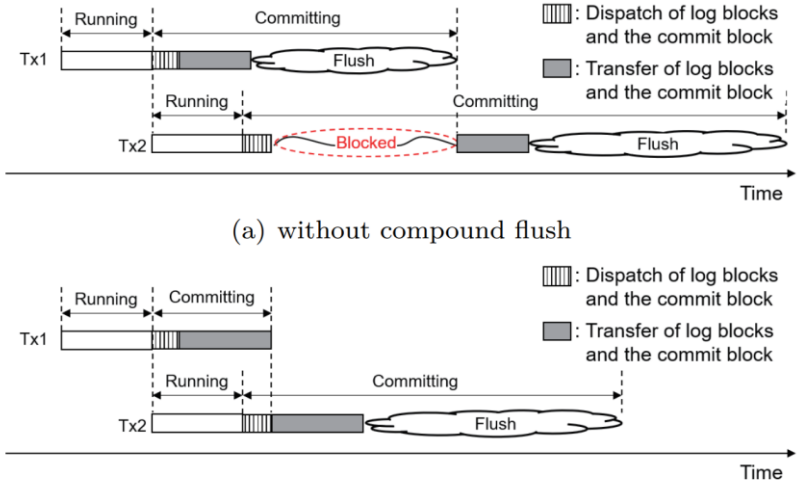


Figure 12: Comparison of the flush procedure with and without Compound Flush

Evaluation and Results

- Four filesystem macro benchmarks – two variants of varmail (varmail-shared and varmail-split) in filebench, dbench, and OLTP-Insert – to cover wide variety of real-world application behaviors.
  - Each benchmark has a different mix of file operations (Table 2) and stresses various parts of the filesystem (Table 3).

Benchmarks	create()	unlink()	write()	read()	fsync()	rename()
varmail	7.7%	7.7%	15.4%	15.4%	15.4%	0%
dbench	16.6%	3.5%	8.6%	27.1%	5.2%	0.7%
OLTP-Insert	0%	0%	77.8%	12.2%	10.0%	0%

Table 2: Ratio of filesystem operations in benchmarks

Benchmarks	Directory contention	In-memory logging	On-disk logging
varmail-shared	High	Moderate	High
varmail-split	No	Moderate	High
dbench	No	Moderate	Moderate
OLTP-Insert	No	low	low

Table 3: Filesystem contention in benchmarks

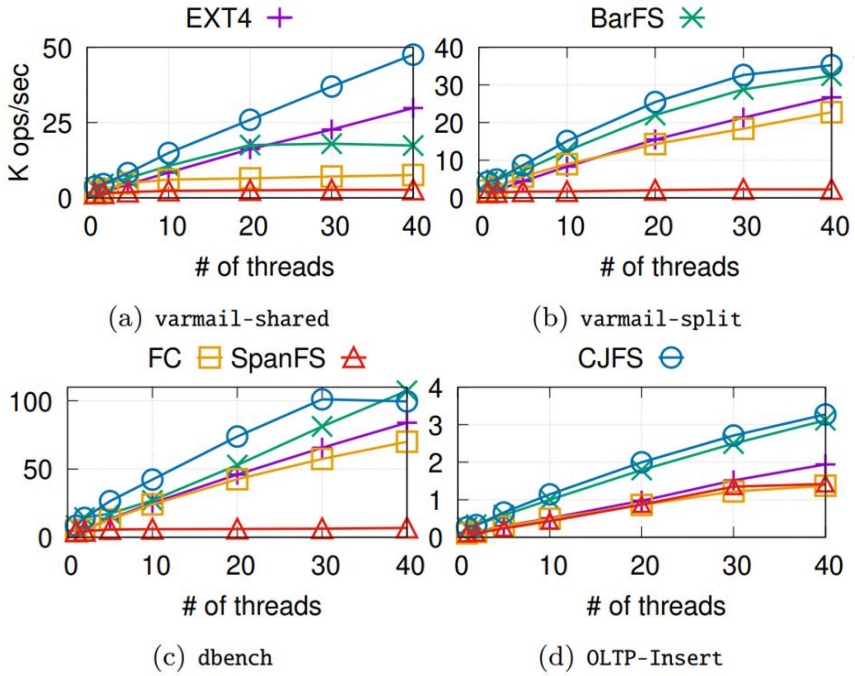


Figure 18: Throughput: EXT4, BarrierFS (BarFS), Fast commit (FC), SpanFS, and CJFS

Evaluation and Results

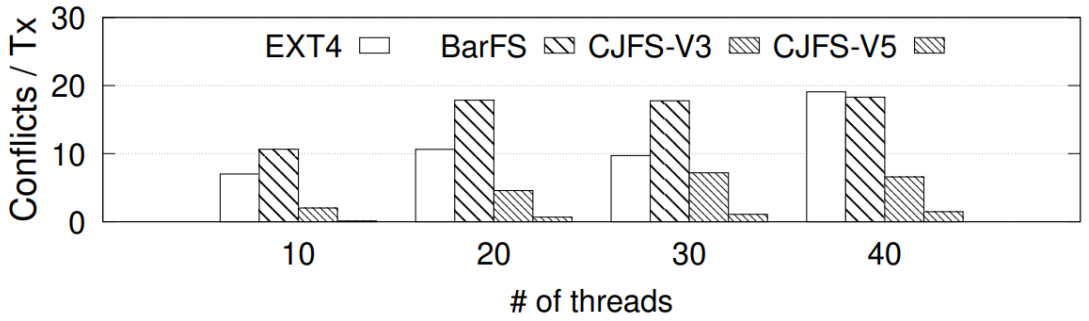
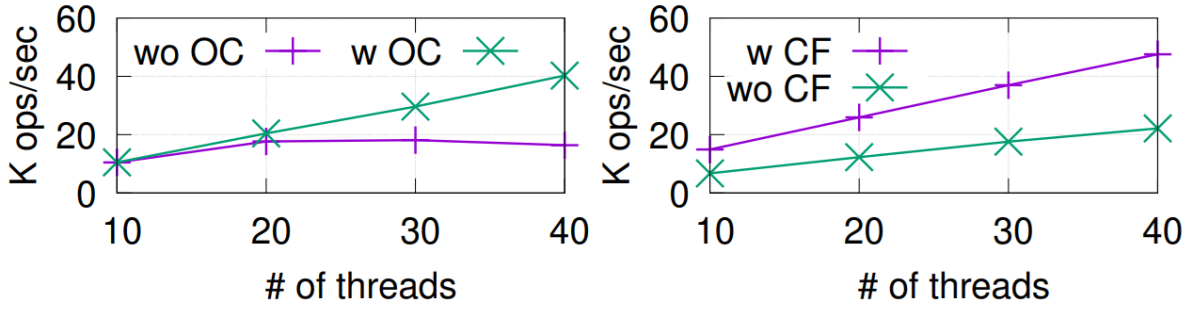
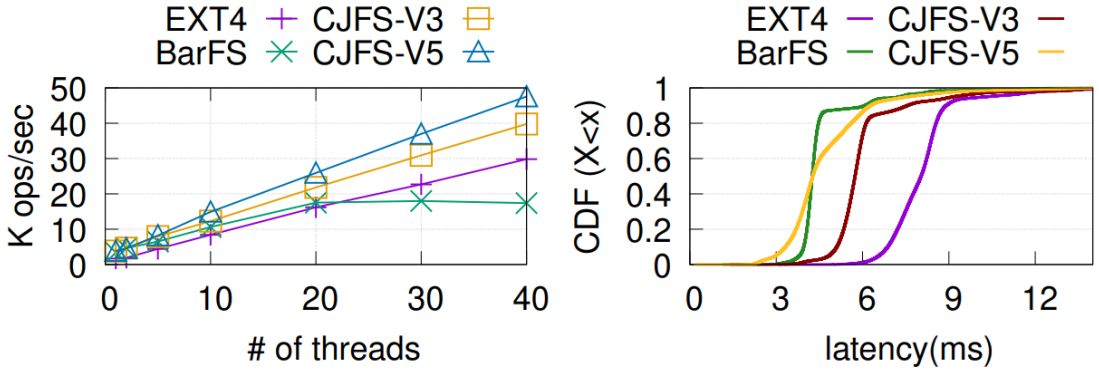


Figure 15: Average number of conflicts per transaction



(a) Opportunistic Coalescing (b) Compound Flush

Figure 16: Effect of Opportunistic Coalescing and Compound Flush for `varmail-shared` in CJFS



(a) Throughput (b) `fsync()` latency (40 threads)

Figure 14: Throughput and Latency of `varmail-shared`: CJFS, BarrierFS (BarFS), and Vanila EXT4 (EXT4)

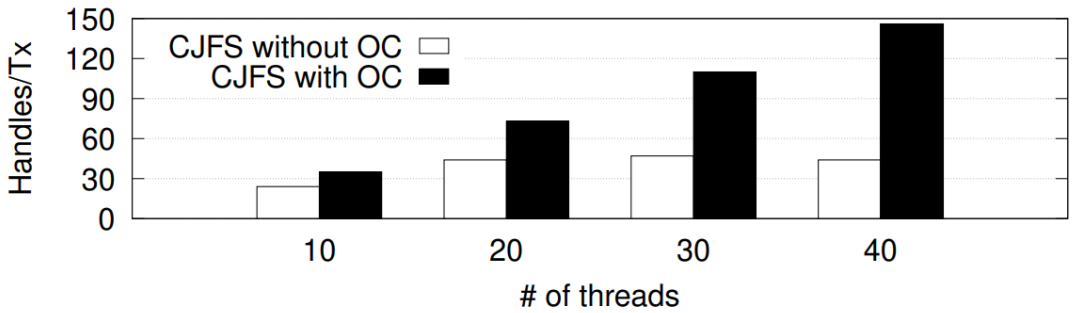


Figure 17: Comparison of coalescing degree with and without Opportunistic Coalescing for `varmail-shared`