High-Performance Transaction Processing in Journaling File Systems

Y. Son, S. Kim, H. Y. Yeom, and H. Han[§]

Seoul National University, Korea

[§]Dongduk Women's University, Korea

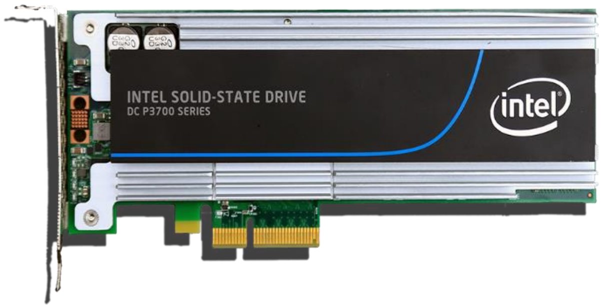
Contents

- Motivation and Background
- Design and Implementation
- Evaluation
- Conclusion

DCSLAB

Motivation and Background

- **Storage technology**
  - High-performance storage devices (e.g., SSDs) provide low-latency, high-throughput, and high I/O parallelism



Highly parallel SSD
(Intel NVMe SSD)

Highly parallel SSD
(Samsung NVMe SSD)

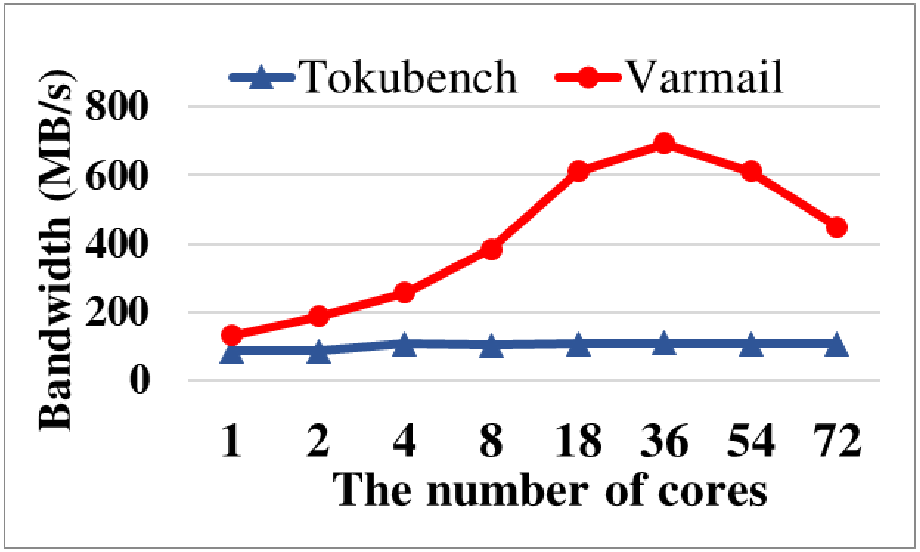
High-Performance SSDs are widely used in
cloud platforms, social network services, and so on

Motivation and Background

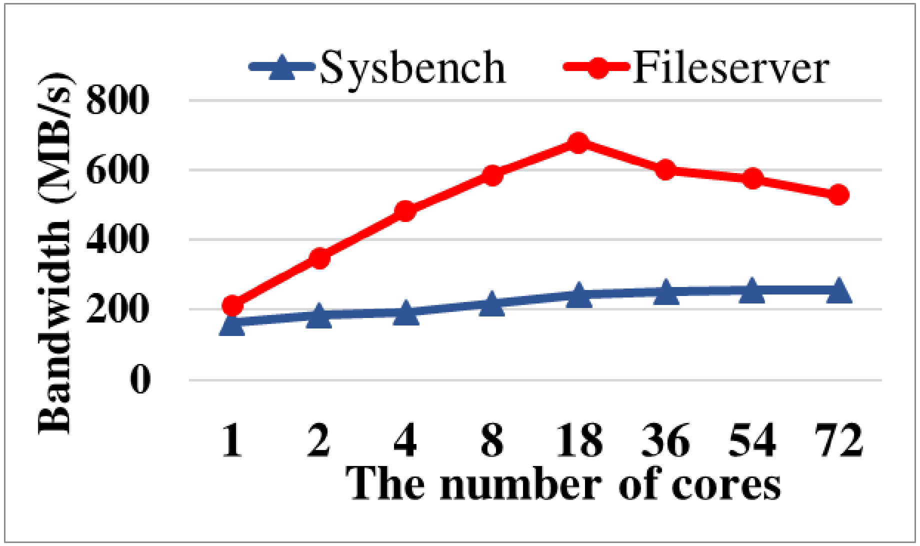
- **Motivational evaluation for highly parallel SSDs**
  - The performance does not scale well or decreases as the number of cores increases

Experimental Setup

72-cores / Intel P3700 / EXT4 file system



Ordered mode



Data journaling mode

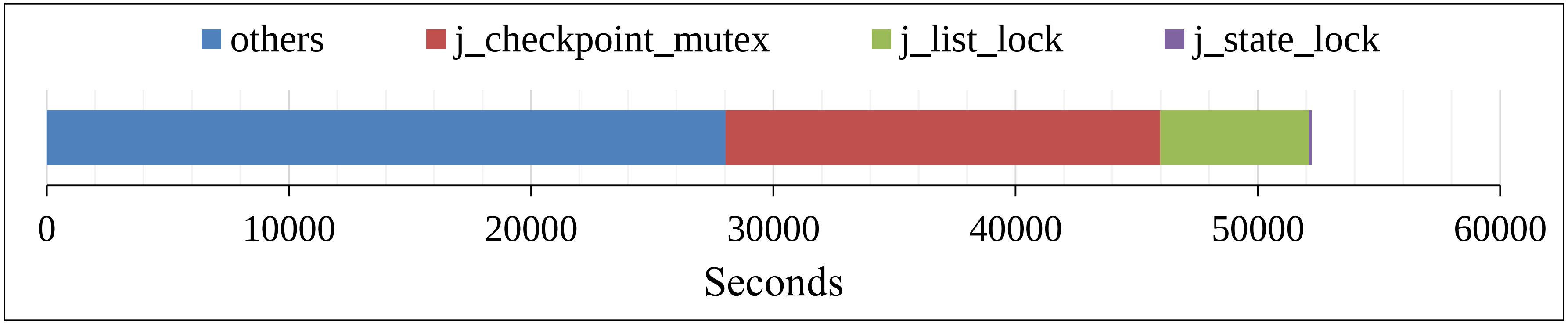
Motivation and Background

- **Existing coarse-grained locking and I/O operations by a single thread in transaction processing**
  - Locks on **transaction processing** in EXT4/JBD2
    - Total write time: 52220s (100%)
    - j_checkpoint_mutex (mutex lock): 17946s (34.40%) **Hot lock**
    - j_list_lock (spin lock): 6140s (11.75%) **Hot lock**
    - j_state_lock (r/w lock): 102s (0.19%)

Execution time breakdown

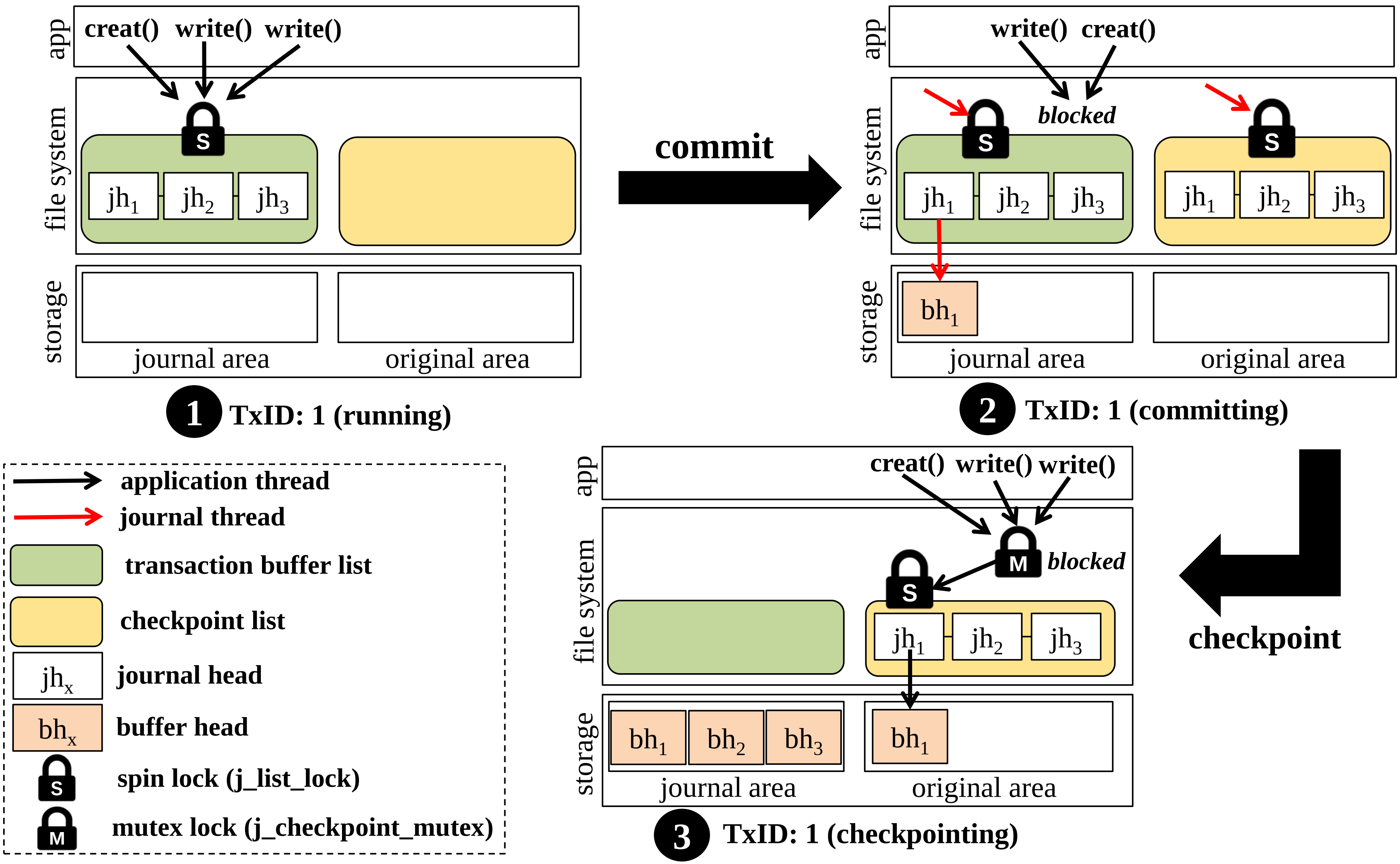
72-cores / Intel P3700 / EXT4 data journaling

sysbench (72threads, total 72 GiB random write)



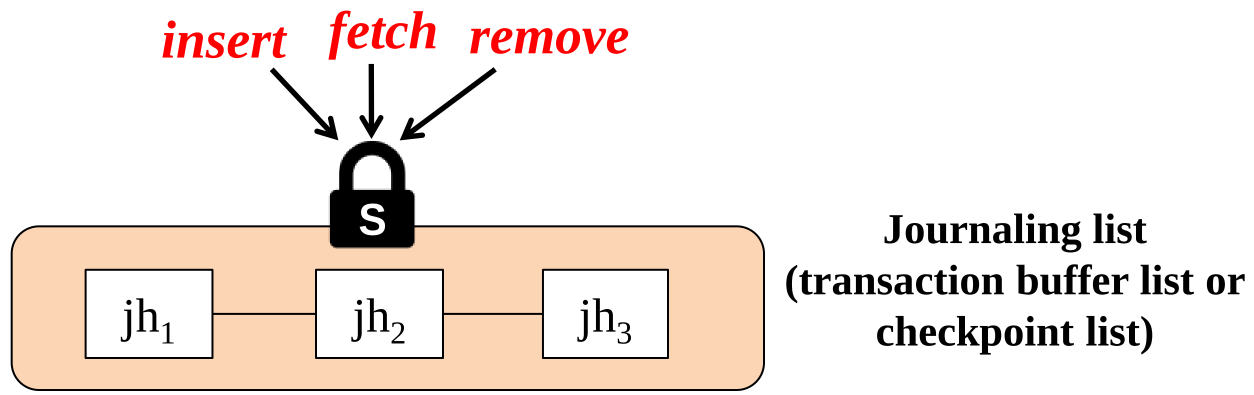
Motivation and Background

- **Overall existing locking and I/O procedure**

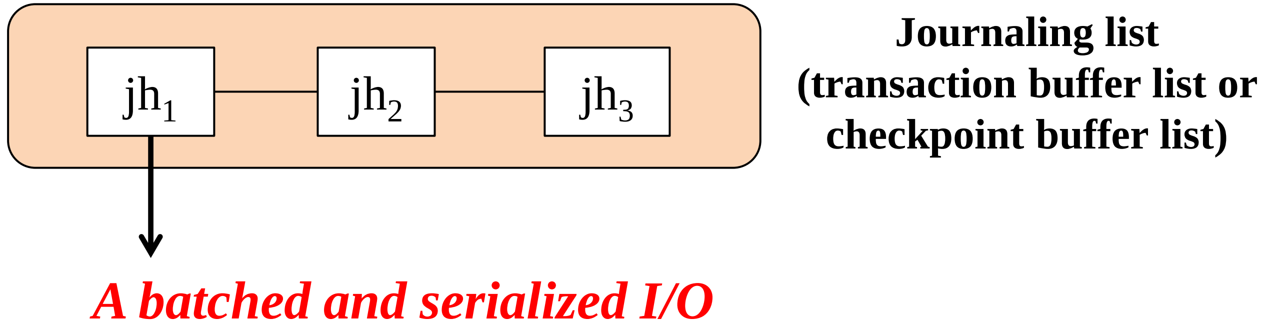


Motivation and Background

- **Coarse-grained locking limits scalability of multi-cores**



- **I/O operation by a single thread limits I/O parallelism of SSDs**

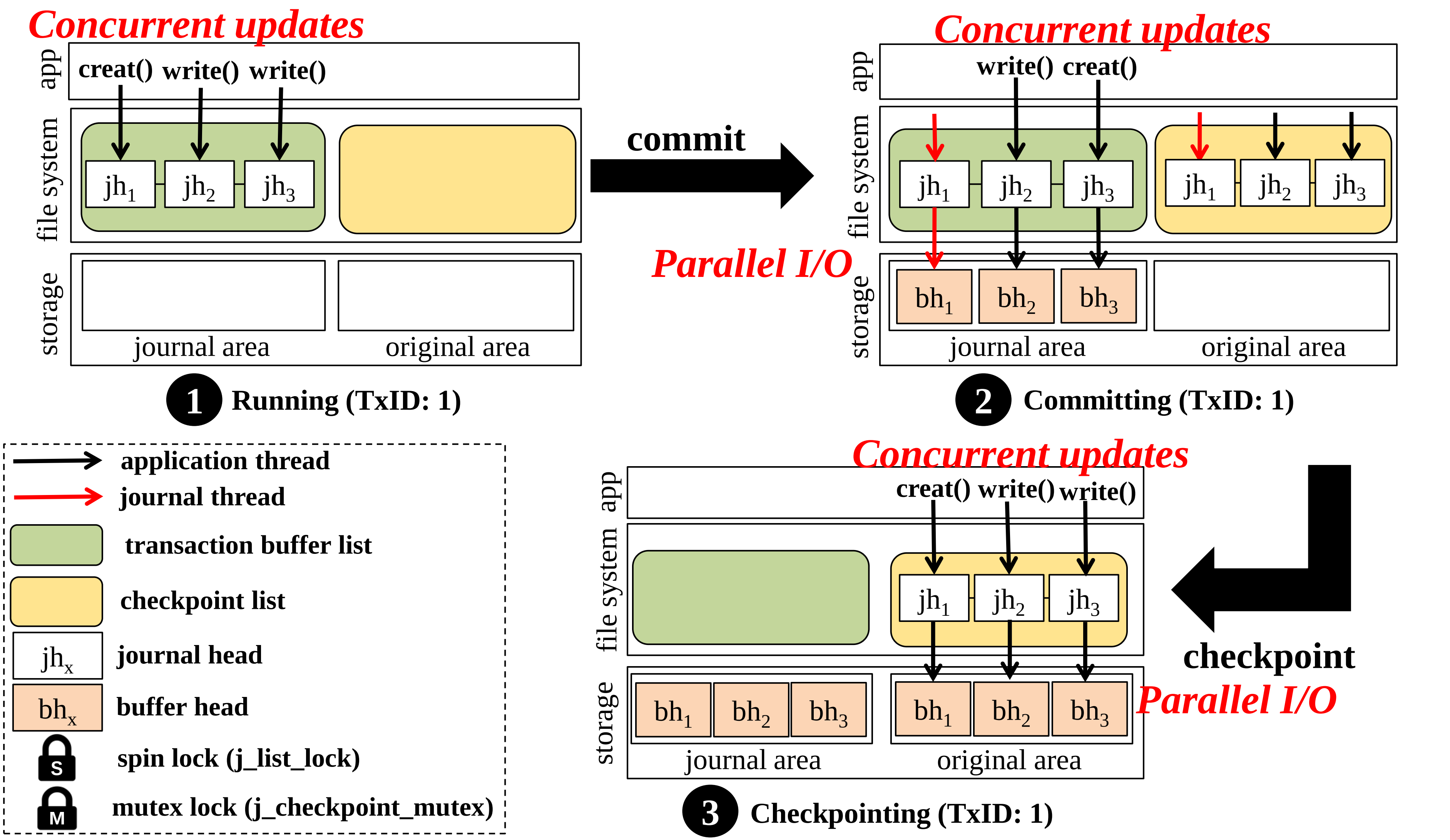


Design and Implementation

- **Goal**
  - Optimizing transaction processing (running, committing, checkpointing) in journaling file systems
- **Our schemes**
  - **Concurrent updates on data structures**
    - Adopting lock-free data structures and operations using atomic instructions
      - Lock-free linked list
        - lock-free insert, remove, fetch
        - Using atomic instructions
          - atomic_add()/atomic_read()/atomic_set()/compare_and_swap()
  - **Parallel I/O in a cooperative manner**
    - Enabling application threads to the journal and checkpoint I/O operations not blocking them
      - Fetching buffers from the shared linked lists, issuing the I/Os, and completing them in parallel

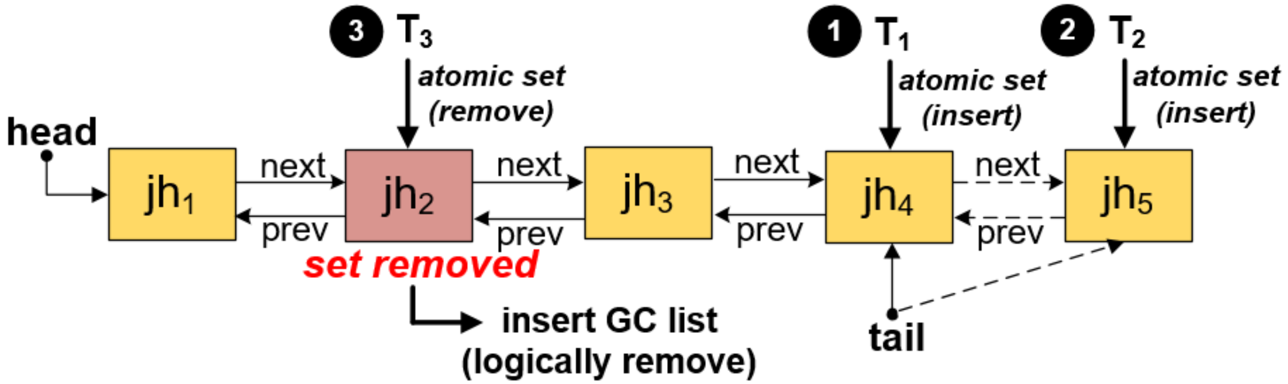
Design and Implementation

- ## Overall Proposed Schemes



Design and Implementation

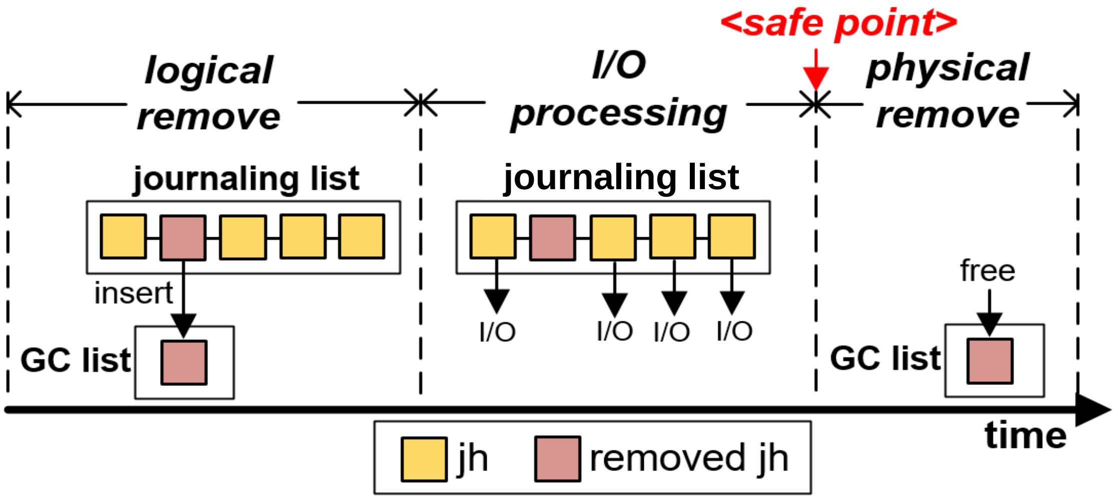
- **Concurrent updates on data structures**
  - Concurrent insert operations
    - Using atomic set instruction




```
add_buffer(jh, head, tail)
{
      jh->prev = atomic_set(tail, jh);
      if(jh->prev == NULL)
            head = jh;
      else
            jh->prev->next = jh;
}
```

Design and Implementation

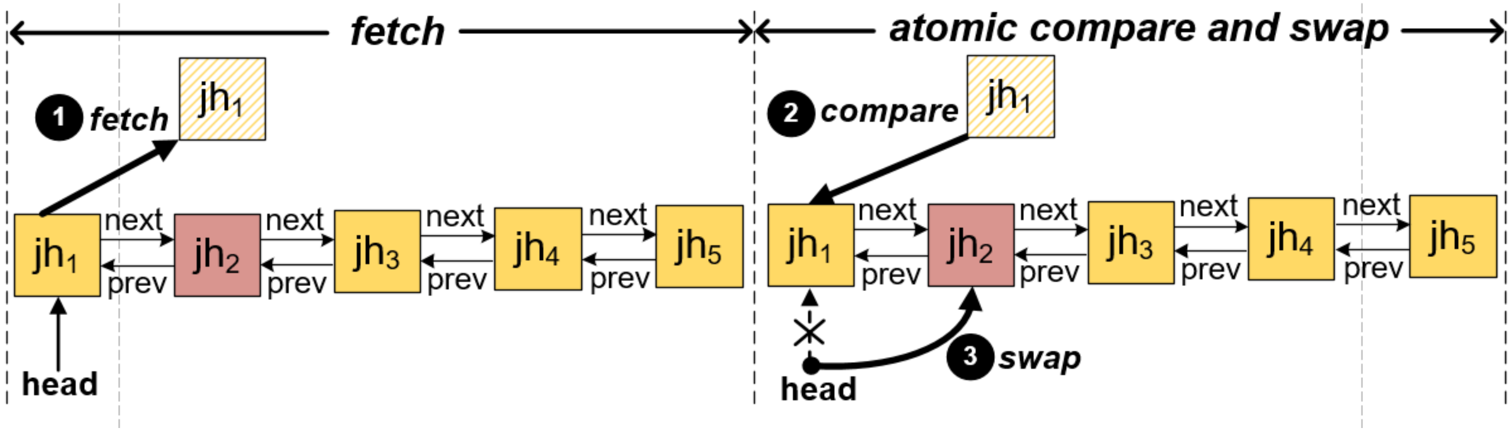
- **Concurrent updates on data structures**
  - Concurrent remove operations (two-phase removal)




```
del_buffer(jh, head, tail)
{
      atomic_set(jh->remove, remove);
      jh->gc_prev = atomic_set(tail, jh);
      if(jh->gc_prev == NULL)
            head = jh;
      else
             jh->gc_prev->gc_next = jh;
}
```

Design and Implementation

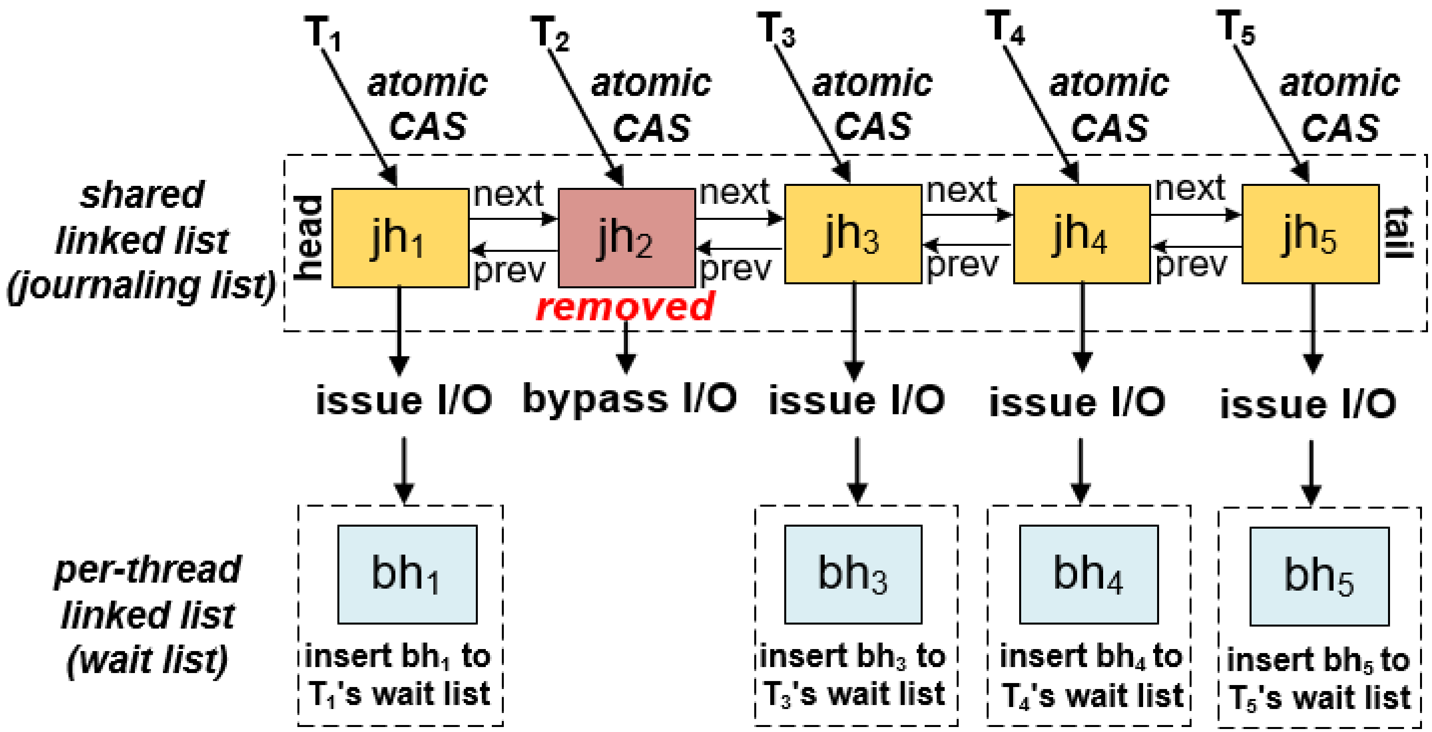
- **Concurrent updates on data structures**
  - Concurrent fetch operations




```
journal_io_start(….)
{
      while((jh = head) != NULL){
            if(atomic_cas(head, jh, jh->next) != jh)
                  continue;
            if(atomic_read(jh->removed) == removed)
                  continue;
            submit_io(…);
}
```

Design and Implementation

- **Parallel I/O operations in a cooperative manner**
  - Allowing the application threads to join the I/Os not blocking them
  - Fetching buffers from the shared linked list concurrently
  - Issuing the I/Os in parallel
  - Completing the I/Os in parallel using per-thread list



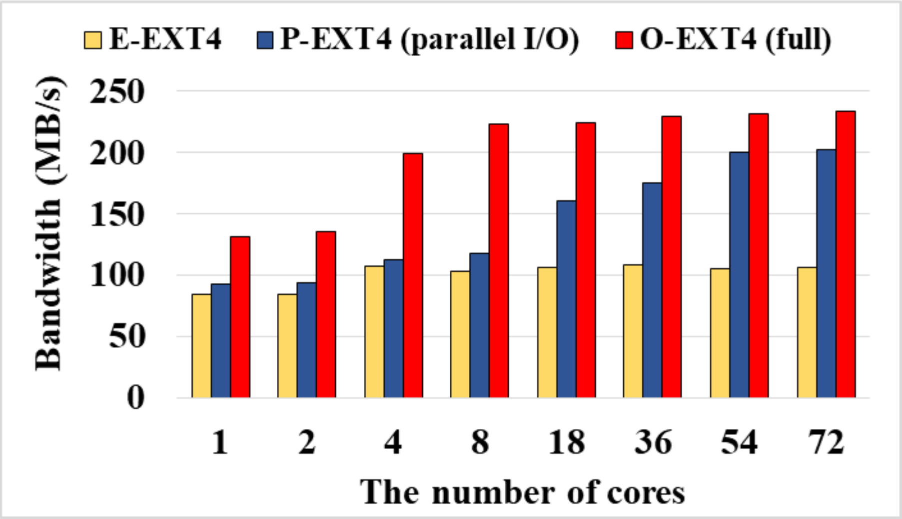
Experimental Setup

- **Hardware**
  - 72-core machine
    - Four Intel Xeon E7-8870 processors (without hyperthreading)
    - 16 GiB DRAM
    - PCI 3.0 interface
  - 800 GiB Intel P3700 NVMe SSD (18-channels)
- **Software**
  - Linux kernel 4.9.1
  - EXT4/JBD2
    - An optimized EXT4 with parallel I/O: **P-EXT4**
    - Fully optimized EXT4: **O-EXT4**
- **Benchmarks**

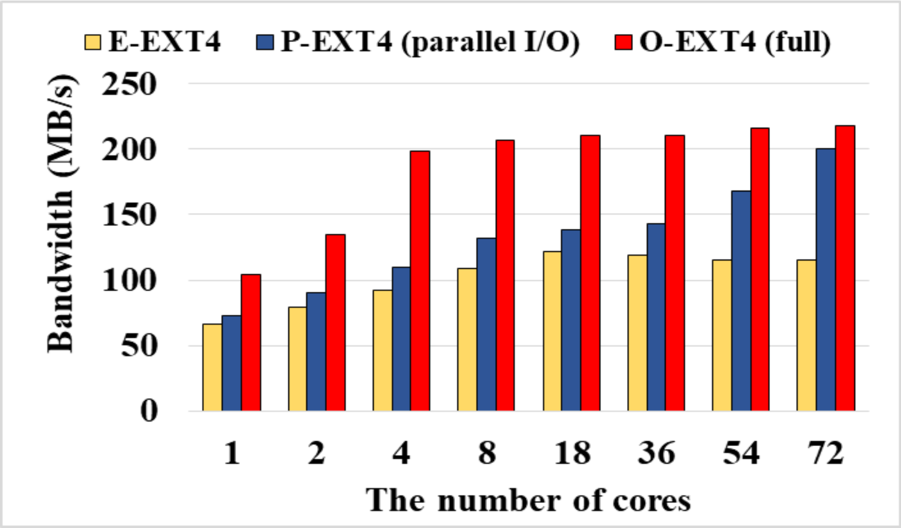
Benchmarks	Descriptions	Parameters
Tokubench (micro)	Metadata-intensive (file creation)	Files: 30,000,000, I/O sizes: 4KiB
Sysbench (micro)	Data-intensive (random write)	Files: 72, Each file size: 1GiB, I/O sizes: 4KiB
Varmail (macro)	Metadata-intensive (read/write ratio = 1:1)	Files: 300,000, Directory width: 10,000
Fileserver (macro)	Data-intensive (read/write ratio = 1:2)	Files: 1,000,000, Directory width: 10,000

Performance Evaluation

- **Tokubench**
  - Ordered mode
    - Improvement: upto 1.9x (P-EXT4), upto 2.2x (O-EXT4)
  - Data journaling mode
    - Improvement: upto 1.73x (P-EXT4), upto 1.88x (O-EXT4)



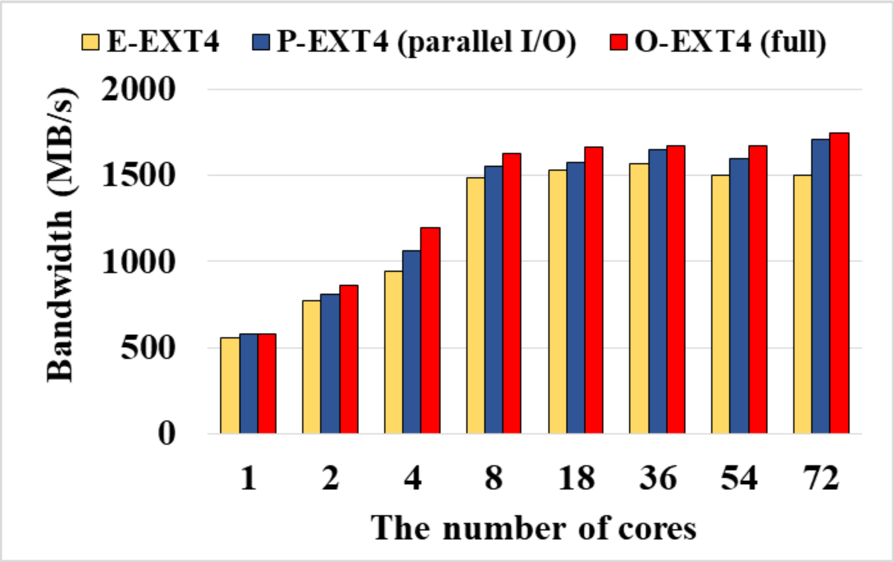
Ordered mode



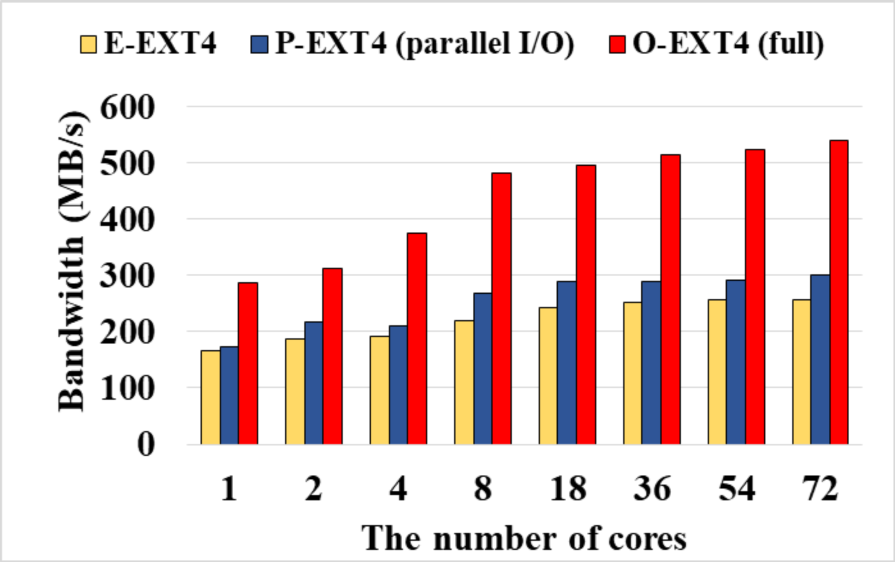
Data journaling mode

Performance Evaluation

- **Sysbench**
  - Ordered mode
    - Improvement: upto 13.8% (P-EXT4), upto 16.3% (O-EXT4)
  - Data journaling mode
    - Improvement: upto 1.17x (P-EXT4), upto 2.1x (O-EXT4)



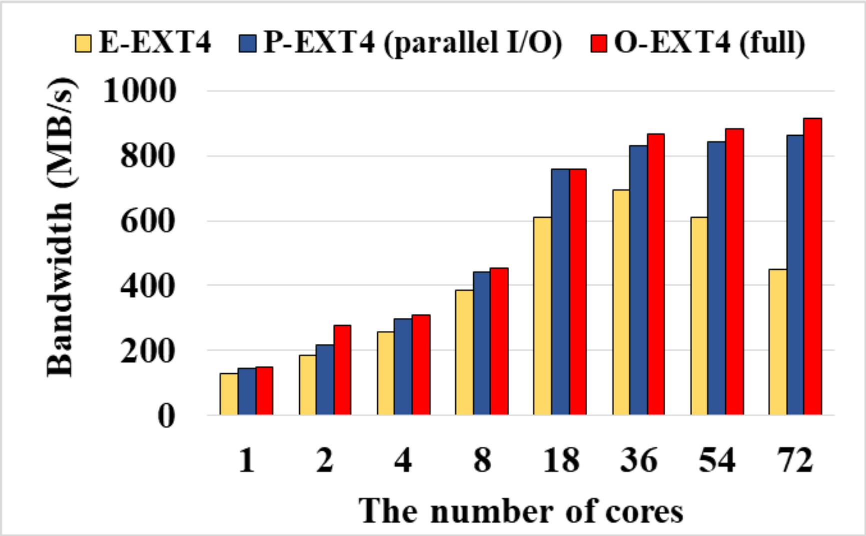
Ordered mode



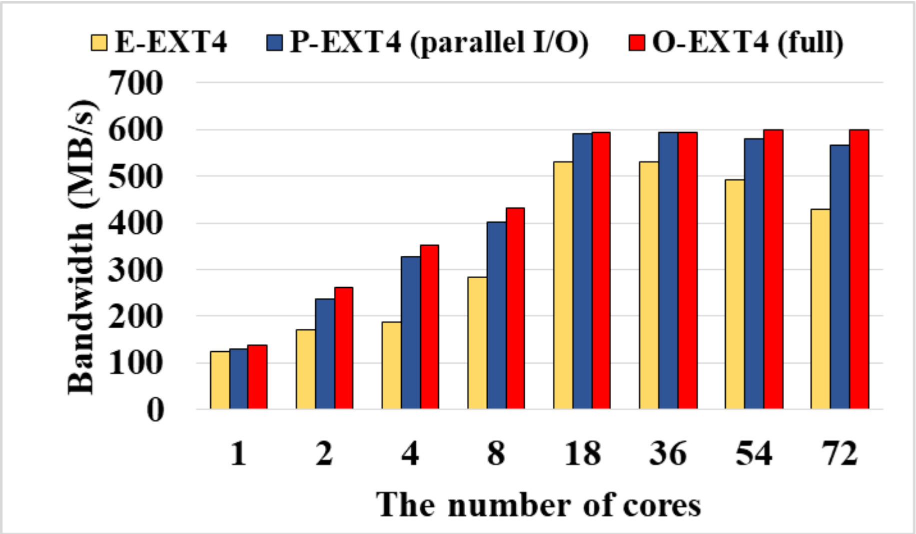
Data journaling mode

Performance Evaluation

- **Varmail**
  - Ordered mode
    - Improvement: upto 1.92x (P-EXT4), upto 2.03x (O-EXT4)
  - Data journaling mode
    - Improvement: upto 31.3% (P-EXT4), upto 39.3% (O-EXT4)



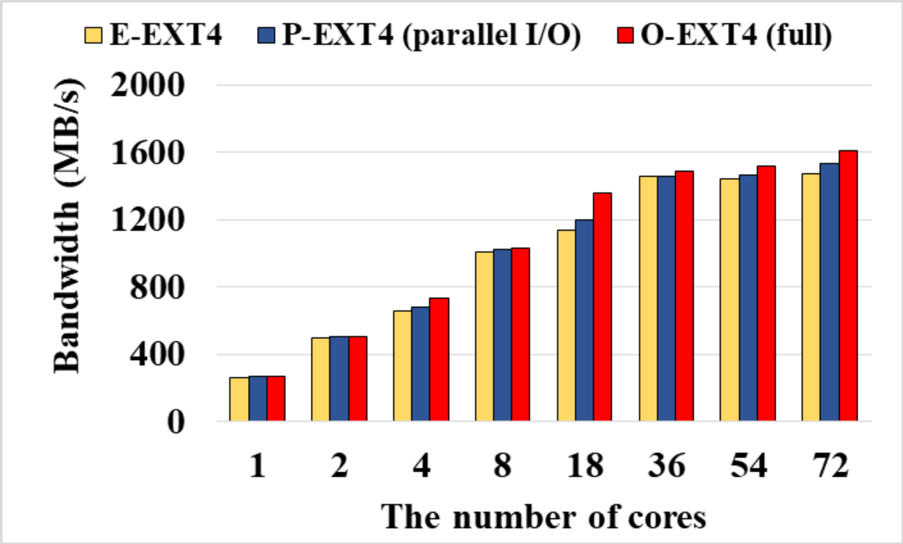
Ordered mode



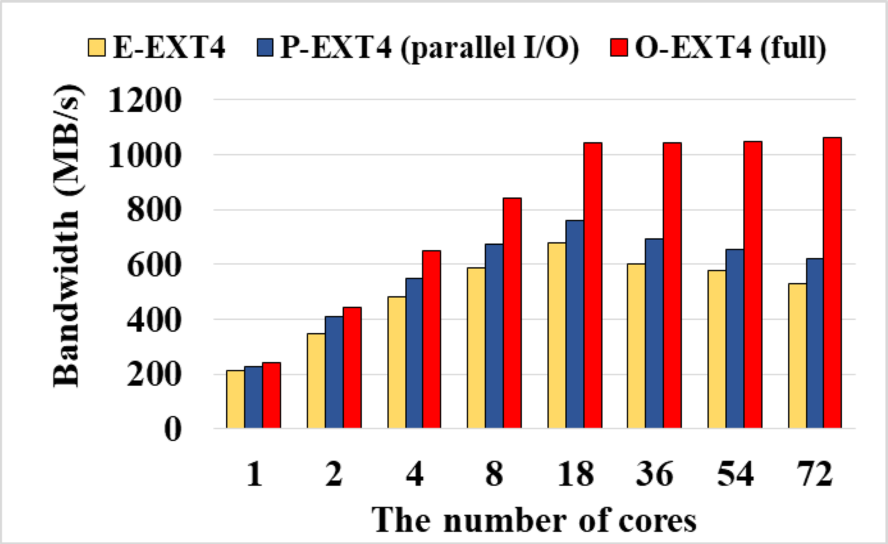
Data journaling mode

Performance Evaluation

- **Fileserver**
  - Ordered mode
    - Improvement: upto 4.3% (P-EXT4), upto 9.6% (O-EXT4)
  - Data journaling mode
    - Improvement: upto 1.45x (P-EXT4), upto 2.01x (O-EXT4)



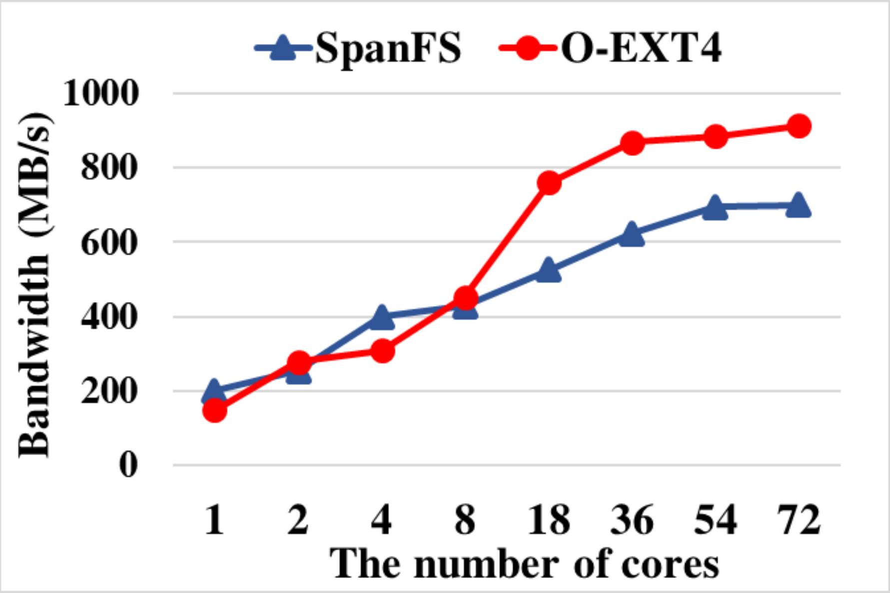
Ordered mode



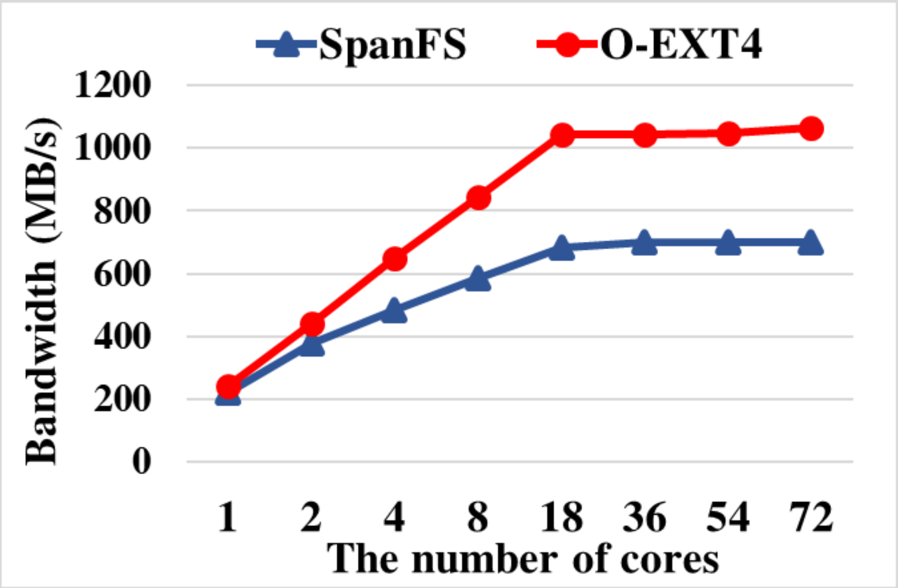
Data journaling mode

Performance Evaluation

- **Comparison with a scalable file system (SpanFS, ATC'15)**
  - Ordered mode
    - Improvement: upto 1.45x
    - The performance of O-EXT4 is similar or slower than SpanFS in the case of small cores
  - Data journaling mode
    - Improvement: upto 1.51x



Ordered mode (varmail)



Data journaling mode (fileserver)

Performance Evaluation

- **Experimental analysis**
  - EXT4 vs. **P-EXT4**
    - Improvement
      - Bandwidth: 16.3%, Write time: 15.7%
  - EXT4 vs. **O-EXT4**
    - Improvement
      - Bandwidth: 2.06x, Write time: 2.08x

File systems	EXT4	P-EXT4	O-EXT4
Device-level BW	692 MB/s	805 MB/s	1426 MB/s
Write time	52220 s (100%)	45124 s (100%)	25078 s (100%)
j_checkpoint_mutex	17946 s (34.4%)	0	0
j_list_lock	6132 s (11.7%)	4890 s (10.8%)	0
j_state_lock	102 s (0.2%)	87 s (0.2%)	182 s (0.7%)
others	28040 s (53.7%)	40147 s (89%)	24896 s (99.3%)

Device-level BW and total execution time of main locks in data journaling mode (sysbench)

Conclusion

- **Motivation and Background**
  - Data structures for transaction processing protected by non-scalable locks
  - Serialized I/O operations by a single thread
- **Approaches**
  - Concurrent updates on data structures
  - Parallel I/O in a cooperative manner
- **Evaluation**
  - Ordered mode: up to 2.2x
  - Data journaling mode: up to 2.1x
- **Future work**
  - Optimizing the locking mechanism for other resources such as file, pa ge cache, etc

THANK YOU for your ATTENTION!