Ext4 Filesystem Scaling

Jan Kára <jack@suse.cz>
SUSE Labs

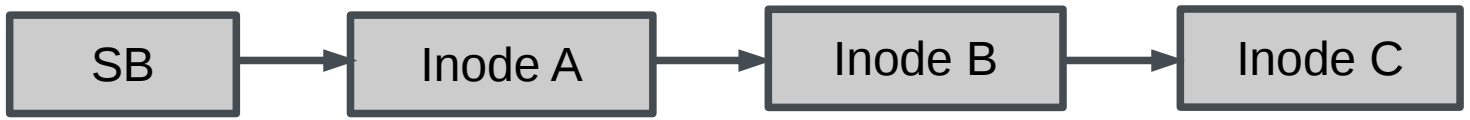
Overview

- Handling of orphan inodes in ext4
- Shrinking cache of logical to physical block mappings
- Cleanup of transaction checkpoint lists

Orphan Inode Handling

Orphan list

- List of inodes for tracking unlinked open files and files with truncate in progress
- In case of crash we walk the orphan list and finish truncates and remove unlinked inodes
- Necessary to avoid leaking such inodes / space on crash
- Added for ext3 filesystem (around 2000)
- List is filesystem-wide ⇒ scalability issue for parallel truncates / unlinks



Orphan list contention

- Orphan stress test with 16 processes on 8 CPU server with 4 GB of RAM, filesystem on ramdisk
- Lockstat except for orphan lock:

```
wait-max    wait-total wait-avg acquisitions hold-max  hold-total hold-avg
12211.31 2288111288.96   244.07     26209088    85.74 62275116.77     2.38
```

- Test execution: 160 s, tasks wait on orphan mutex for 2288 s in total ⇒ 143 s per task.

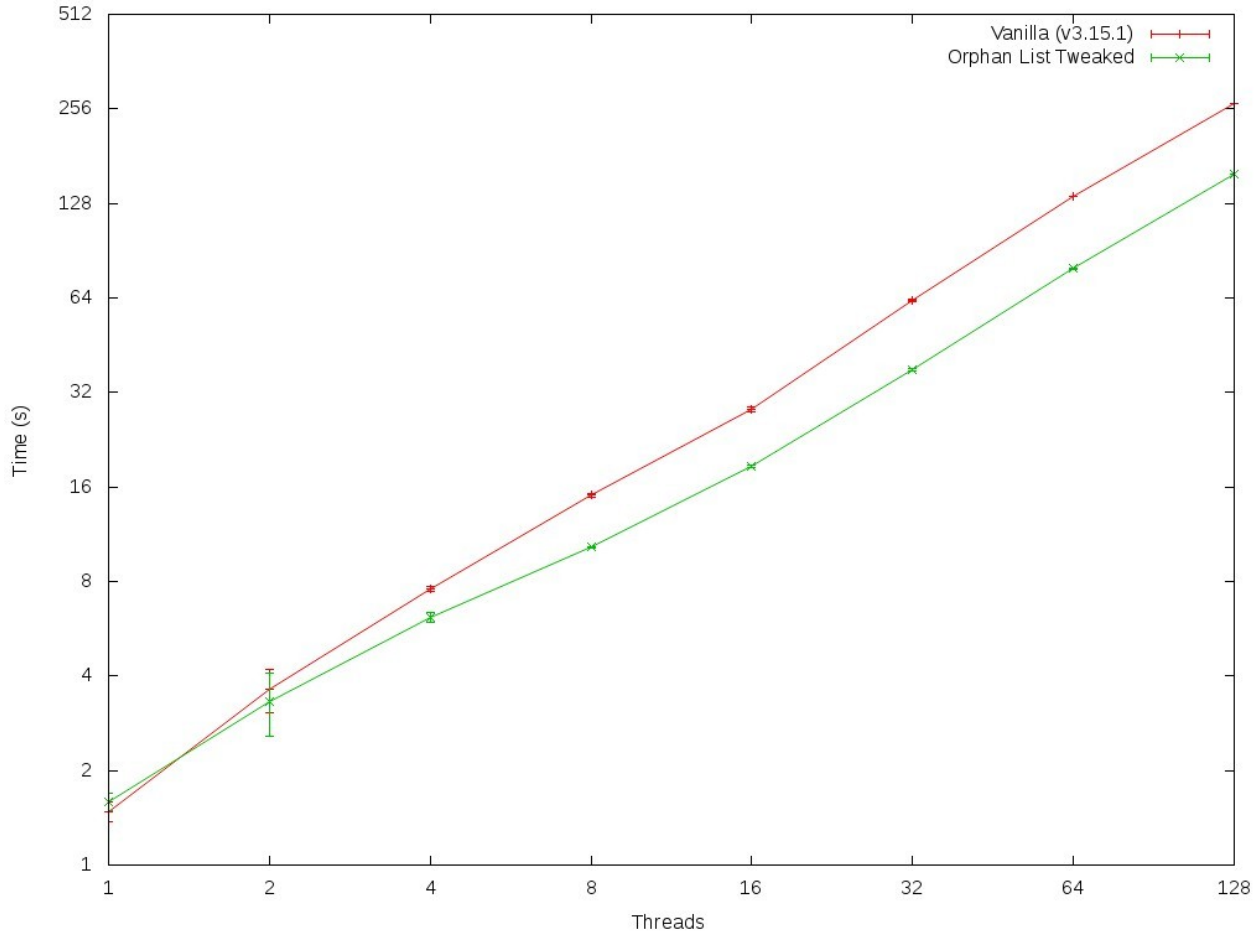
Orphan List Scalability Improvements

- Joint work with Thavatchai Makphaibulchoke
- List handling inherently unscalable – unsolvable without on disk format changes
- Still can do something:
  - Remove unnecessary global lock acquisitions when inode is already part of orphan list (inode modifications itself guarded by inode-local lock)
  - Make critical section shorter
    - Effective for insertion – blocks to modify known in advace
    - Not for deletion – blocks to modify known only after taking global lock
- Merged in 3.16

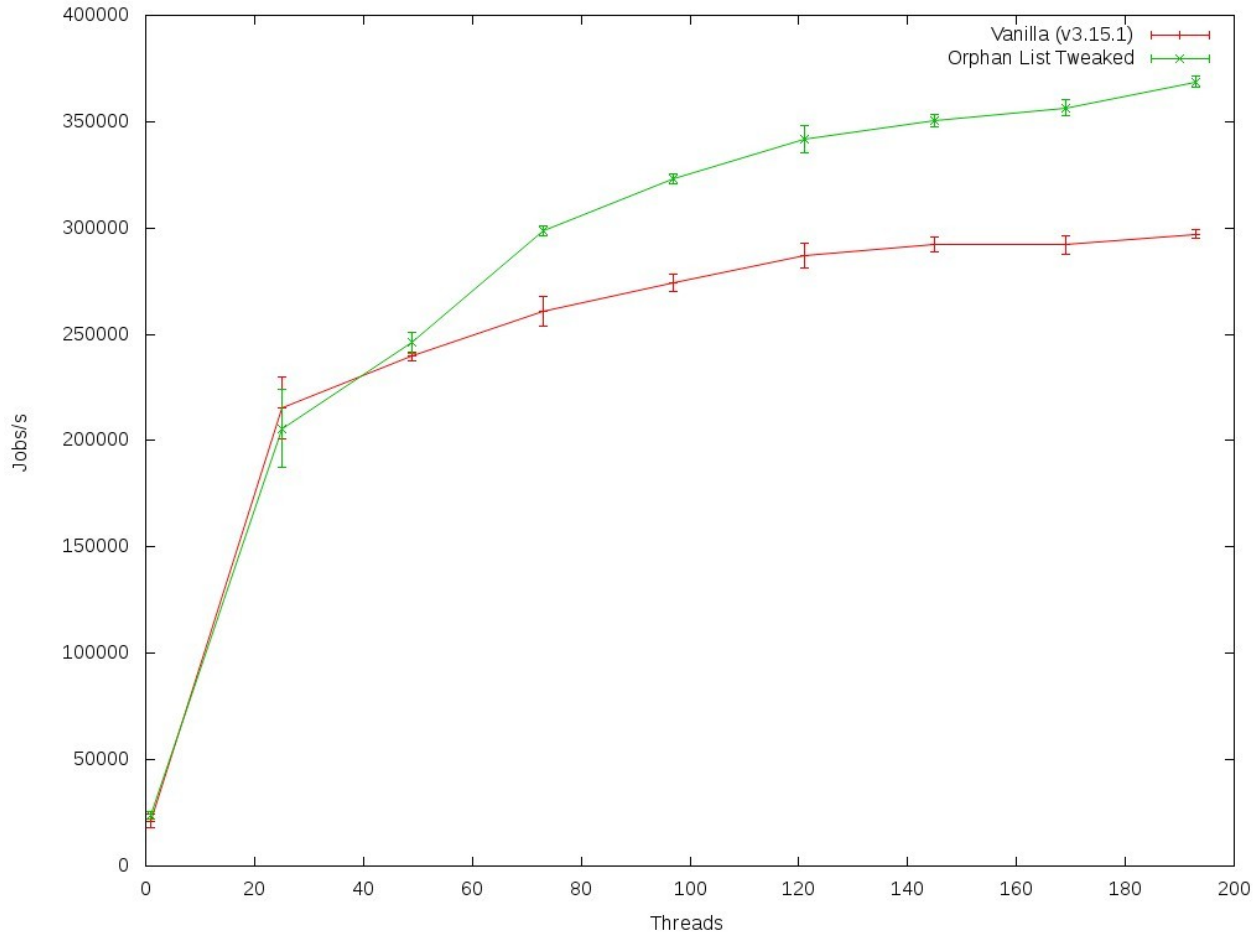
Testing Effect of Changes

- Test data from 48-way Xeon Server with 32 GB RAM
- Ext4 filesystem on ramdisk – preallocated with NUMA interleave policy
- Average of 5 runs
- Run stress-orphan microbenchmark and reaim new_fserver workload
  - Reaim modified to not call sync(1) after every read & write

Effect of Tweaks (stress-orphan)

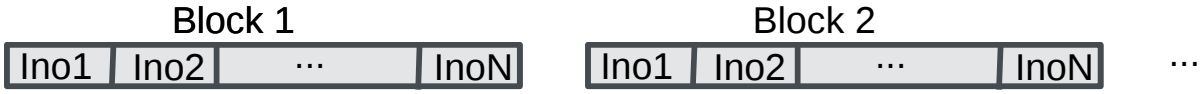


Effect of Tweaks (reaim - fserver)



Orphan File

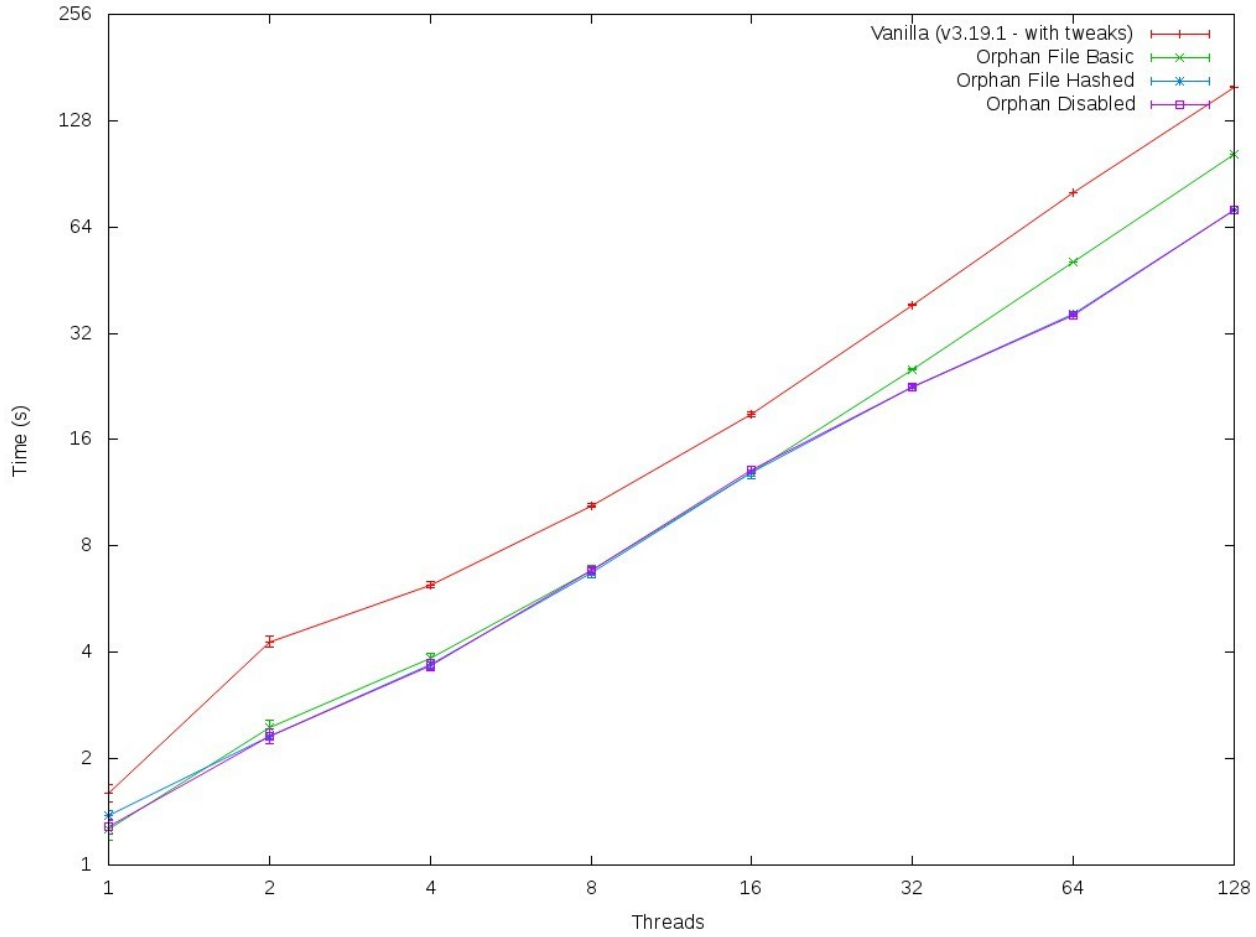
- Experimental patches
- Preallocated system file where we store numbers of orphan inodes
- When we run out of space in orphan file (too many files deleted in parallel) we fall back to old orphan list
- Hack: Skip journalling block if it is part of running transaction
  - Real fix: Improve JBD2 calls to have low overhead if there's nothing to do (block already in the transaction in correct state)



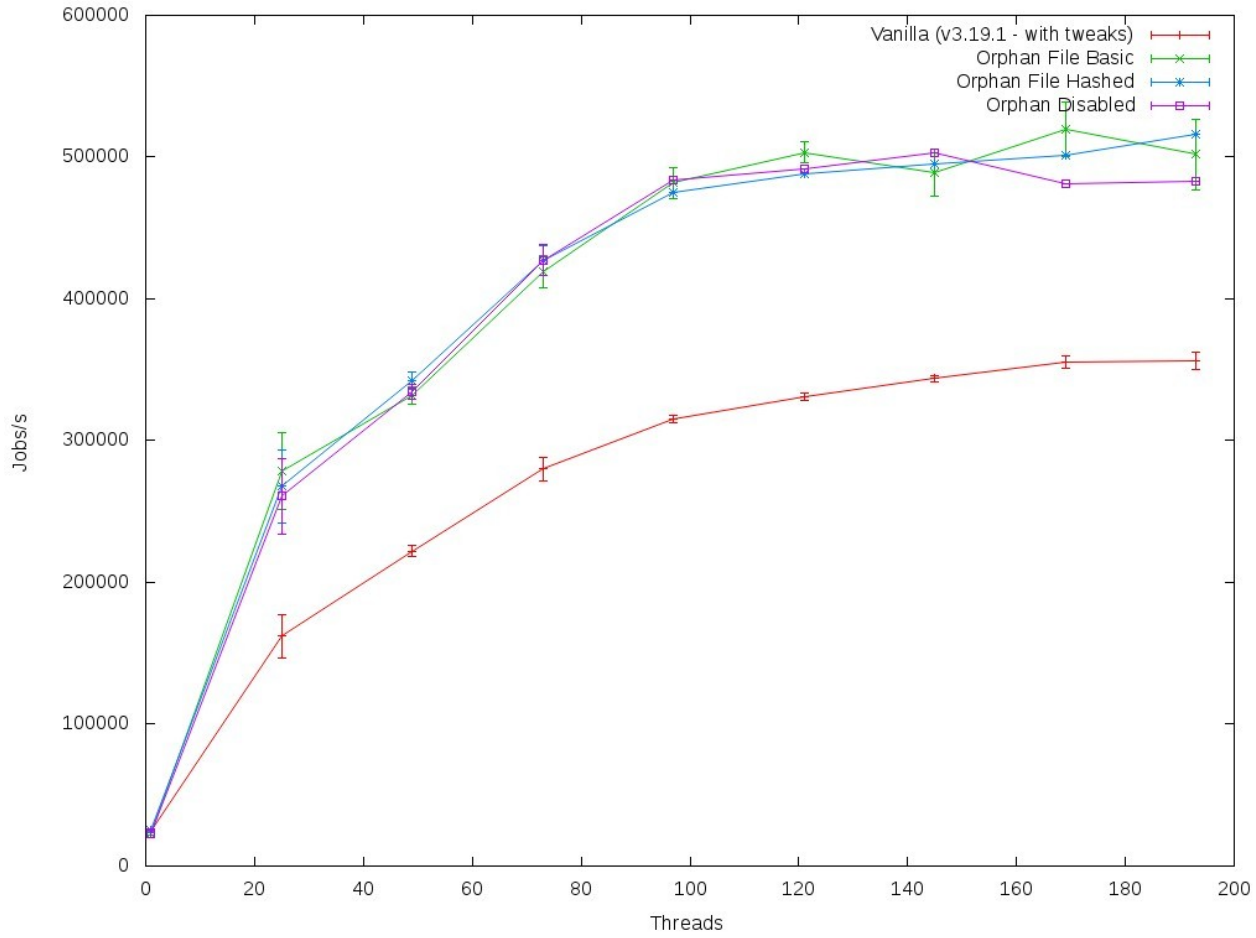
Orphan slot allocation strategies

- Simple way to search for free slot in a file – sequential search from the beginning under a spinlock
- More complex: Hash CPUID to a block in the orphan file (can have easily more blocks for a CPU), start searching at that block.
  - With atomic counters for free entries in a block and cmpxchg() for storing inode number in a slot completely lockless (except for journalling locks in rare cases)

Perf. With Orphan File (stress-orphan)



Perf. With Orphan File (reaim - fserver)



Extent Status Tree Shrinker

Extent Status Tree

- Cache for logical to physical block mapping for files
- For each entry we have logical offset, physical block, length, extent type (delayed allocated, hole, unwritten, written)
- Organized in RB-tree sorted by logical offset
- In case of memory pressure we need to free some entries – shrinker
  - Entries that have delayed allocated type cannot be freed
- MM calls shrinker asking it to scan N objects and free as much as possible

Extent Status Tree Shrinker

- Need to find entries to reclaim without bloating each entry too much
  - Each entry has 16 bytes of useful data
- Keep timestamp when last reclaimable extent was used in each inode
- On reclaim request scan RB trees of inodes with oldest timestamps until N entries were reclaimed
  - Need to sort list of inodes (oops)
  - Scanning may need to skip lots of unreclaimable extents
- Causing large stalls (watchdog triggered) on machines with lots of active inodes

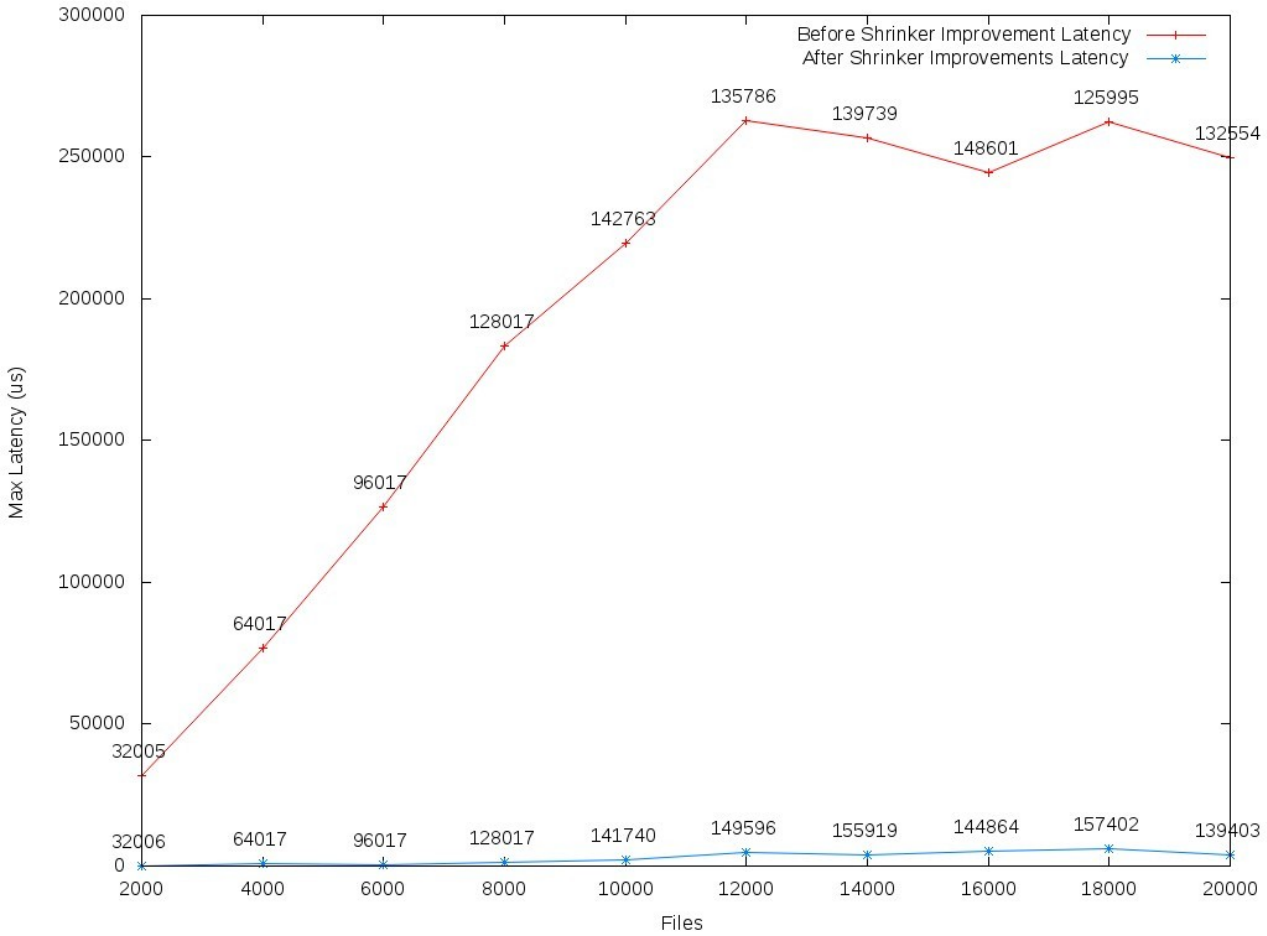
Improving Shrinker

- Joint work with Zheng Liu
- Walk inodes in round-robin order instead of LRU
  - No need for list sorting
- Remember where we stopped scanning inode RB-tree and start next scan there
  - Enough to scan (not reclaim) N entries
- Add simple aging via extent REFERENCED bit
- Merged in 3.19

Shrinker Tests

- Create 32 KB sparse files from 16 processes
- Show maximum latency of a shrinker in 5 minute run
- Large memory allocation once per minute to trigger reclaim
- Test run on 8 CPU machine with 4 GB RAM, filesystem on standard SATA drive

Test With Lots of Files



Test With Two 1GB Files

- Single process creates two 1GB sparse files
- Large allocation to trigger reclaim
- Max latency reduced from 63132 us to 100 us

Checkpoint List Scanning

Checkpoint List Processing

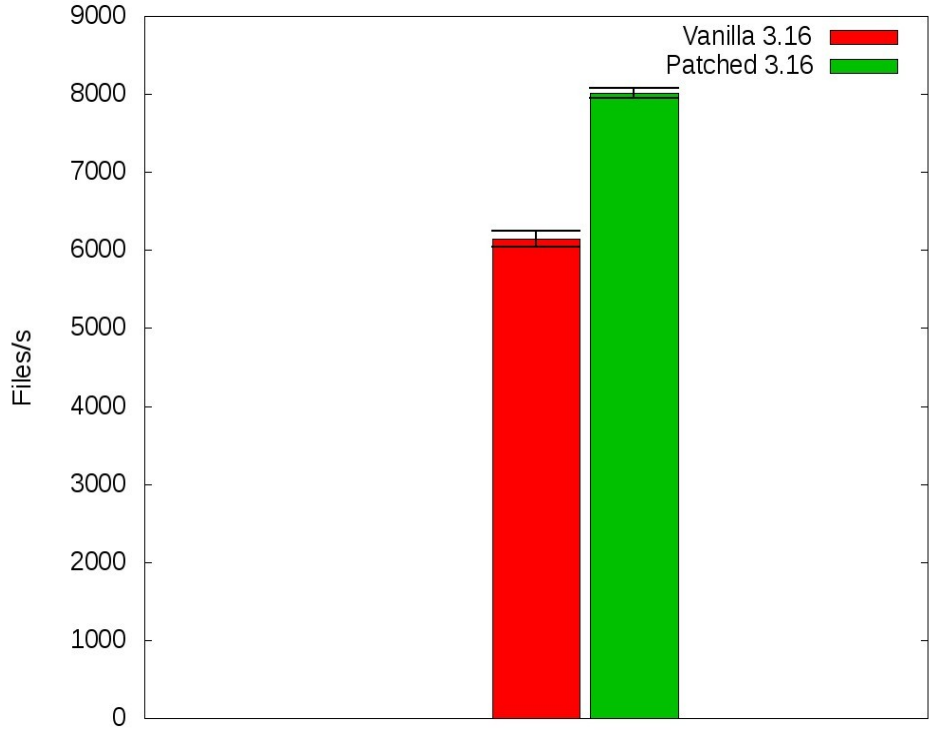
- Joint work with Yuanhan Liu
- JBD2 tracks all buffers that need checkpointing with each transaction
- List cleanup on transaction commit and when running out of journal space
- Expensive for loads with lots of small transactions
  - fs_mark creating 100000 files, 4 KB each, with fsync

```
44.08%  jbd2/ram0-8 [kernel.kallsyms] 0xffffffff81017fe9
23.76%  jbd2/ram0-8 [jbd2]            journal_clean_one_cp_list
17.22%  jbd2/ram0-8 [jbd2]            __jbd2_journal_clean_checkpoint
```

- Scanning list repeatedly not cleaning up anything

Checkpoint List Processing Speedup

- No big point in further scanning once we find buffer that cannot be cleaned up
- Reduces cleanup time from $O(N^2)$ to $O(N)$
- Merged for 3.18



Conclusion

Takeaways

- Even if you cannot fundamentally improve scalability, just reducing length of critical section can help a lot
- But doing things lockless is still faster
- Balance between “sophisticated but slow” and “simple and fast”

Thank you


suse®