Making Production-Ready Filesystems: A case study using Ext4

Theodore Ts'o
Google
January 20, 2010
13:30

What makes a function/library hard to test?

- Premature Optimization
- Large Amounts of Internal State
- Lots of parallelism

What makes a file system hard to make robust?

- Optimization demanded for many workloads
- A file system's job is to store a lot of state
- Lots of parallelism

File systems are like fine wine...

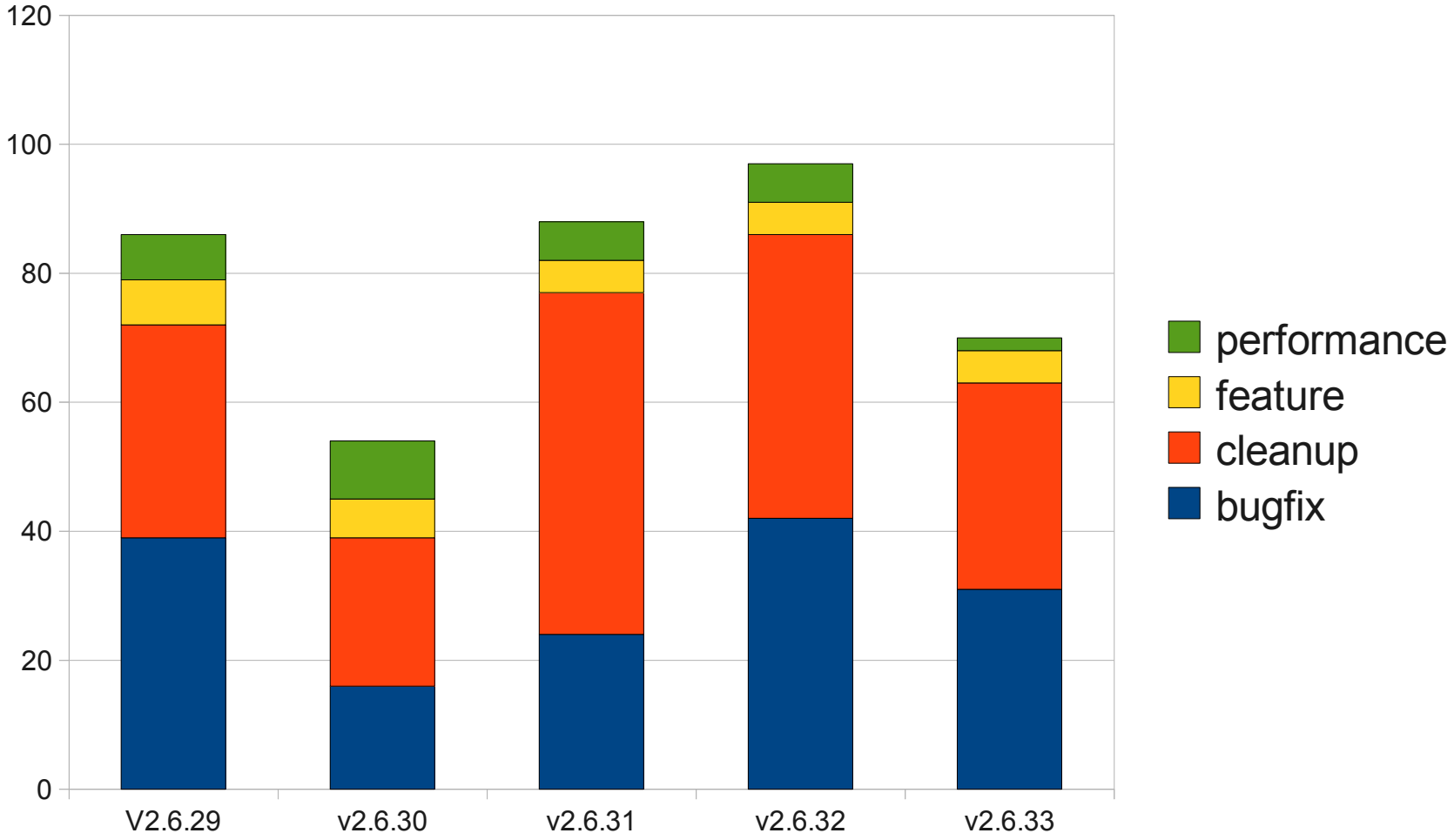
Meet the ZFS team...

Meet the ZFS team...

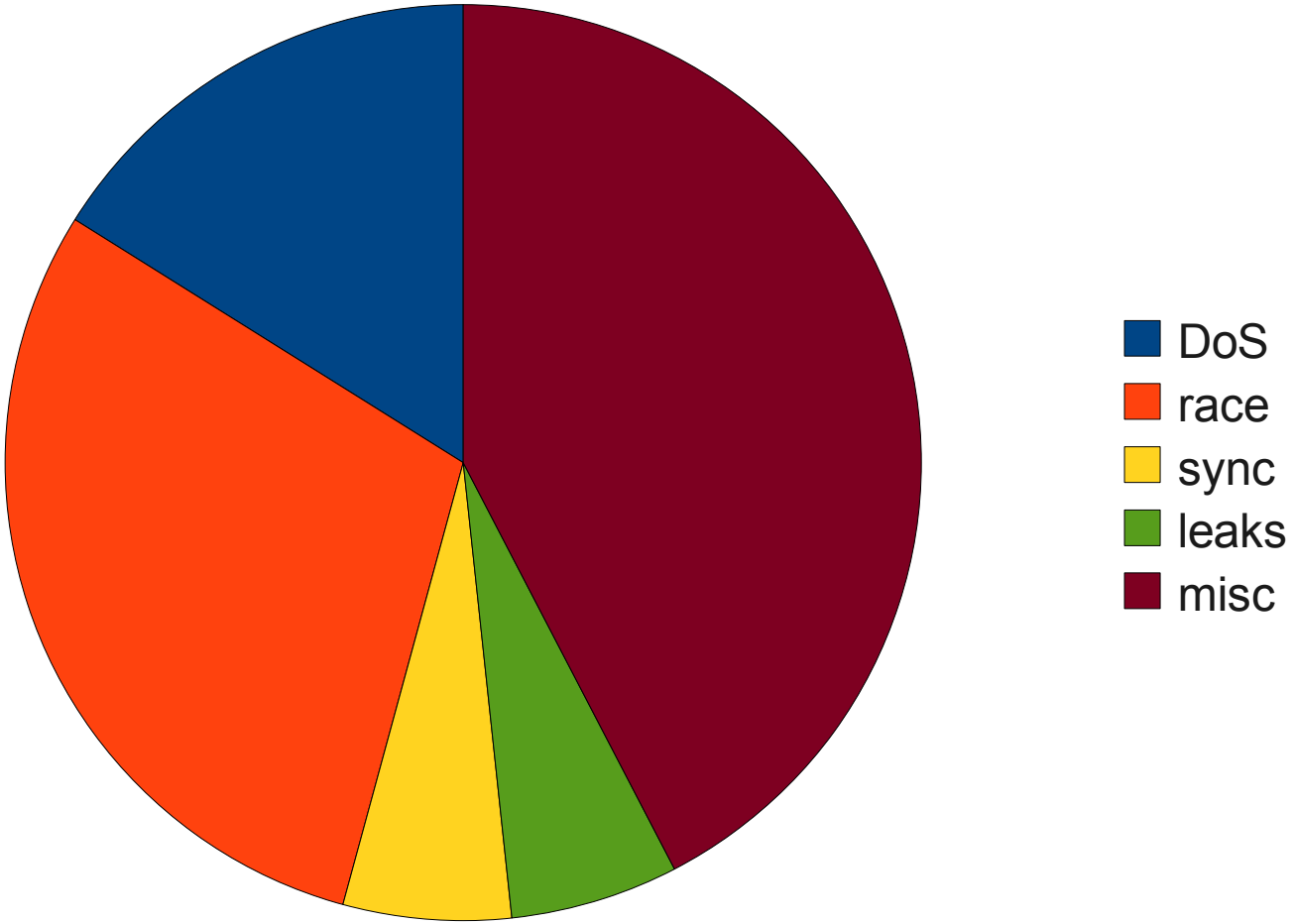
Where are we then with ext4?

- Renamed from ext4dev to ext4 in 2.6.28 (December 25, 2008)
- Shipping in community distributions
  - Fedora 11, Ubuntu 9.10, Open SuSE
  - In RHEL 5, SLES 11 as technology preview
- Hopefully will be showing up in upcoming enterprise/LTS distributions
- Widespread adoption in data centers 12+ months later

Characterizing recent ext4 changes



Break down of bugfixes



Where were the bug fixes?

- Interaction between the new allocator and online resize
- Preallocation races and ENOSPC issues
- On-line defrag
- Fiemap
- Quota
- No Journal mode

What were the new features?

- New tracepoints and features for perfomance tuning / debugging
- Work around non-fsync'ing applications
- On-line defrag
- Fiemap
- Quota
- No-journal mode

Some example bugs that we found

- Fix race in ext4_inode_info.i_cached_extent
  - Fixed in 2.6.30
  - “If two CPU's simultaneously call ext4_ext_get_blocks() at the same time, there is nothing protecting the cached_extent structure from being used and updated at the same time. This could potentially cause the wrong location on disk to be read or written to, including potentially causing the corruption of the block group descriptors and/or inode table.”

How did this bug slip by?!?

- Was part of code donation from Lustre
  - In production use for years there
  - Large code donation; lack of locking missed in the code review
  - Used the file system as an object store
    - Single threaded reads and writes
- Turns out most file system writes tend to be single-threaded.
  - I was using ext4 in production for 9 months; never noticed the problem

Some example bugs that we found

- Fix race in ext4_inode_info.i_cached_extent
- jbd2: fix race between write_metadata_buffer and get_write_access
  - Fixed in 2.6.31
  - “Jbd2_journal_write_metadata_buffer() calls jbd_unlock_bh_state(bh_in) too early; this could potentially allow another thread to call get_write_access on the buffer head, modify the data, and dirty it, and allowing the wrong data to be written into the journal....”

OMG! Sounds serious...

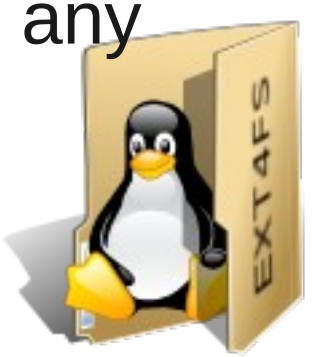
- “...Fortunately, if we lose this race, the only time this will actually cause filesystem corruption is if there is a system crash or other unclean shutdown of the system before the next commit can take place.”
- Turns out this bug was in ext3 too....

Some example bugs that we found

- Fix race in ext4_inode_info.i_cached_extent
- jbd2: fix race between write_metadata_buffer and get_write_access
- Remove bogus BUG() check in ext4_bmap()
  - Fixed in 2.6.29
  - “The code to support no journal ext4 operation added a BUG to ext4_bmap() which fired if there was no journal and the JDATA bit was set in the i_state field. This caused running the filefrag program (which uses the FIMBAP ioctl) to trigger a BUG()”

Some example performance fixes

- avoid unnecessary spinlock in critical POSIX ACL path
  - “If a filesystem supports POSIX ACL's, the VFS layer expects the filesystem to do ACL checks on any files not owned by the caller, and it does this for every single pathname component that it looks up. That obviously can be pretty expensive if the file system isn't careful about it, especially with locking. That's doubly sad, since most files don't have any ACL's.”



Some example performance fixes

- avoid unnecessary spinlock in critical POSIX ACL path
- Fix discard of inode prealloc space with delayed allocation
  - “With delayed allocation we can not discard inode prealloc space during file close, since still have dirty pages for which we haven't allocated blocks yet. With this fix after each get_blocks request we check whether we have no more delalloc blocks and if there are no open fd's for write. If so, we can release the inode prealloc space.”

Some example performance fixes

- avoid unnecessary spinlock in critical POSIX ACL path
- Fix discard of inode prealloc space with delayed allocation
- Adjust ext4_da_writepages() to write out larger contiguous chunks
  - “Work around problems in the writeback code to force out writebacks in larger chunks than just 4mb, which is just too small...”

Conclusion

- File systems are easy!
  - We have 66 of them in the kernel sources
  - Lots of generic library support code; just provide a bmap function and add water...

Conclusion

- ~~File systems are easy!~~
- File systems are **hard**!
  - General purpose file systems have to work well on many different workloads
  - Users want high performance out of their file system
  - Many processes will be hitting it at the same time.

Conclusion

- ~~File systems are easy!~~
- File systems are **hard**!
- Making a production-ready, general purpose file system takes longer and takes more effort than you expect
  - Labor of love
  - Justifying it from a business perspective can be challenging

YouTube Links

- http://www.youtube.com/watch?v=oSs6DcA6dFI
- http://www.youtube.com/watch?v=A6P1ifGjvEE