

ORACLE[®]

Per backing device writeback

Jens Axboe <jens.axboe@oracle.com>
Consulting Member of Staff

Disclaimer!

- I don't really know what I'm talking about
- Diversity is always good
- Expanding your comfort zone is also good

Outline

- Dirty data and cleaning
- Tracking of dirty inodes
- pdflush
- Backing device inode tracking
- Writeback threads
- Various test results

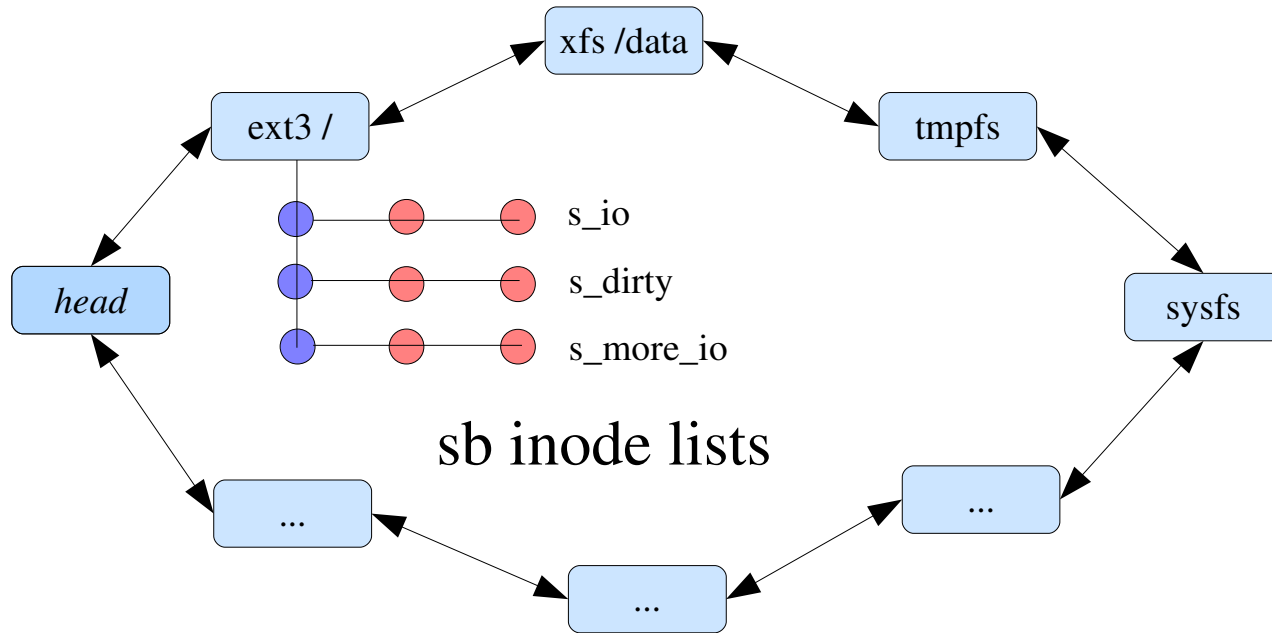
Dirty data

Process

- App does write(2), copy to mmap, splice(2), etc
 - *balance_dirty_pages()*
- chown(2), read(2)
 - Everybody loves atime
- Pages tracked on a per-inode basis
- Buffered writeback organized through 3 lists
 - sb->s_dirty (when first dirtied)
 - sb->s_io (when selected for IO)
 - sb->s_more_io (for requeue purposes (I_SYNC))
- Lists are chronologically ordered by dirty time
 - Except ->s_more_io

Spot the inode

super_blocks list



Dirty data

Cleaning

- Background vs direct cleaning
 - */proc/sys/vm/dirty_background_ratio*
 - */proc/sys/vm/dirty_ratio*
- Kupdate
 - */proc/sys/vm/dirty_expire_centiseecs*
 - *Max age*
 - */proc/sys/vm/dirty_writeback_centiseecs*
 - *Interval between checks and flushes*
- fsync(2) and similar
- WB_SYNC_ALL and WB_SYNC_NONE



Writeback loop

```
For each sb
  For each dirty inode
    For each page in inode
      writeback
```

- Inode starvation
 - MAX_WRITEBACK_PAGES
 - Incomplete writes moved to back of b_dirty

Writeback example

- 5 dirty files
 - 3 4KB (f1..f3)
 - 2 10MB (f4..f5)
- First sweep
 - f1...f3 4kb, f4...f5 4MB
- Second sweep
 - f4...f5 4MB
- Repeat

Writeback control

- `struct writeback_control`
- Passes info down:
 - Pages to write
 - Range cyclic or specific range start/end
 - Nonblocking
 - Integrity
 - Specific age / `for_kupdate`
- And back up:
 - Congestion
 - `more_io`
 - Pages written

Memory pressure

- Concerns all devices
- Scan *super_blocks* from the back
 - Need to hold `sb_lock` spinlock.
- `wbc`
 - `WB_SYNC_NONE`
 - Number of pages to clean
- *generic_sync_sb_inodes(sb, wbc)*
 - Matches `sb/bdi`
 - 'Pins' `bdi`
 - Works `sb->s_io`
 - Stops when `wbc->nr_to_write` is complete

Device specific writeback

- bdi level
- Too many dirty pages
- Same path as memory pressure
 - Same `super_blocks` traversal, `sb_lock`, etc
- `WB_SYNC_ALL` and `WB_SYNC_NONE`
- *generic_sync_sb_inodes()* is a mess

pdflush

- Generic thread pool implementation
- Defaults to 2-8 threads
 - sysfs tunable...
- *pdflush_operation(func, arg)*
 - May fail → only usable for non-data integrity writeback
 - Worker additionally forks new threads (and exits)
- 'Pins' backing devices
- Must not block
- Write congestion
- Use for background and kupdate writeback

pdflush issues

- Non-blocking
 - Request starvation
 - Lumpy/bursty behaviour
 - Sits out
- But blocks anyway
 - ->get_block()
 - Locking
- Tendency to fight each other
- Solution → blocking pdflush! Wait...

Idea...

- How to get rid of congestion and non-blocking
- Per-bdi writeback thread
- Kernel thread count worry
 - Lazy create, sleepy exit

struct backing_dev_info

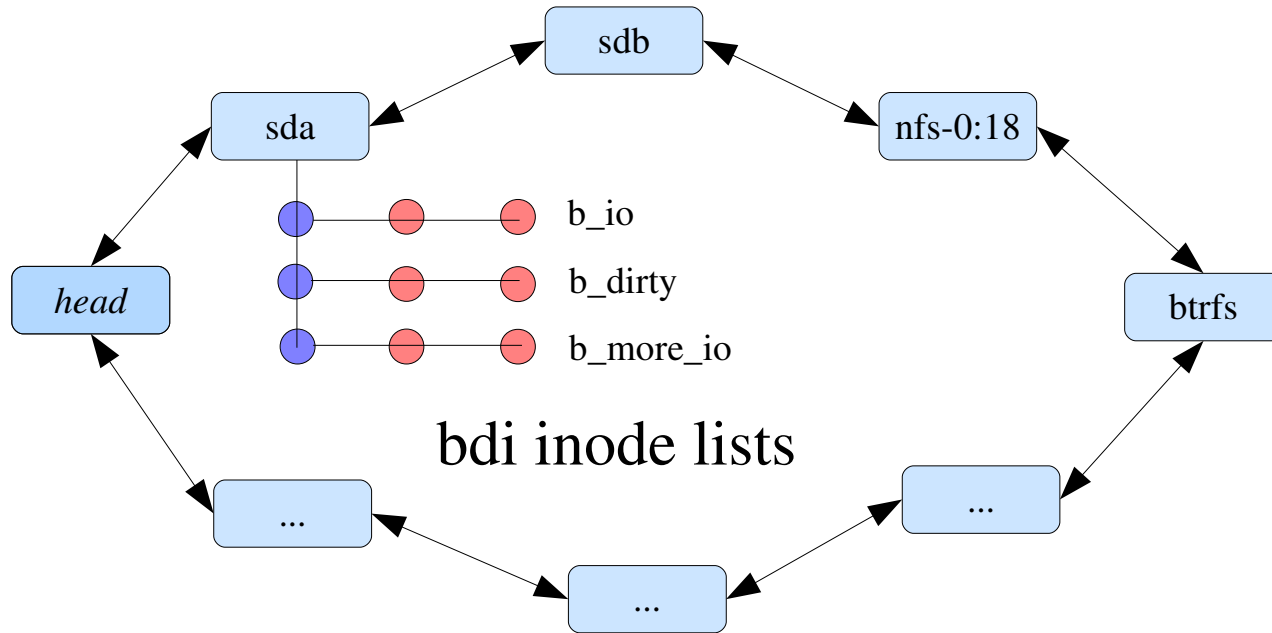
- Embedded in block layer queue
- But can be used anywhere
 - NFS server
 - Btrfs device unification
 - DM/MD etc expose single bdi
- Functions:
 - Congestion/unplug propagation
 - Dirty ratio/threshold management
- Good place to unify dirty data management

Dirty inode management

- Sub-goal: remove dependency on sb list and lock
- Make it “device” local
 - `sb->s_io` → `bdi->b_io`
- Could be done as a preparatory patch
 - No functional change, except `sb_has_dirty_io()`
- `super_block` referencing

Bdi inodes

bdi_list list



Cleaning up the writeback path

- Sync modes different, yet crammed into one path
- Move `writeback_control` structure a level down
 - `struct wb_writeback_args`
- Introduce `bdi_sync_writeback()`
 - Takes *bdi* and *sb* argument
- Introduce `bdi_start_writeback()`
 - Takes *bdi* and *nr_pages* argument
- `bdi_writeback_all()` persists
 - Memory pressure
 - `super_block` specific writeback

New issues

- super_block sync now trickier
 - Bdi dirty inode list could contain many supers
- File system vs file system fairness?
 - Time sorted list should handle that
- No automatic super_block pinning
 - WB_SYNC_ALL ok, WB_SYNC_NONE not so much

Writeback threads

- One per bdi
 - `default_backing_dev_info` is “master” thread
 - Prepared for > 1 thread
- Accepts queued work
 - `WB_SYNC_NONE` completely out-of-line
 - May complete on “work seen”
 - `WB_SYNC_ALL` is waited on
 - Completes on “work complete”
 - Different types of writeback handled
- Memory pressure path now lockless
 - Opportunistic `bdi_start_writeback()`, like `pdflush()`
- Thread itself congestion agnostic

struct bdi_work

```
/* Internal argument wrapper */
struct wb_writeback_args {
    long nr_pages;
    struct super_block *sb;
    enum writeback_sync_modes sync_mode;
    int for_kupdate;
    int range_cyclic;
    int for_background;
};
/* Internal work structure */
struct bdi_work {
    struct list_head list;
    struct rcu_head rcu_head;
    unsigned long seen;
    atomic_t pending;
    struct wb_writeback_args args;
    unsigned long state;
};
```

Work queuing

```
spin_lock(&bdi->wb_lock);  
list_add_tail_rcu(&work->list, &bdi->work_list);  
spin_unlock(&bdi->wb_lock);  
  
if (list_empty(&bdi->wb_list))  
    wake_up_process(default_bdi.task);  
else {  
    if (bdi->task)  
        wake_up_process(bdi->task);  
}
```

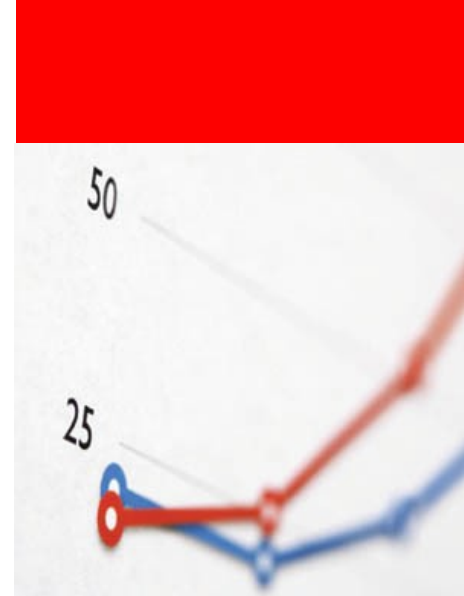
Work queuing continued

- Work items small enough for on-stack alloc
- If thread isn't there, wake up our master thread
 - Master thread auto-forks threads when needed
 - Forward progress guarantee
- Work list itself is also RCU protected
 - Could go away, depends on multi-thread direction
 - Each work item has a 'thread bit mask' and count
- Thread itself decides to exit, if “too idle”

pdflush vs writeback threads

- Small system has same or fewer threads
- Big system has more threads
 - But needs them
 - And exit if idle
- Can block on resources
 - Knowingly
 - or inadvertently, like pdflush
- Good cache behaviour

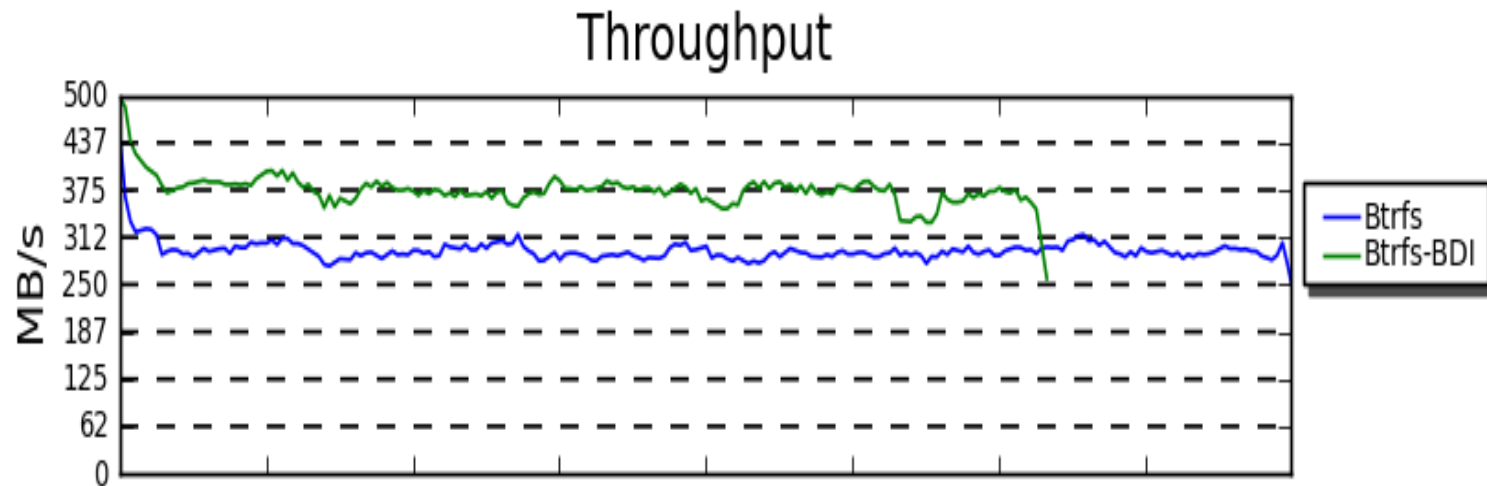
Performance **Results**



Test setup

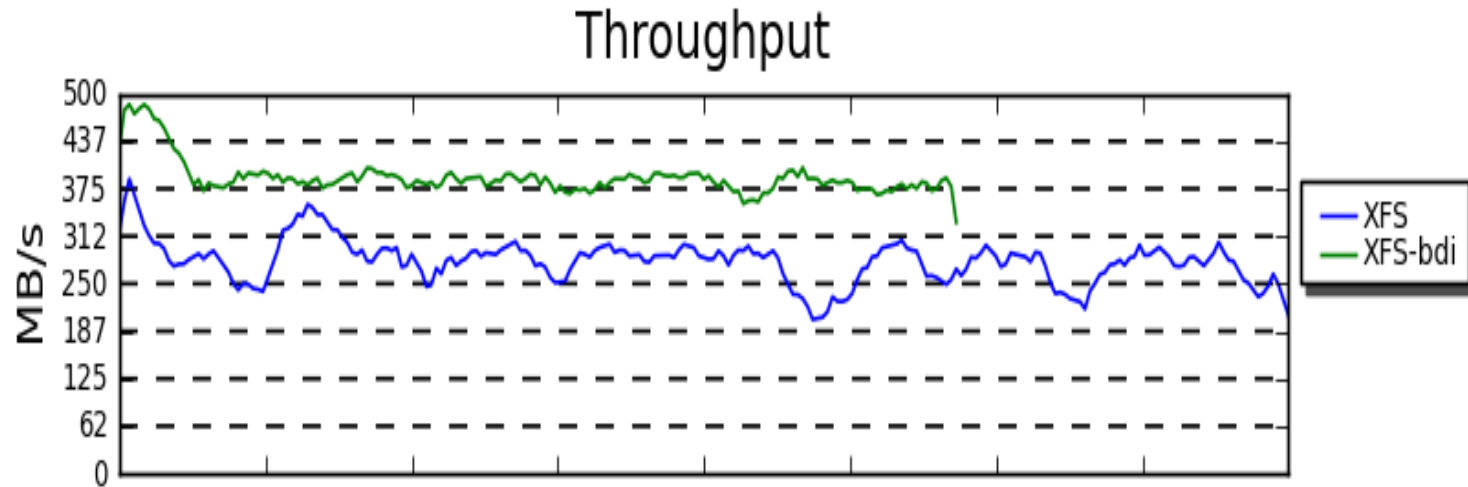
- 32 core / 64 thread Nehalem-EX, 32GB RAM
 - 7 SSD SLC devices
 - XFS and btrfs
- 4 core / 8 thread Nehalem workstation, 4GB RAM
 - Disk array with 5 hard drives
 - XFS and btrfs
- 2.6.31 + btrfs performance branch → *baseline*
 - Baseline + bdi patches from 2.6.32-rc → *bdi*
- Deadline IO scheduler
- fio tool used for benchmarks
- Seekwatcher for pretty pictures and drive side throughput analysis

2 streaming writers, btrfs



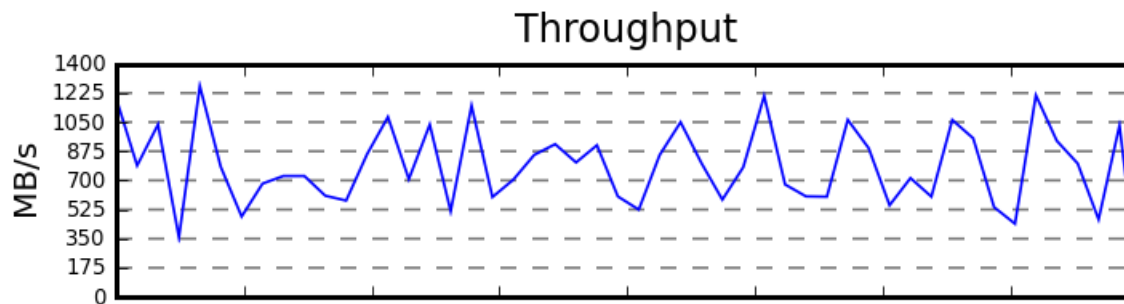
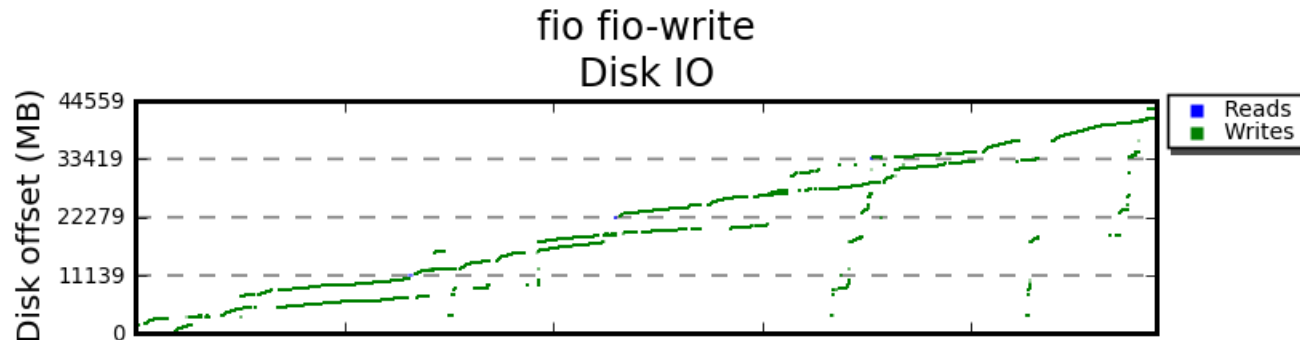
	Avg Seeks/s	Avg MB/s	Avg IO/s	Run time
Btrfs	99.7	293.75	4742.57	223.7
Btrfs-BDI	114.46	372.69	6018.61	176.27

2 streaming writers, XFS



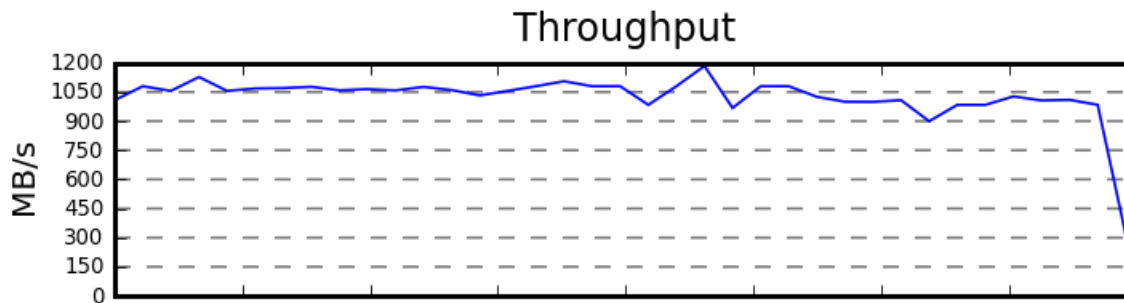
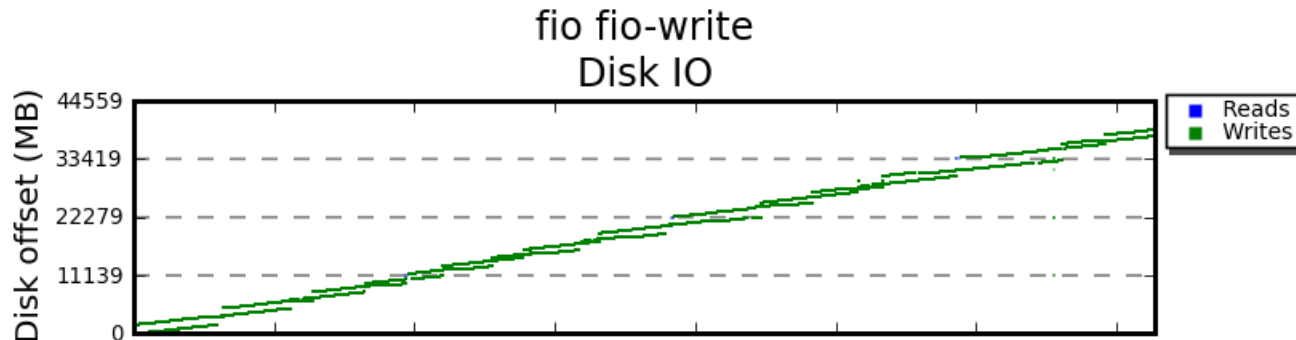
	Avg Seeks/s	Avg MB/s	Avg IO/s	Run time
XFS	145.98	277.59	8314.69	236.09
XFS-bdi	124.01	388.29	16739.2	168.78

2 streaming writers, XFS, mainline



Avg Seeks/s	Avg MB/s	Avg IO/s	Run time
11329.87	796.17	203818.48	48.88

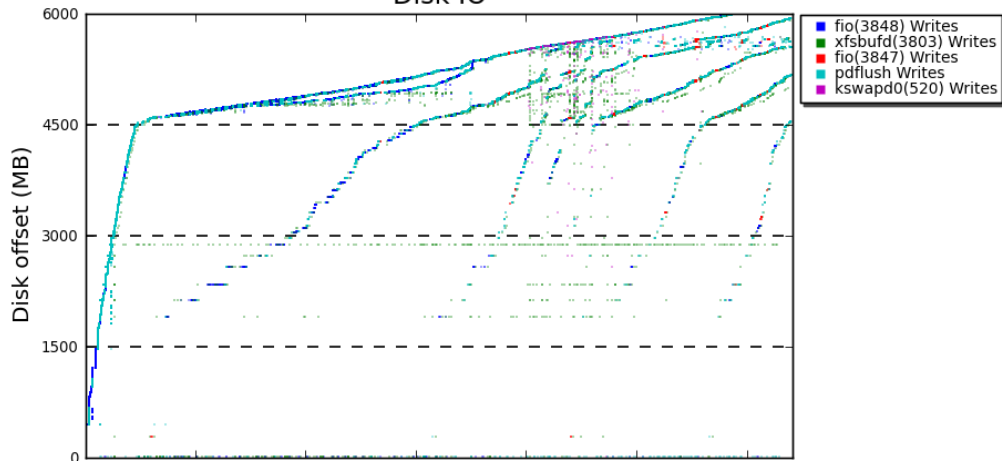
2 streaming writers, XFS, bdi



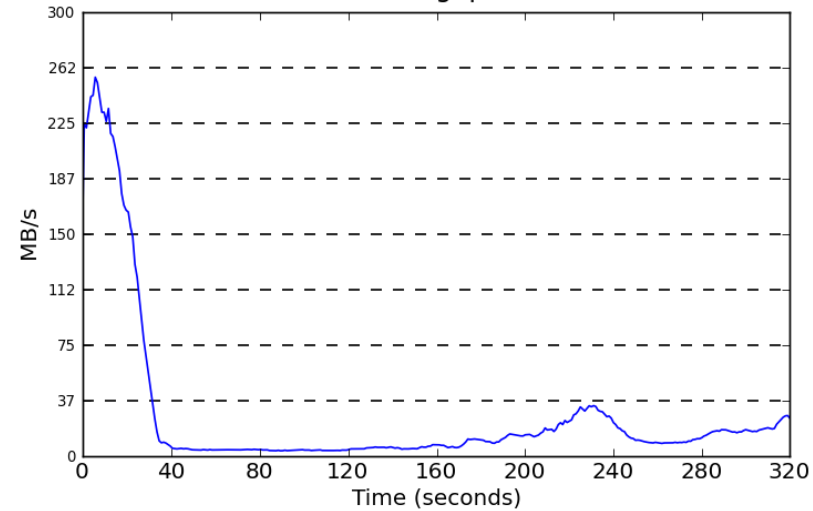
Avg Seeks/s	Avg MB/s	Avg IO/s	Run time
64.35	1042.15	266789.97	36.38

Streaming vs random writer, mainline

Mixed Buffered Random and Streaming IO
Disk IO



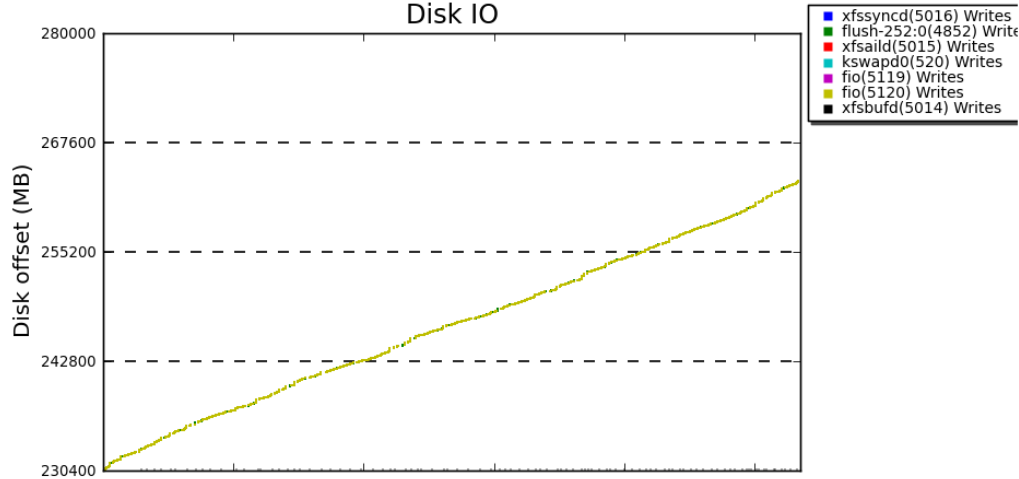
Throughput



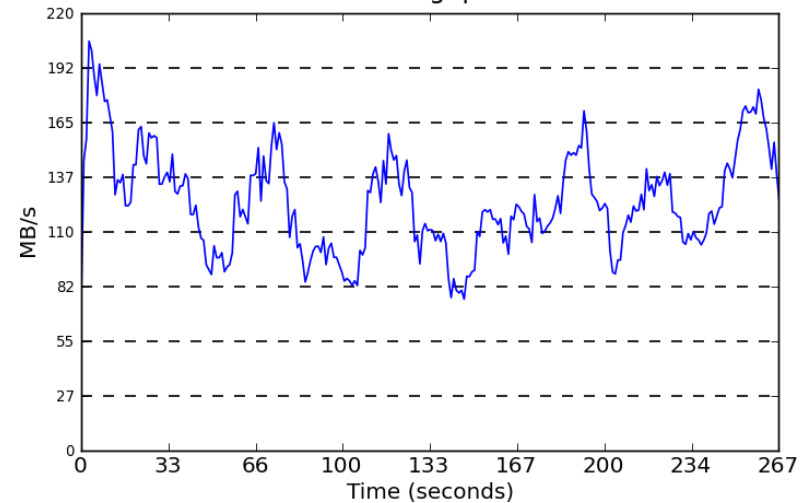
Streaming vs random writer, bdi

“anyone that wants to argue the mainline graph is better is on crack”, Chris Mason

BDI Writeback Mixed Random and Streaming IO
Disk IO



Throughput



Outside results

Shaohua Li <shaohua.li@intel.com> on LKML

Commit d7831a0bdf06b9f722b947bb0c205ff7d77cebd8 causes disk io regression in my test.

*commit d7831a0bdf06b9f722b947bb0c205ff7d77cebd8
Author: Richard Kennedy <richard@rsk.demon.co.uk>
Date: Tue Jun 30 11:41:35 2009 -0700*

mm: prevent balance_dirty_pages() from doing too much work

My system has 12 disks, each disk has two partitions. System runs fio sequence write on all partitions, each partition has 8 jobs.

2.6.31-rc1, fio gives 460m/s disk io

2.6.31-rc2, fio gives about 400m/s disk io. Revert the patch, speed back to 460m/s

Under latest git: fio gives 450m/s disk io; If reverting the patch, the speed is 484m/s.

Room for improvement

TODO

- Size of each writeback request
 - MAX_WRITEBACK_PAGES
- Support for > 1 thread/bdi per consumer
 - Needed for XXGB/sec IO
- Killing ->b_more_io
- More cleaning up of fs-writeback.c
 - → end goal a less fragile infrastructure
- Add writeback tracing
- Testing!

Resources

- Merged in 2.6.32-rc1
- Kernel files
 - `fs/fs-writeback.c`
 - `mm/page-writeback.c`
 - `mm/backing-dev.c`
 - `include/linux/writeback.h`
 - `include/linux/backing-dev.h`
- fio
 - `git clone git://git.kernel.dk/data/git/fio.git`
- seekwatcher
 - <http://oss.oracle.com/~mason/seekwatcher/>

Thanks!

- Chris Mason for prodding me to do this work, testing, and lots of good advice.
- Jan Kara for relentless reviewing and comments.
- Wu Fengguang, Christoph Hellwig

Questions?



ORACLE IS THE **INFORMATION** COMPANY

Thanks!