Rate based Dirty Throttling

Wu Fengguang
<wfg@linux.intel.com>

April 17, 2011

Rationals

Why the complexity?

IO-less reduce XFS contentions and disk seeks

low latency capable of $<$ 10ms pause times

smoothness maintain constant task dirty rate over time

fairness tasks progress at exactly the same rate

scalability 1000+ dd, O(1) algorithm

task IO controller endogenous

cgroup IO controller well integrated

proportional IO controller endogenous

rsync problem

how each fluctuates:
(large ==> small)

Workload
========

50 dd + 1 remote rsync
writing to an XFS

rsync bytes/sec
===============

vanilla	545098.791
Jan	612853.130 (*)
Wu	891014.654

(*) need double check

----- net send per second -----

vanilla	Jan	Wu
130B	538k	806k
538k	538k	807k
273k	543k	1081k
269k	2690k	808k
114B	270k	1076k
114B	270k	676k
114B	812k	943k
114B	808k	807k
114B	539k	1077k
168B	269k	808k
116k	807k	1082k
153k	808k	807k
3230k	543k	1067k
1344k	269k	819k
130B	808k	1081k
2150k	808k	808k
130B	1347k	808k
114B	1888k	1078k
114B	337k	812k
114B	471k	807k
114B	538k	807k
114B	808k	1076k
114B	273k	807k

rsync findings

	vanilla	Jan	Wu
balance_dirty_pages() pause time	0-3s	0-300ms	60-70ms
rsync throughput over Ethernet	1	+12.4%	+63.5%

$\Longrightarrow$

Smooth and **low latency** writeback helps!

Fundemental tradeoffs

Inescapable IO completion fluctuations from FS/storage ......

`Where to embody the fluctuations?`

`(1)` **THRESHOLD** `based throttling`

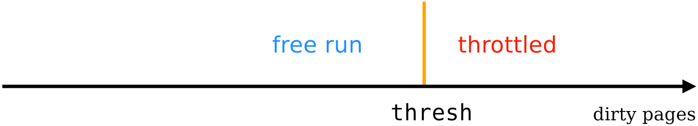
`keep dirty pages at THRESHOLD; fluctuations go to dirty rate`

`(2)` **RATE** `based throttling + gentle` **POSITION CONTROL**

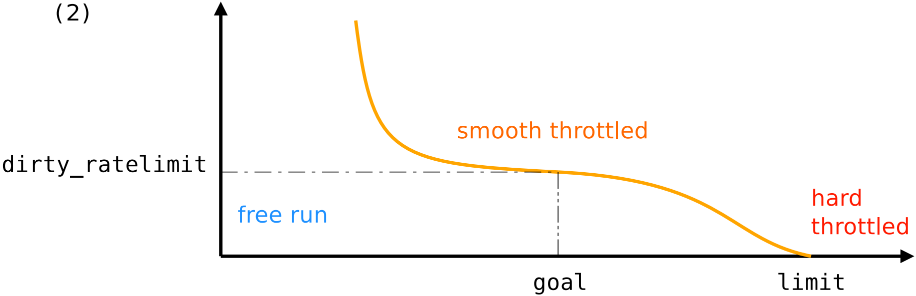
`keep task dirty RATE stable; allow fluctuations in dirty pages`

The change

(1)



(2)



Basic scheme

dirty throttling

```
write() syscall
    balance_dirty_pages(pages_dirtied)
        task_ratelimit = dirty_ratelimit * pos_ratio;
        pause = pages_dirtied / task_ratelimit;
        sleep(pause);
```

balanced state (rate and position)

```
dirty_ratelimit  aka. 'base throttle bandwidth'
== write_bw / N
== (write IO bandwidth) / (# of dd tasks)

pos_ratio == 1.0
```

Principles

- **control** `pages_dirtied` **to get the desired** `pause`

  $\Longrightarrow$ **low latency**

- **stable** `dirty_ratelimit` + **gently reacting** `pos_ratio`

  $\Longrightarrow$ **smoothness**

the balanced throttle bandwidth (theory)

When started N dd, throttle each dd at

```
        task_ratelimit = dirty_ratelimit_0 (any non-zero initial value is OK)
```

After 200ms, we got

```
        dirty_bw = # of pages dirtied by app  / 200ms
        write_bw = # of pages written to disk / 200ms
```

For aggressive dirtiers, the equality holds

```
        dirty_bw == N * task_ratelimit
                 == N * dirty_ratelimit_0                              (1)
```

The balanced throttle bandwidth can be estimated by

```
        ref_ratelimit = dirty_ratelimit_0 * write_bw / dirty_bw       (2)
```

From (1) and (2), we get equality

```
        ref_ratelimit == write_bw / N                                  (3)
```

If the N dd's are all throttled at ref_ratelimit, the dirty/writeback rates will match.

the balanced throttle bandwidth (theory)

`ref_ratelimit` estimated in 200ms!

However, real world is not perfect . . .

the balanced throttle bandwidth (practice)

ITERATIVE METHOD

ref_ratelimit is fluctuating, has estimation errors due to control lags and write_bw errors. It naturally asks for step-by-step approximations:

```
dirty_ratelimit = (dirty_ratelimit * 3 + ref_ratelimit) / 4
```

CONDITIONAL UPDATE

There is no need to update dirty_ratelimit during a stable workload, which only makes it susceptible to noises. So do it defensively and update dirty_ratelimit when

- dirty pages are departing from the global/bdi goals
- dirty pages are near the upper/lower bounds of the control scope

position control

Adjust dirty rate to keep dirty pages around the desired position.

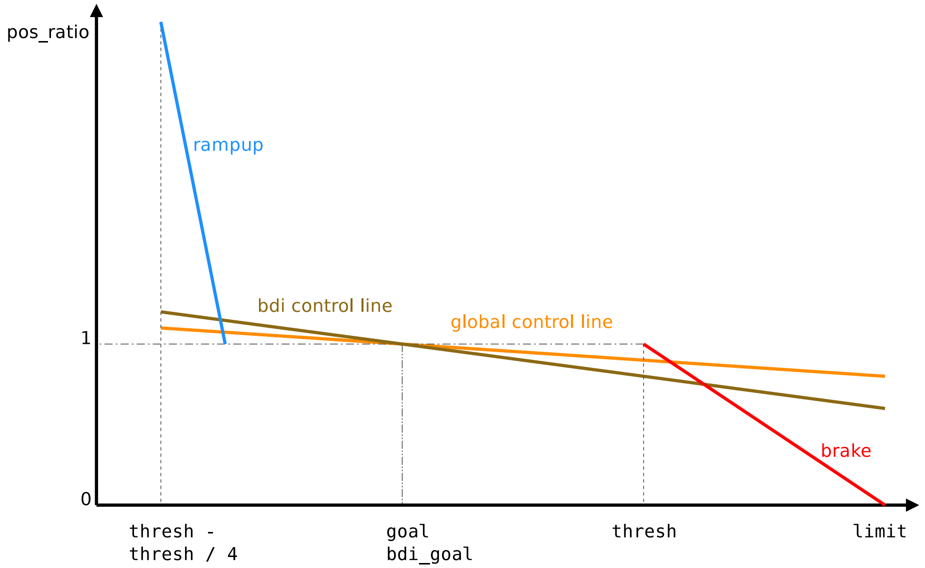
```
pos_ratio = 1.0

// gentle negative feedback control
pos_ratio -= (nr_dirty - goal) / SCALE;
pos_ratio -= (bdi_dirty - bdi_goal) / BDI_SCALE;

// sharp boundary control
if (near global limit)   scale down pos_ratio
if (bdi queue runs low)  scale up   pos_ratio

task_ratelimit = dirty_ratelimit * pos_ratio
```

position control lines



Roll it up

on `write()` syscall

```
balance_dirty_pages(pages_dirtied)
{
    task_ratelimit = bdi->dirty_ratelimit * bdi_position_ratio();
    pause = pages_dirtied / task_ratelimit;
    sleep(pause);
}
```

on every 200`ms`

```
bdi_update_dirty_ratelimit()
{
    bw = bdi->dirty_ratelimit;
    ref_bw = bw * bdi_position_ratio() * write_bw / dirty_bw;

    if (dirty pages unbalanced)
        bdi->dirty_ratelimit = (bw * 3 + ref_bw) / 4;
}
```

Implementation

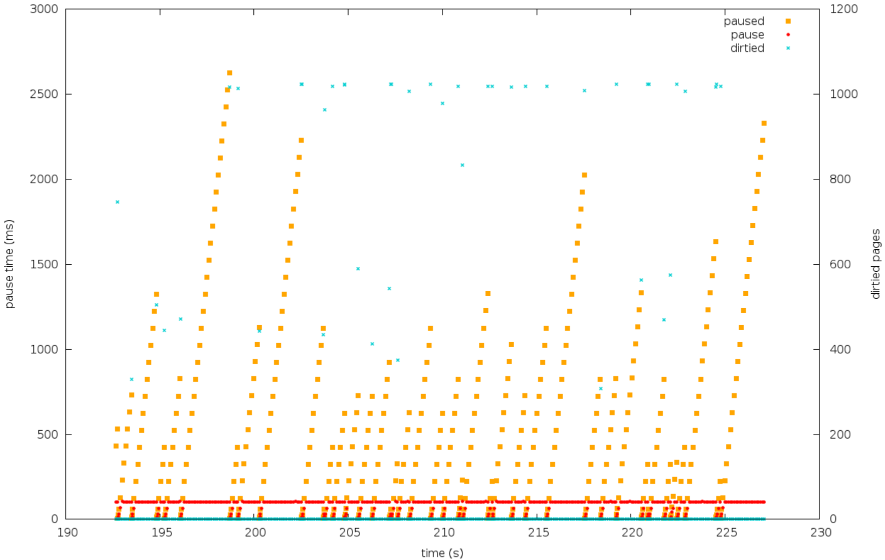
```
mm/page-writeback.c
  799 insertions(+), 196 deletions(-)

  600+ more lines
  200+ dedicated for smoothing/filtering
```

- **smooth and low latency**
- **scale to 1000+ dd**
- **heavily tested and ready for use**
- **future proof**
  - per-task and per-cgroup IO controllers
  - bandwidth and proportional IO controllers

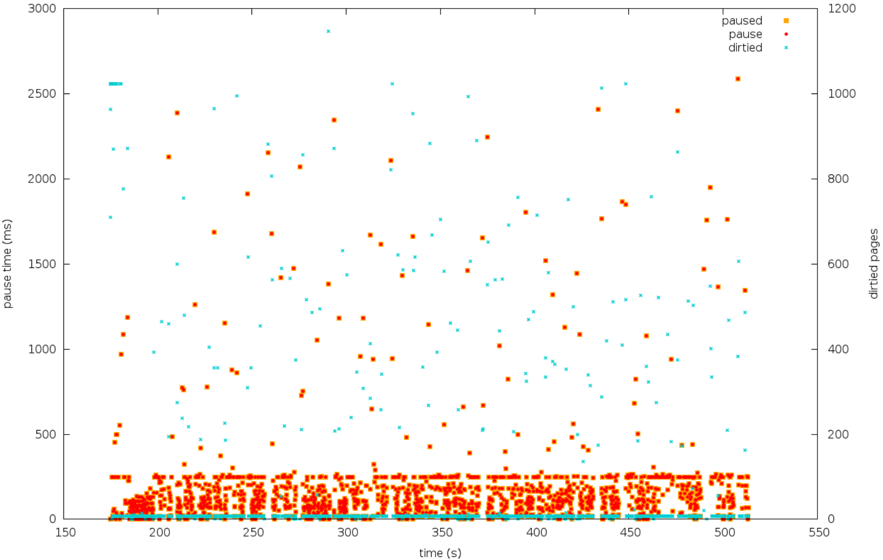
Case Studies

pause time: 0-2500 ms (legacy kernel)



xfs, 8 dd, 4G mem, 2.6.38-rc7

pause time: 0-2500 ms (Jan, JBOD)

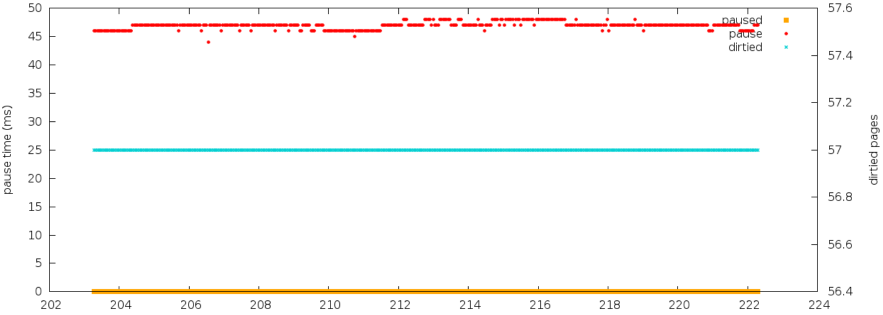


xfs, 10 HDD JBOD, 32 dd on each disk, 6G mem, 2.6.38-rc8-jan-bdp+

pause time: 50ms

- `balance_dirty_pages()` **do sleeps in a** `for(;;)` **loop**
- `pause` **is the pause time in current loop**
- `paused` **is the accumulated pause times in previous loops**

This graph: the task pauses for 45-50ms on every 57 pages dirtied.

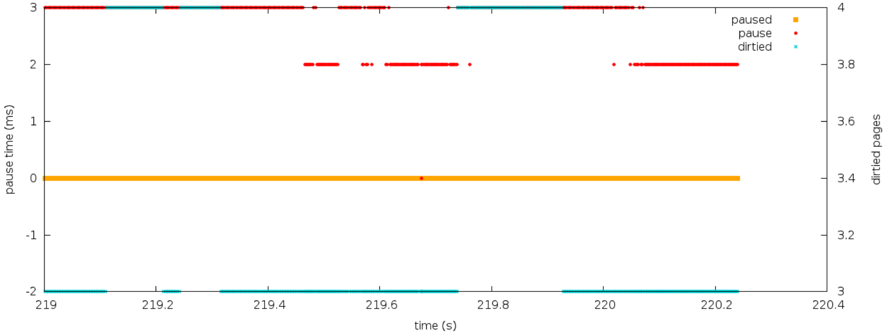


xfs, 10 dd, 4G mem, 2.6.39-rc2-wu-dt7+

pause time: 5ms

To reduce wakeups and CPU overheads, `max_pause()` has a policy to increase the target pause time on growing number of dirtier tasks.

It can actually do lower pause times at will, eg. 5ms:



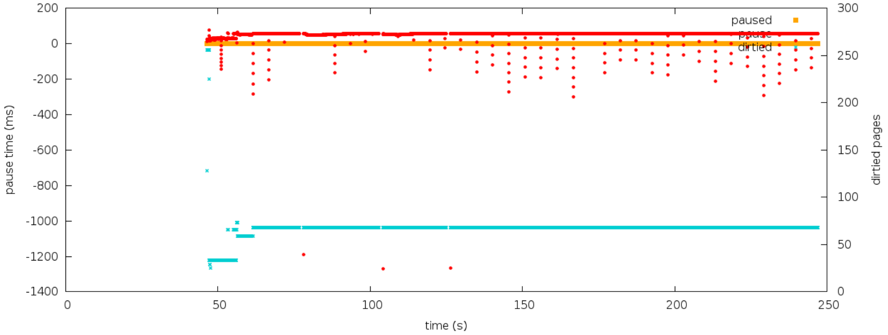
btrfs, 10 dd, 4G mem, 5ms hard coded target pause time

negative pause time

pause < 0 indicate delays outside of `balance_dirty_pages()`

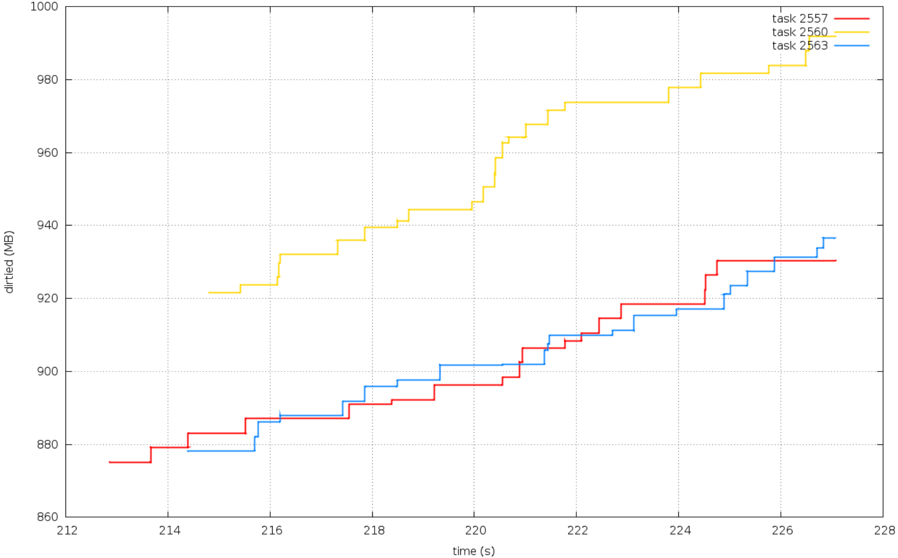
- **user space think time, or**
- `write_begin()` **etc. FS delays**

This graph: each $< 500ms$ ext4 delay shows up as a train of negative pause times; $> 500ms$ delays are not compensated, so are single dots.



ext4, 10 dd, 4G mem, 2.6.39-rc2-wu-dt7+

smoothness: bumping ahead (legacy kernel)

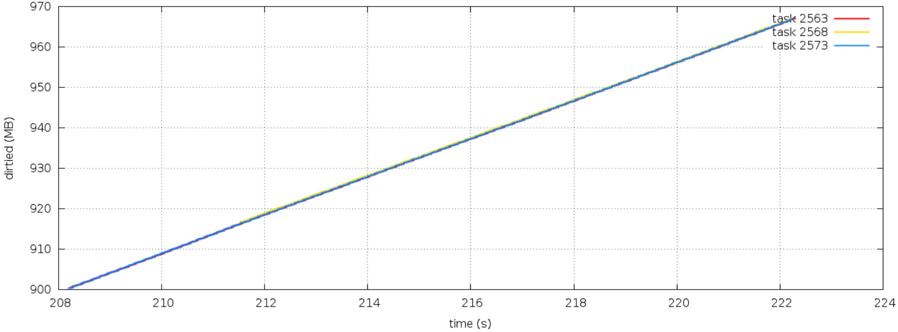


xfs, 8 dd, 4G mem, 2.6.38-rc7

smoothness: straight lines

3 superposed lines

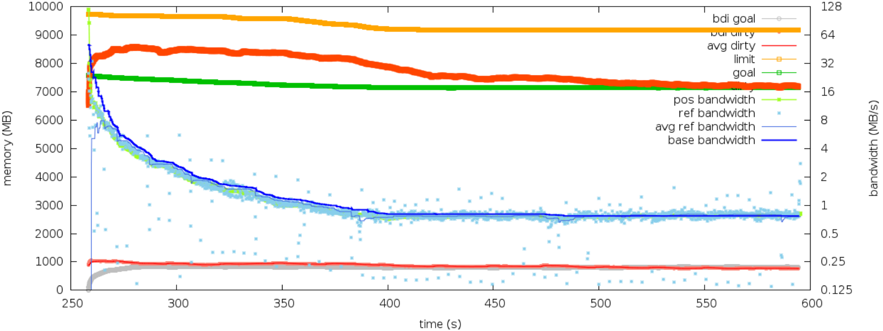
⇒ excellent smoothness and fairness among the 3 dd tasks



xfs, 10 dd, 4G mem, 2.6.39-rc2-wu-dt7+

base bandwidth stability

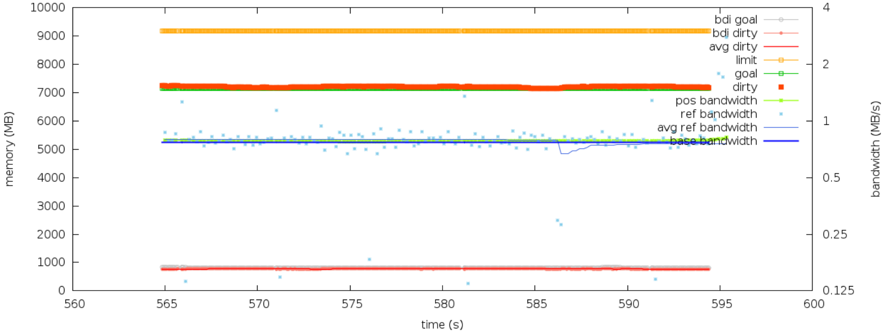
The smoothness originates from the stability of base bandwidth, which won't change as long as being surrounded by the ref bandwidth, avg ref bandwidth and pos bandwidth.



ext4, 10 SSD JBOD, 100 dd on each disk, 64G mem, 2.6.39-rc2-wu-dt7+

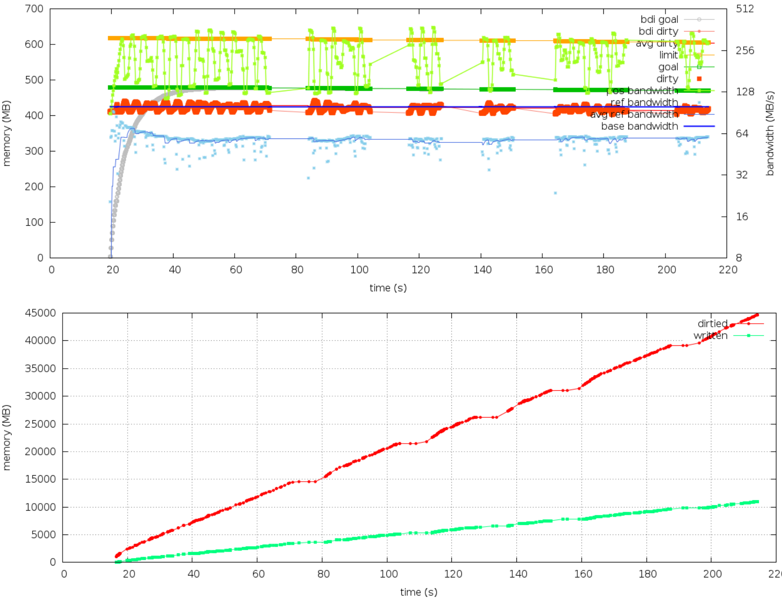
base bandwidth reliability

It's a stable and reliable system. If `ref bw` got systematic errors and drive up/down `base bw`, `dirty pages` will go up/down the `goal` and `pos bw` will in turn go for the opposite side and stop `base bw` from drifting away.



ext4, 10 SSD JBOD, 100 dd on each disk, 64G mem, 2.6.39-rc2-wu-dt7+

base bandwidth reliability (example)

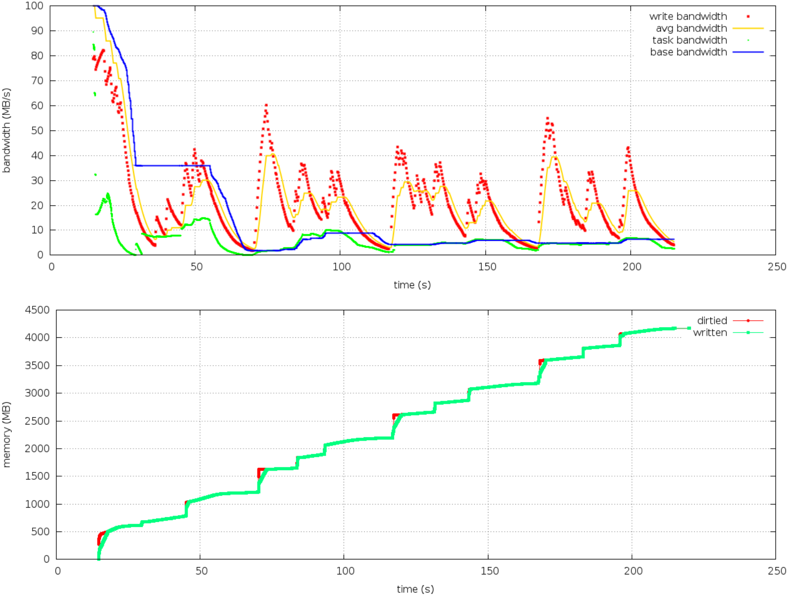


btrfs, 1 dd, 3G mem, 2.6.39-rc3-wu-dt7+: 1kb reads lead to 4 times over-counted dirtied pages and `ref_bw` estimation errors

base bandwidth benefits

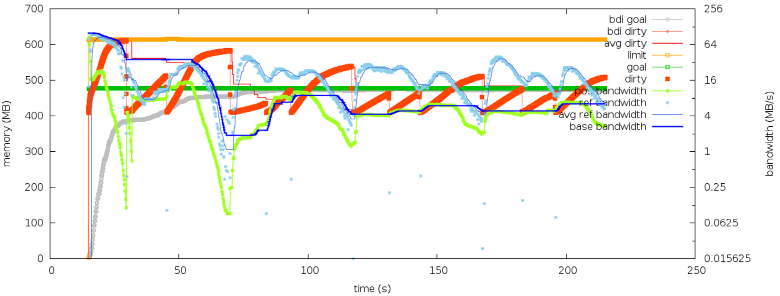
1. **provides per-task bandwidth IO controllers for free**
2. **provides per-task proportional IO controllers for free**
3. **supports per-task policies such as curbing seeky dirtiers more**
4. **per-cgroup IO controllers are demonstrated to be simple**
5. **lockless: the 200ms updates could be moved to each flusher**
6. **adaptiveness: when some task/cgroup is ratelimited by user, the bdi will auto adapt to higher balanced `dirty_ratelimit` for other tasks.**
7. **bumpy workloads: works well on NFS/JBOD; extremely bumpy workloads will be guarded by the boundary control regions.**

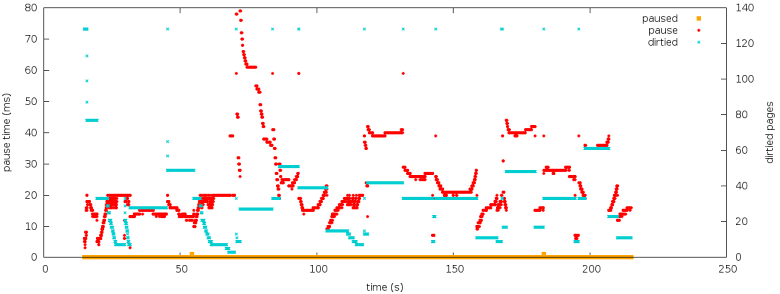
bumpy NFS: bursty IO completions



unmodified NFS, 1 dd, 3G mem, 2.6.39-rc3-wu-dt7+

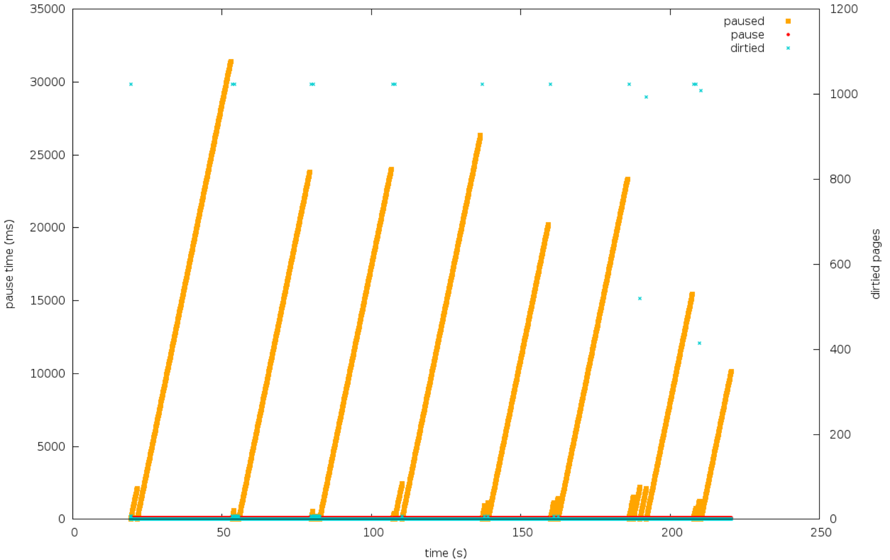
bumpy NFS: 80ms pause





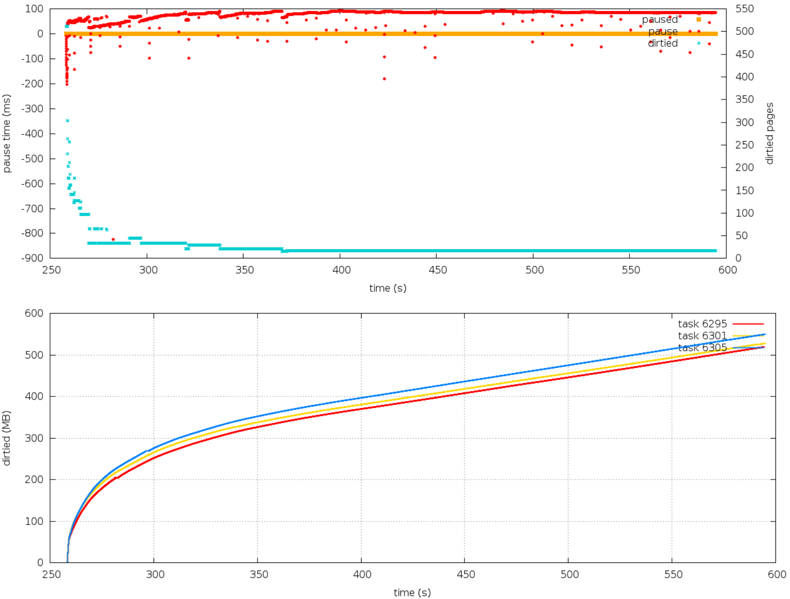
unmodified NFS, 1 dd, 3G mem, 2.6.39-rc3-wu-dt7+

bumpy NFS: 30s pause (legacy and Jan's kernel)



unmodified NFS, 1 dd, 3G mem, 2.6.38-rc7

smooth writeback on JBOD

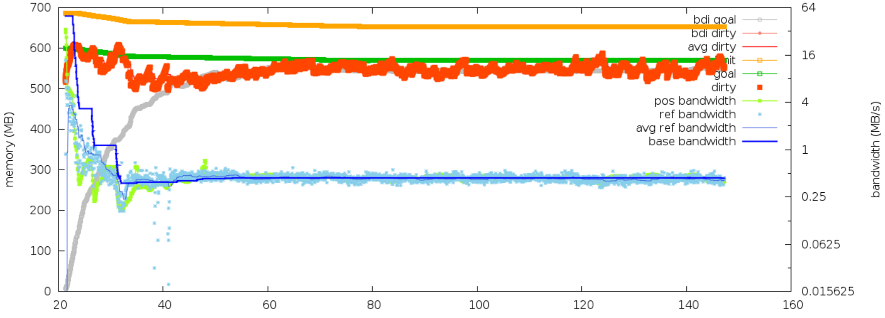


ext4, 10 SSD JBOD, 100 dd on each disk, 64G mem, 2.6.39-rc2-wu-dt7+

fluctuations: 4G ram

No free lunch. The smooth rates are traded by allowing more fluctuations in the number of dirty pages.

Amazingly, increased number of dd tasks hardly leads to increased fluctuations!

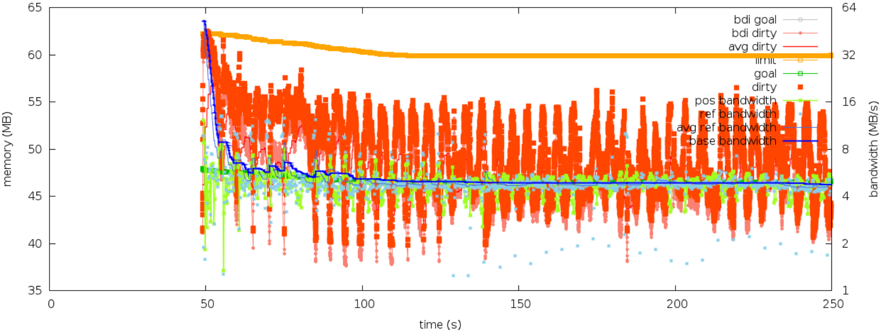


xfs, 100 dd, 4G mem, 2.6.39-rc2-wu-dt7+

fluctuations: 1/10 dirty mem

The fluctuation range is typically within 1s worth of disk writes.

The less memory (or dirty ratio), the more *relative* fluctuations.

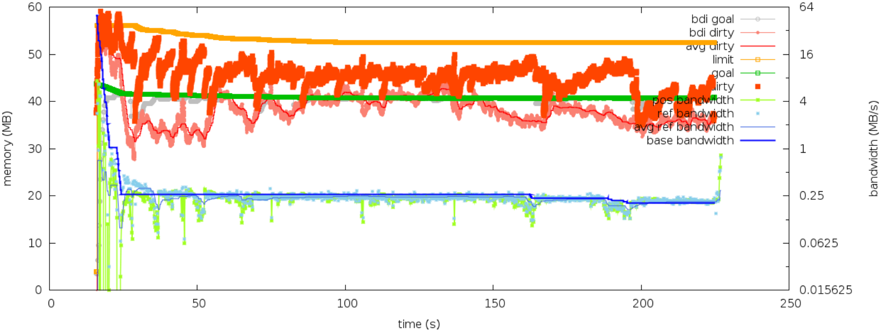


ext4, 10 dd, 3G mem, 2% dirty ratio, 2.6.39-rc3-wu-dt7+

over-dirtying problem (NFS)

```
- up to 20MB gaps between global/bdi dirty pages
- however, (bdi goal == global goal)!
```

NFS occupies quite some dirty pages without lowering the local disk's bdi dirty goal. This pushes global dirty pages high.

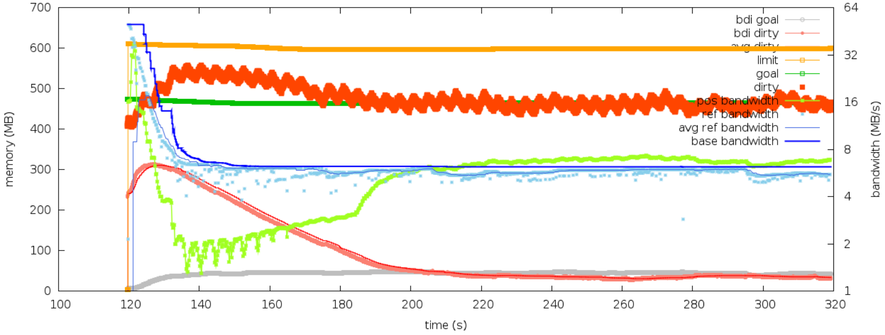


xfs, 100 dd, 3G mem, 2% dirty ratio, 2.6.39-rc2-wu-dt7+

over-dirtying problem (UKEY)

UKEY accumulated much more dirty pages than its bdi goal before it got throttled on (nr_dirty > (dirty_thresh + background_thresh) / 2).

This pushes global dirty pages high before the UKEY's dirty pages drop to normal after 200s.



xfs, 1 dd to UKEY + 1 dd to HDD, 3G mem, 2.6.39-rc2-wu-dt7+

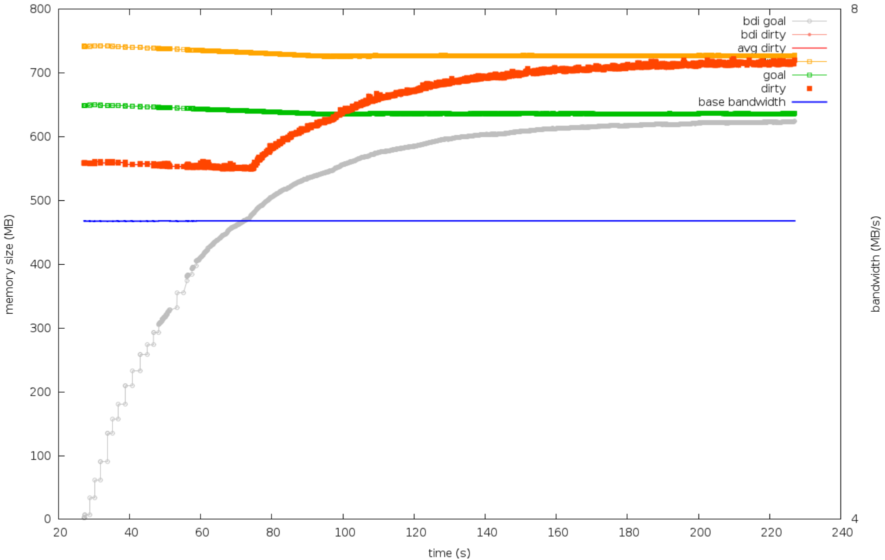
over-dirtying problem (solution)

Setup a brake area under the hard dirty limit. It's a leeway that can guarantee to balance the pressures created by *'dirty pages excessively exceeding bdi goal'*, in the cases of

- **dirty pages accumulated at free-run time on slow devices**
- **sudden storage break down / slow down**
- **sudden workload surges**

Note: it's not a new problem. Legacy kernels will be globally hard throttled and suffer more . . .

rampup time: 200s (vanilla and Jan's kernel)

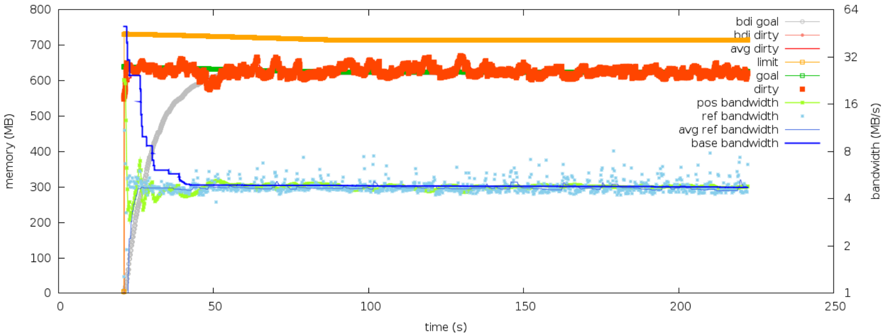


xfs, 8 dd, 4G mem, 2.6.38-rc7

rampup time: 2s

The dirty pages took 200s to rampup, waiting for the bdi dirty proportion to build up. That's too much slow, so it's made faster by 4 times.

Moreover, `bdi goal`'s rampup time is less relevant now. `base bandwidth` will be updated only when `dirty pages` are departing from the `goal`. This has the nice side effect of keeping `dirty pages` to the `goal`.



xfs, 10 dd, 4G mem, 2.6.39-rc2-wu-dt7+

Integrated cgroup IO controller

Demonstrated by 60 lines of code! Basic ideas:

1. maintain per-cgroup base bandwidth (can reuse code)
2. do simple cgroup position control to fix leaks (optional enhancement)

Merits

- **per-bdi nature, works on NFS and Software RAID**
- **no delayed response (working at the right layer)**
- **no page tracking, hence decoupled from memcg**
- **no interactions with FS and CFQ**
- **get proportional IO controller for free**
- **reuse/inherit all the base facilities/functions**

Thank you!