IO-less Dirty Throttling

Fengguang Wu
<fengguang.wu@intel.com>

LINUXCON JAPAN 2012
Write and writeback

USER APPLICATION

dirty pages and
immediately return

KERNEL FLUSHER THREAD

writeout pages after
dirty expired (>30s)

avoid blocking apps
aggregate write IO
The flusher thread(s)

- Initiate writeback IO in background
- One flusher per storage device

$ ps ax

<table>
<thead>
<tr>
<th>PID</th>
<th>TTY</th>
<th>STAT</th>
<th>TIME</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>2322</td>
<td>?</td>
<td>S</td>
<td>0:01</td>
<td>[flush-8:0]</td>
</tr>
<tr>
<td>12681</td>
<td>?</td>
<td>S</td>
<td>0:00</td>
<td>[flush-btrfs-1]</td>
</tr>
</tbody>
</table>
Dirty throttling

To limit dirty pages.. by slowing down heavy dirtiers.
IO-full balance_dirty_pages()

sys_write()
    balance_dirty_pages()

    if (over_bground_thresh())
        start_background_writeback();

    if (dirty exceeded)
        writeback_inodes(pages_dirtied * 3/2);
Problem: disk seeks + lock contentions

flusher       task 1       task 2       task 3

************************ concurrently working on ************************

+---------+ +---------+ +---------+ +---------+
|         | |         | |         | |         |
|         | |         | |         | |         |
|         | |         | |         | |         |
| inode 1 | | inode 2 | | inode 3 | | inode 4 |
|         | |         | |         | |         |
|         | |         | |         | |         |
+---------+ +---------+ +---------+ +---------+

<------------------ disk seeks & small IO ----------------->

<-------------- lock contentions & cache bouncing ------------->
Problem: large latencies

- tasks in deep IO path are *not killable*
- hurts *responsiveness*
- hurts *throughput* (pipeline stalls)

```plaintext
loop {
    READ from net
    WRITE to disk  # long sleeps => idle network
}
```

rsync to a loaded server:

- pause time: 0-3s  0-300ms  60-70ms
- throughput: 1   +12.4%  +63.5%
30s pause (NFS, 1-dd)

NFS, 1 dd, 3G mem, 2.6.38-rc7
3s pause (xfs, 8-dd)

pausible pause
x dinted

pause time (ms)

xfs, 8 dd, 4G mem, 2.6.38-rc7
Let’s do controllable sleeps!

I/O ⇒ SLEEP

```c
if (dirty exceeded)
-
   writeback_inodes(pages_dirtied * 3/2);
+
   sleep(pages_dirtied / task_ratelimit);
```

and rethink dirty balancing ……
Balancing dirty rate

- apps’ dirty progress
- disk write progress
IO-less dirty throttling/balancing

Do proper `sleep()`s during `sys_write()`:

1. Throttle each task’s *dirty rate* around `write_bandwidth / N`.
2. Balance the global *dirty pages* around global dirty target.
3. Balance each disk’s *dirty pages* around the disk’s dirty target.
per-disk write bandwidth shares

Task A
50MB/s
100MB/s
Fast disk

Task B
50MB/s

Task C
10MB/s
10MB/s
Slow ukey

bdi 8:0
N = 2

bdi 8:48
N = 1
The 3 axes of sleeping

In `sys_write()`, when dirtied enough pages:

```
  \                  \                     
  +----> LATENCY CONTROL    \                     
  /                      /  \                     
pages_dirtied

sleep(-----------------------------------------------------)

  pos_ratio    *    balanced_ratelimit
  \            |           |
  v            |           |
= 1.0 if at setpoint       v
> 1.0 if below setpoint    re-estimated
< 1.0 if above setpoint    on every 200ms

POSITION CONTROL    RATE CONTROL```
Controllable `sys_write()` latency

```c
if (enough pages dirtied)
    sleep(pages_dirtied / TASK_RATELIMIT);
```

**Controlling** `pages_dirtied` **controls** pause time.
Reasonable `sys_write()` latencies

100MB/s => insert 10-seconds-long sleeps into 1GB writes

```
10 * sleep(1000ms)      # too long latency
100 * sleep(100ms)
1000 * sleep(10ms)
10000 * sleep(1ms)      # too much CPU overheads
```
Pause time policies

- 10ms for 1-dd
- more dirtier tasks $\Rightarrow$ longer pause time
- 200ms max pause
Regressions on tiny pages_dirtied

**WORKLOAD**  64kb random writes and reads+writes

**PROBLEM**  paused on every few pages dirtied

**SOLUTION**  increase pause time to increase pages_dirtied
This task: pauses 45-50ms on every 57 pages dirtied

- 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
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
Balanced ratelimit

\[
\text{sleep}(\text{pages_dirtied} / \text{balanced_ratelimit});
\]

That throttles the task’s dirty rate to

\[
\text{balanced_ratelimit} = \text{write_bandwidth} / N,
\]

where

\[
N := \text{NUMBER OF DIRTIER TASKS}
\]

is to be estimated.
Estimating N (theory)

When started N dd, throttle each dd at

\[ \text{ratelimit}_0 \] (any non-zero initial value is OK)

After 200ms, we measures

\[ \text{dirty\_rate} = \text{# of pages dirtied by apps} / 200\text{ms} \]

to estimate N:

\[ N = \text{dirty\_rate} / \text{ratelimit}_0 \]

Now the tasks can be rightfully throttled at

\[ \text{balanced\_ratelimit} = \text{write\_bandwidth} / N \]
balanced_ratelimit estimated in 200ms!

However, real world is not perfect ...
Unstable balanced_ratelimit

Ext4, 1 dd to HDD + 1 dd to USB stick, 3G mem, 3.2.0-rc3
COMPRESS SMALL NOISES

`balanced_ratelimit` is fluctuating, has estimation errors due to control lags and `write_bw` errors, asking for step-by-step approximations:

\[ \text{dirty_ratelimit} \Rightarrow \text{balanced_ratelimit} \text{ (the closer, the smaller step)} \]

FILTER OUT LARGE NOISES

There is no need to update `dirty_ratelimit` during a stable workload, which only makes it susceptible to (big) noises. So do it defensively and update `dirty_ratelimit` when `balanced_ratelimit` and `task_ratelimit` are at the same direction of `dirty_ratelimit`.

\[
\begin{align*}
\text{if } (\text{dirty_ratelimit} > \text{both balanced_ratelimit, task_ratelimit}) \\
\text{dirty_ratelimit} & \Rightarrow \text{max(balanced_ratelimit, task_ratelimit)} \\
\text{if } (\text{dirty_ratelimit} < \text{both balanced_ratelimit, task_ratelimit}) \\
\text{dirty_ratelimit} & \Rightarrow \text{min(balanced_ratelimit, task_ratelimit)}
\end{align*}
\]
Balanced dirty pages

- \texttt{task\_ratelimit = balanced\_ratelimit;}
+ \texttt{task\_ratelimit = balanced\_ratelimit * pos\_ratio;}

Negative feedback control

pos\_ratio = 1.0

if (dirty < setpoint) scale up pos\_ratio
if (dirty > setpoint) scale down pos\_ratio

if (bdi\_dirty < bdi\_setpoint) scale up pos\_ratio
if (bdi\_dirty > bdi\_setpoint) scale down pos\_ratio
Paradigm change

(1) Paradigm change diagram showing free run and throttled regions.

(2) Additional diagram showing task rate limit and balanced rate limit with free run and smoothly throttled regions.
Global control line

pos_ratio

| <= global dirty control scope ======> |

freerun^ setpoint^ limit^ dirty pages
span = (8*write_bw) in 1-disk case and transits to (bdi_thresh) in JBOD case.
Over-dirtying example

UKEY accumulated much more initial dirty pages than its bdi goal.

Bring dirty pages down by throttling the task at 1/4 write bandwidth.
Reasonable feedback in all mem/bw combinations

Policies are:

1. global control line: adapts to dirtyable memory size
2. bdi control line: adapts to write bandwidth (except in JBOD)

Designed for:

1. small dirty_setpoint, large write_bw: global line will take control
2. large dirty_setpoint, small write_bw: bdi line will take control
3. JBOD case: bdi_dirty_setpoint fluctuates proportional to dirtyable memory size; do weak bdi position control
1-disk case: fluctuations $\leq$ write bandwidth

ext4, 1 dd, 1GB dirty limit, 2.6.32-rc3
1-disk case: fluctuations $\leq$ write bandwidth

ext4, 100 dd, 1GB dirty limit, 2.6.32-rc3
1-disk case: fluctuations $\leq$ write bandwidth

xfs, 1 dd, 1GB dirty limit, 2.6.32-rc3
1-disk case: fluctuations ≤ write bandwidth
JBOD case: large memory

ext4, 4-HDD JBOD, 1 dd per disk, 8GB dirty limit, 2.6.32-rc3-pause6+
JBOD case: large memory

ext4, 4-HDD JBOD, 10 dd per disk, 8GB dirty limit, 2.6.32-rc3-pause6+
JBOD case: small memory

ext4, 4-HDD JBOD, 1 dd per disk, 100MB dirty limit, 2.6.32-rc3-pause6+
JBOD case: small memory

ext4, 4-HDD JBOD, 10 dd per disk, 100MB dirty limit, 2.6.32-rc3-pause6+
on write() syscall

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

on every 200ms

bdi_update_dirty_ratelimit()
{
    N = dirty_rate / task_ratelimit;
    balanced_ratelimit = write_bw / N;

    update bdi->dirty_ratelimit towards balanced_ratelimit,
    regulated by task_ratelimit
}
What’s next

1. **per-memcg dirty limits**
2. **write_bps throttling IO controller**
3. **proportional weight IO controller** (*challenging*)