Subject: Re: Re: [PATCH][RFC] dirty balancing for cgroups
Posted by KAMEZAWA Hiroyuki on Mon, 14 Jul 2008 14:38:24 GMT
View Forum Message <> Reply to Message

----- Original Message -----
>> See comments in fs/page-writeback.c:: determin_dirtyable_memory()
>> ==
>> /*
>> * Work out the current dirty-memory clamping and background writeout
>> * thresholds.
>> *
>> * The main aim here is to lower them aggressively if there is a lot of map
ped
>> * memory around. To avoid stressing page reclaim with lots of unreclaimab
le
>> * pages. It is better to clamp down on writers than to start swapping, an
d
>> * performing lots of scanning.
>> *
>> * We only allow 1/2 of the currently-unmapped memory to be dirtied.
>> *
>> * We don't permit the clamping level to fall below 5% - that is getting ra
ther
>> * excessive.
>> *
>> * We make sure that the background writeout level is below the adjusted
>> * clamping level.
>> ==
>>
>> "To avoid stressing page reclaim with lots of unreclaimable pages"
>>
>> Then, I think memcg should support this for helping relcaim under memcg.
>
>That comment is unclear at best.
>
>The dirty page limit avoids deadlocks under certain situations, the per
>BDI dirty limit avoids even mode deadlocks by providing isolation
>between BDIs.
>
>
>The fundamental deadlock solved by the dirty page limit is the typical
>reclaim deadlock - needing memory to free memory. It does this by
>ensuring only some part of the total memory used for the page-cache can
>be dirty, thus we always have clean pages around that can be reclaimed
>so we can launder the dirty pages.
>
>This on its own generates a new deadlock for stacked devices, imagine
>device A on top of B. When A generates loads of dirty pages it will

>eventually hit the dirty limit and we'd start to launder them. However
>in order to launder A's dirty pages we'd need to dirty pages for B, but
>we can't since we're at the global limit.
>
>This problem is solved by introducing a per BDI dirty limit, by
>assigning each BDI an individual dirty limit (whoes sum is the total
>dirty limit) we avoid that deadlock. Take the previous example; A would
>start laundering its pages when it hits its own limit, B's operation
>isn't hampered by that.
>
>[even when B's limit is 0 we're able to make progress, since we'll only
> wait for B's dirty page count to decrease - effectively reducing to
> sync writes.]
>
>Of course this raises the question how to assign the various dirty
>limits - any fixed distribution is hard to maintain and suboptimial for
>most workloads.
>
>We solve this by assigning each BDI a fraction proportional to its
>current launder speed. That is to say, if A launders pages twice as fast
>as B does, then A will get 2/3-rd of the total dirty page limit, versus
>1/3-rd for B.
>
>
>Then there is the task dirty stuff - this is basically a 'fix' for the
>problem where a slow writer gets starved by a fast reader. Imagine two
>tasks competing for bandwidth, 1 the fast writer and 2 the slow writer.
>
>1 will dirty loads of pages but all things being equal 2 will have to
>wait for 1's dirty pages.
>
>So what we do is lower the dirty limit for fast writers - so these get
>to wait sooner and slow writers have a little room to make progress
>before they too have to wait.
>
>To properly solve this we'd need to track p_{bdi,task}. However that's
>intracktable. Therefore we approximate that with p_bdi * p_task. This
>approximation looses detail.
>
>Imagine two tasks: 1 and 2, and two BDIs A and B (independent this
>time). If 1 is a (fast) writer to A and 2 is a (slow) writer to B, we
>need not throttle 2 sooner as there is no actual competition.
>
>The full proportional tensor p_{bdi,task} can express this, but the
>simple approximation p_bdi * p_task can not.
>
>The approximation will reduce 1's bandwidth a little even though there
>is no actual competition.

>
>
>Now the problem this patch tries to address...
>
>As you can see you'd need p_{bdi,cgroup,task} for it to work, and the
>obvious approximation p_bdi * p_cgroup * p_task will get even more
>coarse.
>
>You could possibly attempt to do p_{bdi,cgroup} * p_task since the bdi
>and cgroup set are pretty static, but still that would be painful.
>
>So, could you please give some more justification for this work, I'm not
>seeing the value in complicating all this just yet.
>
>
>Thanks for reading this far,
>
Thank you for explanation. Maybe I misunderstood something.

What I thought was to keep some amount of clean pages under memcg
for smooth reclaiming. But after reading your explanation, I'm now
considering again...

about DEADLOCK:

memory cgroup's reclaim path is working as following now.
==
mem_cgroup_charge() or others.
  -> try_to_free_mem_cgroup_pages()
      -> shrink_zone()...
          -> subsystem may call alloc_pages()
              -> ....(*)

Unlike global LRU, memory_cgroup does not affect (*) because add_tp_page_cache
() is called here and maybe no deadlock.

about Memory Reclaim's health:

It doesn't seem good to allow users to make all pages under memcg
to be dirty without penalty(throttling).

Dirty/Written-backed pages are not to be reclaimed by first lru scan because
it has to be written out (and rolated to the tail of LRU.) and clean pages,
which are easy to be dropped, are dropped soon.

Under following situation
 - large file copy under memcg.
 - big coredump under memcg.

I think too much (clean) pages are swapped out by memcg's LRU because
of tons of dirty pages. (but I don't have enough data now. just my concern.)

What can I do against this ? Could you give me a hint ?

Thanks,
-Kame

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers