Subject: Re: [RFC] memory controller : backgorund reclaim and avoid excessive locking [3/5] throttling
Posted by KAMEZAWA Hiroyuki on Thu, 14 Feb 2008 09:38:56 GMT
View Forum Message <> Reply to Message

On Thu, 14 Feb 2008 14:44:50 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> > By this interface, a user can control the # of threads which can enter
> > try_to_free...
> >
>
> What happens to the other threads, do they sleep?
>
yes.

> > Default is 10240 ...a enough big number for unlimited. Maybe this should
> > be changed.
> >
> >
> > Changes from previous one.
> >  - Added an interface to control the limit.
> >  - don't call wake_up at uncharge()...it seems hevay..
> >    Instead of that, sleepers use schedule_timeout(HZ/100). (10ms)
> >
> > Considerations:
> >  - Should we add this 'throttle' to global_lru at first ?
>
> Hmm.. interesting question. I think Rik is looking into some of these issues
>
Maybe. I thought someone posted a patch to -mm but I lost the mail.

> > + * throttling params for reclaim.
> > + */
> > + struct {
> > +  int  limit;
>
> We might want to use something else instead of limit. How about
> max_parallel_reclaimers?
>
ok.

> > +  prepare_to_wait(&mem->throttle.waitq, &wait,
> > +      TASK_INTERRUPTIBLE);
> > +  if (res_counter_check_under_limit(&mem->res)) {
> > +   finish_wait(&mem->throttle.waitq, &wait);
> > +   break;

> > +  }
> > +  /* expect some progress in... */
> > +  schedule_timeout(HZ/50);
>
> Can't we wait on someone to wake us up? Why do we need to do a schedule_timeout?

A method "Someone wakes up" adds new code to uncharge.

 - check value and limt.
 - check waiters
 - wakes them up if any (and touches other cpu's runq->lock.)

Not very light-weight ops. just polling works well in this case, I think.

> And why HZ/50 and not HZ/10? Too many timers distract the system from power
> management :)
ok, HZ/10. (or add some knob.)

> > -  if (try_to_free_mem_cgroup_pages(mem, gfp_mask))
> > +  if (((gfp_mask & (__GFP_FS|__GFP_IO)) != (__GFP_FS|__GFP_IO))
> > +      || mem_cgroup_throttle_reclaim(mem)) {
>
> I think we should split this check and add detailed comments. If we cannot
> sleep, then we cannot be throttled, right?
>
ok.

> > + init_waitqueue_head(&mem->throttle.waitq);
> > + mem->throttle.limit = 10240; /* maybe enough big for no throttle */
>
> Why 10240? So, 10,000 threads can run in parallel, isn't that an overkill?

yes, it's overkill. Because I don't want to define *unlimted* value,
I used a big number.

> How about setting it to number of cpus/2 or something like that?

Hmm, in previous version, cpus/4 was used as default.
some value like cpus/2, nodes/2, seems good. (maybe).

A concern is that memory cgroup can be used with cpuset or
some other controllers. So, the best number can be various under environments.

It's one of choice we set the value as unlimited and makes user set suitable by hand.

Thanks,

-Kame

_____

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