
Subject: Re: [RFC] memory controller : backgorund reclaim and avoid excessive locking [2/5] background reclai

Posted by [KAMEZAWA Hiroyuki](#) on Thu, 14 Feb 2008 08:57:03 GMT

[View Forum Message](#) <> [Reply to Message](#)

On Thu, 14 Feb 2008 14:02:44 +0530

Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> KAMEZAWA Hiroyuki wrote:

> > A patch for background reclaim based on high-low watermak in res_counter.

> > The daemon is called as "memcontd", here.

> >

> > Implements following:

> > * If res->usage is higher than res->hwmark, start memcontd.

> > * memcontd calls try_to_free_pages.

> > * memcontd stops if res->usage is lower than res->lwmark.

> >

>

> Can we call the dameon memcgroupd?

>

ok.

> >

> > + if (res_counter_above_hwmark(&mem->res))

> > + mem_cgroup_schedule_daemon(mem);

> > +

>

> I don't understand this part, some comments might be good here. Why do we wake

> up the dameon in mem_cgroup_isolate_pages? Why not do so under memory pressure

> like the soft limit patches does? I suspect that these patches and soft limit

> patches might have some overlap.

>

Ah, diff option is bad. this is in mem_cgroup_charge_common().

Sorry for confusion.

Final image of this series is.

==

```
while (mem_cgroup_borrow_and_charge(mem, PAGE_SIZE)) {
    int ret;
    if (!(gfp_mask & __GFP_WAIT))
        goto out;
```

```
    if (((gfp_mask & (__GFP_FS|__GFP_IO)) != (__GFP_FS|__GFP_IO))
        || mem_cgroup_throttle_reclaim(mem)) {
        ret = try_to_free_mem_cgroup_pages(mem, gfp_mask);
        atomic_dec(&mem->throttle.reclaimers);
        if (waitqueue_active(&mem->throttle.waitq))
```

```

        wake_up_all(&mem->throttle.waitq);
    if (ret)
        continue;
} else {
    mem_cgroup_wait_reclaim(mem);
    continue;
}

/*
 * try_to_free_mem_cgroup_pages() might not give us a full
 * picture of reclaim. Some pages are reclaimed and might be
 * moved to swap cache or just unmapped from the cgroup.
 * Check the limit again to see if the reclaim reduced the
 * current usage of the cgroup before giving up
 */
if (res_counter_check_under_limit(&mem->res))
    continue;

if (nr_retries < MEM_CGROUP_RECLAIM_RETRIES)
    drain_all_borrow(mem);

if (!nr_retries--) {
    mem_cgroup_out_of_memory(mem, gfp_mask);
    goto out;
}
congestion_wait(WRITE, HZ/10);
}

if (res_counter_above_hwmark(&mem->res))
    mem_cgroup_schedule_daemon(mem);

```

==

```

>> + finish_wait(&mem->daemon.waitq, &wait);
>> + try_to_free_mem_cgroup_pages(mem, GFP_HIGHUSER_MOVABLE);
>> + /* Am I in hurry ? */
>> + if (!res_counter_above_hwmark(&mem->res)) {
>> + /*
>> +  * Extra relaxing..memory reclaim is hevay work.
>> +  * we don't know there is I/O congestion or not.
>> +  * So use just relax rather than congesiton_wait().
>> +  * HZ/10 is widely used value under /mm.
>> +  */
>> + schedule_timeout(HZ/10);
>
> Why not just yeild() here and let another iteration push us back to our low
> water mark?
>

```

just used timeout value which is widely used under /mm.
Hmm, I'll put cond_resched() or yield() here.
And see how cpu is used.

```
> >
> > +static DEFINE_MUTEX(modify_param_mutex);
> > static ssize_t mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
> >     struct file *file, const char __user *userbuf,
> >     size_t nbytes, loff_t *ppos)
> > {
> > - return res_counter_write(&mem_cgroup_from_cont(cont)->res,
> > -   cft->private, userbuf, nbytes, ppos,
> > + int ret;
> > + struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
> > +
> > + mutex_lock(&modify_param_mutex);
> > + /* Attach new background reclaim daemon.
> > +   This must be done before change values (for easy error handling */
> > +
> > + if (cft->private == RES_HWMARK &&
> > +     !mem->daemon.kthread) {
> > +     struct task_struct *thr;
> > +     thr = kthread_run(mem_cgroup_reclaim_daemon, mem, "memcontd");
> > +     if (IS_ERR(thr)) {
> > +         ret = PTR_ERR(thr);
> > +         goto out;
> > +     }
> > +     mem->daemon.kthread = thr;
> > + }
> > + ret = res_counter_write(&mem->res, cft->private, userbuf, nbytes, ppos,
> >     mem_cgroup_write_strategy);
> > +
> > + /* Even on error, don't stop reclaim daemon here. not so problematic. */
> > +
> > +out:
> > + mutex_unlock(&modify_param_mutex);
> > + return ret;
> > }
> >
>
> Could we document the changes to this function. It looks like writing a value to
> the watermarks can cause the dameon to be scheduled.
ok.
(*) creating daemon at initilization is not simple because root cgroup initilization
    is very fast...
```

Thank you.

-Kame

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>
