Subject: Re: [RFC] memory controller : backgorund reclaim and avoid excessive locking [3/5] throttling
Posted by Balbir Singh on Thu, 14 Feb 2008 09:14:50 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> Throttle memory reclaim interface for cgroup.
>
> This patch adds..
> - a limit for simultaneous callers of try_to_free_mem_cgroup_pages().
> - interface for that. memory.shrinkers.
>
> There are some reasons.
> - try_to_free... is very heavy and shoulnd't be called too much at once.
> - When the number of callers of try_to_free.. is big, we'll reclaim
>   too much memory.
>
> 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?

> 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

>   - Do we need more knobs ?
>   - Should default value to be automtically estimated value ?
>   (I used '# of cpus/4' as default in previous version.)
>
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
>
> mm/memcontrol.c | 235 ++++++++++++++++++++++++++++++-------------------
> 1 files changed, 127 insertions(+), 108 deletions(-)
>

> Index: linux-2.6.24-mm1/mm/memcontrol.c
> ===
> --- linux-2.6.24-mm1.orig/mm/memcontrol.c
> +++ linux-2.6.24-mm1/mm/memcontrol.c
> @@ -145,8 +145,17 @@ struct mem_cgroup {
> wait_queue_head_t waitq;
> struct task_struct *kthread;
> } daemon;
> + /*
> + * throttling params for reclaim.
> + */
> + struct {
> + int limit;

We might want to use something else instead of limit. How about
max_parallel_reclaimers?

> + atomic_t reclaimers;
> + wait_queue_head_t waitq;
> + } throttle;
> };
>
> +
> /*
> * We use the lower bit of the page->page_cgroup pointer as a bit spin
> * lock. We need to ensure that page->page_cgroup is atleast two
> @@ -520,6 +529,27 @@ static inline void mem_cgroup_schedule_d
> wake_up_interruptible(&mem->daemon.waitq);
> }
>
> +static inline void mem_cgroup_wait_reclaim(struct mem_cgroup *mem)
> +{
> + DEFINE_WAIT(wait);
> + while (1) {
> + 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?
And why HZ/50 and not HZ/10? Too many timers distract the system from power
management :)

> + finish_wait(&mem->throttle.waitq, &wait);

> + }
> +}
> +
> +static inline int mem_cgroup_throttle_reclaim(struct mem_cgroup *mem)
> +{
> + return atomic_add_unless(&mem->throttle.reclaimers, 1,
> +   mem->throttle.limit);
> +}
>
> unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
>   struct list_head *dst,
> @@ -652,11 +682,22 @@ retry:
>   * the cgroup limit.
>   */
>   while (res_counter_charge(&mem->res, PAGE_SIZE)) {
> + int ret;
>   if (!(gfp_mask & __GFP_WAIT))
>   goto out;
>
> - 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?

> +   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
> @@ -1054,6 +1095,19 @@ static ssize_t mem_force_empty_read(stru
>   return -EINVAL;
> }
>
> +static int mem_throttle_write(struct cgroup *cont, struct cftype *cft, u64 val)
> +{
> + struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
> + int limit = (int)val;
> + mem->throttle.limit = limit;

```
> + return 0;
> +}
> +
> +static u64 mem_throttle_read(struct cgroup *cont, struct cftype *cft)
> +{
> + struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
> + return (u64)mem->throttle.limit;
> +}
>
>  static const struct mem_cgroup_stat_desc {
>   const char *msg;
> @@ -1146,6 +1200,11 @@ static struct cftype mem_cgroup_files[]
>    .read = mem_force_empty_read,
>   },
>   {
> +  .name = "shrinks",
> +  .write_uint = mem_throttle_write,
> +  .read_uint  = mem_throttle_read,
> + },
> + {
>    .name = "stat",
>    .open = mem_control_stat_open,
>   },
> @@ -1215,6 +1274,11 @@ mem_cgroup_create(struct cgroup_subsys *
>     goto free_out;
>   init_waitqueue_head(&mem->daemon.waitq);
>   mem->daemon.kthread = NULL;
> +
> + 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? How about setting it to number of cpus/2 or something like that?

```
> + atomic_set(&mem->throttle.reclaimers, 0);
> +
>   return &mem->css;
>  free_out:
>   for_each_node_state(node, N_POSSIBLE)
>
```

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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