
Subject: [PATCH v5 02/14] memcg: Reclaim when more than one page needed.
Posted by [Glauber Costa](#) on Tue, 16 Oct 2012 10:16:39 GMT
[View Forum Message](#) <> [Reply to Message](#)

From: Suleiman Souhlal <ssouhlal@FreeBSD.org>

mem_cgroup_do_charge() was written before kmem accounting, and expects three cases: being called for 1 page, being called for a stock of 32 pages, or being called for a hugepage. If we call for 2 or 3 pages (and both the stack and several slabs used in process creation are such, at least with the debug options I had), it assumed it's being called for stock and just retried without reclaiming.

Fix that by passing down a minsize argument in addition to the csize.

And what to do about that (csize == PAGE_SIZE && ret) retry? If it's needed at all (and presumably is since it's there, perhaps to handle races), then it should be extended to more than PAGE_SIZE, yet how far? And should there be a retry count limit, of what? For now retry up to COSTLY_ORDER (as page_alloc.c does) and make sure not to do it if __GFP_NORETRY.

[v4: fixed nr pages calculation pointed out by Christoph Lameter]

Signed-off-by: Suleiman Souhlal <suleiman@google.com>

Signed-off-by: Glauber Costa <glommer@parallels.com>

Acked-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

Acked-by: Michal Hocko <mhocko@suse.cz>

Acked-by: Johannes Weiner <hannes@cmpxchg.org>

CC: Tejun Heo <tj@kernel.org>

mm/memcontrol.c | 16 ++++++-----
1 file changed, 9 insertions(+), 7 deletions(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c

index 47cb019..7a9652a 100644

--- a/mm/memcontrol.c

+++ b/mm/memcontrol.c

@@ -2226,7 +2226,8 @@ enum {
};

```
static int mem_cgroup_do_charge(struct mem_cgroup *memcg, gfp_t gfp_mask,  
- unsigned int nr_pages, bool oom_check)  
+ unsigned int nr_pages, unsigned int min_pages,  
+ bool oom_check)  
{  
    unsigned long csize = nr_pages * PAGE_SIZE;  
    struct mem_cgroup *mem_over_limit;
```

```

@@ -2249,18 +2250,18 @@ static int mem_cgroup_do_charge(struct mem_cgroup *memcg,
gfp_t gfp_mask,
} else
    mem_over_limit = mem_cgroup_from_res_counter(fail_res, res);
/*
- * nr_pages can be either a huge page (HPAGE_PMD_NR), a batch
- * of regular pages (CHARGE_BATCH), or a single regular page (1).
- *
- * Never reclaim on behalf of optional batching, retry with a
- * single page instead.
- */
- if (nr_pages == CHARGE_BATCH)
+ if (nr_pages > min_pages)
    return CHARGE_RETRY;

    if (!(gfp_mask & __GFP_WAIT))
        return CHARGE_WOULDBLOCK;

+ if (gfp_mask & __GFP_NORETRY)
+ return CHARGE_NOMEM;
+
    ret = mem_cgroup_reclaim(mem_over_limit, gfp_mask, flags);
    if (mem_cgroup_margin(mem_over_limit) >= nr_pages)
        return CHARGE_RETRY;
@@ -2273,7 +2274,7 @@ static int mem_cgroup_do_charge(struct mem_cgroup *memcg, gfp_t
gfp_mask,
    * unlikely to succeed so close to the limit, and we fall back
    * to regular pages anyway in case of failure.
    */
- if (nr_pages == 1 && ret)
+ if (nr_pages <= (1 << PAGE_ALLOC_COSTLY_ORDER) && ret)
    return CHARGE_RETRY;

/*
@@ -2408,7 +2409,8 @@ again:
    nr_oom_retries = MEM_CGROUP_RECLAIM_RETRIES;
}

- ret = mem_cgroup_do_charge(memcg, gfp_mask, batch, oom_check);
+ ret = mem_cgroup_do_charge(memcg, gfp_mask, batch, nr_pages,
+ oom_check);
    switch (ret) {
    case CHARGE_OK:
        break;
--
1.7.11.7

```
