
Subject: Re: [PATCH -mm 5/5] swapcgroup (v3): implement force_empty
Posted by [KAMEZAWA Hiroyuki](#) on Fri, 04 Jul 2008 10:15:11 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Fri, 4 Jul 2008 15:24:23 +0900

Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp> wrote:

> This patch implements force_empty of swapcgroup.
>
> Currently, it simply uncharges all the charges from the group.
>
> I think there can be other implementations.
>
> What I thought are:
> - move all the charges to its parent.
> - unuse(swap in) all the swap charged to the group.
>
3. move all swap back to memory (see swapoff.)

> But in any case, I think before implementing this way,
> hierarchy and move charges support are needed.
>
> So I think this is enough for the first step.
>

I don't think hierarchy/automatic-load-balancer for swap cg is necessary.
Hmm...but handling limit_change (at least, returns -EBUSY) will be necessary.
Do you consider a some magical way to move pages in swap back to memory ?

In general, I like this set but we can't change the limit on demand. (maybe)
(just putting it to TO-DO-List is okay to me.)

Thanks,
-Kame

>
> Change log
> v2->v3
> - new patch
>
>
> Signed-off-by: Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp>
>
> ---
> include/linux/memcontrol.h | 5 +++++

```

> mm/memcontrol.c      | 47 ++++++-----
> mm/swapfile.c        | 52 ++++++-----
> 3 files changed, 91 insertions(+), 13 deletions(-)
>
> diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
> index 1e6eb0c..8c4bad0 100644
> --- a/include/linux/memcontrol.h
> +++ b/include/linux/memcontrol.h
> @@ -177,6 +177,7 @@ extern void swap_cgroup_uncharge(struct swap_info_struct *si,
>    unsigned long offset);
> extern struct mem_cgroup *page_to_memcg(struct page *page);
> extern int swap_cgroup_vm_swap_full(struct mem_cgroup *memcg);
> +extern void __swap_cgroup_force_empty(struct mem_cgroup *mem);
> #else
> static inline
> struct mem_cgroup **swap_info_clear_memcg(struct swap_info_struct *p)
> @@ -211,6 +212,10 @@ static inline int swap_cgroup_vm_swap_full(struct mem_cgroup
> *memcg)
> {
>     return 0;
> }
> +
> +static inline void __swap_cgroup_force_empty(struct mem_cgroup *mem)
> +{
> +}
> #endif
>
> #endif /* _LINUX_MEMCONTROL_H */
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 160fca1..f93907c 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -826,6 +826,34 @@ static void mem_cgroup_force_empty_list(struct mem_cgroup *mem,
>     spin_unlock_irqrestore(&mz->lru_lock, flags);
> }
>
> +static inline int mem_cgroup_in_use(struct mem_cgroup *mem)
> +{
> +    return mem->res.usage > 0;
> +}
> +
> +static inline int swap_cgroup_in_use(struct mem_cgroup *mem)
> +{
> +#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
> +    return mem->swap_res.usage > 0;
> +#else
> +    return 0;
> +#endif

```

```

> +}
> +
> +static void __mem_cgroup_force_empty(struct mem_cgroup *mem)
> +{
> + int node, zid;
> +
> + for_each_node_state(node, N_POSSIBLE)
> + for (zid = 0; zid < MAX_NR_ZONES; zid++) {
> + struct mem_cgroup_per_zone *mz;
> + enum lru_list l;
> + mz = mem_cgroup_zoneinfo(mem, node, zid);
> + for_each_lru(l)
> + mem_cgroup_force_empty_list(mem, mz, l);
> + }
> +}
> +
> /*
> * make mem_cgroup's charge to be 0 if there is no task.
> * This enables deleting this mem_cgroup.
> @@ -833,7 +861,6 @@ static void mem_cgroup_force_empty_list(struct mem_cgroup *mem,
> static int mem_cgroup_force_empty(struct mem_cgroup *mem)
> {
> int ret = -EBUSY;
> - int node, zid;
>
> css_get(&mem->css);
> /*
> @@ -841,18 +868,17 @@ static int mem_cgroup_force_empty(struct mem_cgroup *mem)
> * active_list <-> inactive_list while we don't take a lock.
> * So, we have to do loop here until all lists are empty.
> */
> - while (mem->res.usage > 0) {
> + while (mem_cgroup_in_use(mem) || swap_cgroup_in_use(mem)) {
> if (atomic_read(&mem->css.cgroup->count) > 0)
> goto out;
> - for_each_node_state(node, N_POSSIBLE)
> - for (zid = 0; zid < MAX_NR_ZONES; zid++) {
> - struct mem_cgroup_per_zone *mz;
> - enum lru_list l;
> - mz = mem_cgroup_zoneinfo(mem, node, zid);
> - for_each_lru(l)
> - mem_cgroup_force_empty_list(mem, mz, l);
> - }
> +
> + if (mem_cgroup_in_use(mem))
> + __mem_cgroup_force_empty(mem);
> +
> + if (swap_cgroup_in_use(mem))

```

```

> + __swap_cgroup_force_empty(mem);
> }
> +
> ret = 0;
> out:
> css_put(&mem->css);
> @@ -1289,6 +1315,7 @@ void swap_cgroup_uncharge(struct swap_info_struct *si,
> * and called swap_entry_free().
> * 2. when this swap entry had been assigned by
> * get_swap_page_of_type() (via SWSUSP?).
> + * 3. force empty can make such entries.
> */
> if (mem) {
> res_counter_uncharge(&mem->swap_res, PAGE_SIZE);
> diff --git a/mm/swapfile.c b/mm/swapfile.c
> index 6a863bc..2263ae8 100644
> --- a/mm/swapfile.c
> +++ b/mm/swapfile.c
> @@ -704,15 +704,30 @@ static int unuse_mm(struct mm_struct *mm,
> }
>
> /*
> + * return the mem_cgroup to which a entry is charged
> + */
> +static inline struct mem_cgroup *entry_to_memcg(swp_entry_t entry)
> +{
> +#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
> + return swap_info[swp_type(entry)].memcg[swp_offset(entry)];
> +#else
> + return NULL;
> +#endif
> +}
> +
> +/*
> * Scan swap_map from current position to next entry still in use.
> + * If mem is specified, the entry should be charged to it.
> * Recycle to start on reaching the end, returning 0 when empty.
> */
> static unsigned int find_next_to_unuse(struct swap_info_struct *si,
> - unsigned int prev)
> + unsigned int prev,
> + struct mem_cgroup *mem)
> {
> unsigned int max = si->max;
> unsigned int i = prev;
> int count;
> + int type = si - swap_info;
>

```

```

> /*
>  * No need for swap_lock here: we're just looking
> @@ -735,7 +750,8 @@ static unsigned int find_next_to_unuse(struct swap_info_struct *si,
>   i = 1;
> }
> count = si->swap_map[i];
> - if (count && count != SWAP_MAP_BAD)
> + if (count && count != SWAP_MAP_BAD
> +   && (!mem || mem == entry_to_memcg(swp_entry(type, i))))
>   break;
> }
> return i;
> @@ -782,7 +798,7 @@ static int try_to_unuse(unsigned int type)
>  * one pass through swap_map is enough, but not necessarily:
>  * there are races when an instance of an entry might be missed.
>  */
> - while ((i = find_next_to_unuse(si, i)) != 0) {
> + while ((i = find_next_to_unuse(si, i, NULL)) != 0) {
>   if (signal_pending(current)) {
>     retval = -EINTR;
>     break;
> @@ -1923,5 +1939,35 @@ struct mem_cgroup *page_to_memcg(struct page *page)
> out:
> return mem;
> }
> +
> +/*
> + * uncharge all the entries that are charged to the group.
> + */
> +void __swap_cgroup_force_empty(struct mem_cgroup *mem)
> +{
> + struct swap_info_struct *p;
> + int type;
> +
> + spin_lock(&swap_lock);
> + for (type = swap_list.head; type >= 0; type = swap_info[type].next) {
> +   p = swap_info + type;
> +
> +   if ((p->flags & SWP_ACTIVE) == SWP_ACTIVE) {
> +     unsigned int i = 0;
> +
> +     spin_unlock(&swap_lock);
> +     while ((i = find_next_to_unuse(p, i, mem)) != 0) {
> +       spin_lock(&swap_lock);
> +       if (p->swap_map[i] && p->memcg[i] == mem)
> +         swap_cgroup_uncharge(p, i);
> +       spin_unlock(&swap_lock);
> +     }

```

```
> + spin_lock(&swap_lock);  
> + }  
> + }  
> + spin_unlock(&swap_lock);  
> +  
> + return;  
> +}  
> #endif  
>  
>
```

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