

---

Subject: Re: [PATCH 5/7] shrink support for memcg kmem controller  
Posted by [KAMEZAWA Hiroyuki](#) on Wed, 22 Feb 2012 01:42:56 GMT

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

---

On Tue, 21 Feb 2012 15:34:37 +0400

Glauber Costa <glommer@parallels.com> wrote:

> This patch adds the shrinker interface to memcg proposed kmem  
> controller. With this, softlimits starts being meaningful. I didn't  
> played to much with softlimits itself, since it is a bit in progress  
> for the general case as well. But this patch at least makes vmscan.c  
> no longer skip shrink\_slab for the memcg case.  
>  
> It also allows us to set the hard limit to a lower value than  
> current usage, as it is possible for the current memcg: a reclaim  
> is carried on, and if we succeed in freeing enough of kernel memory,  
> we can lower the limit.  
>  
> Signed-off-by: Glauber Costa <glommer@parallels.com>  
> CC: Kirill A. Shutemov <kirill@shutemov.name>  
> CC: Greg Thelen <gthelen@google.com>  
> CC: Johannes Weiner <jweiner@redhat.com>  
> CC: Michal Hocko <mhocko@suse.cz>  
> CC: Hiroyuki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>  
> CC: Paul Turner <pjt@google.com>  
> CC: Frederic Weisbecker <fweisbec@gmail.com>  
> CC: Pekka Enberg <penberg@kernel.org>  
> CC: Christoph Lameter <cl@linux.com>  
> ---  
> include/linux/memcontrol.h | 5 +++  
> include/linux/shrinker.h | 4 ++  
> mm/memcontrol.c | 87 ++++++-----  
> mm/vmscan.c | 60 ++++++-----  
> 4 files changed, 150 insertions(+), 6 deletions(-)  
>  
> diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h  
> index 6138d10..246b2d4 100644  
> --- a/include/linux/memcontrol.h  
> +++ b/include/linux/memcontrol.h  
> @@ -33,12 +33,16 @@ struct mm\_struct;  
> struct memcg\_kmem\_cache {  
> struct kmem\_cache \*cache;  
> struct work\_struct destroy;  
> struct list\_head lru;  
> u32 nr\_objects;  
> struct mem\_cgroup \*memcg; /\* Should be able to do without this \*/  
> };  
>

```

> struct memcg_cache_struct {
>     int index;
>     struct kmem_cache *cache;
> +     int (*shrink_fn)(struct shrinker *shrink, struct shrink_control *sc);
> +     struct shrinker shrink;
> };
>
> enum memcg_cache_indexes {
>     @@ -53,6 +57,7 @@ struct mem_cgroup *memcg_from_shrinker(struct shrinker *s);
>     struct memcg_kmem_cache *memcg_cache_get(struct mem_cgroup *memcg, int index);
>     void register_memcg_cache(struct memcg_cache_struct *cache);
>     void memcg_slab_destroy(struct kmem_cache *cache, struct mem_cgroup *memcg);
> +     bool memcg_slab_reclaim(struct mem_cgroup *memcg);
>
>     struct kmem_cache *
>     kmem_cache_dup(struct mem_cgroup *memcg, struct kmem_cache *base);
> diff --git a/include/linux/shrinker.h b/include/linux/shrinker.h
> index 07ceb97..11efdba 100644
> --- a/include/linux/shrinker.h
> +++ b/include/linux/shrinker.h
> @@ -1,6 +1,7 @@
> #ifndef _LINUX_SHRINKER_H
> #define _LINUX_SHRINKER_H
>
> +     struct mem_cgroup;
> /*
>     * This struct is used to pass information from page reclaim to the shrinkers.
>     * We consolidate the values for easier extention later.
>     @@ -10,6 +11,7 @@ struct shrink_control {
>
>     /* How many slab objects shrinker() should scan and try to reclaim */
>     unsigned long nr_to_scan;
> +     struct mem_cgroup *memcg;
> };
>
> /*
>     @@ -40,4 +42,6 @@ struct shrinker {
>     #define DEFAULT_SEEKS 2 /* A good number if you don't know better. */
>     extern void register_shrinker(struct shrinker *);
>     extern void unregister_shrinker(struct shrinker *);
> +
>     +extern void register_shrinker_memcg(struct shrinker *);
> #endif
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 1b1db88..9c89a3c 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
>     @@ -3460,6 +3460,54 @@ static int mem_cgroup_resize_limit(struct mem_cgroup *memcg,

```

```

>     return ret;
> }
>
> +static int mem_cgroup_resize_kmem_limit(struct mem_cgroup *memcg,
> +    unsigned long long val)
> +{
> +
> +    int retry_count;
> +    int ret = 0;
> +    int children = mem_cgroup_count_children(memcg);
> +    u64 curusage, oldusage;
> +
> +    struct shrink_control shrink = {
> +        .gfp_mask = GFP_KERNEL,
> +        .memcg = memcg,
> +    };
> +
> +    /*
> +     * For keeping hierarchical_reclaim simple, how long we should retry
> +     * depends on callers. We set our retry-count to be function
> +     * of # of children which we should visit in this loop.
> +     */
> +    retry_count = MEM_CGROUP_RECLAIM_RETRIES * children;
> +
> +    oldusage = res_counter_read_u64(&memcg->kmem, RES_USAGE);
> +
> +    while (retry_count) {
> +        if (signal_pending(current)) {
> +            ret = -EINTR;
> +            break;
> +        }
> +        mutex_lock(&set_limit_mutex);
> +        ret = res_counter_set_limit(&memcg->kmem, val);
> +        mutex_unlock(&set_limit_mutex);
> +        if (!ret)
> +            break;
> +
> +        shrink_slab(&shrink, 0, 0);
> +
> +        curusage = res_counter_read_u64(&memcg->kmem, RES_USAGE);
> +
> +        /* Usage is reduced ? */
> +        if (curusage >= oldusage)
> +            retry_count--;
> +        else
> +            oldusage = curusage;
> +    }
> +    return ret;

```

```

> +
> +}
> +
> static int mem_cgroup_resize_memsw_limit(struct mem_cgroup *memcg,
>     unsigned long long val)
> {
> @@ -3895,13 +3943,17 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
>     break;
>     if (type == _MEM)
>         ret = mem_cgroup_resize_limit(memcg, val);
>+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>     else if (type == _KMEM) {
>         if (!memcg->kmem_independent_accounting) {
>             ret = -EINVAL;
>             break;
>         }
> -    ret = res_counter_set_limit(&memcg->kmem, val);
> - } else
> +
> +    ret = mem_cgroup_resize_kmem_limit(memcg, val);
> +
> +#endif
> + else
>     ret = mem_cgroup_resize_memsw_limit(memcg, val);
>     break;
>     case RES_SOFT_LIMIT:
> @@ -5007,9 +5059,19 @@ struct memcg_kmem_cache *memcg_cache_get(struct
mem_cgroup *memcg, int index)
>
> void register_memcg_cache(struct memcg_cache_struct *cache)
> {
> + struct shrinker *shrink;
> +
> BUG_ON(kmem_avail_caches[cache->index]);
>
> kmem_avail_caches[cache->index] = cache;
> + if (!kmem_avail_caches[cache->index]->shrink_fn)
> + return;
> +
> + shrink = &kmem_avail_caches[cache->index]->shrink;
> + shrink->seeks = DEFAULT_SEEKS;
> + shrink->shrink = kmem_avail_caches[cache->index]->shrink_fn;
> + shrink->batch = 1024;
> + register_shrinker_memcg(shrink);
> }
>
> #define memcg_kmem(memcg) \
> @@ -5055,8 +5117,21 @@ int memcg_kmem_newpage(struct mem_cgroup *memcg, struct

```

```

page *page, unsigned lon
> {
>     unsigned long size = pages << PAGE_SHIFT;
>     struct res_counter *fail;
> + int ret;
> + bool do_softlimit;
> +
> + ret = res_counter_charge(memcg_kmem(memcg), size, &fail);
> + if (unlikely(mem_cgroup_event_ratelimit(memcg,
> +     MEM_CGROUP_TARGET_THRESH))) {
> +
> +     do_softlimit = mem_cgroup_event_ratelimit(memcg,
> +     MEM_CGROUP_TARGET_SOFTLIMIT);
> +     mem_cgroup_threshold(memcg);
> +     if (unlikely(do_softlimit))
> +         mem_cgroup_update_tree(memcg, page);
> +
}

```

Do we need to have this hook here ?  
(BTW, please don't duplicate...)

```

>
> - return res_counter_charge(memcg_kmem(memcg), size, &fail);
> + return ret;
> }
>
> void memcg_kmem_freepage(struct mem_cgroup *memcg, struct page *page, unsigned long
pages)
> @@ -5083,6 +5158,7 @@ void memcg_create_kmem_caches(struct mem_cgroup *memcg)
>     else
>         memcg->kmem_cache[i].cache = kmem_cache_dup(memcg, cache);
>         INIT_WORK(&memcg->kmem_cache[i].destroy, memcg_cache_destroy);
> +     INIT_LIST_HEAD(&memcg->kmem_cache[i].lru);
>         memcg->kmem_cache[i].memcg = memcg;
>     }
> }
> @@ -5157,6 +5233,11 @@ free_out:
>     return ERR_PTR(error);
> }
>
> +bool memcg_slab_reclaim(struct mem_cgroup *memcg)
> +{
> +    return !memcg->kmem_independent_accounting;
> +}
> +
> void memcg_slab_destroy(struct kmem_cache *cache, struct mem_cgroup *memcg)
> {

```

```

> int i;
> diff --git a/mm/vmscan.c b/mm/vmscan.c
> index c52b235..b9bceb6 100644
> --- a/mm/vmscan.c
> +++ b/mm/vmscan.c
> @@ -159,6 +159,23 @@ long vm_total_pages; /* The total number of pages which the VM
controls */
> static LIST_HEAD(shrinker_list);
> static DECLARE_RWSEM(shrinker_rwsem);
>
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +/*
> + * If we could guarantee the root mem cgroup will always exist, we could just
> + * use the normal shrinker_list, and assume that the root memcg is passed
> + * as a parameter. But we're not quite there yet. Because of that, the shrinkers
> + * from the memcg case can be different from the normal shrinker for the same
> + * object. This is not the ideal situation but is a step towards that.
> +
> + * Also, not all caches will have their memcg version (also likely to change),
> + * so scanning the whole list is a waste.
> +
> + * I am using, however, the same lock for both lists. Updates to it should
> + * be unfrequent, so I don't expect that to generate contention
> +
> +static LIST_HEAD(shrinker_memcg_list);
> +#endif
> +
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR
> static bool global_reclaim(struct scan_control *sc)
> {
> @@ -169,6 +186,11 @@ static bool scanning_global_lru(struct mem_cgroup_zone *mz)
> {
>     return !mz->mem_cgroup;
> }
> +
> +static bool global_slab_reclaim(struct scan_control *sc)
> +{
>     return !memcg_slab_reclaim(sc->target_mem_cgroup);
> }

```

Do we need this new function ? global\_reclaim() isn't enough ?

Thanks,  
-Kame

```

> #else
> static bool global_reclaim(struct scan_control *sc)

```

```

> {
> @@ -179,6 +201,11 @@ static bool scanning_global_lru(struct mem_cgroup_zone *mz)
> {
>   return true;
> }
> +
> +static bool global_slab_reclaim(struct scan_control *sc)
> +{
> +  return true;
> +}
> #endif
>
> static struct zone_reclaim_stat *get_reclaim_stat(struct mem_cgroup_zone *mz)
> @@ -225,6 +252,16 @@ void unregister_shrinker(struct shrinker *shrinker)
> }
> EXPORT_SYMBOL(unregister_shrinker);
>
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +void register_shrinker_memcg(struct shrinker *shrinker)
> +{
> +  atomic_long_set(&shrinker->nr_in_batch, 0);
> +  down_write(&shrinker_rwsem);
> +  list_add_tail(&shrinker->list, &shrinker_memcg_list);
> +  up_write(&shrinker_rwsem);
> +}
> +#endif
> +
> static inline int do_shrinker_shrink(struct shrinker *shrinker,
>           struct shrink_control *sc,
>           unsigned long nr_to_scan)
> @@ -234,6 +271,18 @@ static inline int do_shrinker_shrink(struct shrinker *shrinker,
> }
>
> #define SHRINK_BATCH 128
> +
> +static inline struct list_head
> +*get_shrinker_list(struct shrink_control *shrink)
> +{
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +  if (shrink->memcg)
> +    return &shrinker_memcg_list;
> +  else
> +#endif
> +  return &shrinker_list;
> +}
> +
> /*
>   * Call the shrink functions to age shrinkable caches

```

```

> *
> @@ -259,6 +308,9 @@ unsigned long shrink_slab(struct shrink_control *shrink,
> {
>     struct shrinker *shrinker;
>     unsigned long ret = 0;
> +    struct list_head *slist;
> +
> +    slist = get_shrinker_list(shrink);
>
>     if (nr_pages_scanned == 0)
>         nr_pages_scanned = SWAP_CLUSTER_MAX;
> @@ -269,7 +321,7 @@ unsigned long shrink_slab(struct shrink_control *shrink,
>     goto out;
> }
>
> - list_for_each_entry(shrinker, &shrinker_list, list) {
> + list_for_each_entry(shrinker, slist, list) {
>     unsigned long long delta;
>     long total_scan;
>     long max_pass;
> @@ -2351,9 +2403,9 @@ static unsigned long do_try_to_free_pages(struct zonelist *zonelist,
>
>     /*
>      * Don't shrink slabs when reclaiming memory from
> -     * over limit cgroups
> +     * over limit cgroups, if kernel memory is controlled independently
>     */
> -     if (global_reclaim(sc)) {
> +     if (!global_slab_reclaim(sc)) {
>         unsigned long lru_pages = 0;
>         for_each_zone_zonelist(zone, z, zonelist,
>             gfp_zone(sc->gfp_mask));
> @@ -2362,8 +2414,10 @@ static unsigned long do_try_to_free_pages(struct zonelist *zonelist,
>
>         lru_pages += zone_reclaimable_pages(zone);
>     }
> +     shrink->memcg = sc->target_mem_cgroup;
>
>     shrink_slab(shrink, sc->nr_scanned, lru_pages);
> +
>     if (reclaim_state) {
>         sc->nr_reclaimed += reclaim_state->reclaimed_slab;
>         reclaim_state->reclaimed_slab = 0;
> --
> 1.7.7.6
>
> --
> To unsubscribe from this list: send the line "unsubscribe cgroups" in

```

> the body of a message to majordomo@vger.kernel.org  
> More majordomo info at <http://vger.kernel.org/majordomo-info.html>  
>

---