Subject: Re: [PATCH v2 08/11] memcg: disable kmem code when not in use.
Posted by Michal Hocko on Fri, 17 Aug 2012 07:02:41 GMT
View Forum Message <> Reply to Message

On Thu 09-08-12 17:01:16, Glauber Costa wrote:
> We can use jump labels to patch the code in or out when not used.
>
> Because the assignment: memcg->kmem_accounted = true is done after the
> jump labels increment, we guarantee that the root memcg will always be
> selected until all call sites are patched (see memcg_kmem_enabled).

Not that it would be really important because kmem_accounted goes away
in a subsequent patch but I think the wording is a bit misleading here.
First of all there is no guanratee that kmem_accounted=true is seen
before atomic_inc(&key->enabled) because there is no memory barrier and
the lock serves just a leave barrier. But I do not think this is
important at all because key->enabled is what matters here. Even if
memcg_kmem_enabled is true we do not consider it if the key is disabled,
right?

> This guarantees that no mischarges are applied.
>
> Jump label decrement happens when the last reference count from the
> memcg dies. This will only happen when the caches are all dead.
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> Acked-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> CC: Christoph Lameter <cl@linux.com>
> CC: Pekka Enberg <penberg@cs.helsinki.fi>
> CC: Michal Hocko <mhocko@suse.cz>
> CC: Johannes Weiner <hannes@cmpxchg.org>
> CC: Suleiman Souhlal <suleiman@google.com>

Anyway the code looks correct.
Reviewed-by: Michal Hocko <mhocko@suse.cz>

> ---
> include/linux/memcontrol.h | 5 ++++-
> mm/memcontrol.c | 50 ++----------
> 2 files changed, 44 insertions(+), 11 deletions(-)
>
> diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
> index 75b247e..f39d933 100644
> --- a/include/linux/memcontrol.h
> +++ b/include/linux/memcontrol.h
> @@ -22,6 +22,7 @@
> #include <linux/cgroup.h>
> #include <linux/vm_event_item.h>

> #include <linux/hardirq.h>
> +#include <linux/jump_label.h>
>
> struct mem_cgroup;
> struct page_cgroup;
> @@ -401,7 +402,9 @@ struct sock;
> void sock_update_memcg(struct sock *sk);
> void sock_release_memcg(struct sock *sk);
>
> -#define memcg_kmem_on 1
> +extern struct static_key memcg_kmem_enabled_key;
> +#define memcg_kmem_on static_key_false(&memcg_kmem_enabled_key)
> +
> bool __memcg_kmem_new_page(gfp_t gfp, void *handle, int order);
> void __memcg_kmem_commit_page(struct page *page, void *handle, int order);
> void __memcg_kmem_free_page(struct page *page, int order);
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index e9824c1..3216292 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -437,6 +437,10 @@ struct mem_cgroup *mem_cgroup_from_css(struct
cgroup_subsys_state *s)
> #include <net/sock.h>
> #include <net/ip.h>
>
> +struct static_key memcg_kmem_enabled_key;
> +/* so modules can inline the checks */
> +EXPORT_SYMBOL(memcg_kmem_enabled_key);
> +
> static bool mem_cgroup_is_root(struct mem_cgroup *memcg);
> static int memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, s64 delta);
> static void memcg_uncharge_kmem(struct mem_cgroup *memcg, s64 delta);
> @@ -607,6 +611,16 @@ void __memcg_kmem_free_page(struct page *page, int order)
> mem_cgroup_put(memcg);
> }
> EXPORT_SYMBOL(__memcg_kmem_free_page);
> +
> +static void disarm_kmem_keys(struct mem_cgroup *memcg)
> +{
> + if (memcg->kmem_accounted)
> + static_key_slow_dec(&memcg_kmem_enabled_key);
> +}
> +#else
> +static void disarm_kmem_keys(struct mem_cgroup *memcg)
> +{
> +}
> #endif /* CONFIG_MEMCG_KMEM */
>

```
>  #if defined(CONFIG_INET) && defined(CONFIG_MEMCG_KMEM)
> @@ -622,6 +636,12 @@ static void disarm_sock_keys(struct mem_cgroup *memcg)
>  }
>  #endif
>
> +static void disarm_static_keys(struct mem_cgroup *memcg)
> +{
> + disarm_sock_keys(memcg);
> + disarm_kmem_keys(memcg);
> +}
> +
>  static void drain_all_stock_async(struct mem_cgroup *memcg);
>
>  static struct mem_cgroup_per_zone *
> @@ -4147,6 +4167,24 @@ static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype
*cft,
>   len = scnprintf(str, sizeof(str), "%llu\n", (unsigned long long)val);
>   return simple_read_from_buffer(buf, nbytes, ppos, str, len);
>  }
> +
> +static void memcg_update_kmem_limit(struct mem_cgroup *memcg, u64 val)
> +{
> +#ifdef CONFIG_MEMCG_KMEM
> + /*
> +  * Once enabled, can't be disabled. We could in theory disable it if we
> +  * haven't yet created any caches, or if we can shrink them all to
> +  * death. But it is not worth the trouble.
> +  */
> + mutex_lock(&set_limit_mutex);
> + if (!memcg->kmem_accounted && val != RESOURCE_MAX) {
> +  static_key_slow_inc(&memcg_kmem_enabled_key);
> +  memcg->kmem_accounted = true;
> + }
> + mutex_unlock(&set_limit_mutex);
> +#endif
> +}
> +
>  /*
>   * The user of this function is...
>   * RES_LIMIT.
> @@ -4184,15 +4222,7 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
>     ret = res_counter_set_limit(&memcg->kmem, val);
>     if (ret)
>      break;
> -   /*
> -    * Once enabled, can't be disabled. We could in theory
> -    * disable it if we haven't yet created any caches, or
> -    * if we can shrink them all to death.
```

> -   *
> -   * But it is not worth the trouble
> -   */
> -   if (!memcg->kmem_accounted && val != RESOURCE_MAX)
> -   memcg->kmem_accounted = true;
> +   memcg_update_kmem_limit(memcg, val);
>   } else
>   return -EINVAL;
>   break;
> @@ -5054,7 +5084,7 @@ static void free_work(struct work_struct *work)
>   * to move this code around, and make sure it is outside
>   * the cgroup_lock.
>   */
> - disarm_sock_keys(memcg);
> + disarm_static_keys(memcg);
>   if (size < PAGE_SIZE)
>   kfree(memcg);
>   else
> --
> 1.7.11.2
>
> --
> 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

--
Michal Hocko
SUSE Labs