Subject: Re: [PATCH v4 19/25] memcg: disable kmem code when not in use.
Posted by KAMEZAWA Hiroyuki on Mon, 18 Jun 2012 12:22:14 GMT
View Forum Message <> Reply to Message

(2012/06/18 19:28), 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 mem_cgroup_kmem_enabled).
> 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>
> CC: Christoph Lameter<cl@linux.com>
> CC: Pekka Enberg<penberg@cs.helsinki.fi>
> CC: Michal Hocko<mhocko@suse.cz>
> CC: Kamezawa Hiroyuki<kamezawa.hiroyu@jp.fujitsu.com>
> CC: Johannes Weiner<hannes@cmpxchg.org>
> CC: Suleiman Souhlal<suleiman@google.com>
> ---
> include/linux/memcontrol.h | 5 ++++-
> mm/memcontrol.c | 22 ++++++++++++++++++++++-
> 2 files changed, 25 insertions(+), 2 deletions(-)
>
> diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
> index 27a3f16..47ccd80 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;
> @@ -451,7 +452,6 @@ bool __mem_cgroup_new_kmem_page(gfp_t gfp, void *handle, int
order);
> void __mem_cgroup_commit_kmem_page(struct page *page, void *handle, int order);
> void __mem_cgroup_free_kmem_page(struct page *page, int order);
>
> -#define mem_cgroup_kmem_on 1

```
>   struct kmem_cache *
>   __mem_cgroup_get_kmem_cache(struct kmem_cache *cachep, gfp_t gfp);
>
> @@ -459,6 +459,9 @@ static inline bool has_memcg_flag(gfp_t gfp)
>   {
>    return gfp&  __GFP_SLABMEMCG;
>   }
> +
> +extern struct static_key mem_cgroup_kmem_enabled_key;
> +#define mem_cgroup_kmem_on static_key_false(&mem_cgroup_kmem_enabled_key)
>   #else
>   static inline void mem_cgroup_register_cache(struct mem_cgroup *memcg,
>            struct kmem_cache *s)
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index b47ab87..5295ab6 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -422,6 +422,10 @@ static void mem_cgroup_put(struct mem_cgroup *memcg);
>   #include<net/sock.h>
>   #include<net/ip.h>
>
> +struct static_key mem_cgroup_kmem_enabled_key;
> +/* so modules can inline the checks */
> +EXPORT_SYMBOL(mem_cgroup_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);
> @@ -468,6 +472,12 @@ void sock_release_memcg(struct sock *sk)
>    }
>   }
>
> +static void disarm_static_keys(struct mem_cgroup *memcg)
> +{
> + if (memcg->kmem_accounted)
> +  static_key_slow_dec(&mem_cgroup_kmem_enabled_key);
> +}
> +
>   #ifdef CONFIG_INET
>   struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
>   {
> @@ -831,6 +841,10 @@ static void memcg_slab_init(struct mem_cgroup *memcg)
>    for (i = 0; i<  MAX_KMEM_CACHE_TYPES; i++)
>     memcg->slabs[i] = NULL;
>   }
> +#else
> +static inline void disarm_static_keys(struct mem_cgroup *memcg)
> +{
```

> +}
> #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
>
> static void drain_all_stock_async(struct mem_cgroup *memcg);
> @@ -4344,8 +4358,13 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
> *
> * But it is not worth the trouble
> */
> - if (!memcg->kmem_accounted&& val != RESOURCE_MAX)
> + mutex_lock(&set_limit_mutex);
> + if (!memcg->kmem_accounted&& val != RESOURCE_MAX
> + && !memcg->kmem_accounted) {

I'm sorry why you check the value twice ?

Thanks,
-Kame