Subject: Re: [PATCH 19/23] slab: per-memcg accounting of slab caches
Posted by Suleiman Souhlal on Mon, 30 Apr 2012 21:25:47 GMT
View Forum Message <> Reply to Message

On Sun, Apr 22, 2012 at 4:53 PM, Glauber Costa <glommer@parallels.com> wrote:
> This patch charges allocation of a slab object to a particular
> memcg.
>
> The cache is selected with mem_cgroup_get_kmem_cache(),
> which is the biggest overhead we pay here, because
> it happens at all allocations. However, other than forcing
> a function call, this function is not very expensive, and
> try to return as soon as we realize we are not a memcg cache.
>
> The charge/uncharge functions are heavier, but are only called
> for new page allocations.
>
> Code is heavily inspired by Suleiman's, with adaptations to
> the patchset and minor simplifications by me.
>
> 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/slab_def.h | 66 ++-
> mm/slab.c | 105
++----
> 2 files changed, 162 insertions(+), 9 deletions(-)
>
> diff --git a/include/linux/slab_def.h b/include/linux/slab_def.h
> index 54d25d7..c4f7e45 100644
> --- a/include/linux/slab_def.h
> +++ b/include/linux/slab_def.h
> @@ -51,7 +51,7 @@ struct kmem_cache {
>   void (*ctor)(void *obj);
>
> /* 4) cache creation/removal */
> -   const char *name;
> +   char *name;
>   struct list_head next;
>
> /* 5) statistics */
> @@ -219,4 +219,68 @@ found:
>

```
>  #endif /* CONFIG_NUMA */
>
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +
> +void kmem_cache_drop_ref(struct kmem_cache *cachep);
> +
> +static inline void
> +kmem_cache_get_ref(struct kmem_cache *cachep)
> +{
> +       if (cachep->memcg_params.id == -1 &&
> +           unlikely(!atomic_add_unless(&cachep->memcg_params.refcnt, 1, 0)))
> +               BUG();
> +}
> +
> +static inline void
> +mem_cgroup_put_kmem_cache(struct kmem_cache *cachep)
> +{
> +       rcu_read_unlock();
> +}
> +
> +static inline void
> +mem_cgroup_kmem_cache_prepare_sleep(struct kmem_cache *cachep)
> +{
> +       /*
> +        * Make sure the cache doesn't get freed while we have interrupts
> +        * enabled.
> +        */
> +       kmem_cache_get_ref(cachep);
> +       rcu_read_unlock();
> +}
> +
> +static inline void
> +mem_cgroup_kmem_cache_finish_sleep(struct kmem_cache *cachep)
> +{
> +       rcu_read_lock();
> +       kmem_cache_drop_ref(cachep);
> +}
> +
> +#else /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
> +
> +static inline void
> +kmem_cache_get_ref(struct kmem_cache *cachep)
> +{
> +}
> +
> +static inline void
> +kmem_cache_drop_ref(struct kmem_cache *cachep)
> +{
```

```
> +}
> +
> +static inline void
> +mem_cgroup_put_kmem_cache(struct kmem_cache *cachep)
> +{
> +}
> +
> +static inline void
> +mem_cgroup_kmem_cache_prepare_sleep(struct kmem_cache *cachep)
> +{
> +}
> +
> +static inline void
> +mem_cgroup_kmem_cache_finish_sleep(struct kmem_cache *cachep)
> +{
> +}
> +#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
> +
>  #endif /* _LINUX_SLAB_DEF_H */
> diff --git a/mm/slab.c b/mm/slab.c
> index 13948c3..ac0916b 100644
> --- a/mm/slab.c
> +++ b/mm/slab.c
> @@ -1818,20 +1818,28 @@ static void *kmem_getpages(struct kmem_cache *cachep, gfp_t
flags, int nodeid)
>       if (cachep->flags & SLAB_RECLAIM_ACCOUNT)
>               flags |= __GFP_RECLAIMABLE;
>
> +     nr_pages = (1 << cachep->gfporder);
> +     if (!mem_cgroup_charge_slab(cachep, flags, nr_pages * PAGE_SIZE))
> +             return NULL;
> +
>       page = alloc_pages_exact_node(nodeid, flags | __GFP_NOTRACK,
cachep->gfporder);
>       if (!page) {
>               if (!(flags & __GFP_NOWARN) && printk_ratelimit())
>                       slab_out_of_memory(cachep, flags, nodeid);
> +
> +             mem_cgroup_uncharge_slab(cachep, nr_pages * PAGE_SIZE);
>               return NULL;
>       }
>
> -     nr_pages = (1 << cachep->gfporder);
>       if (cachep->flags & SLAB_RECLAIM_ACCOUNT)
>               add_zone_page_state(page_zone(page),
>                       NR_SLAB_RECLAIMABLE, nr_pages);
>       else
>               add_zone_page_state(page_zone(page),
```

```
>                     NR_SLAB_UNRECLAIMABLE, nr_pages);
> +
> +       kmem_cache_get_ref(cachep);
> +
>        for (i = 0; i < nr_pages; i++)
>                __SetPageSlab(page + i);
>
> @@ -1864,6 +1872,8 @@ static void kmem_freepages(struct kmem_cache *cachep, void
*addr)
>        else
>                sub_zone_page_state(page_zone(page),
>                                NR_SLAB_UNRECLAIMABLE, nr_freed);
> +       mem_cgroup_uncharge_slab(cachep, i * PAGE_SIZE);
> +       kmem_cache_drop_ref(cachep);
>        while (i--) {
>                BUG_ON(!PageSlab(page));
>                __ClearPageSlab(page);
> @@ -2823,12 +2833,28 @@ void kmem_cache_destroy(struct kmem_cache *cachep)
>        if (unlikely(cachep->flags & SLAB_DESTROY_BY_RCU))
>                rcu_barrier();
>
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +       /* Not a memcg cache */
> +       if (cachep->memcg_params.id != -1) {
> +               mem_cgroup_release_cache(cachep);
> +               mem_cgroup_flush_cache_create_queue();
> +       }
> +#endif
>        __kmem_cache_destroy(cachep);
>        mutex_unlock(&cache_chain_mutex);
>        put_online_cpus();
>  }
>  EXPORT_SYMBOL(kmem_cache_destroy);
>
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +void kmem_cache_drop_ref(struct kmem_cache *cachep)
> +{
> +       if (cachep->memcg_params.id == -1 &&
> +           unlikely(atomic_dec_and_test(&cachep->memcg_params.refcnt)))
> +               mem_cgroup_destroy_cache(cachep);
> +}
> +#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
> +
>  /*
>   * Get the memory for a slab management obj.
>   * For a slab cache when the slab descriptor is off-slab, slab descriptors
> @@ -3028,8 +3054,10 @@ static int cache_grow(struct kmem_cache *cachep,
>
```

```
>       offset *= cachep->colour_off;
>
> -      if (local_flags & __GFP_WAIT)
> +      if (local_flags & __GFP_WAIT) {
>             local_irq_enable();
> +             mem_cgroup_kmem_cache_prepare_sleep(cachep);
> +      }
>
>       /*
>        * The test for missing atomic flag is performed here, rather than
> @@ -3058,8 +3086,10 @@ static int cache_grow(struct kmem_cache *cachep,
>
>       cache_init_objs(cachep, slabp);
>
> -      if (local_flags & __GFP_WAIT)
> +      if (local_flags & __GFP_WAIT) {
>             local_irq_disable();
> +             mem_cgroup_kmem_cache_finish_sleep(cachep);
> +      }
>       check_irq_off();
>       spin_lock(&l3->list_lock);
>
> @@ -3072,8 +3102,10 @@ static int cache_grow(struct kmem_cache *cachep,
>  opps1:
>       kmem_freepages(cachep, objp);
>  failed:
> -      if (local_flags & __GFP_WAIT)
> +      if (local_flags & __GFP_WAIT) {
>             local_irq_disable();
> +             mem_cgroup_kmem_cache_finish_sleep(cachep);
> +      }
>       return 0;
>  }
>
> @@ -3834,11 +3866,15 @@ static inline void __cache_free(struct kmem_cache *cachep, void
*objp,
>  */
>  void *kmem_cache_alloc(struct kmem_cache *cachep, gfp_t flags)
>  {
> -      void *ret = __cache_alloc(cachep, flags, __builtin_return_address(0));
> +      void *ret;
> +
> +      rcu_read_lock();
> +      cachep = mem_cgroup_get_kmem_cache(cachep, flags);
> +      rcu_read_unlock();
```

Don't we need to check in_interrupt(), current, __GFP_NOFAIL every
time we call mem_cgroup_cgroup_get_kmem_cache()?

I would personally prefer if those checks were put inside mem_cgroup_get_kmem_cache() instead of having to check for every caller.

-- Suleiman