Subject: Re: [PATCH v5 06/18] consider a memcg parameter in kmem_create_cache
Posted by JoonSoo Kim on Tue, 23 Oct 2012 17:50:04 GMT
View Forum Message <> Reply to Message

2012/10/19 Glauber Costa <glommer@parallels.com>:

```
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 6a1e096..59f6d54 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -339,6 +339,12 @@ struct mem_cgroup {
>  #if defined(CONFIG_MEMCG_KMEM) && defined(CONFIG_INET)
>        struct tcp_memcontrol tcp_mem;
>  #endif
> +#if defined(CONFIG_MEMCG_KMEM)
> +       /* analogous to slab_common's slab_caches list. per-memcg */
> +       struct list_head memcg_slab_caches;
> +       /* Not a spinlock, we can take a lot of time walking the list */
> +       struct mutex slab_caches_mutex;
> +#endif
>  };
```

It is better to change name of "slab_caches_mutex to something
representing slab cache mutex of memcg.

```
> +int memcg_register_cache(struct mem_cgroup *memcg, struct kmem_cache *s)
> +{
> +       size_t size = sizeof(struct memcg_cache_params);
> +
> +       if (!memcg_kmem_enabled())
> +               return 0;
> +
> +       s->memcg_params = kzalloc(size, GFP_KERNEL);
> +       if (!s->memcg_params)
> +               return -ENOMEM;
> +
> +       if (memcg)
> +               s->memcg_params->memcg = memcg;
> +       return 0;
> +}
```

Now, I don't read full-patchset and I have not enough knowledge about cgroup.
So I have a question.
When memcg_kmem_enable, creation kmem_cache of normal user(not
included in cgroup) also allocate memcg_params.
Is it right behavior?

```
> +void memcg_release_cache(struct kmem_cache *s)
> +{
> +	kfree(s->memcg_params);
> +}
> +
>  /*
>   * We need to verify if the allocation against current->mm->owner's memcg is
>   * possible for the given order. But the page is not allocated yet, so we'll
> diff --git a/mm/slab.h b/mm/slab.h
> index 5ee1851..c35ecce 100644
> --- a/mm/slab.h
> +++ b/mm/slab.h
> @@ -35,12 +35,15 @@ extern struct kmem_cache *kmem_cache;
>  /* Functions provided by the slab allocators */
>  extern int __kmem_cache_create(struct kmem_cache *, unsigned long flags);
>
> +struct mem_cgroup;
>  #ifdef CONFIG_SLUB
> -struct kmem_cache *__kmem_cache_alias(const char *name, size_t size,
> -	size_t align, unsigned long flags, void (*ctor)(void *));
> +struct kmem_cache *
> +__kmem_cache_alias(struct mem_cgroup *memcg, const char *name, size_t size,
> +		   size_t align, unsigned long flags, void (*ctor)(void *));
>  #else
> -static inline struct kmem_cache *__kmem_cache_alias(const char *name, size_t size,
> -	size_t align, unsigned long flags, void (*ctor)(void *))
> +static inline struct kmem_cache *
> +__kmem_cache_alias(struct mem_cgroup *memcg, const char *name, size_t size,
> +		   size_t align, unsigned long flags, void (*ctor)(void *))
>  { return NULL; }
>  #endif
>
> @@ -98,11 +101,23 @@ static inline bool is_root_cache(struct kmem_cache *s)
>  {
>  	return !s->memcg_params || s->memcg_params->is_root_cache;
>  }
> +
> +static inline bool cache_match_memcg(struct kmem_cache *cachep,
> +				     struct mem_cgroup *memcg)
> +{
> +	return (is_root_cache(cachep) && !memcg) ||
> +		(cachep->memcg_params->memcg == memcg);
> +}
```

When cachep->memcg_params == NULL and memcg is not NULL, NULL pointer
deref may be occurred.
Is there no situation like above?

```
>  #else
>  static inline bool is_root_cache(struct kmem_cache *s)
>  {
>       return true;
>  }
>
> +static inline bool cache_match_memcg(struct kmem_cache *cachep,
> +                                      struct mem_cgroup *memcg)
> +{
> +       return true;
> +}
>  #endif
>  #endif
> diff --git a/mm/slab_common.c b/mm/slab_common.c
> index bf4b4f1..f97f7b8 100644
> --- a/mm/slab_common.c
> +++ b/mm/slab_common.c
> @@ -18,6 +18,7 @@
>  #include <asm/cacheflush.h>
>  #include <asm/tlbflush.h>
>  #include <asm/page.h>
> +#include <linux/memcontrol.h>
>
>  #include "slab.h"
>
> @@ -27,7 +28,8 @@ DEFINE_MUTEX(slab_mutex);
>  struct kmem_cache *kmem_cache;
>
>  #ifdef CONFIG_DEBUG_VM
> -static int kmem_cache_sanity_check(const char *name, size_t size)
> +static int kmem_cache_sanity_check(struct mem_cgroup *memcg, const char *name,
> +                                    size_t size)
>  {
>       struct kmem_cache *s = NULL;
>
> @@ -53,7 +55,7 @@ static int kmem_cache_sanity_check(const char *name, size_t size)
>                       continue;
>               }
>
> -             if (!strcmp(s->name, name)) {
> +             if (cache_match_memcg(s, memcg) && !strcmp(s->name, name)) {
>                       pr_err("%s (%s): Cache name already exists.\n",
>                               __func__, name);
>                       dump_stack();
> @@ -66,7 +68,8 @@ static int kmem_cache_sanity_check(const char *name, size_t size)
>       return 0;
>  }
>  #else
> -static inline int kmem_cache_sanity_check(const char *name, size_t size)
> +static inline int kmem_cache_sanity_check(struct mem_cgroup *memcg,
> +                                          const char *name, size_t size)
>  {
>         return 0;
>  }
> @@ -97,8 +100,9 @@ static inline int kmem_cache_sanity_check(const char *name, size_t
size)
>   * as davem.
>   */
>
> -struct kmem_cache *kmem_cache_create(const char *name, size_t size, size_t align,
> -               unsigned long flags, void (*ctor)(void *))
> +struct kmem_cache *
> +kmem_cache_create_memcg(struct mem_cgroup *memcg, const char *name, size_t size,
> +                        size_t align, unsigned long flags, void (*ctor)(void *))
>  {
>         struct kmem_cache *s = NULL;
>         int err = 0;
> @@ -106,7 +110,7 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t
size, size_t align
>         get_online_cpus();
>         mutex_lock(&slab_mutex);
>
> -        if (!kmem_cache_sanity_check(name, size) == 0)
> +        if (!kmem_cache_sanity_check(memcg, name, size) == 0)
>                 goto out_locked;
```

This compare is somewhat ugly.
How about "if(kmem_cache_sanity_check(memcg, name, size))"?

```
>         /*
> @@ -117,7 +121,7 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t
size, size_t align
>          */
>         flags &= CACHE_CREATE_MASK;
>
> -        s = __kmem_cache_alias(name, size, align, flags, ctor);
> +        s = __kmem_cache_alias(memcg, name, size, align, flags, ctor);
>         if (s)
>                 goto out_locked;
>
> @@ -126,6 +130,13 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t
size, size_t align
>                 s->object_size = s->size = size;
>                 s->align = align;
>                 s->ctor = ctor;
> +
> +             if (memcg_register_cache(memcg, s)) {
> +                     kmem_cache_free(kmem_cache, s);
> +                     err = -ENOMEM;
> +                     goto out_locked;
> +             }
> +
>               s->name = kstrdup(name, GFP_KERNEL);
>               if (!s->name) {
>                       kmem_cache_free(kmem_cache, s);
> @@ -135,10 +146,11 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t size, size_t align
>  
>               err = __kmem_cache_create(s, flags);
>               if (!err) {
> -
>                       s->refcount = 1;
> -                     list_add(&s->list, &slab_caches);
> -
> +                     if (!memcg)
> +                             list_add(&s->list, &slab_caches);
> +                     else
> +                             memcg_cache_list_add(memcg, s);
>               } else {
>                       kfree(s->name);
>                       kmem_cache_free(kmem_cache, s);
> @@ -166,6 +178,13 @@ out_locked:
>  
>       return s;
>  }
> +
> +struct kmem_cache *
> +kmem_cache_create(const char *name, size_t size, size_t align,
> +               unsigned long flags, void (*ctor)(void *))
> +{
> +     return kmem_cache_create_memcg(NULL, name, size, align, flags, ctor);
> +}
>  EXPORT_SYMBOL(kmem_cache_create);
>  
>  void kmem_cache_destroy(struct kmem_cache *s)
> @@ -180,6 +199,7 @@ void kmem_cache_destroy(struct kmem_cache *s)
>  
>                       list_del(&s->list);
>  
> +                     memcg_release_cache(s);
>                       kfree(s->name);
>                       kmem_cache_free(kmem_cache, s);
>               } else {
> diff --git a/mm/slub.c b/mm/slub.c
> index a34548e..05aefe2 100644
> --- a/mm/slub.c
> +++ b/mm/slub.c
> @@ -31,6 +31,7 @@
>  #include <linux/fault-inject.h>
>  #include <linux/stacktrace.h>
>  #include <linux/prefetch.h>
> +#include <linux/memcontrol.h>
>
>  #include <trace/events/kmem.h>
>
> @@ -3880,7 +3881,7 @@ static int slab_unmergeable(struct kmem_cache *s)
>         return 0;
>  }
>
> -static struct kmem_cache *find_mergeable(size_t size,
> +static struct kmem_cache *find_mergeable(struct mem_cgroup *memcg, size_t size,
>                 size_t align, unsigned long flags, const char *name,
>                 void (*ctor)(void *))
>  {
> @@ -3916,17 +3917,20 @@ static struct kmem_cache *find_mergeable(size_t size,
>                 if (s->size - size >= sizeof(void *))
>                         continue;
>
> +               if (!cache_match_memcg(s, memcg))
> +                       continue;
>                 return s;
>         }
>         return NULL;
>  }
>
> -struct kmem_cache *__kmem_cache_alias(const char *name, size_t size,
> -               size_t align, unsigned long flags, void (*ctor)(void *))
> +struct kmem_cache *
> +__kmem_cache_alias(struct mem_cgroup *memcg, const char *name, size_t size,
> +                  size_t align, unsigned long flags, void (*ctor)(void *))
>  {
>         struct kmem_cache *s;
>
> -       s = find_mergeable(size, align, flags, name, ctor);
> +       s = find_mergeable(memcg, size, align, flags, name, ctor);
>         if (s) {
>                 s->refcount++;
>                 /*
```

If your intention is that find_mergeable() works for memcg-slab-caches properly,
it cannot works properly with this code.
When memcg is not NULL, slab cache is only added to memcg's slab cache list.

find_mergeable() only interate on original-slab-cache list.
So memcg slab cache never be mergeable.