
Subject: Re: [PATCH 09/11] memcg: propagate kmem limiting information to children

Posted by [Tejun Heo](#) on Mon, 25 Jun 2012 18:29:07 GMT

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

Feeling like a nit pervert but..

On Mon, Jun 25, 2012 at 06:15:26PM +0400, Glauber Costa wrote:

```
> @@ -287,7 +287,11 @@ struct mem_cgroup {
>  * Should the accounting and control be hierarchical, per subtree?
>  */
>  bool use_hierarchy;
> - bool kmem_accounted;
> + /*
> +  * bit0: accounted by this cgroup
> +  * bit1: accounted by a parent.
> +  */
> + volatile unsigned long kmem_accounted;
```

Is the volatile declaration really necessary? Why is it necessary?
Why no comment explaining it?

```
> + #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> + static void mem_cgroup_update_kmem_limit(struct mem_cgroup *memcg, u64 val)
> + {
> +  struct mem_cgroup *iter;
> +
> +  mutex_lock(&set_limit_mutex);
> +  if (!test_and_set_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted) &&
> +      val != RESOURCE_MAX) {
> +
> +  /*
> +   * 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
> +   */
> +  static_key_slow_inc(&mem_cgroup_kmem_enabled_key);
> +
> +  if (!memcg->use_hierarchy)
> +      goto out;
> +
> +  for_each_mem_cgroup_tree(iter, memcg) {
> +      if (iter == memcg)
> +          continue;
> +      set_bit(KMEM_ACCOUNTED_PARENT, &iter->kmem_accounted);
> +  }
```

```

> +
> + } else if (test_and_clear_bit(KMEM_ACCOUNTED_THIS, &memcg->kmem_accounted)
> + && val == RESOURCE_MAX) {
> +
> + if (!memcg->use_hierarchy)
> + goto out;
> +
> + for_each_mem_cgroup_tree(iter, memcg) {
> + struct mem_cgroup *parent;

```

Blank line between decl and body please.

```

> + if (iter == memcg)
> + continue;
> + /*
> + * We should only have our parent bit cleared if none of
> + * our parents are accounted. The transversal order of

```

^ type

```

> + * our iter function forces us to always look at the
> + * parents.

```

Also, it's okay here but the text filling in comments and patch descriptions tend to be quite inconsistent. If you're on emacs, alt-q is your friend and I'm sure vim can do text filling pretty nicely too.

```

> + */
> + parent = parent_mem_cgroup(iter);
> + while (parent && (parent != memcg)) {
> + if (test_bit(KMEM_ACCOUNTED_THIS, &parent->kmem_accounted))
> + goto noclear;
> +
> + parent = parent_mem_cgroup(parent);
> + }

```

Better written in for (;)? Also, if we're breaking on parent == memcg, can we ever hit NULL parent in the above loop?

```

> + clear_bit(KMEM_ACCOUNTED_PARENT, &iter->kmem_accounted);
> +noclear:
> + continue;
> + }
> + }
> +out:
> + mutex_unlock(&set_limit_mutex);

```

Can we please branch on val != RECOURSE_MAX first? I'm not even sure

whether the above conditionals are correct. If the user updates an existing kmem limit, the first `test_and_set_bit()` returns non-zero, so the code proceeds onto clearing `KMEM_ACCOUNTED_THIS`, which succeeds but `val == RESOURCE_MAX` fails so it doesn't do anything. If the user changes it again, it will set `ACCOUNTED_THIS` again. So, changing an existing kmem limit toggles `KMEM_ACCOUNTED_THIS`, which just seems wacky to me.

Thanks.

--

tejun
