

---

Subject: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Tue, 18 Sep 2012 14:04:01 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

This patch adds the basic infrastructure for the accounting of the slab caches. To control that, the following files are created:

- \* memory.kmem.usage\_in\_bytes
- \* memory.kmem.limit\_in\_bytes
- \* memory.kmem.failcnt
- \* memory.kmem.max\_usage\_in\_bytes

They have the same meaning of their user memory counterparts. They reflect the state of the "kmem" res\_counter.

The code is not enabled until a limit is set. This can be tested by the flag "kmem\_accounted". This means that after the patch is applied, no behavioral changes exists for whoever is still using memcg to control their memory usage.

We always account to both user and kernel resource\_counters. This effectively means that an independent kernel limit is in place when the limit is set to a lower value than the user memory. A equal or higher value means that the user limit will always hit first, meaning that kmem is effectively unlimited.

People who want to track kernel memory but not limit it, can set this limit to a very high number (like RESOURCE\_MAX - 1page - that no one will ever hit, or equal to the user memory)

Signed-off-by: Glauber Costa <glommer@parallels.com>  
CC: Michal Hocko <mhocko@suse.cz>  
CC: Johannes Weiner <hannes@cmpxchg.org>  
Acked-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

---

mm/memcontrol.c | 64

+++++  
1 file changed, 63 insertions(+), 1 deletion(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c

index d6ad138..f3fd354 100644

--- a/mm/memcontrol.c

+++ b/mm/memcontrol.c

@@ -265,6 +265,10 @@ struct mem\_cgroup {  
};

/\*  
+ \* the counter to account for kernel memory usage.

```

+ */
+ struct res_counter kmem;
+ /*
+  * Per cgroup active and inactive list, similar to the
+  * per zone LRU lists.
+  */
@@ -279,6 +283,7 @@ struct mem_cgroup {
+  * Should the accounting and control be hierarchical, per subtree?
+  */
+ bool use_hierarchy;
+ bool kmem_accounted;

+ bool oom_lock;
+ atomic_t under_oom;
@@ -389,6 +394,7 @@ enum res_type {
+ _MEM,
+ _MEMSWAP,
+ _OOM_TYPE,
+ _KMEM,
+ };

#define MEMFILE_PRIVATE(x, val) ((x) << 16 | (val))
@@ -1439,6 +1445,10 @@ done:
+ res_counter_read_u64(&memcg->memsw, RES_USAGE) >> 10,
+ res_counter_read_u64(&memcg->memsw, RES_LIMIT) >> 10,
+ res_counter_read_u64(&memcg->memsw, RES_FAILCNT));
+ printk(KERN_INFO "kmem: usage %lluB, limit %lluB, failcnt %llu\n",
+ res_counter_read_u64(&memcg->kmem, RES_USAGE) >> 10,
+ res_counter_read_u64(&memcg->kmem, RES_LIMIT) >> 10,
+ res_counter_read_u64(&memcg->kmem, RES_FAILCNT));
+ }

/*
@@ -3946,6 +3956,9 @@ static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype *cft,
+ else
+ val = res_counter_read_u64(&memcg->memsw, name);
+ break;
+ case _KMEM:
+ val = res_counter_read_u64(&memcg->kmem, name);
+ break;
+ default:
+ BUG();
+ }
@@ -3984,8 +3997,18 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
+ break;
+ if (type == _MEM)
+ ret = mem_cgroup_resize_limit(memcg, val);
- else

```

```

+ else if (type == _MEMSWAP)
    ret = mem_cgroup_resize_memsw_limit(memcg, val);
+ else if (type == _KMEM) {
+     ret = res_counter_set_limit(&memcg->kmem, val);
+     if (ret)
+         break;
+
+     /* For simplicity, we won't allow this to be disabled */
+     if (!memcg->kmem_accounted && val != RESOURCE_MAX)
+         memcg->kmem_accounted = true;
+     } else
+     return -EINVAL;
    break;
    case RES_SOFT_LIMIT:
        ret = res_counter_memparse_write_strategy(buffer, &val);
@@ -4051,12 +4074,16 @@ static int mem_cgroup_reset(struct cgroup *cont, unsigned int
event)
    case RES_MAX_USAGE:
        if (type == _MEM)
            res_counter_reset_max(&memcg->res);
+     else if (type == _KMEM)
+         res_counter_reset_max(&memcg->kmem);
+     else
+         res_counter_reset_max(&memcg->memsw);
        break;
    case RES_FAILCNT:
        if (type == _MEM)
            res_counter_reset_failcnt(&memcg->res);
+     else if (type == _KMEM)
+         res_counter_reset_failcnt(&memcg->kmem);
+     else
+         res_counter_reset_failcnt(&memcg->memsw);
        break;
@@ -4618,6 +4645,33 @@ static int mem_cgroup_oom_control_write(struct cgroup *cgrp,
}

#ifdef CONFIG_MEMCG_KMEM
+static struct cftype kmem_cgroup_files[] = {
+ {
+     .name = "kmem.limit_in_bytes",
+     .private = MEMFILE_PRIVATE(_KMEM, RES_LIMIT),
+     .write_string = mem_cgroup_write,
+     .read = mem_cgroup_read,
+ },
+ {
+     .name = "kmem.usage_in_bytes",
+     .private = MEMFILE_PRIVATE(_KMEM, RES_USAGE),
+     .read = mem_cgroup_read,

```

```

+ },
+ {
+ .name = "kmem.failcnt",
+ .private = MEMFILE_PRIVATE(_KMEM, RES_FAILCNT),
+ .trigger = mem_cgroup_reset,
+ .read = mem_cgroup_read,
+ },
+ {
+ .name = "kmem.max_usage_in_bytes",
+ .private = MEMFILE_PRIVATE(_KMEM, RES_MAX_USAGE),
+ .trigger = mem_cgroup_reset,
+ .read = mem_cgroup_read,
+ },
+ {},
+};
+
static int memcg_init_kmem(struct mem_cgroup *memcg, struct cgroup_subsys *ss)
{
    return mem_cgroup_sockets_init(memcg, ss);
@@ -4961,6 +5015,12 @@ mem_cgroup_create(struct cgroup *cont)
    int cpu;
    enable_swap_cgroup();
    parent = NULL;
+
+ #ifdef CONFIG_MEMCG_KMEM
+ WARN_ON(cgroup_add_cftypes(&mem_cgroup_subsys,
+     kmem_cgroup_files));
+ #endif
+
    if (mem_cgroup_soft_limit_tree_init())
        goto free_out;
    root_mem_cgroup = memcg;
@@ -4979,6 +5039,7 @@ mem_cgroup_create(struct cgroup *cont)
    if (parent && parent->use_hierarchy) {
        res_counter_init(&memcg->res, &parent->res);
        res_counter_init(&memcg->memsw, &parent->memsw);
+ res_counter_init(&memcg->kmem, &parent->kmem);
    /*
     * We increment refcnt of the parent to ensure that we can
     * safely access it on res_counter_charge/uncharge.
@@ -4989,6 +5050,7 @@ mem_cgroup_create(struct cgroup *cont)
    } else {
        res_counter_init(&memcg->res, NULL);
        res_counter_init(&memcg->memsw, NULL);
+ res_counter_init(&memcg->kmem, NULL);
    }
    memcg->last_scanned_node = MAX_NUMNODES;
    INIT_LIST_HEAD(&memcg->oom_notify);

```

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure

Posted by [Tejun Heo](#) on Fri, 21 Sep 2012 16:34:04 GMT

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

---

On Tue, Sep 18, 2012 at 06:04:01PM +0400, Glauber Costa wrote:

```
> #ifdef CONFIG_MEMCG_KMEM
> +static struct cftype kmem_cgroup_files[] = {
> + {
> + .name = "kmem.limit_in_bytes",
> + .private = MEMFILE_PRIVATE(_KMEM, RES_LIMIT),
> + .write_string = mem_cgroup_write,
> + .read = mem_cgroup_read,
> + },
> + {
> + .name = "kmem.usage_in_bytes",
> + .private = MEMFILE_PRIVATE(_KMEM, RES_USAGE),
> + .read = mem_cgroup_read,
> + },
> + {
> + .name = "kmem.failcnt",
> + .private = MEMFILE_PRIVATE(_KMEM, RES_FAILCNT),
> + .trigger = mem_cgroup_reset,
> + .read = mem_cgroup_read,
> + },
> + {
> + .name = "kmem.max_usage_in_bytes",
> + .private = MEMFILE_PRIVATE(_KMEM, RES_MAX_USAGE),
> + .trigger = mem_cgroup_reset,
> + .read = mem_cgroup_read,
> + },
> + {},
> +};
> +
> static int memcg_init_kmem(struct mem_cgroup *memcg, struct cgroup_subsys *ss)
> {
>     return mem_cgroup_sockets_init(memcg, ss);
> @@ -4961,6 +5015,12 @@ mem_cgroup_create(struct cgroup *cont)
>     int cpu;
>     enable_swap_cgroup();
>     parent = NULL;
> +
> + #ifdef CONFIG_MEMCG_KMEM
> +     WARN_ON(cgroup_add_cftypes(&mem_cgroup_subsys,
> +         kmem_cgroup_files));
```

> +#endif  
> +

Why not just make it part of mem\_cgroup\_files[]?

Thanks.

--  
tejun

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Mon, 24 Sep 2012 08:09:53 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

```
>> +
>> +#ifdef CONFIG_MEMCG_KMEM
>> + WARN_ON(cgroup_add_cftypes(&mem_cgroup_subsys,
>> +      kmem_cgroup_files));
>> +#endif
>> +
>
> Why not just make it part of mem_cgroup_files[]?
>
> Thanks.
>
```

Done.

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Michal Hocko](#) on Wed, 26 Sep 2012 14:03:47 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue 18-09-12 18:04:01, Glauber Costa wrote:

```
> This patch adds the basic infrastructure for the accounting of the slab
> caches. To control that, the following files are created:
>
> * memory.kmem.usage_in_bytes
> * memory.kmem.limit_in_bytes
> * memory.kmem.failcnt
> * memory.kmem.max_usage_in_bytes
>
> They have the same meaning of their user memory counterparts. They
> reflect the state of the "kmem" res_counter.
>
> The code is not enabled until a limit is set.
```

"Per cgroup slab memory accounting is not enabled until a limit is set for the group. Once the limit is set the accounting cannot be disabled such a group."

Better?

> This can be tested by the flag "kmem\_accounted".

Sounds as if it could be done from userspace (because you were talking about an user interface) which it cannot and we do not see it in this patch because it is not used anywhere. So please be more specific.

> This means that after the patch is applied, no behavioral changes  
> exists for whoever is still using memcg to control their memory usage.  
>  
> We always account to both user and kernel resource\_counters.

This is in contradiction with your claim that there is no behavioral change for memcg users. Please clarify when we use u and when u+k accounting.

"

There is no behavioral change if the kmem accounting is turned off for memcg users but when there is a kmem.limit\_in\_bytes is set then the memory.usage\_in\_bytes will include both user and kmem memory.

"

> This  
> effectively means that an independent kernel limit is in place when the  
> limit is set to a lower value than the user memory. A equal or higher  
> value means that the user limit will always hit first, meaning that kmem  
> is effectively unlimited.  
>  
> People who want to track kernel memory but not limit it, can set this  
> limit to a very high number (like RESOURCE\_MAX - 1page - that no one  
> will ever hit, or equal to the user memory)  
>  
> Signed-off-by: Glauber Costa <glommer@parallels.com>  
> CC: Michal Hocko <mhocko@suse.cz>  
> CC: Johannes Weiner <hannes@cmpxchg.org>  
> Acked-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>  
> ---  
> mm/memcontrol.c | 64  
+++++  
> 1 file changed, 63 insertions(+), 1 deletion(-)  
>  
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c  
> index d6ad138..f3fd354 100644

```

> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -265,6 +265,10 @@ struct mem_cgroup {
> };
>
> /*
> + * the counter to account for kernel memory usage.
> + */
> + struct res_counter kmem;
> + /*
> + * Per cgroup active and inactive list, similar to the
> + * per zone LRU lists.
> + */
> @@ -279,6 +283,7 @@ struct mem_cgroup {
> + * Should the accounting and control be hierarchical, per subtree?
> + */
> + bool use_hierarchy;
> + bool kmem_accounted;
>
> + bool oom_lock;
> + atomic_t under_oom;
> @@ -389,6 +394,7 @@ enum res_type {
> + _MEM,
> + _MEMSWAP,
> + _OOM_TYPE,
> + _KMEM,
> };
>
> #define MEMFILE_PRIVATE(x, val) ((x) << 16 | (val))
> @@ -1439,6 +1445,10 @@ done:
> + res_counter_read_u64(&memcg->memsw, RES_USAGE) >> 10,
> + res_counter_read_u64(&memcg->memsw, RES_LIMIT) >> 10,
> + res_counter_read_u64(&memcg->memsw, RES_FAILCNT));
> + printk(KERN_INFO "kmem: usage %lluKB, limit %lluKB, failcnt %llu\n",
> + res_counter_read_u64(&memcg->kmem, RES_USAGE) >> 10,
> + res_counter_read_u64(&memcg->kmem, RES_LIMIT) >> 10,
> + res_counter_read_u64(&memcg->kmem, RES_FAILCNT));
> }
>
> /*
> @@ -3946,6 +3956,9 @@ static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype
> *cft,
> + else
> + val = res_counter_read_u64(&memcg->memsw, name);
> + break;
> + case _KMEM:
> + val = res_counter_read_u64(&memcg->kmem, name);
> + break;

```



```

> default:
> BUG();
> }
> @@ -3984,8 +3997,18 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
> break;
> if (type == _MEM)
> ret = mem_cgroup_resize_limit(memcg, val);
> - else
> + else if (type == _MEMSWAP)
> ret = mem_cgroup_resize_memsw_limit(memcg, val);
> + else if (type == _KMEM) {
> + ret = res_counter_set_limit(&memcg->kmem, val);
> + if (ret)
> + break;
> +
> + /* For simplicity, we won't allow this to be disabled */
> + if (!memcg->kmem_accounted && val != RESOURCE_MAX)
> + memcg->kmem_accounted = true;
> + } else
> + return -EINVAL;
> break;
> case RES_SOFT_LIMIT:
> ret = res_counter_memparse_write_strategy(buffer, &val);
> @@ -4051,12 +4074,16 @@ static int mem_cgroup_reset(struct cgroup *cont, unsigned int
event)
> case RES_MAX_USAGE:
> if (type == _MEM)
> res_counter_reset_max(&memcg->res);
> + else if (type == _KMEM)
> + res_counter_reset_max(&memcg->kmem);
> else
> res_counter_reset_max(&memcg->memsw);
> break;
> case RES_FAILCNT:
> if (type == _MEM)
> res_counter_reset_failcnt(&memcg->res);
> + else if (type == _KMEM)
> + res_counter_reset_failcnt(&memcg->kmem);
> else
> res_counter_reset_failcnt(&memcg->memsw);
> break;
> @@ -4618,6 +4645,33 @@ static int mem_cgroup_oom_control_write(struct cgroup *cgrp,
> }
>
> #ifdef CONFIG_MEMCG_KMEM

```

Some things are guarded CONFIG\_MEMCG\_KMEM but some are not (e.g. struct mem\_cgroup.kmem). I do understand you want to keep ifdefs on the leash

but we should clean this up one day.

```
> +static struct cftype kmem_cgroup_files[] = {
> + {
> + .name = "kmem.limit_in_bytes",
> + .private = MEMFILE_PRIVATE(_KMEM, RES_LIMIT),
> + .write_string = mem_cgroup_write,
> + .read = mem_cgroup_read,
> + },
> + {
> + .name = "kmem.usage_in_bytes",
> + .private = MEMFILE_PRIVATE(_KMEM, RES_USAGE),
> + .read = mem_cgroup_read,
> + },
> + {
> + .name = "kmem.failcnt",
> + .private = MEMFILE_PRIVATE(_KMEM, RES_FAILCNT),
> + .trigger = mem_cgroup_reset,
> + .read = mem_cgroup_read,
> + },
> + {
> + .name = "kmem.max_usage_in_bytes",
> + .private = MEMFILE_PRIVATE(_KMEM, RES_MAX_USAGE),
> + .trigger = mem_cgroup_reset,
> + .read = mem_cgroup_read,
> + },
> + {},
> +};
> +
> static int memcg_init_kmem(struct mem_cgroup *memcg, struct cgroup_subsys *ss)
> {
>     return mem_cgroup_sockets_init(memcg, ss);
> @@ -4961,6 +5015,12 @@ mem_cgroup_create(struct cgroup *cont)
>     int cpu;
>     enable_swap_cgroup();
>     parent = NULL;
> +
> +#ifdef CONFIG_MEMCG_KMEM
> + WARN_ON(cgroup_add_cftypes(&mem_cgroup_subsys,
> +     kmem_cgroup_files));
> +#endif
> +
>     if (mem_cgroup_soft_limit_tree_init())
>         goto free_out;
>     root_mem_cgroup = memcg;
> @@ -4979,6 +5039,7 @@ mem_cgroup_create(struct cgroup *cont)
>     if (parent && parent->use_hierarchy) {
>         res_counter_init(&memcg->res, &parent->res);
```

```
> res_counter_init(&memcg->memsw, &parent->memsw);
> + res_counter_init(&memcg->kmem, &parent->kmem);
```

Haven't we already discussed that a new memcg should inherit kmem\_accounted from its parent for use\_hierarchy?

Say we have

```
root
|
A (kmem_accounted = 1, use_hierarchy = 1)
\
  B (kmem_accounted = 0)
  \
    C (kmem_accounted = 1)
```

B finds itself in an awkward situation because it doesn't want to account u+k but it ends up doing so because C.

--

Michal Hocko  
SUSE Labs

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure

Posted by [Glauber Costa](#) on Wed, 26 Sep 2012 14:33:10 GMT

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

---

On 09/26/2012 06:03 PM, Michal Hocko wrote:

> On Tue 18-09-12 18:04:01, Glauber Costa wrote:

>> This patch adds the basic infrastructure for the accounting of the slab  
>> caches. To control that, the following files are created:

```
>>
>> * memory.kmem.usage_in_bytes
>> * memory.kmem.limit_in_bytes
>> * memory.kmem.failcnt
>> * memory.kmem.max_usage_in_bytes
>>
```

>> They have the same meaning of their user memory counterparts. They  
>> reflect the state of the "kmem" res\_counter.

>

>> The code is not enabled until a limit is set.

>

> "Per cgroup slab memory accounting is not enabled until a limit is set  
> for the group. Once the limit is set the accounting cannot be disabled  
> such a group."

>

> Better?

>

>> This can be tested by the flag "kmem\_accounted".

>

> Sounds as if it could be done from userspace (because you were talking  
> about an user interface) which it cannot and we do not see it in this  
> patch because it is not used anywhere. So please be more specific.  
>  
>> This means that after the patch is applied, no behavioral changes  
>> exists for whoever is still using memcg to control their memory usage.  
>>  
>> We always account to both user and kernel resource\_counters.  
>  
> This is in contradiction with your claim that there is no behavioral  
> change for memcg users. Please clarify when we use u and when u+k  
> accounting.  
> "  
> There is no behavioral change if the kmem accounting is turned off for  
> memcg users but when there is a kmem.limit\_in\_bytes is set then the  
> memory.usage\_in\_bytes will include both user and kmem memory.  
> "  
>  
>> This  
>> effectively means that an independent kernel limit is in place when the  
>> limit is set to a lower value than the user memory. A equal or higher  
>> value means that the user limit will always hit first, meaning that kmem  
>> is effectively unlimited.  
>>  
>> People who want to track kernel memory but not limit it, can set this  
>> limit to a very high number (like RESOURCE\_MAX - 1page - that no one  
>> will ever hit, or equal to the user memory)  
>>  
>> Signed-off-by: Glauber Costa <glommer@parallels.com>  
>> CC: Michal Hocko <mhocko@suse.cz>  
>> CC: Johannes Weiner <hannes@cmpxchg.org>  
>> Acked-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>  
>> ---  
>> mm/memcontrol.c | 64  
+++++  
>> 1 file changed, 63 insertions(+), 1 deletion(-)  
>>  
>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c  
>> index d6ad138..f3fd354 100644  
>> --- a/mm/memcontrol.c  
>> +++ b/mm/memcontrol.c  
>> @@ -265,6 +265,10 @@ struct mem\_cgroup {  
>> };  
>>  
>> /\*  
>> + \* the counter to account for kernel memory usage.  
>> + \*/  
>> + struct res\_counter kmem;

```

>> + /*
>>  * Per cgroup active and inactive list, similar to the
>>  * per zone LRU lists.
>>  */
>> @@ -279,6 +283,7 @@ struct mem_cgroup {
>>  * Should the accounting and control be hierarchical, per subtree?
>>  */
>>  bool use_hierarchy;
>> + bool kmem_accounted;
>>
>>  bool oom_lock;
>>  atomic_t under_oom;
>> @@ -389,6 +394,7 @@ enum res_type {
>>  _MEM,
>>  _MEMSWAP,
>>  _OOM_TYPE,
>> + _KMEM,
>> };
>>
>> #define MEMFILE_PRIVATE(x, val) ((x) << 16 | (val))
>> @@ -1439,6 +1445,10 @@ done:
>>  res_counter_read_u64(&memcg->memsw, RES_USAGE) >> 10,
>>  res_counter_read_u64(&memcg->memsw, RES_LIMIT) >> 10,
>>  res_counter_read_u64(&memcg->memsw, RES_FAILCNT));
>> + printk(KERN_INFO "kmem: usage %lluKB, limit %lluKB, failcnt %llu\n",
>> + res_counter_read_u64(&memcg->kmem, RES_USAGE) >> 10,
>> + res_counter_read_u64(&memcg->kmem, RES_LIMIT) >> 10,
>> + res_counter_read_u64(&memcg->kmem, RES_FAILCNT));
>> }
>>
>> /*
>> @@ -3946,6 +3956,9 @@ static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype
>> *cft,
>>  else
>>  val = res_counter_read_u64(&memcg->memsw, name);
>>  break;
>> + case _KMEM:
>> + val = res_counter_read_u64(&memcg->kmem, name);
>> + break;
>>  default:
>>  BUG();
>>  }
>> @@ -3984,8 +3997,18 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
>>  break;
>>  if (type == _MEM)
>>  ret = mem_cgroup_resize_limit(memcg, val);
>> - else
>> + else if (type == _MEMSWAP)

```

```

>> ret = mem_cgroup_resize_memsw_limit(memcg, val);
>> + else if (type == _KMEM) {
>> + ret = res_counter_set_limit(&memcg->kmem, val);
>> + if (ret)
>> + break;
>> +
>> + /* For simplicity, we won't allow this to be disabled */
>> + if (!memcg->kmem_accounted && val != RESOURCE_MAX)
>> + memcg->kmem_accounted = true;
>> + } else
>> + return -EINVAL;
>> break;
>> case RES_SOFT_LIMIT:
>> ret = res_counter_memparse_write_strategy(buffer, &val);
>> @@ -4051,12 +4074,16 @@ static int mem_cgroup_reset(struct cgroup *cont, unsigned int
event)
>> case RES_MAX_USAGE:
>> if (type == _MEM)
>> res_counter_reset_max(&memcg->res);
>> + else if (type == _KMEM)
>> + res_counter_reset_max(&memcg->kmem);
>> else
>> res_counter_reset_max(&memcg->memsw);
>> break;
>> case RES_FAILCNT:
>> if (type == _MEM)
>> res_counter_reset_failcnt(&memcg->res);
>> + else if (type == _KMEM)
>> + res_counter_reset_failcnt(&memcg->kmem);
>> else
>> res_counter_reset_failcnt(&memcg->memsw);
>> break;
>> @@ -4618,6 +4645,33 @@ static int mem_cgroup_oom_control_write(struct cgroup *cgrp,
>> }
>>
>> #ifdef CONFIG_MEMCG_KMEM
>
> Some things are guarded CONFIG_MEMCG_KMEM but some are not (e.g. struct
> mem_cgroup.kmem). I do understand you want to keep ifdefs on the leash
> but we should clean this up one day.
>
>> +static struct cftype kmem_cgroup_files[] = {
>> + {
>> + .name = "kmem.limit_in_bytes",
>> + .private = MEMFILE_PRIVATE(_KMEM, RES_LIMIT),
>> + .write_string = mem_cgroup_write,
>> + .read = mem_cgroup_read,
>> + },

```

```

>> + {
>> + .name = "kmem.usage_in_bytes",
>> + .private = MEMFILE_PRIVATE(_KMEM, RES_USAGE),
>> + .read = mem_cgroup_read,
>> + },
>> + {
>> + .name = "kmem.failcnt",
>> + .private = MEMFILE_PRIVATE(_KMEM, RES_FAILCNT),
>> + .trigger = mem_cgroup_reset,
>> + .read = mem_cgroup_read,
>> + },
>> + {
>> + .name = "kmem.max_usage_in_bytes",
>> + .private = MEMFILE_PRIVATE(_KMEM, RES_MAX_USAGE),
>> + .trigger = mem_cgroup_reset,
>> + .read = mem_cgroup_read,
>> + },
>> + {},
>> +};
>> +
>> static int memcg_init_kmem(struct mem_cgroup *memcg, struct cgroup_subsys *ss)
>> {
>>     return mem_cgroup_sockets_init(memcg, ss);
>> @@ -4961,6 +5015,12 @@ mem_cgroup_create(struct cgroup *cont)
>>     int cpu;
>>     enable_swap_cgroup();
>>     parent = NULL;
>> +
>> + #ifdef CONFIG_MEMCG_KMEM
>> +     WARN_ON(cgroup_add_cftypes(&mem_cgroup_subsys,
>> +         kmem_cgroup_files));
>> + #endif
>> +
>>     if (mem_cgroup_soft_limit_tree_init())
>>         goto free_out;
>>     root_mem_cgroup = memcg;
>> @@ -4979,6 +5039,7 @@ mem_cgroup_create(struct cgroup *cont)
>>     if (parent && parent->use_hierarchy) {
>>         res_counter_init(&memcg->res, &parent->res);
>>         res_counter_init(&memcg->memsw, &parent->memsw);
>> +     res_counter_init(&memcg->kmem, &parent->kmem);
>> +
> Haven't we already discussed that a new memcg should inherit kmem_accounted
> from its parent for use_hierarchy?
> Say we have
> root
> |
> A (kmem_accounted = 1, use_hierarchy = 1)

```

```

> \
> B (kmem_accounted = 0)
> \
> C (kmem_accounted = 1)
>
> B find's itself in an awkward situation because it doesn't want to
> account u+k but it ends up doing so because C.
>

```

Ok, I haven't updated it here. But that should be taken care of in the lifecycle patch.

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
 Posted by [Michal Hocko](#) on Wed, 26 Sep 2012 16:01:27 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

```

On Wed 26-09-12 18:33:10, Glauber Costa wrote:
> On 09/26/2012 06:03 PM, Michal Hocko wrote:
> > On Tue 18-09-12 18:04:01, Glauber Costa wrote:
[...
> > @@ -4961,6 +5015,12 @@ mem_cgroup_create(struct cgroup *cont)
> >     int cpu;
> >     enable_swap_cgroup();
> >     parent = NULL;
> > +
> > + #ifdef CONFIG_MEMCG_KMEM
> > + WARN_ON(cgroup_add_cftypes(&mem_cgroup_subsys,
> > +     kmem_cgroup_files));
> > + #endif
> > +
> >     if (mem_cgroup_soft_limit_tree_init())
> >         goto free_out;
> >     root_mem_cgroup = memcg;
> > @@ -4979,6 +5039,7 @@ mem_cgroup_create(struct cgroup *cont)
> >     if (parent && parent->use_hierarchy) {
> >         res_counter_init(&memcg->res, &parent->res);
> >         res_counter_init(&memcg->memsw, &parent->memsw);
> > + res_counter_init(&memcg->kmem, &parent->kmem);
> >
> > Haven't we already discussed that a new memcg should inherit kmem_accounted
> > from its parent for use_hierarchy?
> > Say we have
> > root
> > |
> > A (kmem_accounted = 1, use_hierarchy = 1)
> > \
> > B (kmem_accounted = 0)

```



> > \

> > C (kmem\_accounted = 1)

> >

> > B find's itself in an awkward situation becuae it doesn't want to

> > account u+k but it ends up doing so becuae C.

> >

>

> Ok, I haven't updated it here. But that should be taken care of in the

> lifecycle patch.

I am not sure which patch you are thinking about but I would prefer to have it here because it is safe wrt. races and it is more obvious as well.

--

Michal Hocko  
SUSE Labs

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Wed, 26 Sep 2012 16:36:48 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello, Michal, Glauber.

On Wed, Sep 26, 2012 at 04:03:47PM +0200, Michal Hocko wrote:

> Haven't we already discussed that a new memcg should inherit kmem\_accounted

> from its parent for use\_hierarchy?

> Say we have

> root

> |

> A (kmem\_accounted = 1, use\_hierarchy = 1)

> \

> B (kmem\_accounted = 0)

> \

> C (kmem\_accounted = 1)

>

> B find's itself in an awkward situation becuae it doesn't want to

> account u+k but it ends up doing so becuae C.

Do we really want this level of flexibility? What's wrong with a global switch at the root? I'm not even sure we want this to be optional at all. The only reason I can think of is that it might screw up some configurations in use which are carefully crafted to suit userland-only usage but for that isn't what we need a transition plan rather than another ultra flexible config option that not many really understand the implication of?

In the same vein, do we really need both `.kmem_accounted` and `config` option? If someone is turning on MEMCG, just include `kmem` accounting.

Thanks.

--  
tejun

---

---

Subject: Re: [PATCH v3 04/13] `kmem` accounting basic infrastructure  
Posted by [Glauber Costa](#) on Wed, 26 Sep 2012 17:34:00 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/26/2012 08:01 PM, Michal Hocko wrote:  
> On Wed 26-09-12 18:33:10, Glauber Costa wrote:  
>> On 09/26/2012 06:03 PM, Michal Hocko wrote:  
>>> On Tue 18-09-12 18:04:01, Glauber Costa wrote:  
> [...]  
>>>> @@ -4961,6 +5015,12 @@ mem\_cgroup\_create(struct cgroup \*cont)  
>>>> int cpu;  
>>>> enable\_swap\_cgroup();  
>>>> parent = NULL;  
>>>> +  
>>>> + #ifdef CONFIG\_MEMCG\_KMEM  
>>>> + WARN\_ON(cgroup\_add\_cftypes(&mem\_cgroup\_subsys,  
>>>> + kmem\_cgroup\_files));  
>>>> + #endif  
>>>> +  
>>>> if (mem\_cgroup\_soft\_limit\_tree\_init())  
>>>> goto free\_out;  
>>>> root\_mem\_cgroup = memcg;  
>>>> @@ -4979,6 +5039,7 @@ mem\_cgroup\_create(struct cgroup \*cont)  
>>>> if (parent && parent->use\_hierarchy) {  
>>>> res\_counter\_init(&memcg->res, &parent->res);  
>>>> res\_counter\_init(&memcg->memsw, &parent->memsw);  
>>>> + res\_counter\_init(&memcg->kmem, &parent->kmem);  
>>>>  
>>> Haven't we already discussed that a new `memcg` should inherit `kmem_accounted`  
>>> from its parent for `use_hierarchy`?  
>>> Say we have  
>>> root  
>>> |  
>>> A (`kmem_accounted` = 1, `use_hierarchy` = 1)  
>>> \  
>>> B (`kmem_accounted` = 0)  
>>> \  
>>> C (`kmem_accounted` = 1)  
>>>

>>> B find's itself in an awkward situation becuase it doesn't want to  
>>> account u+k but it ends up doing so becuase C.  
>>>  
>>  
>> Ok, I haven't updated it here. But that should be taken care of in the  
>> lifecycle patch.  
>  
> I am not sure which patch you are thinking about but I would prefer to  
> have it here because it is safe wrt. races and it is more obvious as  
> well.  
>

The patch where I make kmem\_accounted into a bitfield. So any code here  
will eventually disappear.

But BTW, I am not saying I won't update the patch - I like that all  
patches work and make sense in their own, I am just saying that I forgot  
to update this patch, because I added the code in its final version to  
the end and then squashed it.

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Wed, 26 Sep 2012 17:36:36 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/26/2012 08:36 PM, Tejun Heo wrote:

> Hello, Michal, Glauber.  
>  
> On Wed, Sep 26, 2012 at 04:03:47PM +0200, Michal Hocko wrote:  
>> Haven't we already discussed that a new memcg should inherit kmem\_accounted  
>> from its parent for use\_hierarchy?  
>> Say we have  
>> root  
>> |  
>> A (kmem\_accounted = 1, use\_hierarchy = 1)  
>> \  
>> B (kmem\_accounted = 0)  
>> \  
>> C (kmem\_accounted = 1)  
>>  
>> B find's itself in an awkward situation becuase it doesn't want to  
>> account u+k but it ends up doing so becuase C.  
>  
> Do we really want this level of flexibility? What's wrong with a  
> global switch at the root? I'm not even sure we want this to be  
> optional at all. The only reason I can think of is that it might  
> screw up some configurations in use which are carefully crafted to  
> suit userland-only usage but for that isn't what we need a transition

> plan rather than another ultra flexible config option that not many  
> really understand the implication of?  
>  
> In the same vein, do we really need both .kmem\_accounted and config  
> option? If someone is turning on MEMCG, just include kmem accounting.  
>

Yes, we do.

This was discussed multiple times. Our interest is to preserve existing deployed setup, that were tuned in a world where kmem didn't exist. Because we also feed kmem to the user counter, this may very well disrupt their setup.

User memory, unlike kernel memory, may very well be totally in control of the userspace application, so it is not unreasonable to believe that extra pages appearing in a new kernel version may break them.

It is actually a much worse compatibility problem than flipping hierarchy, in comparison

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Wed, 26 Sep 2012 17:44:32 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello, Glauber.

On Wed, Sep 26, 2012 at 10:36 AM, Glauber Costa <glommer@parallels.com> wrote:

> This was discussed multiple times. Our interest is to preserve existing  
> deployed setup, that were tuned in a world where kmem didn't exist.  
> Because we also feed kmem to the user counter, this may very well  
> disrupt their setup.

So, that can be served by .kmem\_accounted at root, no?

> User memory, unlike kernel memory, may very well be totally in control  
> of the userspace application, so it is not unreasonable to believe that  
> extra pages appearing in a new kernel version may break them.  
>  
> It is actually a much worse compatibility problem than flipping  
> hierarchy, in comparison

Again, what's wrong with one switch at the root?

Thanks.

--

tejun

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure

Posted by [Glauber Costa](#) on Wed, 26 Sep 2012 17:53:09 GMT

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

---

On 09/26/2012 09:44 PM, Tejun Heo wrote:

> Hello, Glauber.

>

> On Wed, Sep 26, 2012 at 10:36 AM, Glauber Costa <glommer@parallels.com> wrote:

>> This was discussed multiple times. Our interest is to preserve existing

>> deployed setup, that were tuned in a world where kmem didn't exist.

>> Because we also feed kmem to the user counter, this may very well

>> disrupt their setup.

>

> So, that can be served by .kmem\_accounted at root, no?

>

>> User memory, unlike kernel memory, may very well be totally in control

>> of the userspace application, so it is not unreasonable to believe that

>> extra pages appearing in a new kernel version may break them.

>>

>> It is actually a much worse compatibility problem than flipping

>> hierarchy, in comparison

>

> Again, what's wrong with one switch at the root?

>

I understand your trauma about over flexibility, and you know I share of it. But I don't think there is any need to cap it here. Given kmem accounted is perfectly hierarchical, and there seem to be plenty of people who only care about user memory, I see no reason to disallow a mixed use case here.

I must say that for my particular use case, enabling it unconditionally would just work, so it is not that what I have in mind.

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure

Posted by [Tejun Heo](#) on Wed, 26 Sep 2012 18:01:24 GMT

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

---

Hello,

On Wed, Sep 26, 2012 at 09:53:09PM +0400, Glauber Costa wrote:

> I understand your trauma about over flexibility, and you know I share of

> it. But I don't think there is any need to cap it here. Given kmem

> accounted is perfectly hierarchical, and there seem to be plenty of  
> people who only care about user memory, I see no reason to disallow a  
> mixed use case here.  
>  
> I must say that for my particular use case, enabling it unconditionally  
> would just work, so it is not that what I have in mind.

So, I'm not gonna go as far as pushing for enabling it unconditionally  
but would really like to hear why it's necessary to make it per node  
instead of one global switch. Maybe it has already been discussed to  
hell and back. Care to summarize / point me to it?

Thanks.

--  
tejun

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Wed, 26 Sep 2012 18:56:09 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/26/2012 10:01 PM, Tejun Heo wrote:

> Hello,  
>  
> On Wed, Sep 26, 2012 at 09:53:09PM +0400, Glauber Costa wrote:  
>> I understand your trauma about over flexibility, and you know I share of  
>> it. But I don't think there is any need to cap it here. Given kmem  
>> accounted is perfectly hierarchical, and there seem to be plenty of  
>> people who only care about user memory, I see no reason to disallow a  
>> mixed use case here.  
>>  
>> I must say that for my particular use case, enabling it unconditionally  
>> would just work, so it is not that what I have in mind.  
>  
> So, I'm not gonna go as far as pushing for enabling it unconditionally  
> but would really like to hear why it's necessary to make it per node  
> instead of one global switch. Maybe it has already been discussed to  
> hell and back. Care to summarize / point me to it?  
>

For me, it is the other way around: it makes perfect sense to have a  
per-subtree selection of features where it doesn't hurt us, provided it  
is hierarchical. For the mere fact that not every application is  
interested in this (Michal is the one that is being so far more vocal  
about this not being needed in some use cases), and it is perfectly  
valid to imagine such applications would coexist.

So given the flexibility it brings, the real question is, as I said, backwards: what is it necessary to make it a global switch ?

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure

Posted by [Tejun Heo](#) on Wed, 26 Sep 2012 19:34:17 GMT

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

---

Hello,

On Wed, Sep 26, 2012 at 10:56:09PM +0400, Glauber Costa wrote:

> For me, it is the other way around: it makes perfect sense to have a  
> per-subtree selection of features where it doesn't hurt us, provided it  
> is hierarchical. For the mere fact that not every application is  
> interested in this (Michal is the one that is being so far more vocal  
> about this not being needed in some use cases), and it is perfectly  
> valid to imagine such applications would coexist.

>

> So given the flexibility it brings, the real question is, as I said,  
> backwards: what is it necessary to make it a global switch ?

Because it hurts my head and it's better to keep things simple. We're planning to retire .use\_hierarchy in sub hierarchies and I'd really like to prevent another fiasco like that unless there absolutely is no way around it. Flexibility where necessary is fine but let's please try our best to avoid over-designing things. We've been far too good at getting lost in flexibility maze. Michal, care to chime in?

Thanks.

--

tejun

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure

Posted by [Glauber Costa](#) on Wed, 26 Sep 2012 19:46:37 GMT

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

---

On 09/26/2012 11:34 PM, Tejun Heo wrote:

> Hello,

>

> On Wed, Sep 26, 2012 at 10:56:09PM +0400, Glauber Costa wrote:

>> For me, it is the other way around: it makes perfect sense to have a  
>> per-subtree selection of features where it doesn't hurt us, provided it  
>> is hierarchical. For the mere fact that not every application is  
>> interested in this (Michal is the one that is being so far more vocal  
>> about this not being needed in some use cases), and it is perfectly

>> valid to imagine such applications would coexist.  
>>  
>> So given the flexibility it brings, the real question is, as I said,  
>> backwards: what is it necessary to make it a global switch ?  
>  
> Because it hurts my head and it's better to keep things simple. We're  
> planning to retire .use\_hierarchy in sub hierarchies and I'd really  
> like to prevent another fiasco like that unless there absolutely is no  
> way around it. Flexibility where necessary is fine but let's please  
> try our best to avoid over-designing things. We've been far too good  
> at getting lost in flexibility maze. Michal, care to chime in?  
>

I would very much like to hear Michal here as well, sure.

But as I said in this very beginning of this, you pretty much know that I am heavily involved in trying to get rid of use\_hierarchy, and by no means I consider this en pair with that.

use\_hierarchy is a hack around a core property of cgroups, the fact that they are hierarchical. Its mere existence came to be to overcome a performance limitation.

It puts you in contradictory situation where you have cgroups organized as directories, and then not satisfied in making this hierarchical representation be gravely ignored, forces you to use nonsensical terms like "flat hierarchy", making us grasp at how it is to be a politician once in our lifetimes.

Besides not being part of cgroup core, and respecting very much both cgroups' and basic sanity properties, kmem is an actual feature that some people want, and some people don't. There is no reason to believe that applications that want will live in the same environment with ones that don't want.

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Wed, 26 Sep 2012 19:56:48 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello,

On Wed, Sep 26, 2012 at 11:46:37PM +0400, Glauber Costa wrote:  
> Besides not being part of cgroup core, and respecting very much both  
> cgroups' and basic sanity properties, kmem is an actual feature that  
> some people want, and some people don't. There is no reason to believe  
> that applications that want will live in the same environment with ones  
> that don't want.



I don't know. It definitely is less crazy than `.use_hierarchy` but I wouldn't say it's an inherently different thing. I mean, what does it even mean to have `u+k` limit on one subtree and not on another branch? And we worry about things like what if parent doesn't enable it but its children do.

This is a feature which adds complexity. If the feature is necessary and justified, sure. If not, let's please not and let's err on the side of conservativeness. We can always add it later but the other direction is much harder.

Thanks.

--  
tejun

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Wed, 26 Sep 2012 20:02:14 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/26/2012 11:56 PM, Tejun Heo wrote:

> Hello,

>

> On Wed, Sep 26, 2012 at 11:46:37PM +0400, Glauber Costa wrote:

>> Besides not being part of cgroup core, and respecting very much both  
>> cgroups' and basic sanity properties, kmem is an actual feature that  
>> some people want, and some people don't. There is no reason to believe  
>> that applications that want will live in the same environment with ones  
>> that don't want.

>

> I don't know. It definitely is less crazy than `.use_hierarchy` but I  
> wouldn't say it's an inherently different thing. I mean, what does it  
> even mean to have `u+k` limit on one subtree and not on another branch?  
> And we worry about things like what if parent doesn't enable it but  
> its children do.

>

It is inherently different. To begin with, it actually contemplates two use cases. It is not a work around.

The meaning is also very well defined. The meaning of having this enabled in one subtree and not in other is: Subtree A wants to track kernel memory. Subtree B does not. It's that, and never more than that. There is no maybes and no buts, no magic knobs that makes it behave in a crazy way.

If a children enables it but the parent does not, this does what every tree does: enable it from that point downwards.

> This is a feature which adds complexity. If the feature is necessary  
> and justified, sure. If not, let's please not and let's err on the  
> side of conservativeness. We can always add it later but the other  
> direction is much harder.  
>

I disagree. Having kmem tracking adds complexity. Having to cope with the use case where we turn it on dynamically to cope with the "user page only" use case adds complexity. But I see no significant complexity being added by having it per subtree. Really.

You have the use\_hierarchy fiasco in mind, and I do understand that you are raising the flag and all that.

But think in terms of functionality: This thing here is a lot more similar to swap than use\_hierarchy. Would you argue that memsw should be per-root ?

The reason why it shouldn't: Some people want to limit memory consumption all the way to the swap, some people don't. Same with kmem.

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Wed, 26 Sep 2012 20:16:29 GMT

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

---

On Thu, Sep 27, 2012 at 12:02:14AM +0400, Glauber Costa wrote:

> But think in terms of functionality: This thing here is a lot more  
> similar to swap than use\_hierarchy. Would you argue that memsw should be  
> per-root ?

I'm fairly sure you can make about the same argument about use\_hierarchy. There is a choice to make here and one is simpler than the other. I want the additional complexity justified by actual use cases which isn't too much to ask for especially when the complexity is something visible to userland.

So let's please stop arguing semantics. If this is definitely necessary for some use cases, sure let's have it. If not, let's consider it later. I'll stop responding on "inherent differences." I don't think we'll get anywhere with that.

Michal, Johannes, Kamezawa, what are your thoughts?

Thanks.

--  
tejun

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Wed, 26 Sep 2012 21:24:40 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/27/2012 12:16 AM, Tejun Heo wrote:

> On Thu, Sep 27, 2012 at 12:02:14AM +0400, Glauber Costa wrote:  
>> But think in terms of functionality: This thing here is a lot more  
>> similar to swap than use\_hierarchy. Would you argue that memsw should be  
>> per-root ?  
>  
> I'm fairly sure you can make about the same argument about  
> use\_hierarchy. There is a choice to make here and one is simpler than  
> the other. I want the additional complexity justified by actual use  
> cases which isn't too much to ask for especially when the complexity  
> is something visible to userland.  
>  
> So let's please stop arguing semantics. If this is definitely  
> necessary for some use cases, sure let's have it. If not, let's  
> consider it later. I'll stop responding on "inherent differences." I  
> don't think we'll get anywhere with that.  
>

If you stop responding, we are for sure not getting anywhere. I agree with you here.

Let me point out one issue that you seem to be missing, and you respond or not, your call.

"kmem\_accounted" is not a switch. It is an internal representation only. The semantics, that we discussed exhaustively in San Diego, is that a group that is not limited is not accounted. This is simple and consistent.

Since the limits are still per-cgroup, you are actually proposing more user-visible complexity than me, since you are adding yet another file, with its own semantics.

About use cases, I've already responded: my containers use case is kmem limited. There are people like Michal that specifically asked for user-only semantics to be preserved. So your question for global vs local switch (that again, doesn't exist; only a local \*limit\* exists) should really be posed in the following way:  
"Can two different use cases with different needs be hosted in the same box?"

> Michal, Johannes, Kamezawa, what are your thoughts?

>

waiting! =)

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure

Posted by [Tejun Heo](#) on Wed, 26 Sep 2012 22:10:46 GMT

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

---

Hello, Glauber.

On Thu, Sep 27, 2012 at 01:24:40AM +0400, Glauber Costa wrote:

> "kmem\_accounted" is not a switch. It is an internal representation only.

> The semantics, that we discussed exhaustively in San Diego, is that a

> group that is not limited is not accounted. This is simple and consistent.

>

> Since the limits are still per-cgroup, you are actually proposing more

> user-visible complexity than me, since you are adding yet another file,

> with its own semantics.

I was confused. I thought it was exposed as a switch to userland (it being right below .use\_hierarchy tripped red alert). This is internal flag dependent upon kernel limit being set. My apologies.

So, the proposed behavior is to allow enabling kmemcg anytime but ignore what happened inbetween? Where the knob is changes but the weirdity seems all the same. What prevents us from having a single switch at root which can only be flipped when there's no children?

Backward compatibility is covered with single switch and I really don't think "you can enable limits for kernel memory anytime but we don't keep track of whatever happened before it was flipped the first time because the first time is always special" is a sane thing to expose to userland. Or am I misunderstanding the proposed behavior again?

Thanks.

--

tejun

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure

Posted by [Johannes Weiner](#) on Wed, 26 Sep 2012 22:11:36 GMT

---

On Thu, Sep 27, 2012 at 12:02:14AM +0400, Glauber Costa wrote:

> On 09/26/2012 11:56 PM, Tejun Heo wrote:

> > Hello,

> >

> > On Wed, Sep 26, 2012 at 11:46:37PM +0400, Glauber Costa wrote:

> >> Besides not being part of cgroup core, and respecting very much both

> >> cgroups' and basic sanity properties, kmem is an actual feature that

> >> some people want, and some people don't. There is no reason to believe

> >> that applications that want will live in the same environment with ones

> >> that don't want.

> >

> > I don't know. It definitely is less crazy than .use\_hierarchy but I

> > wouldn't say it's an inherently different thing. I mean, what does it

> > even mean to have u+k limit on one subtree and not on another branch?

> > And we worry about things like what if parent doesn't enable it but

> > its children do.

> >

>

> It is inherently different. To begin with, it actually contemplates two

> use cases. It is not a work around.

>

> The meaning is also very well defined. The meaning of having this

> enabled in one subtree and not in other is: Subtree A wants to track

> kernel memory. Subtree B does not. It's that, and never more than that.

> There is no maybes and no buts, no magic knobs that makes it behave in a

> crazy way.

>

> If a children enables it but the parent does not, this does what every

> tree does: enable it from that point downwards.

>

> > This is a feature which adds complexity. If the feature is necessary

> > and justified, sure. If not, let's please not and let's err on the

> > side of conservativeness. We can always add it later but the other

> > direction is much harder.

>

> I disagree. Having kmem tracking adds complexity. Having to cope with

> the use case where we turn it on dynamically to cope with the "user page

> only" use case adds complexity. But I see no significant complexity

> being added by having it per subtree. Really.

Maybe not in code, but you are adding an extra variable into the system. "One switch per subtree" is more complex than "one switch."

Yes, the toggle is hidden behind setting the limit, but it's still a toggle. The use\_hierarchy complexity comes not from the file that enables it, but from the resulting semantics.

kmem accounting is expensive and we definitely want to allow enabling

it separately from traditional user memory accounting. But I think there is no good reason to not demand an all-or-nothing answer from the admin; either he wants kmem tracking on a machine or not. At least you haven't presented a convincing case, IMO.

I don't think there is strong/any demand for per-node toggles, but once we add this behavior, people will rely on it and expect kmem tracking to stay local and we are stuck with it. Adding it for the reason that people will use it is a self-fulfilling prophecy.

> You have the use\_hierarchy fiasco in mind, and I do understand that you  
> are raising the flag and all that.  
>  
> But think in terms of functionality: This thing here is a lot more  
> similar to swap than use\_hierarchy. Would you argue that memsw should be  
> per-root ?

We actually do have a per-root flag that controls accounting for swap.

> The reason why it shouldn't: Some people want to limit memory  
> consumption all the way to the swap, some people don't. Same with kmem.

That lies in the nature of the interface: we chose k & u+k rather than u & u+k, so our memory.limit\_in\_bytes will necessarily include kmem, while swap is not included there. But I really doubt that there is a strong case for turning on swap accounting intentionally and then limiting memory+swap only on certain subtrees. Where would be the sense in that?

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Wed, 26 Sep 2012 22:29:06 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/27/2012 02:10 AM, Tejun Heo wrote:

> Hello, Glauber.  
>  
> On Thu, Sep 27, 2012 at 01:24:40AM +0400, Glauber Costa wrote:  
>> "kmem\_accounted" is not a switch. It is an internal representation only.  
>> The semantics, that we discussed exhaustively in San Diego, is that a  
>> group that is not limited is not accounted. This is simple and consistent.  
>>  
>> Since the limits are still per-cgroup, you are actually proposing more  
>> user-visible complexity than me, since you are adding yet another file,  
>> with its own semantics.  
>  
> I was confused. I thought it was exposed as a switch to userland (it  
> being right below .use\_hierarchy tripped red alert).

Remember I was the one more vocally and radically so far trying to get rid of use\_hierarchy. I should have been more clear - and I was, as soon as I better understood the nature of your opposition - but this is precisely what I meant by "inherently different".

>  
> So, the proposed behavior is to allow enabling kmemcg anytime but  
> ignore what happened inbetween? Where the knob is changes but the  
> weirdity seems all the same. What prevents us from having a single  
> switch at root which can only be flipped when there's no children?

So I view this very differently from you. We have no root-only switches in memcg. This would be a first, and this is the kind of thing that adds complexity, in my view.

You have someone like libvirt or a systemd service using memcg. It probably starts at boot. Once it is started, it will pretty much prevent switching of any global switch like this.

And then what? If you want a different behavior you need to go kill all your services that are using memcg so you can get the behavior you want? And if they happen to be making a specific flag choice by design, you just say "you really can't run A + B together" ?

I myself think global switches are an unnecessary complication. And let us not talk about use\_hierarchy, please. If it becomes global, it is going to be as part of a phase out plan anyway. The problem with that is not that it is global, is that it shouldn't even exist.

>  
> Backward compatibility is covered with single switch and I really  
> don't think "you can enable limits for kernel memory anytime but we  
> don't keep track of whatever happened before it was flipped the first  
> time because the first time is always special" is a sane thing to  
> expose to userland. Or am I misunderstanding the proposed behavior  
> again?  
>

You do keep track. Before you switch it for the first time, it all belongs to the root memcg.

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Wed, 26 Sep 2012 22:42:35 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello, Glauber.

On Thu, Sep 27, 2012 at 02:29:06AM +0400, Glauber Costa wrote:

> And then what? If you want a different behavior you need to go kill all  
> your services that are using memcg so you can get the behavior you want?  
> And if they happen to be making a specific flag choice by design, you  
> just say "you really can't run A + B together" ?  
>  
> I myself think global switches are an unnecessary complication. And let  
> us not talk about use\_hierarchy, please. If it becomes global, it is  
> going to be as part of a phase out plan anyway. The problem with that is  
> not that it is global, is that it shouldn't even exist.

I would consider it more of a compatibility thing which is set during boot and configurable by sysadmin. Let the newer systems enable it by default on boot and old configs / special ones disable it as necessary.

> > Backward compatibility is covered with single switch and I really  
> > don't think "you can enable limits for kernel memory anytime but we  
> > don't keep track of whatever happened before it was flipped the first  
> > time because the first time is always special" is a sane thing to  
> > expose to userland. Or am I misunderstanding the proposed behavior  
> > again?  
>  
> You do keep track. Before you switch it for the first time, it all  
> belongs to the root memcg.

Well, that's really playing with words. Limit is per cgroup and before the limit is set for the first time, everything is accounted to something else. How is that keeping track?

The proposed behavior seems really crazy to me. Do people really think this is a good idea?

Thanks.

--  
tejun

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Wed, 26 Sep 2012 22:45:52 GMT  
[View Forum Message](#) <> [Reply to Message](#)

On 09/27/2012 02:11 AM, Johannes Weiner wrote:

> On Thu, Sep 27, 2012 at 12:02:14AM +0400, Glauber Costa wrote:  
>> On 09/26/2012 11:56 PM, Tejun Heo wrote:  
>>> Hello,



```

>>>
>>> On Wed, Sep 26, 2012 at 11:46:37PM +0400, Glauber Costa wrote:
>>>> Besides not being part of cgroup core, and respecting very much both
>>>> cgroups' and basic sanity properties, kmem is an actual feature that
>>>> some people want, and some people don't. There is no reason to believe
>>>> that applications that want will live in the same environment with ones
>>>> that don't want.
>>>
>>> I don't know. It definitely is less crazy than .use_hierarchy but I
>>> wouldn't say it's an inherently different thing. I mean, what does it
>>> even mean to have u+k limit on one subtree and not on another branch?
>>> And we worry about things like what if parent doesn't enable it but
>>> its children do.
>>>
>>
>> It is inherently different. To begin with, it actually contemplates two
>> use cases. It is not a work around.
>>
>> The meaning is also very well defined. The meaning of having this
>> enabled in one subtree and not in other is: Subtree A wants to track
>> kernel memory. Subtree B does not. It's that, and never more than that.
>> There is no maybes and no buts, no magic knobs that makes it behave in a
>> crazy way.
>>
>> If a children enables it but the parent does not, this does what every
>> tree does: enable it from that point downwards.
>>
>>> This is a feature which adds complexity. If the feature is necessary
>>> and justified, sure. If not, let's please not and let's err on the
>>> side of conservativeness. We can always add it later but the other
>>> direction is much harder.
>>
>> I disagree. Having kmem tracking adds complexity. Having to cope with
>> the use case where we turn it on dynamically to cope with the "user page
>> only" use case adds complexity. But I see no significant complexity
>> being added by having it per subtree. Really.
>
> Maybe not in code, but you are adding an extra variable into the
> system. "One switch per subtree" is more complex than "one switch."
> Yes, the toggle is hidden behind setting the limit, but it's still a
> toggle. The use_hierarchy complexity comes not from the file that
> enables it, but from the resulting semantics.
>

```

I didn't claim the complexity was in the code. I actually think the other way around that you do, and claim that a global switch is more complex than a per-subtree. All properties we have so far applies to subtrees, due to cgroup's hierarchical nature. We have no global

switches like this so far, and adding one would just add a new concept that wasn't here.

- > kmem accounting is expensive and we definitely want to allow enabling
- > it separately from traditional user memory accounting. But I think
- > there is no good reason to not demand an all-or-nothing answer from
- > the admin; either he wants kmem tracking on a machine or not. At
- > least you haven't presented a convincing case, IMO.
- >
- > I don't think there is strong/any demand for per-node toggles, but
- > once we add this behavior, people will rely on it and expect kmem
- > tracking to stay local and we are stuck with it. Adding it for the
- > reason that people will use it is a self-fulfilling prophecy.

I don't think this is a compatibility only switch. Much has been said in the past about the problem of sharing. A lot of the kernel objects are shared by nature, this is pretty much unavoidable. The answer we have been giving to this inquiry, is that the workloads (us) interested in kmem accounted tend to be quite local in their file accesses (and other kernel objects as well).

It should be obvious that not all workloads are like this, and some of them would actually prefer to have their umem limited only.

I really don't think, and correct me if I am wrong, that the problem lays in "is there a use case for umem?", but rather, if they should be allowed to coexist in a box.

And honestly, it seems to me totally reasonable to avoid restricting people to run as many workloads they think they can in the same box.

- >> You have the use\_hierarchy fiasco in mind, and I do understand that you
- >> are raising the flag and all that.
- >>
- >> But think in terms of functionality: This thing here is a lot more
- >> similar to swap than use\_hierarchy. Would you argue that memsw should be
- >> per-root ?
- >
- > We actually do have a per-root flag that controls accounting for swap.
- >
- >> The reason why it shouldn't: Some people want to limit memory
- >> consumption all the way to the swap, some people don't. Same with kmem.
- >
- > That lies in the nature of the interface: we chose k & u+k rather than
- > u & u+k, so our memory.limit\_in\_bytes will necessarily include kmem,

> while swap is not included there. But I really doubt that there is a  
> strong case for turning on swap accounting intentionally and then  
> limiting memory+swap only on certain subtrees. Where would be the  
> sense in that?

It makes absolute sense. Because until I go set  
memory.memsw.limit\_in\_bytes, my subtree is not limited, which is  
precisely what kmem does.

And the use cases for that are:

- 1) I, application A, want to use 2G of mem, and I can never swap
- 2) I, application B, want to use 2G of mem, but I am fine using extra 1G  
in swap.

There are plenty of workloads in both the "can swap" and "can't swap"  
category around.

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Wed, 26 Sep 2012 22:54:11 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/27/2012 02:42 AM, Tejun Heo wrote:

> Hello, Glauber.

>

> On Thu, Sep 27, 2012 at 02:29:06AM +0400, Glauber Costa wrote:

>> And then what? If you want a different behavior you need to go kill all  
>> your services that are using memcg so you can get the behavior you want?

>> And if they happen to be making a specific flag choice by design, you  
>> just say "you really can't run A + B together" ?

>>

>> I myself think global switches are an unnecessary complication. And let  
>> us not talk about use\_hierarchy, please. If it becomes global, it is  
>> going to be as part of a phase out plan anyway. The problem with that is  
>> not that it is global, is that it shouldn't even exist.

>

> I would consider it more of a compatibility thing which is set during  
> boot and configurable by sysadmin. Let the newer systems enable it by  
> default on boot and old configs / special ones disable it as  
> necessary.

>

I don't. Much has been said in the past about the problem of sharing. A  
lot of the kernel objects are shared by nature, this is pretty much  
unavoidable. The answer we have been giving to this inquiry, is that the  
workloads (us) interested in kmem accounted tend to be quite local in  
their file accesses (and other kernel objects as well).

It should be obvious that not all workloads are like this, and some of them would actually prefer to have their umem limited only.

There is nothing unreasonable in tracking user memory only.

If we have a global switch for "tracking all kernel memory", who would you account the objects that are heavily shared to? I solve this by not tracking kernel memory for cgroups in such workloads. What do you propose?

>>> Backward compatibility is covered with single switch and I really  
>>> don't think "you can enable limits for kernel memory anytime but we  
>>> don't keep track of whatever happened before it was flipped the first  
>>> time because the first time is always special" is a sane thing to  
>>> expose to userland. Or am I misunderstanding the proposed behavior  
>>> again?

>>  
>> You do keep track. Before you switch it for the first time, it all  
>> belongs to the root memcg.

>  
> Well, that's really playing with words. Limit is per cgroup and  
> before the limit is set for the first time, everything is accounted to  
> something else. How is that keeping track?  
>

Even after the limit is set, it is set only by workloads that want kmem to be tracked. If you want to track it during the whole lifetime of the cgroup, you switch it before you put tasks to it. What is so crazy about it?

> The proposed behavior seems really crazy to me. Do people really  
> think this is a good idea?  
>

It is really sad that you lost the opportunity to say that in a room full of mm developers that could add to this discussion in real time, when after an explanation about this was given, Mel asked if anyone would have any objections to this.

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Wed, 26 Sep 2012 23:08:07 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello, Glauber.

On Thu, Sep 27, 2012 at 02:54:11AM +0400, Glauber Costa wrote:  
> I don't. Much has been said in the past about the problem of sharing. A  
> lot of the kernel objects are shared by nature, this is pretty much

> unavoidable. The answer we have been giving to this inquiry, is that the  
 > workloads (us) interested in kmem accounted tend to be quite local in  
 > their file accesses (and other kernel objects as well).  
 >  
 > It should be obvious that not all workloads are like this, and some of  
 > them would actually prefer to have their umem limited only.  
 >  
 > There is nothing unreasonable in tracking user memory only.  
 >  
 > If we have a global switch for "tracking all kernel memory", who would  
 > you account the objects that are heavily shared to? I solve this by not  
 > tracking kernel memory for cgroups in such workloads. What do you propose?

One of the things wrong with that is that it exposes the limitation of the current implementation as interface to userland, which is never a good idea. In addition, how is userland supposed to know which workload is shared kmem heavy or not? Details like that are not even inherent to workloads. It's highly dependent on kernel implementation which may change any day. If we hit workloads like that the right thing to do is improving kmemcg so that such problems don't occur, not exposing another switch.

If we can't make that work in reasonable (doesn't have to be perfect) way, we might as well just give up on kmem controller. If userland has to second-guess kernel implementation details to make use of it, it's useless.

> > Well, that's really playing with words. Limit is per cgroup and  
 > > before the limit is set for the first time, everything is accounted to  
 > > something else. How is that keeping track?  
 > >  
 >  
 > Even after the limit is set, it is set only by workloads that want kmem  
 > to be tracked. If you want to track it during the whole lifetime of the  
 > cgroup, you switch it before you put tasks to it. What is so crazy about it?

The fact that the numbers don't really mean what they apparently should mean.

> > The proposed behavior seems really crazy to me. Do people really  
 > > think this is a good idea?  
 >  
 > It is really sad that you lost the opportunity to say that in a room  
 > full of mm developers that could add to this discussion in real time,  
 > when after an explanation about this was given, Mel asked if anyone  
 > would have any objections to this.

Sure, conferences are useful for building consensus but that's the

extent of it. Sorry that I didn't realize the implications then but conferences don't really add any finality to decisions.

So, this seems properly crazy to me at the similar level of use\_hierarchy fiasco. I'm gonna NACK on this.

Thanks.

--  
tejun

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Wed, 26 Sep 2012 23:20:27 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/27/2012 03:08 AM, Tejun Heo wrote:

> Hello, Glauber.

>

> On Thu, Sep 27, 2012 at 02:54:11AM +0400, Glauber Costa wrote:

>> I don't. Much has been said in the past about the problem of sharing. A

>> lot of the kernel objects are shared by nature, this is pretty much

>> unavoidable. The answer we have been giving to this inquiry, is that the

>> workloads (us) interested in kmem accounted tend to be quite local in

>> their file accesses (and other kernel objects as well).

>>

>> It should be obvious that not all workloads are like this, and some of

>> them would actually prefer to have their umem limited only.

>>

>> There is nothing unreasonable in tracking user memory only.

>>

>> If we have a global switch for "tracking all kernel memory", who would

>> you account the objects that are heavily shared to? I solve this by not

>> tracking kernel memory for cgroups in such workloads. What do you propose?

>

> One of the things wrong with that is that it exposes the limitation of

> the current implementation as interface to userland, which is never a

> good idea. In addition, how is userland supposed to know which

> workload is shared kmem heavy or not? Details like that are not even

> inherent to workloads. It's highly dependent on kernel implementation

> which may change any day. If we hit workloads like that the right

> thing to do is improving kmemcg so that such problems don't occur, not

> exposing another switch.

>

Sorry, there is nothing implementation dependent in here. One of the biggest consumers of all this, are dentries. Dentries are related to the paths you touch. If you touch files in a self-contained directory,

where you don't expect anyone else to touch, this can safely be considered local. If you touch files all around, this can safely be considered not local. Where is the implementation dependent part?

> If we can't make that work in reasonable (doesn't have to be perfect)  
> way, we might as well just give up on kmem controller. If userland  
> has to second-guess kernel implementation details to make use of it,  
> it's useless.

>  
As I said above, it shouldn't.

>>> Well, that's really playing with words. Limit is per cgroup and  
>>> before the limit is set for the first time, everything is accounted to  
>>> something else. How is that keeping track?  
>>>  
>>  
>> Even after the limit is set, it is set only by workloads that want kmem  
>> to be tracked. If you want to track it during the whole lifetime of the  
>> cgroup, you switch it before you put tasks to it. What is so crazy about it?  
>  
> The fact that the numbers don't really mean what they apparently  
> should mean.  
>

This is vague. The usage file in the cgroup means how much kernel memory was used by that cgroup. If it really bothers you that this may not be set through the whole group's lifetime, it is also easily solvable.

> So, this seems properly crazy to me at the similar level of  
> use\_hierarchy fiasco. I'm gonna NACK on this.  
>

As I said: all use cases I particularly care about are covered by a global switch.

I am laying down my views because I really believe they make more sense. But at some point, of course, I'll shut up if I believe I am a lone voice.

I believe it should still be good to hear from mhocko and kame, but from your point of view, would all the rest, plus the introduction of a global switch make it acceptable to you?

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Wed, 26 Sep 2012 23:33:34 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello,



On Thu, Sep 27, 2012 at 03:20:27AM +0400, Glauber Costa wrote:

> > One of the things wrong with that is that it exposes the limitation of  
> > the current implementation as interface to userland, which is never a  
> > good idea. In addition, how is userland supposed to know which  
> > workload is shared kmem heavy or not? Details like that are not even  
> > inherent to workloads. It's highly dependent on kernel implementation  
> > which may change any day. If we hit workloads like that the right  
> > thing to do is improving kmemcg so that such problems don't occur, not  
> > exposing another switch.

>

> Sorry, there is nothing implementation dependent in here. One of the  
> biggest consumers of all this, are dentries. Dentries are related to  
> the paths you touch. If you touch files in a self-contained directory,  
> where you don't expect anyone else to touch, this can safely be  
> considered local. If you touch files all around, this can safely be  
> considered not local. Where is the implementation dependent part?

For things like dentries and inodes and if that really matters, we should be able to account for the usage better, no? And frankly I'm not even sold on that usecase. Unless there's a way to detect and inform about these, userland isn't gonna know that they're doing something which consumes a lot of shared memory even that activity is filesystem walking. You're still asking userland to tune something depending on parameters not easily visible from userland. It's a lose lose situation.

> >> Even after the limit is set, it is set only by workloads that want kmem  
> >> to be tracked. If you want to track it during the whole lifetime of the  
> >> cgroup, you switch it before you put tasks to it. What is so crazy about it?

> >

> > The fact that the numbers don't really mean what they apparently  
> > should mean.

> >

>

> This is vague. The usage file in the cgroup means how much kernel memory  
> was used by that cgroup. If it really bothers you that this may not be  
> set through the whole group's lifetime, it is also easily solvable.

Yes, easily by having a global switch which can be manipulated when there's no children. It really seems like a no brainer to me.

> > So, this seems properly crazy to me at the similar level of  
> > use\_hierarchy fiasco. I'm gonna NACK on this.

>

> As I said: all use cases I particularly care about are covered by a  
> global switch.

>



> I am laying down my views because I really believe they make more sense.  
> But at some point, of course, I'll shut up if I believe I am a lone voice.  
>  
> I believe it should still be good to hear from mhocko and kame, but from  
> your point of view, would all the rest, plus the introduction of a  
> global switch make it acceptable to you?

The only thing I'm whining about is per-node switch + silently ignoring past accounting, so if those two are solved, I think I'm pretty happy with the rest.

Thanks.

--  
tejun

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Michal Hocko](#) on Thu, 27 Sep 2012 12:08:06 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Thu 27-09-12 01:24:40, Glauber Costa wrote:

[...]

> About use cases, I've already responded: my containers use case is kmem  
> limited. There are people like Michal that specifically asked for  
> user-only semantics to be preserved.

Yes, because we have many users (basically almost all) who care only about the user memory because that's what occupies the vast majority of the memory. They usually want to isolate workload which would disrupt the global memory otherwise (e.g. backup process vs. database). You really do not want to pay an additional overhead for kmem accounting here.

> So your question for global vs local switch (that again, doesn't  
> exist; only a local \*limit\* exists) should really be posed in the  
> following way: "Can two different use cases with different needs be  
> hosted in the same box?"

I think this is a good and a relevant question. I think this boils down to whether you want to have trusted and untrusted workloads at the same machine.

Trusted loads usually only need user memory accounting because kmem consumption should be really negligible (unless kernel is doing something really stupid and no kmem limit will help here).

On the other hand, untrusted workloads can do nasty things that administrator has hard time to mitigate and setting a kmem limit can help significantly.

IMHO such a different loads exist on a single machine quite often (Web server and a back up process as the most simplistic one). The per hierarchy accounting, therefore, sounds like a good idea without too much added complexity (actually the only added complexity is in the proper `kmem.limit_in_bytes` handling which is a single place).

So I would rather go with per-hierarchy thing.

> > Michal, Johannes, Kamezawa, what are your thoughts?  
> >  
> waiting! =)

Well, you guys generated a lot of discussion that one has to read through, didn't you :P

--  
Michal Hocko  
SUSE Labs

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Thu, 27 Sep 2012 12:11:00 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

>>> Michal, Johannes, Kamezawa, what are your thoughts?  
>>>  
>> waiting! =)  
>  
> Well, you guys generated a lot of discussion that one has to read  
> through, didn't you :P  
>  
We're quite good at that!

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Michal Hocko](#) on Thu, 27 Sep 2012 12:15:58 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Wed 26-09-12 16:33:34, Tejun Heo wrote:

[...]  
> > > So, this seems properly crazy to me at the similar level of  
> > > use\_hierarchy fiasco. I'm gonna NACK on this.  
> >  
> > As I said: all use cases I particularly care about are covered by a  
> > global switch.  
> >

> > I am laying down my views because I really believe they make more sense.  
> > But at some point, of course, I'll shut up if I believe I am a lone voice.  
> >  
> > I believe it should still be good to hear from mhocko and kame, but from  
> > your point of view, would all the rest, plus the introduction of a  
> > global switch make it acceptable to you?  
>  
> The only thing I'm whining about is per-node switch + silently  
> ignoring past accounting, so if those two are solved, I think I'm  
> pretty happy with the rest.

I think that per-group "switch" is not nice as well but if we make it hierarchy specific (which I am proposing for quite some time) and do not let enable accounting for a group with tasks then we get both flexibility and reasonable semantic. A global switch sounds too coars to me and it really not necessary.

Would this work with you?

--

Michal Hocko  
SUSE Labs

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Thu, 27 Sep 2012 12:20:55 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/27/2012 04:15 PM, Michal Hocko wrote:

> On Wed 26-09-12 16:33:34, Tejun Heo wrote:

> [...]

>>>> So, this seems properly crazy to me at the similar level of  
>>>> use\_hierarchy fiasco. I'm gonna NACK on this.

>>>

>>> As I said: all use cases I particularly care about are covered by a  
>>> global switch.

>>>

>>> I am laying down my views because I really believe they make more sense.

>>> But at some point, of course, I'll shut up if I believe I am a lone voice.

>>>

>>> I believe it should still be good to hear from mhocko and kame, but from  
>>> your point of view, would all the rest, plus the introduction of a  
>>> global switch make it acceptable to you?

>>

>> The only thing I'm whining about is per-node switch + silently  
>> ignoring past accounting, so if those two are solved, I think I'm  
>> pretty happy with the rest.

>

> I think that per-group "switch" is not nice as well but if we make it

> hierarchy specific (which I am proposing for quite some time) and do not  
> let enable accounting for a group with tasks then we get both  
> flexibility and reasonable semantic. A global switch sounds too coars to  
> me and it really not necessary.  
>  
> Would this work with you?  
>

How exactly would that work? AFAIK, we have a single memcg root, we  
can't have multiple memcg hierarchies in a system. Am I missing something?

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Thu, 27 Sep 2012 12:40:03 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/27/2012 04:40 PM, Michal Hocko wrote:

> On Thu 27-09-12 16:20:55, Glauber Costa wrote:  
>> On 09/27/2012 04:15 PM, Michal Hocko wrote:  
>>> On Wed 26-09-12 16:33:34, Tejun Heo wrote:  
>>> [...]  
>>>>> So, this seems properly crazy to me at the similar level of  
>>>>> use\_hierarchy fiasco. I'm gonna NACK on this.  
>>>>>  
>>>>> As I said: all use cases I particularly care about are covered by a  
>>>>> global switch.  
>>>>>  
>>>>> I am laying down my views because I really believe they make more sense.  
>>>>> But at some point, of course, I'll shut up if I believe I am a lone voice.  
>>>>>  
>>>>> I believe it should still be good to hear from mhocko and kame, but from  
>>>>> your point of view, would all the rest, plus the introduction of a  
>>>>> global switch make it acceptable to you?  
>>>>>  
>>>> The only thing I'm whining about is per-node switch + silently  
>>>> ignoring past accounting, so if those two are solved, I think I'm  
>>>> pretty happy with the rest.  
>>>>  
>>> I think that per-group "switch" is not nice as well but if we make it  
>>> hierarchy specific (which I am proposing for quite some time) and do not  
>>> let enable accounting for a group with tasks then we get both  
>>> flexibility and reasonable semantic. A global switch sounds too coars to  
>>> me and it really not necessary.  
>>>  
>>> Would this work with you?  
>>>  
>>  
>> How exactly would that work? AFAIK, we have a single memcg root, we

>> can't have multiple memcg hierarchies in a system. Am I missing something?  
>  
> Well root is so different that we could consider the first level as the  
> real roots for hierarchies.  
>  
So let's favor clarity: What you are proposing is that the first level  
can have a switch for that, and the first level only. Is that right ?

At first, I just want to understand what exactly is your proposal. This  
is not an endorsement of lack thereof.

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Michal Hocko](#) on Thu, 27 Sep 2012 12:40:31 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Thu 27-09-12 16:20:55, Glauber Costa wrote:  
> On 09/27/2012 04:15 PM, Michal Hocko wrote:  
> > On Wed 26-09-12 16:33:34, Tejun Heo wrote:  
> > [...]  
> >>>> So, this seems properly crazy to me at the similar level of  
> >>>> use\_hierarchy fiasco. I'm gonna NACK on this.  
> >>>  
> >>> As I said: all use cases I particularly care about are covered by a  
> >>> global switch.  
> >>>  
> >>> I am laying down my views because I really believe they make more sense.  
> >>> But at some point, of course, I'll shut up if I believe I am a lone voice.  
> >>>  
> >>> I believe it should still be good to hear from mhocko and kame, but from  
> >>> your point of view, would all the rest, plus the introduction of a  
> >>> global switch make it acceptable to you?  
> >>  
> >> The only thing I'm whining about is per-node switch + silently  
> >> ignoring past accounting, so if those two are solved, I think I'm  
> >> pretty happy with the rest.  
> >  
> > I think that per-group "switch" is not nice as well but if we make it  
> > hierarchy specific (which I am proposing for quite some time) and do not  
> > let enable accounting for a group with tasks then we get both  
> > flexibility and reasonable semantic. A global switch sounds too coars to  
> > me and it really not necessary.  
> >  
> > Would this work with you?  
> >  
>  
> How exactly would that work? AFAIK, we have a single memcg root, we  
> can't have multiple memcg hierarchies in a system. Am I missing something?

Well root is so different that we could consider the first level as the real roots for hierarchies.

--

Michal Hocko  
SUSE Labs

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Michal Hocko](#) on Thu, 27 Sep 2012 12:54:15 GMT

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

---

On Thu 27-09-12 16:40:03, Glauber Costa wrote:

> On 09/27/2012 04:40 PM, Michal Hocko wrote:

> > On Thu 27-09-12 16:20:55, Glauber Costa wrote:

> >> On 09/27/2012 04:15 PM, Michal Hocko wrote:

> >>> On Wed 26-09-12 16:33:34, Tejun Heo wrote:

> >>> [...]

> >>>>> So, this seems properly crazy to me at the similar level of

> >>>>> use\_hierarchy fiasco. I'm gonna NACK on this.

> >>>>>

> >>>>> As I said: all use cases I particularly care about are covered by a

> >>>>> global switch.

> >>>>>

> >>>>> I am laying down my views because I really believe they make more sense.

> >>>>> But at some point, of course, I'll shut up if I believe I am a lone voice.

> >>>>>

> >>>>> I believe it should still be good to hear from mhocko and kame, but from

> >>>>> your point of view, would all the rest, plus the introduction of a

> >>>>> global switch make it acceptable to you?

> >>>>>

> >>>>> The only thing I'm whining about is per-node switch + silently

> >>>>> ignoring past accounting, so if those two are solved, I think I'm

> >>>>> pretty happy with the rest.

> >>>>>

> >>>>> I think that per-group "switch" is not nice as well but if we make it

> >>>>> hierarchy specific (which I am proposing for quite some time) and do not

> >>>>> let enable accounting for a group with tasks then we get both

> >>>>> flexibility and reasonable semantic. A global switch sounds too coars to

> >>>>> me and it really not necessary.

> >>>>>

> >>>>> Would this work with you?

> >>>>>

> >>>>>

> >> How exactly would that work? AFAIK, we have a single memcg root, we

> >> can't have multiple memcg hierarchies in a system. Am I missing something?

> >

> > Well root is so different that we could consider the first level as the

> > real roots for hierarchies.  
> >  
> So let's favor clarity: What you are proposing is that the first level  
> can have a switch for that, and the first level only. Is that right ?

I do not want any more switches. I am fine with your "set the limit and start accounting approach" and then inherit the `_internal_` flag down the hierarchy.

If you are in a child and want to set the limit then you can do that only if your parent is accounted already (so that you can have your own limit). We will need the same thing for `oom_control` and swappiness.

--

Michal Hocko  
SUSE Labs

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Mel Gorman](#) on Thu, 27 Sep 2012 14:28:22 GMT

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

---

On Wed, Sep 26, 2012 at 04:08:07PM -0700, Tejun Heo wrote:

> Hello, Glauber.

>

> On Thu, Sep 27, 2012 at 02:54:11AM +0400, Glauber Costa wrote:

> > I don't. Much has been said in the past about the problem of sharing. A

> > lot of the kernel objects are shared by nature, this is pretty much

> > unavoidable. The answer we have been giving to this inquiry, is that the

> > workloads (us) interested in kmem accounted tend to be quite local in

> > their file accesses (and other kernel objects as well).

> >

> > It should be obvious that not all workloads are like this, and some of

> > them would actually prefer to have their umem limited only.

> >

> > There is nothing unreasonable in tracking user memory only.

> >

> > If we have a global switch for "tracking all kernel memory", who would

> > you account the objects that are heavily shared to? I solve this by not

> > tracking kernel memory for cgroups in such workloads. What do you propose?

>

> One of the things wrong with that is that it exposes the limitation of

> the current implementation as interface to userland, which is never a

> good idea.

I think the limitations have been fairly clearly explained and any admin using the interface is going to have \*some\* familiarity with the limitations.

> In addition, how is userland supposed to know which

> workload is shared kmem heavy or not?

By using a bit of common sense.

An application may not be able to figure this out but the administrator is going to be able to make a very educated guess. If processes running within two containers are not sharing a filesystem hierarchy for example then it'll be clear they are not sharing dentries.

If there was a suspicion they were then it could be analysed with something like SystemTap probing when files are opened and see if files are being opened that are shared between containers.

It's not super-easy but it's not impossible either and I fail to see why it's such a big deal for you.

>Details like that are not even

> inherent to workloads. It's highly dependent on kernel implementation  
> which may change any day. If we hit workloads like that the right  
> thing to do is improving kmemcg so that such problems don't occur, not  
> exposing another switch.

>

> If we can't make that work in reasonable (doesn't have to be perfect)  
> way, we might as well just give up on kmem controller. If userland  
> has to second-guess kernel implementation details to make use of it,  
> it's useless.

>

> > > Well, that's really playing with words. Limit is per cgroup and  
> > > before the limit is set for the first time, everything is accounted to  
> > > something else. How is that keeping track?

> > >

> >

> > Even after the limit is set, it is set only by workloads that want kmem  
> > to be tracked. If you want to track it during the whole lifetime of the  
> > cgroup, you switch it before you put tasks to it. What is so crazy about it?

>

> The fact that the numbers don't really mean what they apparently  
> should mean.

>

I think it is a reasonable limitation that only some kernel allocations are accounted for although I'll freely admit I'm not a cgroup or memcg user either.

My understanding is that this comes down to cost -- accounting for the kernel memory usage is expensive so it is limited only to the allocations that are easy to abuse by an unprivileged process. Hence this is initially concerned with stack pages with dentries and TCP usage to follow in later patches.



Further I would expect that an administrator would be aware of these limitations and set `kmem_accounting` at cgroup creation time before any processes start. Maybe that should be enforced but it's not a fundamental problem.

Due to the cost of accounting, I can see why it would be desirable to enable `kmem_accounting` for some cgroup trees and not others. It is not unreasonable to expect that an administrator might want to account within one cgroup where processes are accessing millions of files without impairing the performance of another cgroup that is mostly using anonymous memory.

> > > The proposed behavior seems really crazy to me. Do people really  
> > > think this is a good idea?  
> >  
> > It is really sad that you lost the opportunity to say that in a room  
> > full of mm developers that could add to this discussion in real time,  
> > when after an explanation about this was given, Mel asked if anyone  
> > would have any objections to this.  
>  
> Sure, conferences are useful for building consensus but that's the  
> extent of it. Sorry that I didn't realize the implications then but  
> conferences don't really add any finality to decisions.  
>  
> So, this seems properly crazy to me at the similar level of  
> `use_hierarchy` fiasco. I'm gonna NACK on this.  
>

I think you're over-reacting to say the very least :|

--

Mel Gorman  
SUSE Labs

---

Subject: Re: [PATCH v3 04/13] `kmem` accounting basic infrastructure  
Posted by [Tejun Heo](#) on Thu, 27 Sep 2012 14:33:00 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello, Michal.

On Thu, Sep 27, 2012 at 02:08:06PM +0200, Michal Hocko wrote:

> Yes, because we have many users (basically almost all) who care only  
> about the user memory because that's what occupies the vast majority of  
> the memory. They usually want to isolate workload which would disrupt  
> the global memory otherwise (e.g. backup process vs. database). You  
> really do not want to pay an additional overhead for `kmem` accounting

> here.

I'm not too convinced. First of all, the overhead added by kmemcg isn't big. The hot path overhead is quite minimal - it doesn't do much more than indirecting one more time. In terms of memory usage, it sure could lead to a bit more fragmentation but even if it gets to several megs per cgroup, I don't think that's something excessive. So, there is overhead but I don't believe it to be prohibitive.

> > So your question for global vs local switch (that again, doesn't  
> > exist; only a local \*limit\* exists) should really be posed in the  
> > following way: "Can two different use cases with different needs be  
> > hosted in the same box?"  
>  
> I think this is a good and a relevant question. I think this boils down  
> to whether you want to have trusted and untrusted workloads at the same  
> machine.  
> Trusted loads usually only need user memory accounting because kmem  
> consumption should be really negligible (unless kernel is doing  
> something really stupid and no kmem limit will help here).  
> On the other hand, untrusted workloads can do nasty things that  
> administrator has hard time to mitigate and setting a kmem limit can  
> help significantly.  
>  
> IMHO such a different loads exist on a single machine quite often (Web  
> server and a back up process as the most simplistic one). The per  
> hierarchy accounting, therefore, sounds like a good idea without too  
> much added complexity (actually the only added complexity is in the  
> proper kmem.limit\_in\_bytes handling which is a single place).

The distinction between "trusted" and "untrusted" is something artificially created due to the assumed deficiency of kmemcg implementation. Making things like this visible to userland is a bad idea because it locks us into a place where we can't or don't need to improve the said deficiencies and end up pushing the difficult problems to somewhere else where it will likely be implemented in a shabbier way. There sure are cases when such approach simply cannot be avoided, but I really don't think that's the case here - the overhead already seems to be at an acceptable level and we're not taking away the escape switch.

This is userland visible API. We better err on the side of being conservative than going overboard with flexibility. Even if we eventually need to make this switching fully hierarchical, we really should be doing,

1. Implement simple global switching and look for problem cases.

2. Analyze them and see whether the problem case can't be solved in a better, more intelligent way.
3. If the problem is something structurally inherent or reasonably too difficult to solve any other way, consider dumping the problem as config parameters to userland.

We can always expand the flexibility. Let's do the simple thing first. As an added bonus, it would enable using static\_keys for accounting branches too.

Thanks.

--

tejun

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Mel Gorman](#) on Thu, 27 Sep 2012 14:43:07 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Thu, Sep 27, 2012 at 07:33:00AM -0700, Tejun Heo wrote:

> Hello, Michal.

>

> On Thu, Sep 27, 2012 at 02:08:06PM +0200, Michal Hocko wrote:

> > Yes, because we have many users (basically almost all) who care only  
> > about the user memory because that's what occupies the vast majority of  
> > the memory. They usually want to isolate workload which would disrupt  
> > the global memory otherwise (e.g. backup process vs. database). You  
> > really do not want to pay an additional overhead for kmem accounting  
> > here.

>

> I'm not too convinced. First of all, the overhead added by kmemcg  
> isn't big.

Really?

If kmemcg was globally accounted then every \_\_GFP\_KMEMCG allocation in the page allocator potentially ends up down in \_\_memcg\_kmem\_newpage\_charge which

1. takes RCU read lock
2. looks up cgroup from task
3. takes a reference count
4. memcg\_charge\_kmem -> \_\_mem\_cgroup\_try\_charge
5. release reference count

That's a \*LOT\* of work to incur for cgroups that do not care about kernel

accounting. This is why I thought it was reasonable that the kmem accounting not be global.

--

Mel Gorman  
SUSE Labs

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Thu, 27 Sep 2012 14:49:42 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello, Mel.

On Thu, Sep 27, 2012 at 03:28:22PM +0100, Mel Gorman wrote:

> > In addition, how is userland supposed to know which  
> > workload is shared kmem heavy or not?  
>  
> By using a bit of common sense.  
>  
> An application may not be able to figure this out but the administrator  
> is going to be able to make a very educated guess. If processes running  
> within two containers are not sharing a filesystem hierarchy for example  
> then it'll be clear they are not sharing dentries.  
>  
> If there was a suspicion they were then it could be analysed with  
> something like SystemTap probing when files are opened and see if files  
> are being opened that are shared between containers.  
>  
> It's not super-easy but it's not impossible either and I fail to see why  
> it's such a big deal for you.

Because we're not even trying to actually solve the problem but just dumping it to userland. If dentry/inode usage is the only case we're being worried about, there can be better ways to solve it or at least we should strive for that.

Also, the problem is not that it is impossible if you know and carefully plan for things beforehand (that would be one extremely competent admin) but that the problem is undiscoverable. With kmemcg accounting disabled, there's no way to tell a looking cgroup the admin thinks running something which doesn't tax kmem much could be generating a ton without the admin ever noticing.

> > The fact that the numbers don't really mean what they apparently  
> > should mean.  
>  
> I think it is a reasonable limitation that only some kernel allocations are

> accounted for although I'll freely admit I'm not a cgroup or memcg user  
> either.  
>  
> My understanding is that this comes down to cost -- accounting for the  
> kernel memory usage is expensive so it is limited only to the allocations  
> that are easy to abuse by an unprivileged process. Hence this is  
> initially concerned with stack pages with dentries and TCP usage to  
> follow in later patches.

I think the cost isn't too prohibitive considering it's already using memcg. Charging / uncharging happens only as pages enter and leave slab caches and the hot path overhead is essentially single indirection. Glauber's benchmark seemed pretty reasonable to me and I don't yet think that warrants exposing this subtle tree of configuration.

> > Sure, conferences are useful for building consensus but that's the  
> > extent of it. Sorry that I didn't realize the implications then but  
> > conferences don't really add any finality to decisions.  
> >  
> > So, this seems properly crazy to me at the similar level of  
> > use\_hierarchy fiasco. I'm gonna NACK on this.  
>  
> I think you're over-reacting to say the very least :|

The part I nacked is enabling kmemcg on a populated cgroup and then starting accounting from then without any apparent indication that any past allocation hasn't been considered. You end up with numbers which nobody can't tell what they really mean and there's no mechanism to guarantee any kind of ordering between populating the cgroup and configuring it and there's \*no\* way to find out what happened afterwards neither. This is properly crazy and definitely deserves a nack.

Thanks.

--  
tejun

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Thu, 27 Sep 2012 14:57:59 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/27/2012 06:49 PM, Tejun Heo wrote:

> Hello, Mel.

>

> On Thu, Sep 27, 2012 at 03:28:22PM +0100, Mel Gorman wrote:

>>> In addition, how is userland supposed to know which  
>>> workload is shared kmem heavy or not?  
>>  
>> By using a bit of common sense.  
>>  
>> An application may not be able to figure this out but the administrator  
>> is going to be able to make a very educated guess. If processes running  
>> within two containers are not sharing a filesystem hierarchy for example  
>> then it'll be clear they are not sharing dentries.  
>>  
>> If there was a suspicion they were then it could be analysed with  
>> something like SystemTap probing when files are opened and see if files  
>> are being opened that are shared between containers.  
>>  
>> It's not super-easy but it's not impossible either and I fail to see why  
>> it's such a big deal for you.  
>  
> Because we're not even trying to actually solve the problem but just  
> dumping it to userland. If dentry/inode usage is the only case we're  
> being worried about, there can be better ways to solve it or at least  
> we should strive for that.  
>

Not only it is not the only case we care about, this is not even touched in this series. (It is only touched in the next one). This one, for instance, cares about the stack. The reason everything is being dumped into "kmem", is precisely to make things simpler. I argue that at some point it makes sense to draw a line, and "kmem" is a much better line than any fine grained control - precisely because it is conceptually easier to grasp.

> Also, the problem is not that it is impossible if you know and  
> carefully plan for things beforehand (that would be one extremely  
> competent admin) but that the problem is undiscoverable. With kmemcg  
> accounting disabled, there's no way to tell a looking cgroup the admin  
> thinks running something which doesn't tax kmem much could be  
> generating a ton without the admin ever noticing.  
>  
>>> The fact that the numbers don't really mean what they apparently  
>>> should mean.  
>>  
>> I think it is a reasonable limitation that only some kernel allocations are  
>> accounted for although I'll freely admit I'm not a cgroup or memcg user  
>> either.  
>>  
>> My understanding is that this comes down to cost -- accounting for the  
>> kernel memory usage is expensive so it is limited only to the allocations

>> that are easy to abuse by an unprivileged process. Hence this is  
>> initially concerned with stack pages with dentries and TCP usage to  
>> follow in later patches.  
>  
> I think the cost isn't too prohibitive considering it's already using  
> memcg. Charging / uncharging happens only as pages enter and leave  
> slab caches and the hot path overhead is essentially single  
> indirection. Glauber's benchmark seemed pretty reasonable to me and I  
> don't yet think that warrants exposing this subtle tree of  
> configuration.  
>

Only so we can get some numbers: the cost is really minor if this is all disabled. If this is fully enabled, it can get to some 2 or 3 %, which may or may not be acceptable to an application. But for me this is not even about cost, and that's why I haven't brought it up so far

>>> Sure, conferences are useful for building consensus but that's the  
>>> extent of it. Sorry that I didn't realize the implications then but  
>>> conferences don't really add any finality to decisions.

>>>

>>> So, this seems properly crazy to me at the similar level of  
>>> use\_hierarchy fiasco. I'm gonna NACK on this.

>>

>> I think you're over-reacting to say the very least :|

>

> The part I nacked is enabling kmemcg on a populated cgroup and then  
> starting accounting from then without any apparent indication that any  
> past allocation hasn't been considered. You end up with numbers which  
> nobody can't tell what they really mean and there's no mechanism to  
> guarantee any kind of ordering between populating the cgroup and  
> configuring it and there's \*no\* way to find out what happened  
> afterwards neither. This is properly crazy and definitely deserves a  
> nack.

>

Mel suggestion of not allowing this to happen once the cgroup has tasks takes care of this, and is something I thought of myself.

This would remove this particular piece of objection, and maintain the per-subtree control.

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure

Posted by [Tejun Heo](#) on Thu, 27 Sep 2012 14:58:02 GMT

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

Hello, Mel.

On Thu, Sep 27, 2012 at 03:43:07PM +0100, Mel Gorman wrote:

> > I'm not too convinced. First of all, the overhead added by kmemcg  
> > isn't big.  
>  
> Really?  
>  
> If kmemcg was globally accounted then every \_\_GFP\_KMEMCG allocation in  
> the page allocator potentially ends up down in  
> \_\_memcg\_kmem\_newpage\_charge which  
>  
> 1. takes RCU read lock  
> 2. looks up cgroup from task  
> 3. takes a reference count  
> 4. memcg\_charge\_kmem -> \_\_mem\_cgroup\_try\_charge  
> 5. release reference count  
>  
> That's a \*LOT\* of work to incur for cgroups that do not care about kernel  
> accounting. This is why I thought it was reasonable that the kmem accounting  
> not be global.

But that happens only when pages enter and leave slab and if it still is significant, we can try to further optimize charging. Given that this is only for cases where memcg is already in use and we provide a switch to disable it globally, I really don't think this warrants implementing fully hierarchy configuration.

Thanks.

--

tejun

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Michal Hocko](#) on Thu, 27 Sep 2012 15:09:50 GMT

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

---

On Thu 27-09-12 07:33:00, Tejun Heo wrote:

> Hello, Michal.  
>  
> On Thu, Sep 27, 2012 at 02:08:06PM +0200, Michal Hocko wrote:  
> > Yes, because we have many users (basically almost all) who care only  
> > about the user memory because that's what occupies the vast majority of  
> > the memory. They usually want to isolate workload which would disrupt  
> > the global memory otherwise (e.g. backup process vs. database). You  
> > really do not want to pay an additional overhead for kmem accounting  
> > here.  
>



> I'm not too convinced. First of all, the overhead added by kmemcg  
> isn't big.

You are probably talking about memory overhead which is indeed not that big (except for possible side effects as fragmentation which you mention below). But the runtime overhead is present, as far as I understand from what Glauber said. But, on the other hand, it is fair to say that those who `_want_` use the feature should pay for it.

> The hot path overhead is quite minimal - it doesn't do much more than  
> indirecting one more time. In terms of memory usage, it sure could  
> lead to a bit more fragmentation but even if it gets to several megs  
> per cgroup, I don't think that's something excessive. So, there is  
> overhead but I don't believe it to be prohibitive.

Remember that users do not want to pay even "something minimal" when the feature is not needed.

> > > So your question for global vs local switch (that again, doesn't  
> > > exist; only a local *\*limit\** exists) should really be posed in the  
> > > following way: "Can two different use cases with different needs be  
> > > hosted in the same box?"  
> >  
> > I think this is a good and a relevant question. I think this boils down  
> > to whether you want to have trusted and untrusted workloads at the same  
> > machine.  
> > Trusted loads usually only need user memory accounting because kmem  
> > consumption should be really negligible (unless kernel is doing  
> > something really stupid and no kmem limit will help here).  
> > On the other hand, untrusted workloads can do nasty things that  
> > administrator has hard time to mitigate and setting a kmem limit can  
> > help significantly.  
> >  
> > IMHO such a different loads exist on a single machine quite often (Web  
> > server and a back up process as the most simplistic one). The per  
> > hierarchy accounting, therefore, sounds like a good idea without too  
> > much added complexity (actually the only added complexity is in the  
> > proper `kmem.limit_in_bytes` handling which is a single place).  
>  
> The distinction between "trusted" and "untrusted" is something  
> artificially created due to the assumed deficiency of kmemcg  
> implementation.

Not really. It doesn't have to do anything with the overhead (be it memory or runtime). It really boils down to "do I need/want it at all". Why would I want to think about how much kernel memory is in use in the first place? Or do you think that user memory accounting should be deprecated?

> Making things like this visible to userland is a bad  
> idea because it locks us into a place where we can't or don't need to  
> improve the said deficiencies and end up pushing the difficult  
> problems to somewhere else where it will likely be implemented in a  
> shabbier way. There sure are cases when such approach simply cannot  
> be avoided, but I really don't think that's the case here - the  
> overhead already seems to be at an acceptable level and we're not  
> taking away the escape switch.  
>  
> This is userland visible API.

I am not sure which API visible part you have in mind but  
kmem.limit\_in\_bytes will be there whether we go with global knob or "no  
limit no accounting" approach.

> We better err on the side of being conservative than going overboard  
> with flexibility. Even if we eventually need to make this switching  
> fully hierarchical, we really should be doing,  
>  
> 1. Implement simple global switching and look for problem cases.  
>  
> 2. Analyze them and see whether the problem case can't be solved in a  
> better, more intelligent way.  
>  
> 3. If the problem is something structurally inherent or reasonably too  
> difficult to solve any other way, consider dumping the problem as  
> config parameters to userland.  
>  
> We can always expand the flexibility. Let's do the simple thing  
> first. As an added bonus, it would enable using static\_keys for  
> accounting branches too.

While I do agree with you in general and being careful is at place in  
this area as time shown several times, this seems to be too restrictive  
in this particular case.

We won't save almost no code with the global knob so I am not sure  
what we are actually saving here. Global knob will just give us all or  
nothing semantic without making the whole thing simpler. You will stick  
with static branches and check whether the group is accountable anyway,  
right?

--

Michal Hocko  
SUSE Labs

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure

Hello,

On Thu, Sep 27, 2012 at 06:57:59PM +0400, Glauber Costa wrote:

> > Because we're not even trying to actually solve the problem but just  
> > dumping it to userland. If dentry/inode usage is the only case we're  
> > being worried about, there can be better ways to solve it or at least  
> > we should strive for that.

>

> Not only it is not the only case we care about, this is not even touched  
> in this series. (It is only touched in the next one). This one, for  
> instance, cares about the stack. The reason everything is being dumped  
> into "kmem", is precisely to make things simpler. I argue that at some  
> point it makes sense to draw a line, and "kmem" is a much better line  
> than any fine grained control - precisely because it is conceptually  
> easier to grasp.

Can you please give other examples of cases where this type of issue exists (plenty of shared kernel data structure which is inherent to the workload at hand)? Until now, this has been the only example for this type of issues.

> > I think the cost isn't too prohibitive considering it's already using  
> > memcg. Charging / uncharging happens only as pages enter and leave  
> > slab caches and the hot path overhead is essentially single  
> > indirection. Glauber's benchmark seemed pretty reasonable to me and I  
> > don't yet think that warrants exposing this subtle tree of  
> > configuration.

>

> Only so we can get some numbers: the cost is really minor if this is all  
> disabled. If this is fully enable, it can get to some 2 or 3 %, which  
> may or may not be acceptable to an application. But for me this is not  
> even about cost, and that's why I haven't brought it up so far

It seems like Mel's concern is mostly based on performance overhead concerns tho.

> > The part I nacked is enabling kmemcg on a populated cgroup and then  
> > starting accounting from then without any apparent indication that any  
> > past allocation hasn't been considered. You end up with numbers which  
> > nobody can't tell what they really mean and there's no mechanism to  
> > guarantee any kind of ordering between populating the cgroup and  
> > configuring it and there's \*no\* way to find out what happened  
> > afterwards neither. This is properly crazy and definitely deserves a  
> > nack.

> >

>

> Mel suggestion of not allowing this to happen once the cgroup has tasks  
> takes care of this, and is something I thought of myself.

You mean Michal's? It should also disallow switching if there are children cgroups, right?

> This would remove this particular piece of objection, and maintain the  
> per-subtree control.

Yeah, I don't see anything broken with that although I'll try to argue for simpler one a bit more.

Thanks.

--  
tejun

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Michal Hocko](#) on Thu, 27 Sep 2012 17:56:43 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Thu 27-09-12 10:46:05, Tejun Heo wrote:

[...]

> > > The part I nacked is enabling kmemcg on a populated cgroup and then  
> > > starting accounting from then without any apparent indication that any  
> > > past allocation hasn't been considered. You end up with numbers which  
> > > nobody can't tell what they really mean and there's no mechanism to  
> > > guarantee any kind of ordering between populating the cgroup and  
> > > configuring it and there's \*no\* way to find out what happened  
> > > afterwards neither. This is properly crazy and definitely deserves a  
> > > nack.

> > >

> >

> > Mel suggestion of not allowing this to happen once the cgroup has tasks  
> > takes care of this, and is something I thought of myself.

>

> You mean Michal's? It should also disallow switching if there are  
> children cgroups, right?

Right.

--  
Michal Hocko  
SUSE Labs

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Thu, 27 Sep 2012 18:30:36 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/27/2012 06:58 PM, Tejun Heo wrote:

> Hello, Mel.

>

> On Thu, Sep 27, 2012 at 03:43:07PM +0100, Mel Gorman wrote:

>>> I'm not too convinced. First of all, the overhead added by kmemcg  
>>> isn't big.

>>

>> Really?

>>

>> If kmemcg was globally accounted then every \_\_GFP\_KMEMCG allocation in  
>> the page allocator potentially ends up down in  
>> \_\_memcg\_kmem\_newpage\_charge which

>>

>> 1. takes RCU read lock

>> 2. looks up cgroup from task

>> 3. takes a reference count

>> 4. memcg\_charge\_kmem -> \_\_mem\_cgroup\_try\_charge

>> 5. release reference count

>>

>> That's a \*LOT\* of work to incur for cgroups that do not care about kernel  
>> accounting. This is why I thought it was reasonable that the kmem accounting  
>> not be global.

>

> But that happens only when pages enter and leave slab and if it still

> is significant, we can try to further optimize charging. Given that

> this is only for cases where memcg is already in use and we provide a

> switch to disable it globally, I really don't think this warrants

> implementing fully hierarchy configuration.

>

Not totally true. We still have to match every allocation to the right  
cache, and that is actually our heaviest hit, responsible for the 2, 3 %  
we're seeing when this is enabled. It is the kind of path so hot that  
people frown upon branches being added, so I don't think we'll ever get  
this close to being free.

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Thu, 27 Sep 2012 18:45:01 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/27/2012 09:46 PM, Tejun Heo wrote:

> Hello,

>

> On Thu, Sep 27, 2012 at 06:57:59PM +0400, Glauber Costa wrote:  
>>> Because we're not even trying to actually solve the problem but just  
>>> dumping it to userland. If dentry/inode usage is the only case we're  
>>> being worried about, there can be better ways to solve it or at least  
>>> we should strive for that.  
>>  
>> Not only it is not the only case we care about, this is not even touched  
>> in this series. (It is only touched in the next one). This one, for  
>> instance, cares about the stack. The reason everything is being dumped  
>> into "kmem", is precisely to make things simpler. I argue that at some  
>> point it makes sense to draw a line, and "kmem" is a much better line  
>> than any fine grained control - precisely because it is conceptually  
>> easier to grasp.  
>  
> Can you please give other examples of cases where this type of issue  
> exists (plenty of shared kernel data structure which is inherent to  
> the workload at hand)? Until now, this has been the only example for  
> this type of issues.  
>

Yes. the namespace related caches (\*), all kinds of sockets and network structures, other file system structures like file struct, vm areas, and pretty much everything a full container does.

(\*) we run full userspace, so we have namespaces + cgroups combination.

>>> I think the cost isn't too prohibitive considering it's already using  
>>> memcg. Charging / uncharging happens only as pages enter and leave  
>>> slab caches and the hot path overhead is essentially single  
>>> indirection. Glauber's benchmark seemed pretty reasonable to me and I  
>>> don't yet think that warrants exposing this subtle tree of  
>>> configuration.  
>>  
>> Only so we can get some numbers: the cost is really minor if this is all  
>> disabled. If this is fully enable, it can get to some 2 or 3 %, which  
>> may or may not be acceptable to an application. But for me this is not  
>> even about cost, and that's why I haven't brought it up so far  
>  
> It seems like Mel's concern is mostly based on performance overhead  
> concerns tho.  
>  
>>> The part I nacked is enabling kmemcg on a populated cgroup and then  
>>> starting accounting from then without any apparent indication that any  
>>> past allocation hasn't been considered. You end up with numbers which  
>>> nobody can't tell what they really mean and there's no mechanism to  
>>> guarantee any kind of ordering between populating the cgroup and  
>>> configuring it and there's \*no\* way to find out what happened  
>>> afterwards neither. This is properly crazy and definitely deserves a

>>> nack.  
>>>  
>>  
>> Mel suggestion of not allowing this to happen once the cgroup has tasks  
>> takes care of this, and is something I thought of myself.  
>  
> You mean Michal's? It should also disallow switching if there are  
> children cgroups, right?  
>

No, I meant Mel, quoting this:

"Further I would expect that an administrator would be aware of these limitations and set kmem\_accounting at cgroup creation time before any processes start. Maybe that should be enforced but it's not a fundamental problem."

But I guess it is pretty much the same thing Michal proposes, in essence.

Or IOW, if your concern is with the fact that charges may have happened in the past before this is enabled, we can make sure this cannot happen by disallowing the limit to be set if currently unset (value changes are obviously fine) if you have children or any tasks already in the group.

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Sun, 30 Sep 2012 07:57:00 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello, Glauber.

On Thu, Sep 27, 2012 at 10:45:01PM +0400, Glauber Costa wrote:

> > Can you please give other examples of cases where this type of issue  
> > exists (plenty of shared kernel data structure which is inherent to  
> > the workload at hand)? Until now, this has been the only example for  
> > this type of issues.  
>  
> Yes. the namespace related caches (\*), all kinds of sockets and network  
> structures, other file system structures like file struct, vm areas, and  
> pretty much everything a full container does.  
>  
> (\*) we run full userspace, so we have namespaces + cgroups combination.

This is probably me being dumb but wouldn't resources used by full namespaces be mostly independent? Which parts get shared? Also, if you do full namespace, isn't it more likely that you would want fuller resource isolation too?

> >> Mel suggestion of not allowing this to happen once the cgroup has tasks  
> >> takes care of this, and is something I thought of myself.  
> >  
> > You mean Michal's? It should also disallow switching if there are  
> > children cgroups, right?  
>  
> No, I meant Mel, quoting this:  
>  
> "Further I would expect that an administrator would be aware of these  
> limitations and set kmem\_accounting at cgroup creation time before any  
> processes start. Maybe that should be enforced but it's not a  
> fundamental problem."  
>  
> But I guess it is pretty much the same thing Michal proposes, in essence.  
>  
> Or IOW, if your concern is with the fact that charges may have happened  
> in the past before this is enabled, we can make sure this cannot happen  
> by disallowing the limit to be set if currently unset (value changes are  
> obviously fine) if you have children or any tasks already in the group.

Yeah, please do that.

Thanks.

--  
tejun

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Sun, 30 Sep 2012 08:02:49 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Sun, Sep 30, 2012 at 04:57:00PM +0900, Tejun Heo wrote:  
> On Thu, Sep 27, 2012 at 10:45:01PM +0400, Glauber Costa wrote:  
> > > Can you please give other examples of cases where this type of issue  
> > > exists (plenty of shared kernel data structure which is inherent to  
> > > the workload at hand)? Until now, this has been the only example for  
> > > this type of issues.  
> >  
> > Yes. the namespace related caches (\*), all kinds of sockets and network  
> > structures, other file system structures like file struct, vm areas, and  
> > pretty much everything a full container does.  
> >  
> > (\*) we run full userspace, so we have namespaces + cgroups combination.  
>  
> This is probably me being dumb but wouldn't resources used by full  
> namespaces be mostly independent? Which parts get shared? Also, if  
> you do full namespace, isn't it more likely that you would want fuller



> resource isolation too?

Just a thought about dentry/inode. Would it make sense to count total number of references per cgroup and charge the total amount according to that? Reference counts are how the shared ownership is represented after all. Counting total per cgroup isn't accurate and pathological cases could be weird tho.

Thanks.

--  
tejun

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Sun, 30 Sep 2012 08:23:58 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello, Glauber.

On Thu, Sep 27, 2012 at 10:30:36PM +0400, Glauber Costa wrote:

> > But that happens only when pages enter and leave slab and if it still  
> > is significant, we can try to further optimize charging. Given that  
> > this is only for cases where memcg is already in use and we provide a  
> > switch to disable it globally, I really don't think this warrants  
> > implementing fully hierarchy configuration.

>  
> Not totally true. We still have to match every allocation to the right  
> cache, and that is actually our heaviest hit, responsible for the 2, 3 %  
> we're seeing when this is enabled. It is the kind of path so hot that  
> people frown upon branches being added, so I don't think we'll ever get  
> this close to being free.

Sure, depending on workload, any addition to alloc/free could be noticeable. I don't know. I'll write more about it when replying to Michal's message. BTW, `__memcg_kmem_get_cache()` does seem a bit heavy. I wonder whether indexing from cache side would make it cheaper? e.g. something like the following.

```
kmem_cache *__memcg_kmem_get_cache(cachep, gfp)
{
    struct kmem_cache *c;

    c = cachep->memcg_params->caches[percpu_read(kmemcg_slab_idx)];
    if (likely(c))
        return c;
    /* try to create and then fall back to cachep */
}
```

where kmemcg\_slab\_idx is updated from sched notifier (or maybe add and use current->kmemcg\_slab\_idx?). You would still need \_\_GFP\_\* and in\_interrupt() tests but current->mm and PF\_KTHREAD tests can be rolled into index selection.

Thanks.

--  
tejun

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Sun, 30 Sep 2012 08:47:50 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello, Michal.

On Thu, Sep 27, 2012 at 05:09:50PM +0200, Michal Hocko wrote:

> On Thu 27-09-12 07:33:00, Tejun Heo wrote:

> > I'm not too convinced. First of all, the overhead added by kmemcg  
> > isn't big.

>

> You are probably talking about memory overhead which is indeed not that  
> big (except for possible side effects as fragmentation which you mention  
> below). But the runtime overhead is present, as far as I understand from  
> what Glauber said. But, on the other hand, it is fair to say that those  
> who want use the feature should pay for it.

Yeah, as long as the overhead is reasonable and it doesn't affect non-users, I think we should put more emphasis on simplicity. cgroup is pretty hairy (from both implementation and interface POVs) to begin with. Unfortunately, what's reasonable or how much more emphasis varies widely depending on who one asks.

> > The hot path overhead is quite minimal - it doesn't do much more than  
> > indirecting one more time. In terms of memory usage, it sure could  
> > lead to a bit more fragmentation but even if it gets to several megs  
> > per cgroup, I don't think that's something excessive. So, there is  
> > overhead but I don't believe it to be prohibitive.

>

> Remember that users do not want to pay even "something minimal" when the  
> feature is not needed.

Yeah but, if we can get it down to, say, around 1% under most workloads for memcg users, it is quite questionable to introduce full hierarchical configuration to allow avoiding that, isn't it?

> > The distinction between "trusted" and "untrusted" is something  
> > artificially created due to the assumed deficiency of kmemcg  
> > implementation.  
>  
> Not really. It doesn't have to do anything with the overhead (be it  
> memory or runtime). It really boils down to "do I need/want it at all".  
> Why would I want to think about how much kernel memory is in use in the  
> first place? Or do you think that user memory accounting should be  
> deprecated?

But you can apply the same "do I need/want it at all" question to the configuration parameter too. I can see your point but the decision seems muddy to me, and if muddy, I prefer to err on the side of being too conservative.

> > This is userland visible API.  
>  
> I am not sure which API visible part you have in mind but  
> kmem.limit\_in\_bytes will be there whether we go with global knob or "no  
> limit no accounting" approach.

I mean full hierarchical configuration of it. It becomes something which each memcg user cares about instead of something which the base system / admin flips on system boot.

> > We can always expand the flexibility. Let's do the simple thing  
> > first. As an added bonus, it would enable using static\_keys for  
> > accounting branches too.  
>  
> While I do agree with you in general and being careful is at place in  
> this area as time shown several times, this seems to be too restrictive  
> in this particular case.  
> We won't save almost no code with the global knob so I am not sure  
> what we are actually saving here. Global knob will just give us all or  
> nothing semantic without making the whole thing simpler. You will stick  
> with static branches and check whether the group is accountable anyway,  
> right?

The thing is about the same argument can be made about .use\_hierarchy too. It doesn't necessarily make the code much harder. Especially because the code is structured with that feature on mind, removing .use\_hierarchy might not remove whole lot of code; however, the wider range of behavior which got exposed through that poses a much larger problem when we try to make modifications on related behaviors. We get a lot more locked down by seemingly not too much code and our long term maintainability / sustainability suffers as a result.

I'm not trying to say this is as bad as .use\_hierarchy but want to

point out that memcg and cgroup in general have had pretty strong tendency to choose overly flexible and complex designs and interfaces and it's probably about time we become more careful especially about stuff which is visible to userland.

Thanks.

--

tejun

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure

Posted by [James Bottomley](#) on Sun, 30 Sep 2012 08:56:28 GMT

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

---

On Sun, 2012-09-30 at 17:02 +0900, Tejun Heo wrote:

> On Sun, Sep 30, 2012 at 04:57:00PM +0900, Tejun Heo wrote:

> > On Thu, Sep 27, 2012 at 10:45:01PM +0400, Glauber Costa wrote:

> > > Can you please give other examples of cases where this type of issue  
> > > exists (plenty of shared kernel data structure which is inherent to  
> > > the workload at hand)? Until now, this has been the only example for  
> > > this type of issues.

> > >

> > > Yes. the namespace related caches (\*), all kinds of sockets and network  
> > > structures, other file system structures like file struct, vm areas, and  
> > > pretty much everything a full container does.

> > >

> > > (\*) we run full userspace, so we have namespaces + cgroups combination.

> >

> > This is probably me being dumb but wouldn't resources used by full  
> > namespaces be mostly independent? Which parts get shared? Also, if  
> > you do full namespace, isn't it more likely that you would want fuller  
> > resource isolation too?

>

> Just a thought about dentry/inode. Would it make sense to count total  
> number of references per cgroup and charge the total amount according  
> to that? Reference counts are how the shared ownership is represented  
> after all. Counting total per cgroup isn't accurate and pathological  
> cases could be weird tho.

The beancounter approach originally used by OpenVZ does exactly this. There are two specific problems, though, firstly you can't count references in generic code, so now you have to extend the cgroup tentacles into every object, an invasiveness which people didn't really like. Secondly split accounting causes oddities too, like your total kernel memory usage can appear to go down even though you do nothing just because someone else added a share. Worse, if someone drops the reference, your usage can go up, even though you did nothing, and push

you over your limit, at which point action gets taken against the container. This leads to nasty system unpredictability (The whole point of cgroup isolation is supposed to be preventing resource usage in one cgroup from affecting that in another).

We discussed this pretty heavily at the Containers Mini Summit in Santa Rosa. The emergent consensus was that no-one really likes first use accounting, but it does solve all the problems and it has the fewest unexpected side effects.

If you have an alternative that wasn't considered then, I'm sure everyone would be interested, but it isn't split accounting.

James

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Sun, 30 Sep 2012 10:37:32 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello, James.

On Sun, Sep 30, 2012 at 09:56:28AM +0100, James Bottomley wrote:  
> The beancounter approach originally used by OpenVZ does exactly this.  
> There are two specific problems, though, firstly you can't count  
> references in generic code, so now you have to extend the cgroup  
> tentacles into every object, an invasiveness which people didn't really  
> like.

Yeah, it will need some hooks. For dentry and inode, I think it would be pretty well isolated tho. Wasn't it?

> Secondly split accounting causes oddities too, like your total  
> kernel memory usage can appear to go down even though you do nothing  
> just because someone else added a share. Worse, if someone drops the  
> reference, your usage can go up, even though you did nothing, and push  
> you over your limit, at which point action gets taken against the  
> container. This leads to nasty system unpredictability (The whole point  
> of cgroup isolation is supposed to be preventing resource usage in one  
> cgroup from affecting that in another).

In a sense, the fluctuating amount is the actual resource burden the cgroup is putting on the system, so maybe it just needs to be handled better or maybe we should charge fixed amount per refcnt? I don't know.

> We discussed this pretty heavily at the Containers Mini Summit in Santa  
> Rosa. The emergent consensus was that no-one really likes first use

> accounting, but it does solve all the problems and it has the fewest  
> unexpected side effects.

But that's like fitting the problem to the mechanism. Maybe that is the best which can be done, but the side effect there is way-off accounting under pretty common workload, which sounds pretty nasty to me.

Thanks.

--

tejun

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [James Bottomley](#) on Sun, 30 Sep 2012 11:25:52 GMT

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

---

On Sun, 2012-09-30 at 19:37 +0900, Tejun Heo wrote:

> Hello, James.

>

> On Sun, Sep 30, 2012 at 09:56:28AM +0100, James Bottomley wrote:

> > The beancounter approach originally used by OpenVZ does exactly this.

> > There are two specific problems, though, firstly you can't count

> > references in generic code, so now you have to extend the cgroup

> > tentacles into every object, an invasiveness which people didn't really

> > like.

>

> Yeah, it will need some hooks. For dentry and inode, I think it would

> be pretty well isolated tho. Wasn't it?

But you've got to ask yourself who cares about accurate accounting per container of dentry and inode objects? They're not objects that any administrator is used to limiting. What we at parallels care about isn't accurately accounting them, it's that one container can't DoS another by exhausting system resources. That's achieved equally well by first charge slab accounting, so we don't really have an interest in pushing object accounting code for which there's no use case.

> > Secondly split accounting causes oddities too, like your total

> > kernel memory usage can appear to go down even though you do nothing

> > just because someone else added a share. Worse, if someone drops the

> > reference, your usage can go up, even though you did nothing, and push

> > you over your limit, at which point action gets taken against the

> > container. This leads to nasty system unpredictability (The whole point

> > of cgroup isolation is supposed to be preventing resource usage in one

> > cgroup from affecting that in another).

>

> In a sense, the fluctuating amount is the actual resource burden the  
> cgroup is putting on the system, so maybe it just needs to be handled  
> better or maybe we should charge fixed amount per refcnt? I don't  
> know.

Yes, we considered single charge per reference accounting as well (although I don't believe anyone went as far as producing an implementation). The problem here is now that the sum of the container resources no-longer bears any relation to the host consumption. This makes it very difficult for virtualisation orchestration systems to make accurate decisions when doing dynamic resource scheduling (DRS).

Conversely, as ugly as you think it, first use accounting is actually pretty good at identifying problem containers (at least with regard to memory usage) for DRS because containers which are stretching the memory tend to accumulate the greatest number of first charges over the system lifetime.

> > We discussed this pretty heavily at the Containers Mini Summit in Santa  
> > Rosa. The emergent consensus was that no-one really likes first use  
> > accounting, but it does solve all the problems and it has the fewest  
> > unexpected side effects.  
>  
> But that's like fitting the problem to the mechanism. Maybe that is  
> the best which can be done, but the side effect there is way-off  
> accounting under pretty common workload, which sounds pretty nasty to  
> me.

All we need kernel memory accounting and limiting for is DoS prevention. There aren't really any system administrators who care about Kernel Memory accounting (at least until the system goes oom) because there are no absolute knobs for it (all there is are a set of weird and wonderful heuristics, like dirty limit ratio and drop caches). Kernel memory usage has a whole set of regulatory infrastructure for trying to make it transparent to the user.

Don't get me wrong: if there were some easy way to get proper memory accounting for free, we'd be happy but, because it has no practical application for any of our customers, there's a limited price we're willing to pay to get it.

James

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Mon, 01 Oct 2012 00:57:17 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---



Hello, James.

On Sun, Sep 30, 2012 at 12:25:52PM +0100, James Bottomley wrote:

- > But you've got to ask yourself who cares about accurate accounting per
- > container of dentry and inode objects? They're not objects that any
- > administrator is used to limiting. What we at parallels care about
- > isn't accurately accounting them, it's that one container can't DoS
- > another by exhausting system resources. That's achieved equally well by
- > first charge slab accounting, so we don't really have an interest in
- > pushing object accounting code for which there's no use case.

Isn't it more because the use cases you have on mind don't share dentries/inodes too much? Wildly incorrect accounting definitely degrades container isolation and can lead to unexpected behaviors.

- > All we need kernel memory accounting and limiting for is DoS prevention.
- > There aren't really any system administrators who care about Kernel
- > Memory accounting (at least until the system goes oom) because there are
- > no absolute knobs for it (all there is are a set of weird and wonderful
- > heuristics, like dirty limit ratio and drop caches). Kernel memory

I think that's because the mechanism currently doesn't exist. If one wants to control how memory is distributed across different cgroups, it's logical to control kernel memory too. The resource in question is the actual memory after all. I think at least google would be interested in it, so, no, I don't agree that nobody wants it. If that is the case, we're working towards the wrong direction.

- > usage has a whole set of regulatory infrastructure for trying to make it
- > transparent to the user.
- >
- > Don't get me wrong: if there were some easy way to get proper memory
- > accounting for free, we'd be happy but, because it has no practical
- > application for any of our customers, there's a limited price we're
- > willing to pay to get it.

Even on purely technical ground, it could be that first-use is the right trade off if other more accurate approaches are too difficult and most workloads are happy with such approach. I'm still a bit weary to base userland interface decisions on that tho.

Thanks.

--  
tejun



Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Mon, 01 Oct 2012 08:36:37 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/30/2012 11:57 AM, Tejun Heo wrote:

> Hello, Glauber.

>

> On Thu, Sep 27, 2012 at 10:45:01PM +0400, Glauber Costa wrote:

>>> Can you please give other examples of cases where this type of issue  
>>> exists (plenty of shared kernel data structure which is inherent to  
>>> the workload at hand)? Until now, this has been the only example for  
>>> this type of issues.

>>

>> Yes. the namespace related caches (\*), all kinds of sockets and network  
>> structures, other file system structures like file struct, vm areas, and  
>> pretty much everything a full container does.

>>

>> (\*) we run full userspace, so we have namespaces + cgroups combination.

>

> This is probably me being dumb but wouldn't resources used by full  
> namespaces be mostly independent? Which parts get shared? Also, if  
> you do full namespace, isn't it more likely that you would want fuller  
> resource isolation too?

>

Not necessarily. Namespaces are pretty flexible. If you are using the network namespace, for instance, you can create interfaces, routes, addresses, etc. But because this deals with the network only, there is nothing unreasonable in saying that your webserver and database lives in the same network (which is a different network namespace), but are entitled to different memory limits - which is cgroups realm. With application-only containers being championed these days by multiple users, I would expect this situation to become non-negligible.

The full container scenario, of course, is very different. Most of the accesses tends to be local.

I will second what Michal said, since I believe this is also very important: User memory is completely at the control of the application, while kernel memory is not, and never will. It is perfectly fine to imagine applications that want its memory to be limited by a very predictable amount, and there are no reasons to believe that those cannot live in the same box as full containers - the biggest example of kmem interested folks. It could be, for instance, that a management tool for containers lives in there, and that application wants to be umem limited but not kmem limited. (If it goes touching files and data inside each container, for instance, it is obviously not local)

>>>> Mel suggestion of not allowing this to happen once the cgroup has tasks

>>>> takes care of this, and is something I thought of myself.  
>>>  
>>> You mean Michal's? It should also disallow switching if there are  
>>> children cgroups, right?  
>>  
>> No, I meant Mel, quoting this:  
>>  
>> "Further I would expect that an administrator would be aware of these  
>> limitations and set kmem\_accounting at cgroup creation time before any  
>> processes start. Maybe that should be enforced but it's not a  
>> fundamental problem."  
>>  
>> But I guess it is pretty much the same thing Michal proposes, in essence.  
>>  
>> Or IOW, if your concern is with the fact that charges may have happened  
>> in the past before this is enabled, we can make sure this cannot happen  
>> by disallowing the limit to be set if currently unset (value changes are  
>> obviously fine) if you have children or any tasks already in the group.  
>  
> Yeah, please do that.  
>

I did already, patches soon! =)

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Mon, 01 Oct 2012 08:43:24 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 10/01/2012 04:57 AM, Tejun Heo wrote:

> Hello, James.  
>  
> On Sun, Sep 30, 2012 at 12:25:52PM +0100, James Bottomley wrote:  
>> But you've got to ask yourself who cares about accurate accounting per  
>> container of dentry and inode objects? They're not objects that any  
>> administrator is used to limiting. What we at parallels care about  
>> isn't accurately accounting them, it's that one container can't DoS  
>> another by exhausting system resources. That's achieved equally well by  
>> first charge slab accounting, so we don't really have an interest in  
>> pushing object accounting code for which there's no use case.  
>  
> Isn't it more because the use cases you have on mind don't share  
> dentries/inodes too much? Wildly incorrect accounting definitely  
> degrades container isolation and can lead to unexpected behaviors.  
>  
>> All we need kernel memory accounting and limiting for is DoS prevention.  
>> There aren't really any system administrators who care about Kernel  
>> Memory accounting (at least until the system goes oom) because there are

>> no absolute knobs for it (all there is are a set of weird and wonderful  
>> heuristics, like dirty limit ratio and drop caches). Kernel memory  
>  
> I think that's because the mechanism currently doesn't exist. If one  
> wants to control how memory is distributed across different cgroups,  
> it's logical to control kernel memory too. The resource in question  
> is the actual memory after all. I think at least google would be  
> interested in it, so, no, I don't agree that nobody wants it. If that  
> is the case, we're working towards the wrong direction.  
>  
>> usage has a whole set of regulatory infrastructure for trying to make it  
>> transparent to the user.  
>>  
>> Don't get me wrong: if there were some easy way to get proper memory  
>> accounting for free, we'd be happy but, because it has no practical  
>> application for any of our customers, there's a limited price we're  
>> willing to pay to get it.  
>  
> Even on purely technical ground, it could be that first-use is the  
> right trade off if other more accurate approaches are too difficult  
> and most workloads are happy with such approach. I'm still a bit  
> weary to base userland interface decisions on that tho.  
>

For the record, user memory also suffers a bit from being always constrained to first-touch accounting. Greg Thelen is working on alternative solutions to make first-accounting the default in a configurable environment, as he explained in the kernel summit.

When that happens, kernel memory can take advantage of it for free.

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Mon, 01 Oct 2012 08:45:11 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/30/2012 12:23 PM, Tejun Heo wrote:

> Hello, Glauber.  
>  
> On Thu, Sep 27, 2012 at 10:30:36PM +0400, Glauber Costa wrote:  
>>> But that happens only when pages enter and leave slab and if it still  
>>> is significant, we can try to further optimize charging. Given that  
>>> this is only for cases where memcg is already in use and we provide a  
>>> switch to disable it globally, I really don't think this warrants  
>>> implementing fully hierarchy configuration.  
>>  
>> Not totally true. We still have to match every allocation to the right  
>> cache, and that is actually our heaviest hit, responsible for the 2, 3 %

>> we're seeing when this is enabled. It is the kind of path so hot that  
 >> people frown upon branches being added, so I don't think we'll ever get  
 >> this close to being free.

>

> Sure, depending on workload, any addition to alloc/free could be  
 > noticeable. I don't know. I'll write more about it when replying to  
 > Michal's message. BTW, `__memcg_kmem_get_cache()` does seem a bit  
 > heavy. I wonder whether indexing from cache side would make it  
 > cheaper? e.g. something like the following.

>

```
> kmem_cache *__memcg_kmem_get_cache(cachep, gfp)
> {
>   struct kmem_cache *c;
>
>   c = cachep->memcg_params->caches[percpu_read(kmemcg_slab_idx)];
>   if (likely(c))
>       return c;
>   /* try to create and then fall back to cachep */
> }
```

>

> where `kmemcg_slab_idx` is updated from sched notifier (or maybe add and  
 > use `current->kmemcg_slab_idx?`). You would still need `__GFP_*` and  
 > `in_interrupt()` tests but `current->mm` and `PF_KTHREAD` tests can be  
 > rolled into index selection.

>

How big would this array be? there can be a lot more `kmem_caches` than  
 there are `memcgs`. That is why it is done from `memcg` side.

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
 Posted by [Glauber Costa](#) on Mon, 01 Oct 2012 08:46:02 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 09/30/2012 02:37 PM, Tejun Heo wrote:

> Hello, James.

>

> On Sun, Sep 30, 2012 at 09:56:28AM +0100, James Bottomley wrote:

>> The beancounter approach originally used by OpenVZ does exactly this.

>> There are two specific problems, though, firstly you can't count  
 >> references in generic code, so now you have to extend the cgroup  
 >> tentacles into every object, an invasiveness which people didn't really  
 >> like.

>

> Yeah, it will need some hooks. For dentry and inode, I think it would  
 > be pretty well isolated tho. Wasn't it?

>

We would still need something for the stack. For open files, and for everything that becomes a potential problem. We then end up with 35 different knobs instead of one. One of the perceived advantages of this approach, is that it condenses as much data as a single knob as possible, reducing complexity and over flexibility.

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Michal Hocko](#) on Mon, 01 Oct 2012 09:27:56 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hi,

On Sun 30-09-12 17:47:50, Tejun Heo wrote:

[...]

> > > The hot path overhead is quite minimal - it doesn't do much more than  
> > > indirecting one more time. In terms of memory usage, it sure could  
> > > lead to a bit more fragmentation but even if it gets to several megs  
> > > per cgroup, I don't think that's something excessive. So, there is  
> > > overhead but I don't believe it to be prohibitive.

> >

> > Remember that users do not want to pay even "something minimal" when the  
> > feature is not needed.

>

> Yeah but, if we can get it down to, say, around 1% under most  
> workloads for memcg users, it is quite questionable to introduce full  
> hierarchical configuration to allow avoiding that, isn't it?

Remember that the kmem memory is still accounted to u+k if it is enabled which could be a no-go because some workloads (I have provided an example that those which are trusted are generally safe to ignore kernel memory overhead) simply don't want to consider additional memory which is mostly invisible for them.

> > > The distinction between "trusted" and "untrusted" is something  
> > > artificially created due to the assumed deficiency of kmemcg  
> > > implementation.

> >

> > Not really. It doesn't have to do anything with the overhead (be it  
> > memory or runtime). It really boils down to "do I need/want it at all".  
> > Why would I want to think about how much kernel memory is in use in the  
> > first place? Or do you think that user memory accounting should be  
> > deprecated?

>

> But you can apply the same "do I need/want it at all" question to the  
> configuration parameter too.

Yes but, as I've said, the global configuration parameter is too

coarse. You can have a mix of trusted and untrusted workloads at the same machine (e.g. web server which is inherently untrusted) and trusted (local batch jobs which just needs a special LRU aging).

> I can see your point but the decision seems muddy to me, and if muddy,  
> I prefer to err on the side of being too conservative.  
>  
> > > This is userland visible API.  
> >  
> > I am not sure which API visible part you have in mind but  
> > kmem.limit\_in\_bytes will be there whether we go with global knob or "no  
> > limit no accounting" approach.  
>  
> I mean full hierarchical configuration of it. It becomes something  
> which each memcg user cares about instead of something which the base  
> system / admin flips on system boot.  
>  
> > > We can always expand the flexibility. Let's do the simple thing  
> > > first. As an added bonus, it would enable using static\_keys for  
> > > accounting branches too.  
> >  
> > While I do agree with you in general and being careful is at place in  
> > this area as time shown several times, this seems to be too restrictive  
> > in this particular case.  
> > We won't save almost no code with the global knob so I am not sure  
> > what we are actually saving here. Global knob will just give us all or  
> > nothing semantic without making the whole thing simpler. You will stick  
> > with static branches and check whether the group is accountable anyway,  
> > right?  
>  
> The thing is about the same argument can be made about .use\_hierarchy  
> too. It doesn't necessarily make the code much harder. Especially  
> because the code is structured with that feature on mind, removing  
> .use\_hierarchy might not remove whole lot of code; however, the wider  
> range of behavior which got exposed through that poses a much larger  
> problem when we try to make modifications on related behaviors. We  
> get a lot more locked down by seemingly not too much code and our long  
> term maintainability / sustainability suffers as a result.

I think that comparing kmem accounting with use\_hierarchy is not fair.  
Glauber tried to explain why already so I will not repeat it here.  
I will just mention one thing. use\_hierarchy has been introduced because  
hierarchies were expensive at the time. kmem accounting is about should  
we do u or u+k accounting. So there is a crucial difference.

> I'm not trying to say this is as bad as .use\_hierarchy but want to  
> point out that memcg and cgroup in general have had pretty strong  
> tendency to choose overly flexible and complex designs and interfaces

> and it's probably about time we become more careful especially about  
> stuff which is visible to userland.

That is right but I think that the current discussion shows that a mixed (kmem disabled and kmem enabled hierarchies) workloads are far from being theoretical and a global knob is just too coarse. I am afraid we will see "we want that per hierarchy" requests shortly and that would just add a new confusion where global knob would complicate it considerably (do we really want on/off/per\_hierarchy global knob?).

--

Michal Hocko  
SUSE Labs

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Wed, 03 Oct 2012 22:43:16 GMT

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

---

Hey, Michal.

On Mon, Oct 01, 2012 at 11:27:56AM +0200, Michal Hocko wrote:

> > Yeah but, if we can get it down to, say, around 1% under most  
> > workloads for memcg users, it is quite questionable to introduce full  
> > hierarchical configuration to allow avoiding that, isn't it?  
>  
> Remember that the kmem memory is still accounted to u+k if it is enabled  
> which could be a no-go because some workloads (I have provided an  
> example that those which are trusted are generally safe to ignore kernel  
> memory overhead) simply don't want to consider additional memory which  
> is mostly invisible for them.

Maybe it's because my exposure to cgroup usage is different from yours but the argument that not accounting kernel memory is something inherently beneficial is lost on me. For compatibility, overhead and/or implementation complexity issues, yeah, sure, we can't always (well more like usually) have all we want but I don't get how not accounting kernel memory is something inherently necessary or beneficial. This is all about provisioning physical memory to different groups of users and memory is memory (that's why u+k makes sense, right?). Without kmemcg enabled, we not only lack a way to control kernel memory usage but also a way to even watch per-group usages.

> > But you can apply the same "do I need/want it at all" question to the  
> > configuration parameter too.  
>  
> Yes but, as I've said, the global configuration parameter is too  
> coarse. You can have a mix of trusted and untrusted workloads at the



- > same machine (e.g. web server which is inherently untrusted) and trusted
- > (local batch jobs which just needs a special LRU aging).

This too stems from the same difference stated above. You think there's inherent distinction between trusted and untrusted workloads and they need different features from the kernel while I think why trust anyone if you can untrust everyone and consider the knob as a compatibility thing.

- > I think that comparing kmem accounting with use\_hierarchy is not fair.
- > Glauber tried to explain why already so I will not repeat it here.
- > I will just mention one thing. use\_hierarchy has been introduced because
- > hierarchies were expensive at the time. kmem accounting is about should
- > we do u or u+k accounting. So there is a crucial difference.

It may be less crazy but I think there are enough commonalities to at least make a comparison. Mel seems to think it's mostly about performance overhead. You think that not accounting kmem is something inherently necessary.

- > That is right but I think that the current discussion shows that a mixed
- > (kmem disabled and kmem enabled hierarchies) workloads are far from
- > being theoretical and a global knob is just too coarse. I am afraid we

I'm not sure there's much evidence in this thread. The strongest up to this point seems to be performance overhead / difficulty of general enough implementation. As for "trusted" workload, what are the inherent benefits of trusting if you don't have to?

- > will see "we want that per hierarchy" requests shortly and that would
- > just add a new confusion where global knob would complicate it
- > considerably (do we really want on/off/per\_hierarchy global knob?).

Hmmm? The global knob is just the same per\_hierarchy knob at the root. It's hierarchical after all.

Anyways, as long as the "we silently ignore what happened before being enabled" is gone, I won't fight this anymore. It isn't broken after all. But, please think about making things simpler in general, cgroup is riddled with mis-designed complexities and memcg seems to be leading the charge at times.

Thanks.

--

tejun



Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Wed, 03 Oct 2012 22:54:58 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello, Glauber.

On Mon, Oct 01, 2012 at 12:45:11PM +0400, Glauber Costa wrote:

> > where kmemcg\_slab\_idx is updated from sched notifier (or maybe add and  
> > use current->kmemcg\_slab\_idx?). You would still need \_\_GFP\_\* and  
> > in\_interrupt() tests but current->mm and PF\_KTHREAD tests can be  
> > rolled into index selection.  
>  
> How big would this array be? there can be a lot more kmem\_caches than  
> there are memcgs. That is why it is done from memcg side.

The total number of memcgs are pretty limited due to the ID thing,  
right? And kmemcg is only applied to subset of caches. I don't think  
the array size would be a problem in terms of memory overhead, would  
it? If so, RCU synchronize and dynamically grow them?

Thanks.

--  
tejun

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Wed, 03 Oct 2012 22:59:30 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello, Glauber.

On Mon, Oct 01, 2012 at 12:46:02PM +0400, Glauber Costa wrote:

> > Yeah, it will need some hooks. For dentry and inode, I think it would  
> > be pretty well isolated tho. Wasn't it?  
>  
> We would still need something for the stack. For open files, and for  
> everything that becomes a potential problem. We then end up with 35  
> different knobs instead of one. One of the perceived advantages of this  
> approach, is that it condenses as much data as a single knob as  
> possible, reducing complexity and over flexibility.

Oh, I didn't mean to use object-specific counting for all of them.  
Most resources don't have such common misaccounting problem. I mean,  
for stack, it doesn't exist by definition (other than cgroup  
migration). There's no reason to use anything other than first-use  
kmem based accounting for them. My point was that for particularly  
problematic ones like dentry/inode, it might be better to treat them

differently.

Thanks.

--

tejun

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Glauber Costa](#) on Thu, 04 Oct 2012 11:55:14 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On 10/04/2012 02:54 AM, Tejun Heo wrote:

> Hello, Glauber.

>

> On Mon, Oct 01, 2012 at 12:45:11PM +0400, Glauber Costa wrote:

>>> where kmemcg\_slab\_idx is updated from sched notifier (or maybe add and  
>>> use current->kmemcg\_slab\_idx?). You would still need \_\_GFP\_\* and  
>>> in\_interrupt() tests but current->mm and PF\_KTHREAD tests can be  
>>> rolled into index selection.

>>

>> How big would this array be? there can be a lot more kmem\_caches than  
>> there are memcgs. That is why it is done from memcg side.

>

> The total number of memcgs are pretty limited due to the ID thing,  
> right? And kmemcg is only applied to subset of caches. I don't think  
> the array size would be a problem in terms of memory overhead, would  
> it? If so, RCU synchronize and dynamically grow them?

>

> Thanks.

>

I don't want to assume the number of memcgs will always be that limited.  
Sure, the ID limitation sounds pretty much a big one, but people doing  
VMs usually want to stack as many VMs as they possibly can in an  
environment, and the less things preventing that from happening, the better.

That said, now that I've experimented with this a bit, indexing from the  
cache may have some advantages: it can get too complicated to propagate  
new caches appearing to all memcgs that already in-flight. We don't have  
this problem from the cache side, because instances of it are guaranteed  
not to exist at this point by definition.

I don't want to bloat unrelated kmem\_cache structures, so I can't embed  
a memcg array in there: I would have to have a pointer to a memcg array  
that gets assigned at first use. But if we don't want to have a static  
number, as you and christoph already frowned upon heavily, we may have  
to do that memcg side as well.

The array gets bigger, though, because it pretty much has to be enough to accomodate all `css_ids`. Even now, they are more than the 400 I used in this patchset. Not allocating all of them at once will lead to more complication and pointer chasing in here.

I'll take a look at the alternatives today and tomorrow.

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure

Posted by [Michal Hocko](#) on Fri, 05 Oct 2012 13:47:23 GMT

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

---

On Thu 04-10-12 07:43:16, Tejun Heo wrote:

[...]

> > That is right but I think that the current discussion shows that a mixed  
> > (kmem disabled and kmem enabled hierarchies) workloads are far from  
> > being theoretical and a global knob is just too coarse. I am afraid we  
>  
> I'm not sure there's much evidence in this thread. The strongest upto  
> this point seems to be performance overhead / difficulty of general  
> enough implementation. As for "trusted" workload, what are the  
> inherent benefits of trusting if you don't have to?

One advantage is that you do not have to consider kernel memory allocations (which are inherently bound to the kernel version) so the sizing is much easier and version independent. If you set a limit to XY because you know what you are doing you certainly do not want to regress (e.g. because of unnecessary reclaim) just because a certain kernel allocation got bigger, right?

> > will see "we want that per hierarchy" requests shortly and that would  
> > just add a new confusion where global knob would complicate it  
> > considerably (do we really want on/off/per\_hierarchy global knob?).  
>  
> Hmm? The global knob is just the same per\_hierarchy knob at the  
> root. It's hierarchical after all.

When you said global knob I imagined mount or boot option. If you want to have yet another `memory.enable_kmem` then IMHO it is much easier to use set-accounted semantic (which is hierarchical as well).

> Anyways, as long as the "we silently ignore what happened before being  
> enabled" is gone, I won't fight this anymore. It isn't broken after  
> all.

OK, it is good that we settled this.

> But, please think about making things simpler in general, cgroup  
> is riddled with mis-designed complexities and memcg seems to be  
> leading the charge at times.

Yes this is an evolution and it seems that we are slowly getting there.

>  
> Thanks.

--

Michal Hocko  
SUSE Labs

---

---

Subject: Re: [PATCH v3 04/13] kmem accounting basic infrastructure  
Posted by [Tejun Heo](#) on Sat, 06 Oct 2012 02:19:24 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hello, Glauber.

On Thu, Oct 04, 2012 at 03:55:14PM +0400, Glauber Costa wrote:

> I don't want to bloat unrelated kmem\_cache structures, so I can't embed  
> a memcg array in there: I would have to have a pointer to a memcg array  
> that gets assigned at first use. But if we don't want to have a static  
> number, as you and christoph already frowned upon heavily, we may have  
> to do that memcg side as well.

>

> The array gets bigger, though, because it pretty much has to be enough  
> to accomodate all css\_ids. Even now, they are more than the 400 I used  
> in this patchset. Not allocating all of them at once will lead to more  
> complication and pointer chasing in here.

I don't think it would require more pointer chasing. At the simplest,  
we can just compare the array size each time. If you wanna be more  
efficient, all arrays can be kept at the same size and resized when  
the number of memcgs cross the current number. The only runtime  
overhead would be one pointer deref which I don't think can be avoided  
regardless of the indexing direction.

Thanks.

--

tejun

---