

---

Subject: Re: [RFC/PATCH] cgroup swap subsystem  
Posted by [Paul Menage](#) on Wed, 05 Mar 2008 06:36:19 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hi Daisuke,

Most of my comments below are to do with style issues with cgroups, rather than the details of the memory management code.

2008/3/4 Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp>:

```
> +/*
> + * A page_cgroup page is associated with every page descriptor. The
> + * page_cgroup helps us identify information about the cgroup
> + */
> +struct page_cgroup {
> +    struct list_head lru;        /* per cgroup LRU list */
> +    struct page *page;
> +    struct mem_cgroup *mem_cgroup;
> +#ifdef CONFIG_CGROUP_SWAP_LIMIT
> +    struct mm_struct *pc_mm;
> +#endif
> +    atomic_t ref_cnt;           /* Helpful when pages move b/w */
> +                                /* mapped and cached states */
> +    int    flags;
> +};
>
> +
> +#ifdef CONFIG_CGROUP_SWAP_LIMIT
> +struct swap_cgroup {
> +    struct cgroup_subsys_state css;
> +    struct res_counter res;
> +};
> +
> +static inline struct swap_cgroup *swap_cgroup_from_cgrp(struct cgroup *cgrp)
> +{
> +    return container_of(cgroup_subsys_state(cgrp, swap_subsys_id),
> +                        struct swap_cgroup,
> +                        css);
> +}
> +
> +static inline struct swap_cgroup *swap_cgroup_from_task(struct task_struct *p)
> +{
> +    return container_of(task_subsys_state(p, swap_subsys_id),
> +                        struct swap_cgroup, css);
> +}
```

Can't these definitions be moved into swap\_limit.c?

```

> @@ -254,15 +243,27 @@ struct mem_cgroup *mem_cgroup_from_task(
> void mm_init_cgroup(struct mm_struct *mm, struct task_struct *p)
> {
>     struct mem_cgroup *mem;
> #ifdef CONFIG_CGROUP_SWAP_LIMIT
> +     struct swap_cgroup *swap;
> #endif
>
>     mem = mem_cgroup_from_task(p);
>     css_get(&mem->css);
>     mm->mem_cgroup = mem;
> +
> #ifdef CONFIG_CGROUP_SWAP_LIMIT
> +     swap = swap_cgroup_from_task(p);
> +     css_get(&swap->css);
> +     mm->swap_cgroup = swap;
> #endif

```

My feeling is that it would be cleaner to move this code into swap\_limit.c, and have a separate mm\_init\_swap\_cgroup() function. (And a mm\_free\_swap\_cgroup() function).

```

> +     pc = page_get_page_cgroup(page);
> +     if (WARN_ON(!pc))
> +         mm = &init_mm;
> +     else
> +         mm = pc->pc_mm;
> +     BUG_ON(!mm);

```

Is this safe against races with the mem.force\_empty operation?

```

> +
> +     rcu_read_lock();
> +     swap = rcu_dereference(mm->swap_cgroup);
> +     rcu_read_unlock();
> +     BUG_ON(!swap);

```

Is it safe to do rcu\_read\_unlock() while you are still planning to operate on the value of "swap"?

```

> +
> +static ssize_t swap_cgroup_read(struct cgroup *cgrp,
> +                               struct cftype *cft, struct file *file,
> +                               char __user *userbuf, size_t nbytes,
> +                               loff_t *ppos)
> +{
> +     return res_counter_read(&swap_cgroup_from_cgrp(cgrp)->res,
> +                             cft->private, userbuf, nbytes, ppos,

```

```
> +          NULL);
> +}
```

Can you use the cgroups read\_u64 method, and just call res\_counter\_read\_u64?

```
> +
> +static int swap_cgroup_write_strategy(char *buf, unsigned long long *tmp)
> +{
> +    *tmp = memparse(buf, &buf);
> +    if (*buf != '\0')
> +        return -EINVAL;
> +
> +    /*
> +     * Round up the value to the closest page size
> +     */
> +    *tmp = ((*tmp + PAGE_SIZE - 1) >> PAGE_SHIFT) << PAGE_SHIFT;
> +    return 0;
> +}
```

This is the same as mem\_cgroup\_write\_strategy. As part of your patch, can you create a res\_counter\_write\_pagealign() strategy function in res\_counter.c and use it from the memory and swap cgroups?

```
> +
> +#ifdef CONFIG_CGROUP_SWAP_LIMIT
> +    p->swap_cgroup = vmalloc(maxpages * sizeof(*swap_cgroup));
> +    if (!(p->swap_cgroup)) {
> +        error = -ENOMEM;
> +        goto bad_swap;
> +    }
> +    memset(p->swap_cgroup, 0, maxpages * sizeof(*swap_cgroup));
> +#endif
```

It would be nice to only allocate these the first time the swap cgroup subsystem becomes active, to avoid the overhead for people not using it; even better if you can free it again if the swap subsystem becomes inactive again.

Paul

---

Containers mailing list  
Containers@lists.linux-foundation.org  
<https://lists.linux-foundation.org/mailman/listinfo/containers>

---

---

Subject: Re: Re: [RFC/PATCH] cgroup swap subsystem  
Posted by [Paul Menage](#) on Thu, 06 Mar 2008 08:52:41 GMT

---

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

---

On Thu, Mar 6, 2008 at 12:50 AM, Pavel Emelyanov <xemul@openvz.org> wrote:

> > The change that you're referring to is allowing a cgroup to have a  
> > total memory limit for itself and all its children, and then giving  
> > that cgroup's children separate memory limits within that overall  
> > limit?  
>  
> Yup. Isn't this reasonable?

Yes, sounds like a good plan.

Paul

---

Containers mailing list  
Containers@lists.linux-foundation.org  
<https://lists.linux-foundation.org/mailman/listinfo/containers>

---

---

Subject: Re: [RFC/PATCH] cgroup swap subsystem  
Posted by [Daisuke Nishimura](#) on Thu, 06 Mar 2008 12:20:58 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hi.

Paul Menage wrote:

```
>> + pc = page_get_page_cgroup(page);  
>> + if (WARN_ON(!pc))  
>> +     mm = &init_mm;  
>> + else  
>> +     mm = pc->pc_mm;  
>> + BUG_ON(!mm);
```

>  
> Is this safe against races with the mem.force\_empty operation?

>  
I've not considered yet about force\_empty operation  
of memory subsystem.

Thank you for pointing it out.

```
>> +  
>> + rcu_read_lock();  
>> + swap = rcu_dereference(mm->swap_cgroup);  
>> + rcu_read_unlock();  
>> + BUG_ON(!swap);
```

>  
> Is it safe to do rcu\_read\_unlock() while you are still planning to  
> operate on the value of "swap"?

>

You are right.

I think I should `css_get()` before `rcu_read_unlock()` as memory subsystem does.

```
>> +
>> + #ifdef CONFIG_CGROUP_SWAP_LIMIT
>> +     p->swap_cgroup = vmalloc(maxpages * sizeof(*swap_cgroup));
>> +     if (!(p->swap_cgroup)) {
>> +         error = -ENOMEM;
>> +         goto bad_swap;
>> +     }
>> +     memset(p->swap_cgroup, 0, maxpages * sizeof(*swap_cgroup));
>> + #endif
```

>  
> It would be nice to only allocate these the first time the swap cgroup  
> subsystem becomes active, to avoid the overhead for people not using  
> it; even better if you can free it again if the swap subsystem becomes  
> inactive again.

>  
> Hmm.. good idea.

I think this is possible by adding a flag file, like "swap.enable\_limit",  
to the top of cgroup directory, and charging all the swap entries  
which are used when the flag is enabled to the top cgroup.

Thanks,  
Daisuke Nishimura.

---

Containers mailing list  
Containers@lists.linux-foundation.org  
<https://lists.linux-foundation.org/mailman/listinfo/containers>

---

---

Subject: Re: Re: [RFC/PATCH] cgroup swap subsystem  
Posted by [KAMEZAWA Hiroyuki](#) on Thu, 06 Mar 2008 12:56:56 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

>> At first look, remembering mm struct is not very good.  
>> Remembering swap controller itself is better.

>  
> The swap\_cgroup when the page (and page\_cgroup) is allocated and  
> the swap\_cgroup when the page is going to be swapped out may be  
> different by `swap_cgroup_move_task()`, so I think swap\_cgroup  
> to be charged should be determined at the point of swapout.

>  
> Accounting swap against an entity which allocs anon memory is

not strange. Problem here is `move_task` itself.  
Now, charges against anon is not moved when a task which uses it is moved. please fix this behavior first if you think this is problematic.

But, finally, a daemon driven by process event connector determines the group before process starts using anon. It's doubtful that it's worth to add complicated/costly ones.

Thanks,  
-Kame

---

Containers mailing list  
Containers@lists.linux-foundation.org  
<https://lists.linux-foundation.org/mailman/listinfo/containers>

---

---

Subject: Re: [RFC/PATCH] cgroup swap subsystem  
Posted by [Daisuke Nishimura](#) on Fri, 07 Mar 2008 08:22:29 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hi.

kamezawa.hiroyu@jp.fujitsu.com wrote:

>>> At first look, remembering mm struct is not very good.  
>>> Remembering swap controller itself is better.  
>> The swap\_cgroup when the page(and page\_cgroup) is allocated and  
>> the swap\_cgroup when the page is going to be swapped out may be  
>> different by `swap_cgroup_move_task()`, so I think swap\_cgroup  
>> to be charged should be determined at the point of swapout.  
>>  
> Accounting swap against an entity which allocs anon memory is  
> not strange. Problem here is `move_task` itself.  
> Now, charges against anon is not moved when a task which uses it  
> is moved. please fix this behavior first if you think this is  
> problematic.  
>  
> But, finally, a daemon driven by process event connector  
> determines the group before process starts using anon. It's  
> doubtful that it's worth to add complicated/costly ones.  
>

I agree with you.

I think the current behavior of `move_task` is problematic,  
and should fix it.  
But fixing it would be difficult and add a costly process,

so I should consider more.

Thanks,  
Daisuke Nishimura.

---

Containers mailing list  
Containers@lists.linux-foundation.org  
<https://lists.linux-foundation.org/mailman/listinfo/containers>

---

---

Subject: Re: Re: [RFC/PATCH] cgroup swap subsystem  
Posted by [yamamoto](#) on Wed, 12 Mar 2008 22:57:40 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

> >> At first look, remembering mm struct is not very good.  
> >> Remembering swap controller itself is better.  
> >  
> >The swap\_cgroup when the page(and page\_cgroup) is allocated and  
> >the swap\_cgroup when the page is going to be swapped out may be  
> >different by swap\_cgroup\_move\_task(), so I think swap\_cgroup  
> >to be charged should be determined at the point of swapout.  
> >  
> Accounting swap against an entity which allocs anon memory is  
> not strange. Problem here is move\_task itself.  
> Now, charges against anon is not moved when a task which uses it  
> is moved. please fix this behavior first if you think this is  
> problematic.  
>  
> But, finally, a daemon driven by process event connector  
> determines the group before process starts using anon. It's  
> doubtful that it's worth to add complicated/costly ones.  
>  
>  
> Thanks,  
> -Kame

doesn't PEC work asynchronously and allows processes to use  
anonymous memory before being moved by the daemon?

YAMAMOTO Takashi

---

Containers mailing list  
Containers@lists.linux-foundation.org  
<https://lists.linux-foundation.org/mailman/listinfo/containers>

---