

---

Subject: Re: [RFC] [PATCH] memory controller background reclamation

Posted by [Balbir Singh](#) on Fri, 23 Nov 2007 03:46:12 GMT

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

---

YAMAMOTO Takashi wrote:

> changes from the previous:

> - rebase to linux-2.6.24-rc2-mm1 + kamezawa patchset.

> - create workqueue when setting high watermark, rather than cgroup creation

> which is earlier on boot.

>

> YAMAMOTO Takashi

>

>

> Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

> ---

>

> --- linux-2.6.24-rc2-mm1-kame-pd/include/linux/res\_counter.h.BACKUP 2007-11-14  
16:05:48.000000000 +0900

> +++ linux-2.6.24-rc2-mm1-kame-pd/include/linux/res\_counter.h 2007-11-22  
15:14:32.000000000 +0900

> @@ -32,6 +32,13 @@ struct res\_counter {

> \* the number of unsuccessful attempts to consume the resource

> \*/

> unsigned long long failcnt;

> +

> + /\*

> + \* watermarks

> + \*/

> + unsigned long long high\_watermark;

> + unsigned long long low\_watermark;

> +

> /\*

> \* the lock to protect all of the above.

> \* the routines below consider this to be IRQ-safe

> @@ -66,6 +73,8 @@ enum {

> RES\_USAGE,

> RES\_LIMIT,

> RES\_FAILCNT,

> + RES\_HIGH\_WATERMARK,

> + RES\_LOW\_WATERMARK,

> };

>

> /\*

> @@ -124,4 +133,26 @@ static inline bool res\_counter\_check\_und

> return ret;

> }

>

> +static inline bool res\_counter\_below\_low\_watermark(struct res\_counter \*cnt)

```

> +{
> + bool ret;
> + unsigned long flags;
> +
> + spin_lock_irqsave(&cnt->lock, flags);
> + ret = cnt->usage < cnt->low_watermark;
> + spin_unlock_irqrestore(&cnt->lock, flags);
> + return ret;
> +}
> +
> +static inline bool res_counter_above_high_watermark(struct res_counter *cnt)
> +{
> + bool ret;
> + unsigned long flags;
> +
> + spin_lock_irqsave(&cnt->lock, flags);
> + ret = cnt->usage > cnt->high_watermark;
> + spin_unlock_irqrestore(&cnt->lock, flags);
> + return ret;
> +}
> +
> #endif
> --- linux-2.6.24-rc2-mm1-kame-pd/kernel/res_counter.c.BACKUP 2007-11-14
16:05:52.000000000 +0900
> +++ linux-2.6.24-rc2-mm1-kame-pd/kernel/res_counter.c 2007-11-22 15:14:32.000000000
+0900
> @@ -17,6 +17,8 @@ void res_counter_init(struct res_counter
> {
> spin_lock_init(&counter->lock);
> counter->limit = (unsigned long long)LLONG_MAX;
> + counter->high_watermark = (unsigned long long)LLONG_MAX;
> + counter->low_watermark = (unsigned long long)LLONG_MAX;

```

Should low watermark also be LLONG\_MAX?

```

> }
>
> int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
> @@ -69,6 +71,10 @@ res_counter_member(struct res_counter *c
> return &counter->limit;
> case RES_FAILCNT:
> return &counter->failcnt;
> + case RES_HIGH_WATERMARK:
> + return &counter->high_watermark;
> + case RES_LOW_WATERMARK:
> + return &counter->low_watermark;
> };
>

```

```

> BUG();
> @@ -99,6 +105,7 @@ ssize_t res_counter_write(struct res_cou
> int ret;
> char *buf, *end;
> unsigned long long tmp, *val;
> + unsigned long flags;
>
> buf = kmalloc(nbytes + 1, GFP_KERNEL);
> ret = -ENOMEM;
> @@ -122,9 +129,29 @@ ssize_t res_counter_write(struct res_cou
> goto out_free;
> }
>
> + spin_lock_irqsave(&counter->lock, flags);
> val = res_counter_member(counter, member);
> + /* ensure low_watermark <= high_watermark <= limit */
> + switch (member) {
> + case RES_LIMIT:
> + if (tmp < counter->high_watermark)
> + goto out_locked;
> + break;
> + case RES_HIGH_WATERMARK:
> + if (tmp > counter->limit || tmp < counter->low_watermark)
> + goto out_locked;
> + break;
> + case RES_LOW_WATERMARK:
> + if (tmp > counter->high_watermark)
> + goto out_locked;
> + break;
> + }
> *val = tmp;
> + BUG_ON(counter->high_watermark > counter->limit);
> + BUG_ON(counter->low_watermark > counter->high_watermark);
> ret = nbytes;
> +out_locked:
> + spin_unlock_irqrestore(&counter->lock, flags);
> out_free:
> kfree(buf);
> out:
> --- linux-2.6.24-rc2-mm1-kame-pd/mm/memcontrol.c.BACKUP 2007-11-20 13:11:09.000000000
+0900
> +++ linux-2.6.24-rc2-mm1-kame-pd/mm/memcontrol.c 2007-11-22 16:29:26.000000000 +0900
> @@ -28,6 +28,7 @@
> #include <linux/rcupdate.h>
> #include <linux/swap.h>
> #include <linux/spinlock.h>
> +#include <linux/workqueue.h>
> #include <linux/fs.h>

```

```

> #include <linux/seq_file.h>
>
> @@ -138,6 +139,10 @@ struct mem_cgroup {
> * statistics.
> */
> struct mem_cgroup_stat stat;
> + /*
> + * background reclamation.
> + */
> + struct work_struct reclaim_work;
> };
>
> /*
> @@ -240,6 +245,21 @@ static unsigned long mem_cgroup_get_all_
>
> static struct mem_cgroup init_mem_cgroup;
>
> +static DEFINE_MUTEX(mem_cgroup_workqueue_init_lock);
> +static struct workqueue_struct *mem_cgroup_workqueue;
> +
> +static void mem_cgroup_create_workqueue(void)
> +{
> +
> + if (mem_cgroup_workqueue != NULL)
> + return;
> +
> + mutex_lock(&mem_cgroup_workqueue_init_lock);
> + if (mem_cgroup_workqueue == NULL)
> + mem_cgroup_workqueue = create_workqueue("mem_cgroup");
> + mutex_unlock(&mem_cgroup_workqueue_init_lock);
> +}
> +
> static inline
> struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont)
> {
> @@ -566,6 +586,50 @@ unsigned long mem_cgroup_isolate_pages(u
> return nr_taken;
> }
>
> +static void
> +mem_cgroup_schedule_reclaim(struct mem_cgroup *mem)
> +{
> +
> + if (mem_cgroup_workqueue == NULL) {
> + BUG_ON(mem->css.cgroup->parent != NULL);
> + return;
> + }
> +

```

```

> + if (work_pending(&mem->reclaim_work))
> + return;
> +
> + css_get(&mem->css); /* XXX need some thoughts wrt cgroup removal. */
> + /*
> + * XXX workqueue is not an ideal mechanism for our purpose.
> + * revisit later.
> + */
> + if (!queue_work(mem_cgroup_workqueue, &mem->reclaim_work))
> + css_put(&mem->css);
> +}
> +
> +static void
> +mem_cgroup_reclaim(struct work_struct *work)
> +{
> + struct mem_cgroup * const mem =
> + container_of(work, struct mem_cgroup, reclaim_work);
> + int batch_count = 128; /* XXX arbitrary */

```

Could we define and use something like MEM\_CGROUP\_BATCH\_COUNT for now? Later we could consider and see if it needs to be tunable. numbers are hard to read in code.

```

> +
> + for (; batch_count > 0; batch_count--) {
> + if (res_counter_below_low_watermark(&mem->res))
> + break;

```

Shouldn't we also check to see that we start reclaim in background only when we are above the high watermark?

```

> + /*
> + * XXX try_to_free_foo is not a correct mechanism to
> + * use here. eg. ALLOCSTALL counter
> + * revisit later.
> + */
> + if (!try_to_free_mem_cgroup_pages(mem, GFP_KERNEL))
> + break;
> +}
> + if (batch_count == 0)
> + mem_cgroup_schedule_reclaim(mem);
> + css_put(&mem->css);
> +}
> +
> + /*
> + * Charge the memory controller for page usage.
> + * Return
> + @@ -631,6 +695,12 @@ retry:

```

```

> rcu_read_unlock();
>
> /*
> + * schedule background reclaim if we are above the high watermark.
> + */
> + if (res_counter_above_high_watermark(&mem->res))
> + mem_cgroup_schedule_reclaim(mem);
> +
> + /*
> * If we created the page_cgroup, we should free it on exceeding
> * the cgroup limit.
> */
> @@ -939,9 +1009,16 @@ static ssize_t mem_cgroup_write(struct c
> struct file *file, const char __user *userbuf,
> size_t nbytes, loff_t *ppos)
> {
> - return res_counter_write(&mem_cgroup_from_cont(cont)->res,
> + ssize_t ret;
> +
> + ret = res_counter_write(&mem_cgroup_from_cont(cont)->res,
> cft->private, userbuf, nbytes, ppos,
> mem_cgroup_write_strategy);
> +
> + if (ret >= 0 && cft->private == RES_HIGH_WATERMARK)
> + mem_cgroup_create_workqueue();
> +
> + return ret;
> }
>
> static ssize_t mem_control_type_write(struct cgroup *cont,
> @@ -1097,6 +1174,18 @@ static struct cftype mem_cgroup_files[]
> .read = mem_cgroup_read,
> },
> {
> + .name = "high_watermark_in_bytes",
> + .private = RES_HIGH_WATERMARK,
> + .write = mem_cgroup_write,
> + .read = mem_cgroup_read,
> + },
> + {
> + .name = "low_watermark_in_bytes",
> + .private = RES_LOW_WATERMARK,
> + .write = mem_cgroup_write,
> + .read = mem_cgroup_read,
> + },
> + {
> .name = "control_type",
> .write = mem_control_type_write,

```

```
> .read = mem_control_type_read,  
> @@ -1161,6 +1250,8 @@ mem_cgroup_create(struct cgroup_subsys *  
> if (alloc_mem_cgroup_per_zone_info(mem, node))  
> goto free_out;  
>  
> + INIT_WORK(&mem->reclaim_work, mem_cgroup_reclaim);  
> +  
> return &mem->css;  
> free_out:  
> for_each_node_state(node, N_POSSIBLE)
```

I'll start some tests on these patches.

--

Warm Regards,  
Balbir Singh  
Linux Technology Center  
IBM, ISTL

---

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

---