Subject: Re: [RFC 2/4] memcg: high-low watermark
Posted by Balbir Singh on Tue, 27 May 2008 16:26:17 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> Add high/low watermarks to res_counter.
> *This patch itself has no behavior changes to memory resource controller.
>
> Changelog: very old one -> this one (v1)
> - watarmark_state is removed and all state check is done under lock.
> - changed res_counter_charge() interface. The only user is memory
>   resource controller. Anyway, returning -ENOMEM here is a bit starnge.
> - Added watermark enable/disable flag for someone don't want watermarks.
> - Restarted against 2.6.25-mm1.
> - some subsystem which doesn't want high-low watermark can work withou it.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> From: YAMAMOTO Takashi <yamamoto@valinux.co.jp>
>

The From: line should be the first line IIRC.

> ---
> include/linux/res_counter.h | 41 ++++++++++++++++++++++++++---
> kernel/res_counter.c | 66 ++----
> mm/memcontrol.c | 2 -
> 3 files changed, 99 insertions(+), 10 deletions(-)
>
> Index: mm-2.6.26-rc2-mm1/include/linux/res_counter.h
> ===
> --- mm-2.6.26-rc2-mm1.orig/include/linux/res_counter.h
> +++ mm-2.6.26-rc2-mm1/include/linux/res_counter.h
> @@ -16,6 +16,16 @@
> #include <linux/cgroup.h>
>
> /*
> + * status of resource coutner's usage.
> + */
> +enum res_state {
> + RES_BELOW_LOW, /* usage < lwmark */
> + RES_BELOW_HIGH, /* lwmark < usage < hwmark */
> + RES_BELOW_LIMIT, /* hwmark < usage < limit. */
> + RES_OVER_LIMIT, /* only used at chage. */
> +};
> +
> +/*
>   * The core object. the cgroup that wishes to account for some
>   * resource may include this counter into its structures and use

>   * the helpers described beyond
> @@ -39,6 +49,12 @@ struct res_counter {
>   */
>   unsigned long long failcnt;
>   /*
> + * watermarks. needs to keep lwmark <= hwmark <= limit.
> + */
> + unsigned long long hwmark;
> + unsigned long long lwmark;
> + int   use_watermark;

Is it routine to comment this way? I prefer not to have spaces in the type and
the member, makes it easier for my eyes.

> + /*
>   * the lock to protect all of the above.
>   * the routines below consider this to be IRQ-safe
>   */
> @@ -76,13 +92,18 @@ enum {
>   RES_MAX_USAGE,
>   RES_LIMIT,
>   RES_FAILCNT,
> + RES_HWMARK,
> + RES_LWMARK,
> };
>
> /*
>   * helpers for accounting
> + * res_counter_init() ... initialize counter and disable watermarks.
> + * res_counter_init_wmark() ... initialize counter and enable watermarks.
>   */
>
> void res_counter_init(struct res_counter *counter);
> +void res_counter_init_wmark(struct res_counter *counter);
>
> /*
>   * charge - try to consume more resource.
> @@ -93,11 +114,21 @@ void res_counter_init(struct res_counter
>   *
>   * returns 0 on success and <0 if the counter->usage will exceed the
>   * counter->limit _locked call expects the counter->lock to be taken
> + * return values:
> + * If watermark is disabled,
> + * RES_BELOW_LIMIT -- usage is smaller than limt, success.

     ^^^^ typo

> + * RES_OVER_LIMIT   -- usage is bigger than limit, failed.

> + *
> + * If watermark is enabled,
> + * RES_BELOW_LOW   -- usage is smaller than low watermark, success
> + * RES_BELOW_HIGH   -- usage is smaller than high watermark, success.
> + * RES_BELOW_LIMIT -- usage is smaller than limt, success.
> + * RES_OVER_LIMIT   -- usage is bigger than limit, failed.
>   */
>
> -int __must_check res_counter_charge_locked(struct res_counter *counter,
> - unsigned long val);
> -int __must_check res_counter_charge(struct res_counter *counter,
> +enum res_state __must_check
> +res_counter_charge_locked(struct res_counter *counter, unsigned long val);
> +enum res_state __must_check res_counter_charge(struct res_counter *counter,
>   unsigned long val);
>
> /*
> @@ -164,4 +195,7 @@ static inline int res_counter_empty(stru
>   spin_unlock_irqrestore(&cnt->lock, flags);
>   return ret;
> }
> +
> +enum res_state res_counter_state(struct res_counter *counter);
> +
> #endif
> Index: mm-2.6.26-rc2-mm1/kernel/res_counter.c
> ===
> --- mm-2.6.26-rc2-mm1.orig/kernel/res_counter.c
> +++ mm-2.6.26-rc2-mm1/kernel/res_counter.c
> @@ -18,22 +18,40 @@ void res_counter_init(struct res_counter
> {
>   spin_lock_init(&counter->lock);
>   counter->limit = (unsigned long long)LLONG_MAX;
> + counter->use_watermark = 0;
> }
>
> -int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
> +void res_counter_init_wmark(struct res_counter *counter)
> +{
> + spin_lock_init(&counter->lock);
> + counter->limit = (unsigned long long)LLONG_MAX;
> + counter->hwmark = (unsigned long long)LLONG_MAX;
> + counter->lwmark = (unsigned long long)LLONG_MAX;
> + counter->use_watermark = 1;
> +}
> +
> +enum res_state
> +res_counter_charge_locked(struct res_counter *counter, unsigned long val)

```
>  {
>   if (counter->usage + val > counter->limit) {
>    counter->failcnt++;
> -  return -ENOMEM;
> +  return RES_OVER_LIMIT;
>   }
>
>   counter->usage += val;
>   if (counter->usage > counter->max_usage)
>    counter->max_usage = counter->usage;
> - return 0;
> + if (counter->use_watermark) {
> +  if (counter->usage <= counter->lwmark)
> +   return RES_BELOW_LOW;
> +  if (counter->usage <= counter->hwmark)
> +   return RES_BELOW_HIGH;
> + }
> + return RES_BELOW_LIMIT;
>  }
>
> -int res_counter_charge(struct res_counter *counter, unsigned long val)
> +enum res_state
> +res_counter_charge(struct res_counter *counter, unsigned long val)
>  {
>   int ret;
>   unsigned long flags;
> @@ -44,6 +62,23 @@ int res_counter_charge(struct res_counte
>   return ret;
>  }
>
> +enum res_state res_counter_state(struct res_counter *counter)
> +{
> + unsigned long flags;
> + enum res_state ret = RES_BELOW_LIMIT;
> +
> + spin_lock_irqsave(&counter->lock, flags);
> + if (counter->use_watermark) {
> +  if (counter->usage <= counter->lwmark)
> +   ret = RES_BELOW_LOW;
> +  else if (counter->usage <= counter->hwmark)
> +   ret = RES_BELOW_HIGH;
> + }
> + spin_unlock_irqrestore(&counter->lock, flags);
> + return ret;
> +}
> +
```

When do we return RES_OVER_LIMIT? Are we missing that here?

```
> +
>  void res_counter_uncharge_locked(struct res_counter *counter, unsigned long val)
>  {
>   if (WARN_ON(counter->usage < val))
> @@ -74,6 +109,10 @@ res_counter_member(struct res_counter *c
>    return &counter->limit;
>   case RES_FAILCNT:
>    return &counter->failcnt;
> + case RES_HWMARK:
> +  return &counter->hwmark;
> + case RES_LWMARK:
> +  return &counter->lwmark;
>   };
>
>   BUG();
> @@ -134,10 +173,27 @@ ssize_t res_counter_write(struct res_cou
>     goto out_free;
>   }
>   spin_lock_irqsave(&counter->lock, flags);
> + switch (member) {
> +  case RES_LIMIT:
> +   if (counter->use_watermark && counter->hwmark > tmp)
> +    goto unlock_free;
```

We need to document such API changes in the Documentation/controllers/memory.txt file.

```
> +   break;
> +  case RES_HWMARK:
> +   if (tmp < counter->lwmark  || tmp > counter->limit)
> +    goto unlock_free;
> +   break;
> +  case RES_LWMARK:
> +   if (tmp > counter->hwmark)
> +    goto unlock_free;
> +   break;
> +  default:
> +   break;
> + }
>   val = res_counter_member(counter, member);
>   *val = tmp;
> - spin_unlock_irqrestore(&counter->lock, flags);
>   ret = nbytes;
> +unlock_free:
> + spin_unlock_irqrestore(&counter->lock, flags);
>  out_free:
>   kfree(buf);
```

> out:
> Index: mm-2.6.26-rc2-mm1/mm/memcontrol.c
> ===
> --- mm-2.6.26-rc2-mm1.orig/mm/memcontrol.c
> +++ mm-2.6.26-rc2-mm1/mm/memcontrol.c
> @@ -559,7 +559,7 @@ static int mem_cgroup_charge_common(stru
>   css_get(&memcg->css);
>   }
>
> - while (res_counter_charge(&mem->res, PAGE_SIZE)) {
> + while (res_counter_charge(&mem->res, PAGE_SIZE) == RES_OVER_LIMIT) {
>   if (!(gfp_mask & __GFP_WAIT))
>   goto out;
>

Otherwise looks good so far. Need to look at the background reclaim code.

--
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