Subject: Re: [RFC] memory controller : backgorund reclaim and avoid excessive locking [1/5] high-low watermar
Posted by Balbir Singh on Thu, 14 Feb 2008 08:48:33 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> Add High/Low watermark support to res_counter.
>
> This adds high/low watermark support to res_counter.
>
> Adds new member hwmark, lwmark, wmark_state to res_counter struct.
>
> Users of res_counter must keep lwmark <= hwmark <= limit.
> (set params as lwmark == hwmark == limit will disable this feature.)
>
> wmark_state is added to read wmark status without lock.
>
> Codes will be like this. (check status after charge.)
>
>  if (res_counter_charge(cnt...)) { <----(need lock)
>  .....
>  }
>  if (res_counter_above_hwmark(cnt)) <---- (no lock)
>  trigger background jobs...
>
> If I have to check under lock, please teach me.
>

If there are several processes running in parallel in the same cgroup, the end result might not be so nice, specially if the usage is close to the watermarks. I suspect that we should  be OK for now, but might be worth keeping in mind.

> counter->hwmark and counter->lwmark is not automatically adjusted
> when limit is changed. So, users must change lwmark, hwmark before
> changing limit.
>
> Changelog
>  * made variable names shorter.
>  * adjusted to 2.6.24-mm1
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
>  include/linux/res_counter.h |  30 ++++++++++++++++++++++++++
>  kernel/res_counter.c        |  53 ++++++++++++++++++++++++++++++++++++++++++++-----
>  2 files changed, 78 insertions(+), 5 deletions(-)
>
> Index: linux-2.6.24-mm1/include/linux/res_counter.h

```
> ===================================================================
> --- linux-2.6.24-mm1.orig/include/linux/res_counter.h
> +++ linux-2.6.24-mm1/include/linux/res_counter.h
> @@ -19,6 +19,16 @@
>   * the helpers described beyond
>   */
>
> +/*
> + * Watermark status.
> + */
> +
> +enum watermark_state {
> + RES_WMARK_BELOW_LOW = 0, /* usage < low <= high <= max */
> + RES_WMARK_ABOVE_LOW, /* low <= usage < high <= max */
> + RES_WMARK_ABOVE_HIGH, /* low <= high <= usage <= max */
> +};
> +
>  struct res_counter {
>   /*
>    * the current resource consumption level
> @@ -33,10 +43,18 @@ struct res_counter {
>    */
>   unsigned long long failcnt;
>   /*
> +  * for supporting High/Low watermak
> +  * Must keep low <= high <= limit.
> +  */
> + unsigned long long hwmark;
> + unsigned long long lwmark;
> + enum watermark_state wmark_state; /* changed at charge/uncharge */
> + /*
>    * the lock to protect all of the above.
>    * the routines below consider this to be IRQ-safe
>    */
>   spinlock_t lock;
> +
>  };
>
>  /*
> @@ -66,6 +84,8 @@ enum {
>   RES_USAGE,
>   RES_LIMIT,
>   RES_FAILCNT,
> + RES_HWMARK,
> + RES_LWMARK,
>  };
>
>  /*
```

> @@ -124,4 +144,14 @@ static inline bool res_counter_check_und
>  return ret;
>  }
>
> +static inline bool res_counter_below_lwmark(struct res_counter *cnt)
> +{
> + smp_rmb();
> + return (cnt->wmark_state == RES_WMARK_BELOW_LOW);
> +}
> +static inline bool res_counter_above_hwmark(struct res_counter *cnt)
> +{
> + smp_rmb();
> + return (cnt->wmark_state == RES_WMARK_ABOVE_HIGH);
> +}
>  #endif
> Index: linux-2.6.24-mm1/kernel/res_counter.c
> ===================================================================
> --- linux-2.6.24-mm1.orig/kernel/res_counter.c
> +++ linux-2.6.24-mm1/kernel/res_counter.c
> @@ -17,16 +17,29 @@ void res_counter_init(struct res_counter
>  {
>  spin_lock_init(&counter->lock);
>  counter->limit = (unsigned long long)LLONG_MAX;
> + counter->lwmark = (unsigned long long)LLONG_MAX;
> + counter->hwmark = (unsigned long long)LLONG_MAX;
> + counter->wmark_state = RES_WMARK_BELOW_LOW;
>  }
>
>  int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
>  {
> - if (counter->usage + val > counter->limit) {
> + unsigned long long newval = counter->usage + val;
> + if (newval > counter->limit) {
>  counter->failcnt++;
>  return -ENOMEM;
>  }
>
> - counter->usage += val;
> +  if (newval > counter->hwmark) {
> +  counter->wmark_state = RES_WMARK_ABOVE_HIGH;
> +  smp_wmb();

Do we need a barrier here? I suspect not, could you please document as to why a barrier is needed?

> + } else if (newval > counter->lwmark) {
> +  counter->wmark_state = RES_WMARK_ABOVE_LOW;
> +  smp_wmb();

>> Ditto

> + }
> +
> + counter->usage = newval;
> +
>  return 0;
>  }
>
> @@ -43,10 +56,18 @@ int res_counter_charge(struct res_counte
>
>  void res_counter_uncharge_locked(struct res_counter *counter, unsigned long val)
>  {
> + unsigned long long newval = counter->usage - val;
>  if (WARN_ON(counter->usage < val))
> -  val = counter->usage;
> +  newval = 0;
>
> - counter->usage -= val;
> + if (newval < counter->lwmark) {
> +  counter->wmark_state = RES_WMARK_BELOW_LOW;
> +  smp_wmb();

>> Ditto

> + } else if (newval < counter->hwmark) {
> +  counter->wmark_state = RES_WMARK_ABOVE_LOW;
> +  smp_wmb();

>> Ditto

> + }
> + counter->usage = newval;
>  }
>
>  void res_counter_uncharge(struct res_counter *counter, unsigned long val)
> @@ -69,6 +90,10 @@ 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();

```
> @@ -123,10 +148,28 @@ ssize_t res_counter_write(struct res_cou
>     goto out_free;
>   }
>   spin_lock_irqsave(&counter->lock, flags);
> + switch (member) {
> +  case RES_LIMIT:
> +   if (counter->hwmark > tmp)
> +    goto unlock_free;
> +   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:
>
```

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL