Subject: Re: [PATCH v6 02/10] foundations of per-cgroup memory pressure controlling.
Posted by Glauber Costa on Tue, 29 Nov 2011 09:19:12 GMT
View Forum Message <> Reply to Message

On 11/28/2011 12:55 AM, KAMEZAWA Hiroyuki wrote:
> On Fri, 25 Nov 2011 15:38:08 -0200
> Glauber Costa<glommer@parallels.com> wrote:
>
>> This patch replaces all uses of struct sock fields' memory_pressure,
>> memory_allocated, sockets_allocated, and sysctl_mem to acessor
>> macros. Those macros can either receive a socket argument, or a mem_cgroup
>> argument, depending on the context they live in.
>>
>> Since we're only doing a macro wrapping here, no performance impact at all is
>> expected in the case where we don't have cgroups disabled.
>>
>> Signed-off-by: Glauber Costa<glommer@parallels.com>
>> CC: David S. Miller<davem@davemloft.net>
>> CC: Hiroyouki Kamezawa<kamezawa.hiroyu@jp.fujitsu.com>
>> CC: Eric W. Biederman<ebiederm@xmission.com>
>> CC: Eric Dumazet<eric.dumazet@gmail.com>
>
> I have some comments on the style. Maybe a nitpick but many patches were
> sent for fixing conding style in memcg recently.
>
> +static inline int *sk_memory_pressure(const struct sock *sk)
> +{
> + return sk->sk_prot->memory_pressure;
> +}
> +
> +static inline long sk_prot_mem(const struct sock *sk, int index)
> +{
> + long *prot = sk->sk_prot->sysctl_mem;
> + return prot[index];
> +}
> +
>
> I don't think sk_prot_mem() is an easy to undestand name.
> sk_prot_memory_limit() ?
>
>> +static inline int
>> +kcg_sockets_allocated_sum_positive(struct proto *prot, struct mem_cgroup *cg)
>> +{
>> + return percpu_counter_sum_positive(prot->sockets_allocated);
>> +}
>> +
>> +static inline long

>> +kcg_memory_allocated(struct proto *prot, struct mem_cgroup *cg)
>> +{
>> + return atomic_long_read(prot->memory_allocated);
>> +}
>>
>
> I don't like 'kcg'. What it means ?
> memory_cgrou_prot_socekts_allocated() ? and
> memory_cgroup_prot_memory_allocated() ?
>
> And the variable for memory cgroup should be 'memcg'.
> http://www.spinics.net/lists/linux-mm/msg26781.html
> So, please rename.
>
>
>
>> #ifdef CONFIG_PROC_FS
>> /* Called with local bh disabled */
>> diff --git a/include/net/tcp.h b/include/net/tcp.h
>> index e147f42..ccaa3b6 100644
> <snip>
>
> seq_printf(seq, "RAW: inuse %d\n",
>> diff --git a/net/ipv4/tcp.c b/net/ipv4/tcp.c
>> index 34f5db1..89a2bfe 100644
>> --- a/net/ipv4/tcp.c
>> +++ b/net/ipv4/tcp.c
>> @@ -319,9 +319,11 @@ EXPORT_SYMBOL(tcp_memory_pressure);
>>
>> void tcp_enter_memory_pressure(struct sock *sk)
>> {
>> - if (!tcp_memory_pressure) {
>> + int *memory_pressure = sk_memory_pressure(sk);
>> +
>
> Don't you need !memory_pressure check here ?

Not really. Note that the original tcp code doesn't have it as well.
The generic networking code deals with many protocols, not all of them
have memory pressure functionality implemented. Therefore, a check is
needed. If we're inside tcp boundaries, we can pretty much assume
memory_pressure is present.

> Hmm, can this function be
>
> +static inline int *sk_memory_pressure(const struct sock *sk)
> +{
> + return sk->sk_prot->memory_pressure;

> +}
>
> as
>
> static inline bool sk_under_prot_memory_pressure(const struct sock *sk)
> {
> if (sk->sk_prot->memory_pressure&&
> *sk->sk_prot->memory_pressure)
> return true;
>
> return false;
> }
>
> and have sk_set/unset_prot_memory_pressure(), ?

Yes, it could. Would it be preferable for you?
>
>
>> + if (!*memory_pressure) {
>> NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPMEMORYPRESSURES);
>> - tcp_memory_pressure = 1;
>> + *memory_pressure = 1;
>> }
>> }
>> EXPORT_SYMBOL(tcp_enter_memory_pressure);
>> diff --git a/net/ipv4/tcp_input.c b/net/ipv4/tcp_input.c
>> index 52b5c2d..3df862d 100644
>> --- a/net/ipv4/tcp_input.c
>> +++ b/net/ipv4/tcp_input.c
>> @@ -322,7 +322,7 @@ static void tcp_grow_window(struct sock *sk, const struct sk_buff
*skb)
>> /* Check #1 */
>> if (tp->rcv_ssthresh< tp->window_clamp&&
>> (int)tp->rcv_ssthresh< tcp_space(sk)&&
>> - !tcp_memory_pressure) {
>> + !sk_memory_pressure(sk)) {
>
> Don't you need to check !*sk_memory_pressure(sk) ?

good catch!
yes.

>
>
>
>> int incr;
>>
>> /* Check #2. Increase window, if skb with such overhead

```
>> @@ -411,8 +411,8 @@ static void tcp_clamp_window(struct sock *sk)
>>
>>    if (sk->sk_rcvbuf<  sysctl_tcp_rmem[2]&&
>>    !(sk->sk_userlocks&  SOCK_RCVBUF_LOCK)&&
>> -     !tcp_memory_pressure&&
>> -     atomic_long_read(&tcp_memory_allocated)<  sysctl_tcp_mem[0]) {
>> +     !sk_memory_pressure(sk)&&
>> +     sk_memory_allocated(sk)<  sk_prot_mem(sk, 0)) {
>>     sk->sk_rcvbuf = min(atomic_read(&sk->sk_rmem_alloc),
>>           sysctl_tcp_rmem[2]);
>>    }
>> @@ -4864,7 +4864,7 @@ static int tcp_prune_queue(struct sock *sk)
>>
>>    if (atomic_read(&sk->sk_rmem_alloc)>= sk->sk_rcvbuf)
>>     tcp_clamp_window(sk);
>> - else if (tcp_memory_pressure)
>> + else if (sk_memory_pressure(sk))
>>     tp->rcv_ssthresh = min(tp->rcv_ssthresh, 4U * tp->advmss);
>
> Ditto.
>
>
>>
>>    tcp_collapse_ofo_queue(sk);
>> @@ -4930,11 +4930,11 @@ static int tcp_should_expand_sndbuf(const struct sock *sk)
>>     return 0;
>>
>>    /* If we are under global TCP memory pressure, do not expand.  */
>> - if (tcp_memory_pressure)
>> + if (sk_memory_pressure(sk))
>>     return 0;
>
> again.
>
> Thanks,
> -Kame
>
Ok, I will fix this and respin it.
```