
Subject: [PATCH v6 02/10] foundations of per-cgroup memory pressure controlling.
Posted by [Glauber Costa](#) on Fri, 25 Nov 2011 17:38:08 GMT

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

This patch replaces all uses of struct sock fields' memory_pressure, memory_allocated, sockets_allocated, and sysctl_mem to accessor 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>

```
include/net/sock.h | 64 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
include/net/tcp.h  | 3 +-
net/core/sock.c    | 55 ++++++++++++++++++++++++++++++++++++++-----
net/ipv4/proc.c    | 7 +++-
net/ipv4/tcp.c     | 6 +++-
net/ipv4/tcp_input.c | 12 ++++----
net/ipv4/tcp_ipv4.c | 4 +-
net/ipv4/tcp_output.c | 2 +-
net/ipv4/tcp_timer.c | 2 +-
net/ipv6/tcp_ipv6.c | 2 +-
10 files changed, 118 insertions(+), 39 deletions(-)
```

diff --git a/include/net/sock.h b/include/net/sock.h

index c6658be..0d054e0 100644

--- a/include/net/sock.h

+++ b/include/net/sock.h

@@ -54,6 +54,7 @@

#include <linux/security.h>

#include <linux/slab.h>

#include <linux/uaccess.h>

+#include <linux/memcontrol.h>

#include <linux/filter.h>

#include <linux/rculist_nulls.h>

@@ -863,6 +864,69 @@ static inline void sk_refcnt_debug_release(const struct sock *sk)

#define sk_refcnt_debug_release(sk) do { } while (0)

#endif /* SOCK_REFCNT_DEBUG */

+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];
+}
+
+static inline long
+sk_memory_allocated(const struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+ return atomic_long_read(prot->memory_allocated);
+}
+
+static inline long
+sk_memory_allocated_add(struct sock *sk, int amt)
+{
+ struct proto *prot = sk->sk_prot;
+ return atomic_long_add_return(amt, prot->memory_allocated);
+}
+
+static inline void
+sk_memory_allocated_sub(struct sock *sk, int amt)
+{
+ struct proto *prot = sk->sk_prot;
+ atomic_long_sub(amt, prot->memory_allocated);
+}
+
+static inline void sk_sockets_allocated_dec(struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+ percpu_counter_dec(prot->sockets_allocated);
+}
+
+static inline void sk_sockets_allocated_inc(struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+ percpu_counter_inc(prot->sockets_allocated);
+}
+
+static inline int
+sk_sockets_allocated_read_positive(struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+
+ return percpu_counter_sum_positive(prot->sockets_allocated);

```

```

+}
+
+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);
+}

#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
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -44,6 +44,7 @@
#include <net/dst.h>

#include <linux/seq_file.h>
#include <linux/memcontrol.h>

extern struct inet_hashinfo tcp_hashinfo;

@@ -285,7 +286,7 @@ static inline bool tcp_too_many_orphans(struct sock *sk, int shift)
{

if (sk->sk_wmem_queued > SOCK_MIN_SNDBUF &&
- atomic_long_read(&tcp_memory_allocated) > sysctl_tcp_mem[2])
+ sk_memory_allocated(sk) > sk_prot_mem(sk, 2))
return true;
return false;
}
diff --git a/net/core/sock.c b/net/core/sock.c
index 4ed7b1d..26bdb1c 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -1288,7 +1288,7 @@ struct sock *sk_clone(const struct sock *sk, const gfp_t priority)
newsk->sk_wq = NULL;

if (newsk->sk_prot->sockets_allocated)
- percpu_counter_inc(newsk->sk_prot->sockets_allocated);
+ sk_sockets_allocated_inc(newsk);

```

```

    if (sock_flag(newsk, SOCK_TIMESTAMP) ||
        sock_flag(newsk, SOCK_TIMESTAMPING_RX_SOFTWARE))
@@ -1677,30 +1677,32 @@ int __sk_mem_schedule(struct sock *sk, int size, int kind)
    struct proto *prot = sk->sk_prot;
    int amt = sk_mem_pages(size);
    long allocated;
+ int *memory_pressure;

    sk->sk_forward_alloc += amt * SK_MEM_QUANTUM;
- allocated = atomic_long_add_return(amt, prot->memory_allocated);
+
+ memory_pressure = sk_memory_pressure(sk);
+ allocated = sk_memory_allocated_add(sk, amt);

    /* Under limit. */
- if (allocated <= prot->sysctl_mem[0]) {
- if (prot->memory_pressure && *memory_pressure)
- *prot->memory_pressure = 0;
- return 1;
- }
+ if (allocated <= sk_prot_mem(sk, 0))
+ if (memory_pressure && *memory_pressure)
+ *memory_pressure = 0;

    /* Under pressure. */
- if (allocated > prot->sysctl_mem[1])
+ if (allocated > sk_prot_mem(sk, 1))
    if (prot->enter_memory_pressure)
        prot->enter_memory_pressure(sk);

    /* Over hard limit. */
- if (allocated > prot->sysctl_mem[2])
+ if (allocated > sk_prot_mem(sk, 2))
    goto suppress_allocation;

    /* guarantee minimum buffer size under pressure */
    if (kind == SK_MEM_RECV) {
        if (atomic_read(&sk->sk_rmem_alloc) < prot->sysctl_rmem[0])
            return 1;
+
    } else { /* SK_MEM_SEND */
        if (sk->sk_type == SOCK_STREAM) {
            if (sk->sk_wmem_queued < prot->sysctl_wmem[0])
@@ -1710,13 +1712,13 @@ int __sk_mem_schedule(struct sock *sk, int size, int kind)
            return 1;
        }
    }

- if (prot->memory_pressure) {

```

```

+ if (memory_pressure) {
    int alloc;

- if (!*prot->memory_pressure)
+ if (!*memory_pressure)
    return 1;
- alloc = percpu_counter_read_positive(prot->sockets_allocated);
- if (prot->sysctl_mem[2] > alloc *
+ alloc = sk_sockets_allocated_read_positive(sk);
+ if (sk_prot_mem(sk, 2) > alloc *
    sk_mem_pages(sk->sk_wmem_queued +
    atomic_read(&sk->sk_rmem_alloc) +
    sk->sk_forward_alloc))
@@ -1739,7 +1741,9 @@ suppress_allocation:

    /* Alas. Undo changes. */
    sk->sk_forward_alloc -= amt * SK_MEM_QUANTUM;
- atomic_long_sub(amt, prot->memory_allocated);
+
+ sk_memory_allocated_sub(sk, amt);
+
    return 0;
}
EXPORT_SYMBOL(__sk_mem_schedule);
@@ -1750,15 +1754,15 @@ EXPORT_SYMBOL(__sk_mem_schedule);
*/
void __sk_mem_reclaim(struct sock *sk)
{
- struct proto *prot = sk->sk_prot;
+ int *memory_pressure = sk_memory_pressure(sk);

- atomic_long_sub(sk->sk_forward_alloc >> SK_MEM_QUANTUM_SHIFT,
-   prot->memory_allocated);
+ sk_memory_allocated_sub(sk,
+   sk->sk_forward_alloc >> SK_MEM_QUANTUM_SHIFT);
    sk->sk_forward_alloc &= SK_MEM_QUANTUM - 1;

- if (prot->memory_pressure && *prot->memory_pressure &&
-   (atomic_long_read(prot->memory_allocated) < prot->sysctl_mem[0]))
-   *prot->memory_pressure = 0;
+ if (memory_pressure && *memory_pressure &&
+   (sk_memory_allocated(sk) < sk_prot_mem(sk, 0)))
+   *memory_pressure = 0;
}
EXPORT_SYMBOL(__sk_mem_reclaim);

@@ -2477,13 +2481,20 @@ static char proto_method_implemented(const void *method)

```

```

static void proto_seq_printf(struct seq_file *seq, struct proto *proto)
{
+ struct mem_cgroup *cg = mem_cgroup_from_task(current);
+ int *memory_pressure = NULL;
+
+ if (proto->memory_pressure)
+ memory_pressure = proto->memory_pressure;
+
seq_printf(seq, "%-9s %4u %6d %6ld %-3s %6u %-3s %-10s "
"%2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c\n",
proto->name,
proto->obj_size,
sock_prot_inuse_get(seq_file_net(seq), proto),
- proto->memory_allocated != NULL ? atomic_long_read(proto->memory_allocated) : -1L,
- proto->memory_pressure != NULL ? *proto->memory_pressure ? "yes" : "no" : "NI",
+ proto->memory_allocated != NULL ?
+ kcg_memory_allocated(proto, cg) : -1L,
+ memory_pressure != NULL ? *memory_pressure ? "yes" : "no" : "NI",
proto->max_header,
proto->slab == NULL ? "no" : "yes",
module_name(proto->owner),
diff --git a/net/ipv4/proc.c b/net/ipv4/proc.c
index 4bfad5d..535456d 100644
--- a/net/ipv4/proc.c
+++ b/net/ipv4/proc.c
@@ -52,20 +52,21 @@ static int sockstat_seq_show(struct seq_file *seq, void *v)
{
struct net *net = seq->private;
int orphans, sockets;
+ struct mem_cgroup *cg = mem_cgroup_from_task(current);

local_bh_disable();
orphans = percpu_counter_sum_positive(&tcp_orphan_count);
- sockets = percpu_counter_sum_positive(&tcp_sockets_allocated);
+ sockets = kcg_sockets_allocated_sum_positive(&tcp_prot, cg);
local_bh_enable();

socket_seq_show(seq);
seq_printf(seq, "TCP: inuse %d orphan %d tw %d alloc %d mem %ld\n",
sock_prot_inuse_get(net, &tcp_prot), orphans,
tcp_death_row.tw_count, sockets,
- atomic_long_read(&tcp_memory_allocated));
+ kcg_memory_allocated(&tcp_prot, cg));
seq_printf(seq, "UDP: inuse %d mem %ld\n",
sock_prot_inuse_get(net, &udp_prot),
- atomic_long_read(&udp_memory_allocated));
+ kcg_memory_allocated(&udp_prot, cg));

```

```

seq_printf(seq, "UDPLITE: inuse %d\n",
    sock_prot_inuse_get(net, &udplite_prot));
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);
+
+ 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)) {
    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);

    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;

    /* If we are under soft global TCP memory pressure, do not expand. */
- if (atomic_long_read(&tcp_memory_allocated) >= sysctl_tcp_mem[0])
+ if (sk_memory_allocated(sk) >= sk_prot_mem(sk, 0))
    return 0;

    /* If we filled the congestion window, do not expand. */
diff --git a/net/ipv4/tcp_ipv4.c b/net/ipv4/tcp_ipv4.c
index 0ea10ee..f124a4b 100644
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -1914,7 +1914,7 @@ static int tcp_v4_init_sock(struct sock *sk)
    sk->sk_rcvbuf = sysctl_tcp_rmem[1];

    local_bh_disable();
- percpu_counter_inc(&tcp_sockets_allocated);
+ sk_sockets_allocated_inc(sk);
    local_bh_enable();

    return 0;
@@ -1970,7 +1970,7 @@ void tcp_v4_destroy_sock(struct sock *sk)
    tp->cookie_values = NULL;
}

- percpu_counter_dec(&tcp_sockets_allocated);
+ sk_sockets_allocated_dec(sk);
}
EXPORT_SYMBOL(tcp_v4_destroy_sock);

diff --git a/net/ipv4/tcp_output.c b/net/ipv4/tcp_output.c
index 980b98f..04e229b 100644
--- a/net/ipv4/tcp_output.c
+++ b/net/ipv4/tcp_output.c
@@ -1919,7 +1919,7 @@ u32 __tcp_select_window(struct sock *sk)
    if (free_space < (full_space >> 1)) {

```



```

icsk->icsk_ack.quick = 0;

- if (tcp_memory_pressure)
+ if (sk_memory_pressure(sk))
    tp->rcv_ssthresh = min(tp->rcv_ssthresh,
        4U * tp->advmss);

diff --git a/net/ipv4/tcp_timer.c b/net/ipv4/tcp_timer.c
index 2e0f0af..c9f830c 100644
--- a/net/ipv4/tcp_timer.c
+++ b/net/ipv4/tcp_timer.c
@@ -261,7 +261,7 @@ static void tcp_delack_timer(unsigned long data)
 }

out:
- if (tcp_memory_pressure)
+ if (sk_memory_pressure(sk))
    sk_mem_reclaim(sk);
out_unlock:
    bh_unlock_sock(sk);
diff --git a/net/ipv6/tcp_ipv6.c b/net/ipv6/tcp_ipv6.c
index 10b2b31..3a08fcd 100644
--- a/net/ipv6/tcp_ipv6.c
+++ b/net/ipv6/tcp_ipv6.c
@@ -1995,7 +1995,7 @@ static int tcp_v6_init_sock(struct sock *sk)
    sk->sk_rcvbuf = sysctl_tcp_rmem[1];

    local_bh_disable();
- percpu_counter_inc(&tcp_sockets_allocated);
+ sk_sockets_allocated_inc(sk);
    local_bh_enable();

    return 0;
--
1.7.6.4

```
