Subject: [PATCH v5 0/8] per-cgroup tcp buffer pressure settings
Posted by Glauber Costa on Tue, 04 Oct 2011 12:17:52 GMT
View Forum Message <> Reply to Message

[[v3: merge Kirill's suggestions, + a destroy-related bugfix]]
[[v4: Fix a bug with non-mounted cgroups + disallow task movement]]
[[v5: Compile bug with modular ipv6 + tcp files in bytes]]

Kame, Kirill,

I am submitting this again merging most of your comments. I've decided to
leave some of them out:
 * I am not using res_counters for allocated_memory. Besides being more
   expensive than what we need, to make it work in a nice way, we'd have
   to change the !cgroup code, including other protocols than tcp. Also,

 * I am not using failcnt and max_usage_in_bytes for it. I believe the value
   of those lies more in the allocation than in the pressure control. Besides,
   fail conditions lie mostly outside of the memory cgroup's control. (Actually,
   a soft_limit makes a lot of sense, and I do plan to introduce it in a follow
   up series)

If you agree with the above, and there are any other pressing issues, let me
know and I will address them ASAP. Otherwise, let's discuss it. I'm always open.

All:

This patch introduces per-cgroup tcp buffers limitation. This allows
sysadmins to specify a maximum amount of kernel memory that
tcp connections can use at any point in time. TCP is the main interest
in this work, but extending it to other protocols would be easy.

For this to work, I am hooking it into memcg, after the introdution of
an extension for tracking and controlling objects in kernel memory.
Since they are usually not found in page granularity, and are fundamentally
different from userspace memory (not swappable, can't overcommit), they
need their special place inside the Memory Controller.

Right now, the kmem extension is quite basic, and just lays down the
basic infrastucture for the ongoing work.

Although it does not account kernel memory allocated - I preferred to
keep this series simple and leave accounting to the slab allocations when
they arrive.

What it does is to piggyback in the memory control mechanism already present in
/proc/sys/net/ipv4/tcp_mem. There is a soft limit, and a hard limit,
that will suppress allocation when reached. For each non-root cgroup, however,

the file kmem.tcp_maxmem will be used to cap those values.

The usage I have in mind here is containers. Each container will
define its own values for soft and hard limits, but none of them will
be possibly bigger than the value the box' sysadmin specified from
the outside.

To test for any performance impacts of this patch, I used netperf's
TCP_RR benchmark on localhost, so we can have both recv and snd in action.
For this iteration, I am using the 1% confidence interval as suggested by Rick.

Command line used was ./src/netperf -t TCP_RR -H localhost -i 30,3 -I 99,1 and the
results: (I haven't re-run this since nothing major changed from last version, nothing
in core)

Without the patch
==========

Local /Remote
Socket Size Request Resp. Elapsed Trans.
Send Recv Size Size Time Rate
bytes Bytes bytes bytes secs. per sec

16384 87380 1 1 10.00 35356.22
16384 87380

With the patch
==========

Local /Remote
Socket Size Request Resp. Elapsed Trans.
Send Recv Size Size Time Rate
bytes Bytes bytes bytes secs. per sec

16384 87380 1 1 10.00 35399.12
16384 87380

The difference is less than 0.5 %

A simple test with a 1000 level nesting yields more or less the same
difference:

1000 level nesting
==========

Local /Remote
Socket Size Request Resp. Elapsed Trans.

```
Send   Recv  Size    Size    Time     Rate
bytes  Bytes bytes   bytes   secs.    per sec

16384  87380  1       1       10.00    35304.35
16384  87380
```

Glauber Costa (8):
  Basic kernel memory functionality for the Memory Controller
  socket: initial cgroup code.
  foundations of per-cgroup memory pressure controlling.
  per-cgroup tcp buffers control
  per-netns ipv4 sysctl_tcp_mem
  tcp buffer limitation: per-cgroup limit
  Display current tcp memory allocation in kmem cgroup
  Disable task moving when using kernel memory accounting

```
 Documentation/cgroups/memory.txt |   38 ++++-
 crypto/af_alg.c                  |    7 +-
 include/linux/memcontrol.h       |   56 ++++++
 include/net/netns/ipv4.h         |    1 +
 include/net/sock.h               |  127 ++++++++++++-
 include/net/tcp.h                |   29 +++-
 include/net/udp.h                |    3 +-
 include/trace/events/sock.h      |   10 +-
 init/Kconfig                     |   14 ++
 mm/memcontrol.c                  |  371 ++++++++++++++++++++++++++++++++++++++--
 net/core/sock.c                  |  104 ++++++++---
 net/decnet/af_decnet.c           |   21 ++-
 net/ipv4/proc.c                  |    7 +-
 net/ipv4/sysctl_net_ipv4.c       |   71 +++++++-
 net/ipv4/tcp.c                   |   58 ++++---
 net/ipv4/tcp_input.c             |   12 +-
 net/ipv4/tcp_ipv4.c              |   24 ++-
 net/ipv4/tcp_output.c            |    2 +-
 net/ipv4/tcp_timer.c             |    2 +-
 net/ipv4/udp.c                   |   20 ++-
 net/ipv6/tcp_ipv6.c              |   20 ++-
 net/ipv6/udp.c                   |    4 +-
 net/sctp/socket.c                |   35 +++-
 23 files changed, 905 insertions(+), 131 deletions(-)
```

--
1.7.6

Subject: [PATCH v5 2/8] socket: initial cgroup code.
Posted by Glauber Costa on Tue, 04 Oct 2011 12:17:54 GMT
View Forum Message <> Reply to Message

We aim to control the amount of kernel memory pinned at any
time by tcp sockets. To lay the foundations for this work,
this patch adds a pointer to the kmem_cgroup to the socket
structure.

Signed-off-by: Glauber Costa <glommer@parallels.com>
Acked-by: Kirill A. Shutemov<kirill@shutemov.name>
Reviewed-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujtsu.com>
CC: David S. Miller <davem@davemloft.net>
CC: Eric W. Biederman <ebiederm@xmission.com>
---
 include/linux/memcontrol.h | 15 +++++++++++++++
 include/net/sock.h | 2 ++
 mm/memcontrol.c | 36 ++++++++++++++++++++++++++++++++++++
 net/core/sock.c | 3 +++
 4 files changed, 56 insertions(+), 0 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 343bd76..88aea1b 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -376,5 +376,20 @@ mem_cgroup_print_bad_page(struct page *page)
 }
 #endif

+#ifdef CONFIG_INET
+struct sock;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+void sock_update_memcg(struct sock *sk);
+void sock_release_memcg(struct sock *sk);
+
+#else
+static inline void sock_update_memcg(struct sock *sk)
+{
+}
+static inline void sock_release_memcg(struct sock *sk)
+{
+}
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+#endif /* CONFIG_INET */
 #endif /* _LINUX_MEMCONTROL_H */

diff --git a/include/net/sock.h b/include/net/sock.h
index 8e4062f..afe1467 100644
--- a/include/net/sock.h

```
+++ b/include/net/sock.h
@@ -228,6 +228,7 @@ struct sock_common {
   * @sk_security: used by security modules
   * @sk_mark: generic packet mark
   * @sk_classid: this socket's cgroup classid
+  * @sk_cgrp: this socket's kernel memory (kmem) cgroup
   * @sk_write_pending: a write to stream socket waits to start
   * @sk_state_change: callback to indicate change in the state of the sock
   * @sk_data_ready: callback to indicate there is data to be processed
@@ -339,6 +340,7 @@ struct sock {
 #endif
  __u32   sk_mark;
  u32   sk_classid;
+ struct mem_cgroup *sk_cgrp;
  void   (*sk_state_change)(struct sock *sk);
  void   (*sk_data_ready)(struct sock *sk, int bytes);
  void   (*sk_write_space)(struct sock *sk);
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 0871e3f..b1dcba9 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -296,6 +296,42 @@ struct mem_cgroup {
 };

+/* Writing them here to avoid exposing memcg's inner layout */
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+#ifdef CONFIG_INET
+#include <net/sock.h>
+
+void sock_update_memcg(struct sock *sk)
+{
+ /* right now a socket spends its whole life in the same cgroup */
+ if (sk->sk_cgrp) {
+  WARN_ON(1);
+  return;
+ }
+
+ rcu_read_lock();
+ sk->sk_cgrp = mem_cgroup_from_task(current);
+
+ /*
+  * We don't need to protect against anything task-related, because
+  * we are basically stuck with the sock pointer that won't change,
+  * even if the task that originated the socket changes cgroups.
+  *
+  * What we do have to guarantee, is that the chain leading us to
+  * the top level won't change under our noses. Incrementing the
```

```
+  * reference count via cgroup_exclude_rmdir guarantees that.
+  */
+ cgroup_exclude_rmdir(mem_cgroup_css(sk->sk_cgrp));
+ rcu_read_unlock();
+}
+
+void sock_release_memcg(struct sock *sk)
+{
+ cgroup_release_and_wakeup_rmdir(mem_cgroup_css(sk->sk_cgrp));
+}
+#endif /* CONFIG_INET */
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+
 /* Stuffs for move charges at task migration. */
 /*
  * Types of charges to be moved. "move_charge_at_immitgrate" is treated as a
diff --git a/net/core/sock.c b/net/core/sock.c
index bc745d0..5426ba0 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -125,6 +125,7 @@
 #include <net/xfrm.h>
 #include <linux/ipsec.h>
 #include <net/cls_cgroup.h>
+#include <linux/memcontrol.h>

 #include <linux/filter.h>

@@ -1141,6 +1142,7 @@ struct sock *sk_alloc(struct net *net, int family, gfp_t priority,
   atomic_set(&sk->sk_wmem_alloc, 1);

   sock_update_classid(sk);
+  sock_update_memcg(sk);
  }

  return sk;
@@ -1172,6 +1174,7 @@ static void __sk_free(struct sock *sk)
   put_cred(sk->sk_peer_cred);
  put_pid(sk->sk_peer_pid);
  put_net(sock_net(sk));
+ sock_release_memcg(sk);
  sk_prot_free(sk->sk_prot_creator, sk);
 }

--
1.7.6
```

Subject: [PATCH v5 3/8] foundations of per-cgroup memory pressure controlling.
Posted by Glauber Costa on Tue, 04 Oct 2011 12:17:55 GMT
View Forum Message <> Reply to Message

This patch converts struct sock fields memory_pressure,
memory_allocated, sockets_allocated, and sysctl_mem (now prot_mem)
to function pointers, receiving a struct mem_cgroup parameter.

enter_memory_pressure is kept the same, since all its callers
have socket a context, and the kmem_cgroup can be derived from
the socket itself.

To keep things working, the patch convert all users of those fields
to use acessor functions.

In my benchmarks I didn't see a significant performance difference
with this patch applied compared to a baseline (around 1 % diff, thus
inside error margin).

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>
---
 crypto/af_alg.c | 7 ++-
 include/linux/memcontrol.h | 29 ++++++++++-
 include/net/sock.h | 112 ++--
 include/net/tcp.h | 11 +++--
 include/net/udp.h | 3 +-
 include/trace/events/sock.h | 10 ++--
 mm/memcontrol.c | 45 ++++++++++++++++++
 net/core/sock.c | 62 +++++++++++++++++++++++++
 net/decnet/af_decnet.c | 21 +++++++-
 net/ipv4/proc.c | 7 ++-
 net/ipv4/tcp.c | 27 +++++++++-
 net/ipv4/tcp_input.c | 12 ++--
 net/ipv4/tcp_ipv4.c | 12 ++--
 net/ipv4/tcp_output.c | 2 +-
 net/ipv4/tcp_timer.c | 2 +-
 net/ipv4/udp.c | 20 ++++++--
 net/ipv6/tcp_ipv6.c | 10 ++--
 net/ipv6/udp.c | 4 +-
 net/sctp/socket.c | 35 ++++++++++---
 19 files changed, 345 insertions(+), 86 deletions(-)

diff --git a/crypto/af_alg.c b/crypto/af_alg.c
index ac33d5f..9f41324 100644
--- a/crypto/af_alg.c
+++ b/crypto/af_alg.c

```
@@ -29,10 +29,15 @@ struct alg_type_list {

 static atomic_long_t alg_memory_allocated;

+static atomic_long_t *memory_allocated_alg(struct mem_cgroup *memcg)
+{
+ return &alg_memory_allocated;
+}
+
 static struct proto alg_proto = {
   .name    = "ALG",
   .owner   = THIS_MODULE,
- .memory_allocated = &alg_memory_allocated,
+ .memory_allocated = memory_allocated_alg,
   .obj_size  = sizeof(struct alg_sock),
 };

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 88aea1b..2012060 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -361,6 +361,10 @@ static inline
 void mem_cgroup_count_vm_event(struct mm_struct *mm, enum vm_event_item idx)
 {
 }
+static inline struct mem_cgroup *mem_cgroup_from_task(struct task_struct *p)
+{
+ return NULL;
+}
 #endif /* CONFIG_CGROUP_MEM_CONT */

 #if !defined(CONFIG_CGROUP_MEM_RES_CTLR) || !defined(CONFIG_DEBUG_VM)
@@ -378,11 +382,34 @@ mem_cgroup_print_bad_page(struct page *page)

 #ifdef CONFIG_INET
 struct sock;
+struct proto;
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
 void sock_update_memcg(struct sock *sk);
 void sock_release_memcg(struct sock *sk);
-
+void memcg_sock_mem_alloc(struct mem_cgroup *memcg, struct proto *prot,
+      int amt, int *parent_failure);
+void memcg_sock_mem_free(struct mem_cgroup *memcg, struct proto *prot, int amt);
+void memcg_sockets_allocated_dec(struct mem_cgroup *memcg, struct proto *prot);
+void memcg_sockets_allocated_inc(struct mem_cgroup *memcg, struct proto *prot);
 #else
+/* memcontrol includes sockets.h, that includes memcontrol.h ... */
```

```
+static inline void memcg_sock_mem_alloc(struct mem_cgroup *memcg,
+     struct proto *prot, int amt,
+     int *parent_failure)
+{
+}
+static inline void memcg_sock_mem_free(struct mem_cgroup *memcg,
+           struct proto *prot, int amt)
+{
+}
+static inline void memcg_sockets_allocated_dec(struct mem_cgroup *memcg,
+            struct proto *prot)
+{
+}
+static inline void memcg_sockets_allocated_inc(struct mem_cgroup *memcg,
+            struct proto *prot)
+{
+}
 static inline void sock_update_memcg(struct sock *sk)
 {
 }
diff --git a/include/net/sock.h b/include/net/sock.h
index afe1467..c6983cf 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/cgroup.h>

 #include <linux/filter.h>
 #include <linux/rculist_nulls.h>
@@ -168,6 +169,8 @@ struct sock_common {
  /* public: */
 };

+struct mem_cgroup;
+
 /**
   * struct sock - network layer representation of sockets
   * @__sk_common: shared layout with inet_timewait_sock
@@ -786,18 +789,32 @@ struct proto {
  unsigned int  inuse_idx;
 #endif

+ /*
+  * per-cgroup memory tracking:
+  *
```

```
+  * The following functions track memory consumption of network buffers
+  * by cgroup (kmem_cgroup) for the current protocol. As of the rest
+  * of the fields in this structure, not all protocols are required
+  * to implement them. Protocols that don't want to do per-cgroup
+  * memory pressure management, can just assume the root cgroup is used.
+  *
+  */
   /* Memory pressure */
   void   (*enter_memory_pressure)(struct sock *sk);
- atomic_long_t  *memory_allocated; /* Current allocated memory. */
- struct percpu_counter *sockets_allocated; /* Current number of sockets. */
+ /* Pointer to the current memory allocation of this cgroup. */
+ atomic_long_t  *(*memory_allocated)(struct mem_cgroup *memcg);
+ /* Pointer to the current number of sockets in this cgroup. */
+ struct percpu_counter *(*sockets_allocated)(struct mem_cgroup *memcg);
   /*
-  * Pressure flag: try to collapse.
+  * Per cgroup pointer to the pressure flag: try to collapse.
    * Technical note: it is used by multiple contexts non atomically.
    * All the __sk_mem_schedule() is of this nature: accounting
    * is strict, actions are advisory and have some latency.
    */
- int   *memory_pressure;
- long   *sysctl_mem;
+ int   *(*memory_pressure)(struct mem_cgroup *memcg);
+ /* Pointer to the per-cgroup version of the the sysctl_mem field */
+ long   *(*prot_mem)(struct mem_cgroup *memcg);
+
   int   *sysctl_wmem;
   int   *sysctl_rmem;
   int   max_header;
@@ -856,6 +873,91 @@ static inline void sk_refcnt_debug_release(const struct sock *sk)
  #define sk_refcnt_debug_release(sk) do { } while (0)
  #endif /* SOCK_REFCNT_DEBUG */

+#include <linux/memcontrol.h>
+static inline int *sk_memory_pressure(struct sock *sk)
+{
+ int *ret = NULL;
+ if (sk->sk_prot->memory_pressure)
+  ret = sk->sk_prot->memory_pressure(sk->sk_cgrp);
+ return ret;
+}
+
+static inline long sk_prot_mem(struct sock *sk, int index)
+{
+ long *prot = sk->sk_prot->prot_mem(sk->sk_cgrp);
+ return prot[index];
```

```
+}
+
+static inline long
+sk_memory_allocated(struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+ struct mem_cgroup *cg = sk->sk_cgrp;
+
+ return atomic_long_read(prot->memory_allocated(cg));
+}
+
+static inline long
+sk_memory_allocated_add(struct sock *sk, int amt, int *parent_failure)
+{
+ struct proto *prot = sk->sk_prot;
+ struct mem_cgroup *cg = sk->sk_cgrp;
+ long allocated;
+
+ allocated = atomic_long_add_return(amt, prot->memory_allocated(cg));
+ memcg_sock_mem_alloc(cg, prot, amt, parent_failure);
+ return allocated;
+}
+
+static inline void
+sk_memory_allocated_sub(struct sock *sk, int amt)
+{
+ struct proto *prot = sk->sk_prot;
+ struct mem_cgroup *cg = sk->sk_cgrp;
+
+ atomic_long_sub(amt, prot->memory_allocated(cg));
+ memcg_sock_mem_free(cg, prot, amt);
+}
+
+static inline void sk_sockets_allocated_dec(struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+ struct mem_cgroup *cg = sk->sk_cgrp;
+
+ percpu_counter_dec(prot->sockets_allocated(cg));
+ memcg_sockets_allocated_dec(cg, prot);
+}
+
+static inline void sk_sockets_allocated_inc(struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+ struct mem_cgroup *cg = sk->sk_cgrp;
+
+ percpu_counter_inc(prot->sockets_allocated(cg));
```

```
+ memcg_sockets_allocated_inc(cg, prot);
+}
+
+static inline int
+sk_sockets_allocated_read_positive(struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+ struct mem_cgroup *cg = sk->sk_cgrp;
+
+ return percpu_counter_sum_positive(prot->sockets_allocated(cg));
+}
+
+static inline int
+kcg_sockets_allocated_sum_positive(struct proto *prot, struct mem_cgroup *cg)
+{
+ return percpu_counter_sum_positive(prot->sockets_allocated(cg));
+}
+
+static inline long
+kcg_memory_allocated(struct proto *prot, struct mem_cgroup *cg)
+{
+ return atomic_long_read(prot->memory_allocated(cg));
+}
+

 #ifdef CONFIG_PROC_FS
 /* Called with local bh disabled */
diff --git a/include/net/tcp.h b/include/net/tcp.h
index acc620a..2cbbc13 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -45,6 +45,7 @@
 #include <net/dst.h>

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

 extern struct inet_hashinfo tcp_hashinfo;

@@ -253,9 +254,11 @@ extern int sysctl_tcp_cookie_size;
 extern int sysctl_tcp_thin_linear_timeouts;
 extern int sysctl_tcp_thin_dupack;

-extern atomic_long_t tcp_memory_allocated;
-extern struct percpu_counter tcp_sockets_allocated;
-extern int tcp_memory_pressure;
+struct mem_cgroup;
+extern long *tcp_sysctl_mem(struct mem_cgroup *memcg);
```

```
+struct percpu_counter *sockets_allocated_tcp(struct mem_cgroup *memcg);
+int *memory_pressure_tcp(struct mem_cgroup *memcg);
+atomic_long_t *memory_allocated_tcp(struct mem_cgroup *memcg);

 /*
  * The next routines deal with comparing 32 bit unsigned ints
@@ -286,7 +289,7 @@ 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/include/net/udp.h b/include/net/udp.h
index 67ea6fc..b96ed51 100644
--- a/include/net/udp.h
+++ b/include/net/udp.h
@@ -105,7 +105,8 @@ static inline struct udp_hslot *udp_hashslot2(struct udp_table *table,

 extern struct proto udp_prot;

-extern atomic_long_t udp_memory_allocated;
+atomic_long_t *memory_allocated_udp(struct mem_cgroup *memcg);
+long *udp_sysctl_mem(struct mem_cgroup *memcg);

 /* sysctl variables for udp */
 extern long sysctl_udp_mem[3];
diff --git a/include/trace/events/sock.h b/include/trace/events/sock.h
index 779abb9..12a6083 100644
--- a/include/trace/events/sock.h
+++ b/include/trace/events/sock.h
@@ -37,7 +37,7 @@ TRACE_EVENT(sock_exceed_buf_limit,

  TP_STRUCT__entry(
   __array(char, name, 32)
-  __field(long *, sysctl_mem)
+  __field(long *, prot_mem)
   __field(long, allocated)
   __field(int, sysctl_rmem)
   __field(int, rmem_alloc)
@@ -45,7 +45,7 @@ TRACE_EVENT(sock_exceed_buf_limit,

  TP_fast_assign(
   strncpy(__entry->name, prot->name, 32);
-  __entry->sysctl_mem = prot->sysctl_mem;
+  __entry->prot_mem = sk->sk_prot->prot_mem(sk->sk_cgrp);
```

```
   __entry->allocated = allocated;
   __entry->sysctl_rmem = prot->sysctl_rmem[0];
   __entry->rmem_alloc = atomic_read(&sk->sk_rmem_alloc);
@@ -54,9 +54,9 @@ TRACE_EVENT(sock_exceed_buf_limit,
  TP_printk("proto:%s sysctl_mem=%ld,%ld,%ld allocated=%ld "
   "sysctl_rmem=%d rmem_alloc=%d",
   __entry->name,
-  __entry->sysctl_mem[0],
-  __entry->sysctl_mem[1],
-  __entry->sysctl_mem[2],
+  __entry->prot_mem[0],
+  __entry->prot_mem[1],
+  __entry->prot_mem[2],
   __entry->allocated,
   __entry->sysctl_rmem,
   __entry->rmem_alloc)
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index b1dcba9..60ab41d 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -296,6 +296,7 @@ struct mem_cgroup {
  spinlock_t pcp_counter_lock;
 };

+static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg);
 /* Writing them here to avoid exposing memcg's inner layout */
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
 #ifdef CONFIG_INET
@@ -329,6 +330,49 @@ void sock_release_memcg(struct sock *sk)
 {
  cgroup_release_and_wakeup_rmdir(mem_cgroup_css(sk->sk_cgrp));
 }
+
+void memcg_sock_mem_alloc(struct mem_cgroup *memcg, struct proto *prot,
+     int amt, int *parent_failure)
+{
+ memcg = parent_mem_cgroup(memcg);
+ for (; memcg != NULL; memcg = parent_mem_cgroup(memcg)) {
+  long alloc;
+  long *prot_mem = prot->prot_mem(memcg);
+  /*
+   * Large nestings are not the common case, and stopping in the
+   * middle would be complicated enough, that we bill it all the
+   * way through the root, and if needed, unbill everything later
+   */
+  alloc = atomic_long_add_return(amt,
+           prot->memory_allocated(memcg));
+  *parent_failure |= (alloc > prot_mem[2]);
```

```
+ }
+}
+EXPORT_SYMBOL(memcg_sock_mem_alloc);
+
+void memcg_sock_mem_free(struct mem_cgroup *memcg, struct proto *prot, int amt)
+{
+ memcg = parent_mem_cgroup(memcg);
+ for (; memcg != NULL; memcg = parent_mem_cgroup(memcg))
+  atomic_long_sub(amt, prot->memory_allocated(memcg));
+}
+EXPORT_SYMBOL(memcg_sock_mem_free);
+
+void memcg_sockets_allocated_dec(struct mem_cgroup *memcg, struct proto *prot)
+{
+ memcg = parent_mem_cgroup(memcg);
+ for (; memcg; memcg = parent_mem_cgroup(memcg))
+  percpu_counter_dec(prot->sockets_allocated(memcg));
+}
+EXPORT_SYMBOL(memcg_sockets_allocated_dec);
+
+void memcg_sockets_allocated_inc(struct mem_cgroup *memcg, struct proto *prot)
+{
+ memcg = parent_mem_cgroup(memcg);
+ for (; memcg; memcg = parent_mem_cgroup(memcg))
+  percpu_counter_inc(prot->sockets_allocated(memcg));
+}
+EXPORT_SYMBOL(memcg_sockets_allocated_inc);
 #endif /* CONFIG_INET */
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

@@ -414,7 +458,6 @@ enum mem_type {

 static void mem_cgroup_get(struct mem_cgroup *mem);
 static void mem_cgroup_put(struct mem_cgroup *mem);
-static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *mem);
 static void drain_all_stock_async(struct mem_cgroup *mem);

 static struct mem_cgroup_per_zone *
diff --git a/net/core/sock.c b/net/core/sock.c
index 5426ba0..6e3ace7 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -1293,7 +1293,7 @@ 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))
@@ -1684,30 +1684,33 @@ 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;
+ int parent_failure = 0;

  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, &parent_failure);
+
+ /* Over hard limit (we, or our parents) */
+ if (parent_failure || (allocated > sk_prot_mem(sk, 2)))
+  goto suppress_allocation;

  /* Under limit. */
- if (allocated <= prot->sysctl_mem[0]) {
-  if (prot->memory_pressure && *prot->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])
-  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])
```

```
@@ -1717,13 +1720,13 @@ 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))
@@ -1746,7 +1749,9 @@ 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);
@@ -1757,15 +1762,15 @@ 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);

@@ -2484,13 +2489,20 @@ 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(cg);
+
  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
%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/decnet/af_decnet.c b/net/decnet/af_decnet.c
index 19acd00..39f83b4 100644
--- a/net/decnet/af_decnet.c
+++ b/net/decnet/af_decnet.c
@@ -458,13 +458,28 @@ static void dn_enter_memory_pressure(struct sock *sk)
  }
 }

+static atomic_long_t *memory_allocated_dn(struct mem_cgroup *memcg)
+{
+ return &decnet_memory_allocated;
+}
+
+static int *memory_pressure_dn(struct mem_cgroup *memcg)
+{
+ return &dn_memory_pressure;
+}
+
+static long *dn_sysctl_mem(struct mem_cgroup *memcg)
+{
+ return sysctl_decnet_mem;
```

```
+}
+
 static struct proto dn_proto = {
  .name   = "NSP",
  .owner   = THIS_MODULE,
  .enter_memory_pressure = dn_enter_memory_pressure,
- .memory_pressure = &dn_memory_pressure,
- .memory_allocated = &decnet_memory_allocated,
- .sysctl_mem  = sysctl_decnet_mem,
+ .memory_pressure = memory_pressure_dn,
+ .memory_allocated = memory_allocated_dn,
+ .prot_mem  = dn_sysctl_mem,
  .sysctl_wmem  = sysctl_decnet_wmem,
  .sysctl_rmem  = sysctl_decnet_rmem,
  .max_header  = DN_MAX_NSP_DATA_HEADER + 64,
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 46febca..ca82b90 100644
--- a/net/ipv4/tcp.c
+++ b/net/ipv4/tcp.c
```

```
@@ -291,13 +291,11 @@ EXPORT_SYMBOL(sysctl_tcp_rmem);
 EXPORT_SYMBOL(sysctl_tcp_wmem);

 atomic_long_t tcp_memory_allocated; /* Current allocated memory. */
-EXPORT_SYMBOL(tcp_memory_allocated);

 /*
  * Current number of TCP sockets.
  */
 struct percpu_counter tcp_sockets_allocated;
-EXPORT_SYMBOL(tcp_sockets_allocated);

 /*
  * TCP splice context
@@ -315,7 +313,18 @@ struct tcp_splice_state {
  * is strict, actions are advisory and have some latency.
  */
 int tcp_memory_pressure __read_mostly;
-EXPORT_SYMBOL(tcp_memory_pressure);
+
+int *memory_pressure_tcp(struct mem_cgroup *memcg)
+{
+ return &tcp_memory_pressure;
+}
+EXPORT_SYMBOL(memory_pressure_tcp);
+
+struct percpu_counter *sockets_allocated_tcp(struct mem_cgroup *memcg)
+{
+ return &tcp_sockets_allocated;
+}
+EXPORT_SYMBOL(sockets_allocated_tcp);

 void tcp_enter_memory_pressure(struct sock *sk)
 {
@@ -326,6 +335,18 @@ void tcp_enter_memory_pressure(struct sock *sk)
 }
 EXPORT_SYMBOL(tcp_enter_memory_pressure);

+long *tcp_sysctl_mem(struct mem_cgroup *memcg)
+{
+ return sysctl_tcp_mem;
+}
+EXPORT_SYMBOL(tcp_sysctl_mem);
+
+atomic_long_t *memory_allocated_tcp(struct mem_cgroup *memcg)
+{
+ return &tcp_memory_allocated;
+}
```

```
+EXPORT_SYMBOL(memory_allocated_tcp);
+
 /* Convert seconds to retransmits based on initial and max timeout */
 static u8 secs_to_retrans(int seconds, int timeout, int rto_max)
 {
diff --git a/net/ipv4/tcp_input.c b/net/ipv4/tcp_input.c
index 21fab3e..1529fbe 100644
--- a/net/ipv4/tcp_input.c
+++ b/net/ipv4/tcp_input.c
@@ -316,7 +316,7 @@ static void tcp_grow_window(struct sock *sk, 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
@@ -398,8 +398,8 @@ 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]);
  }
@@ -4806,7 +4806,7 @@ 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);
@@ -4872,11 +4872,11 @@ static int tcp_should_expand_sndbuf(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 c34f015..3227f6a 100644
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -1908,7 +1908,7 @@ 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;
@@ -1964,7 +1964,7 @@ 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);

@@ -2605,11 +2605,11 @@ struct proto tcp_prot = {
  .unhash   = inet_unhash,
  .get_port  = inet_csk_get_port,
  .enter_memory_pressure = tcp_enter_memory_pressure,
- .sockets_allocated = &tcp_sockets_allocated,
+ .memory_pressure = memory_pressure_tcp,
+ .sockets_allocated = sockets_allocated_tcp,
  .orphan_count  = &tcp_orphan_count,
- .memory_allocated = &tcp_memory_allocated,
- .memory_pressure = &tcp_memory_pressure,
- .sysctl_mem  = sysctl_tcp_mem,
+ .memory_allocated = memory_allocated_tcp,
+ .prot_mem  = tcp_sysctl_mem,
  .sysctl_wmem  = sysctl_tcp_wmem,
  .sysctl_rmem  = sysctl_tcp_rmem,
  .max_header  = MAX_TCP_HEADER,
diff --git a/net/ipv4/tcp_output.c b/net/ipv4/tcp_output.c
index 882e0b0..06aeb31 100644
--- a/net/ipv4/tcp_output.c
+++ b/net/ipv4/tcp_output.c
@@ -1912,7 +1912,7 @@ 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 ecd44b0..2c67617 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/ipv4/udp.c b/net/ipv4/udp.c
index 1b5a193..f8d72ce 100644
--- a/net/ipv4/udp.c
+++ b/net/ipv4/udp.c
@@ -120,9 +120,6 @@ EXPORT_SYMBOL(sysctl_udp_rmem_min);
 int sysctl_udp_wmem_min __read_mostly;
 EXPORT_SYMBOL(sysctl_udp_wmem_min);

-atomic_long_t udp_memory_allocated;
-EXPORT_SYMBOL(udp_memory_allocated);
-
 #define MAX_UDP_PORTS 65536
 #define PORTS_PER_CHAIN (MAX_UDP_PORTS / UDP_HTABLE_SIZE_MIN)

@@ -1918,6 +1915,19 @@ unsigned int udp_poll(struct file *file, struct socket *sock, poll_table
*wait)
 }
 EXPORT_SYMBOL(udp_poll);

+static atomic_long_t udp_memory_allocated;
+atomic_long_t *memory_allocated_udp(struct mem_cgroup *memcg)
+{
+ return &udp_memory_allocated;
+}
+EXPORT_SYMBOL(memory_allocated_udp);
+
+long *udp_sysctl_mem(struct mem_cgroup *memcg)
+{
```

```
+ return sysctl_udp_mem;
+}
+EXPORT_SYMBOL(udp_sysctl_mem);
+
 struct proto udp_prot = {
  .name      = "UDP",
  .owner     = THIS_MODULE,
@@ -1936,8 +1946,8 @@ struct proto udp_prot = {
  .unhash     = udp_lib_unhash,
  .rehash     = udp_v4_rehash,
  .get_port    = udp_v4_get_port,
- .memory_allocated  = &udp_memory_allocated,
- .sysctl_mem    = sysctl_udp_mem,
+ .memory_allocated  = &memory_allocated_udp,
+ .prot_mem    = udp_sysctl_mem,
  .sysctl_wmem    = &sysctl_udp_wmem_min,
  .sysctl_rmem    = &sysctl_udp_rmem_min,
  .obj_size    = sizeof(struct udp_sock),
diff --git a/net/ipv6/tcp_ipv6.c b/net/ipv6/tcp_ipv6.c
index 3c9fa61..962fb56 100644
--- a/net/ipv6/tcp_ipv6.c
+++ b/net/ipv6/tcp_ipv6.c
@@ -1987,7 +1987,7 @@ 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;
@@ -2196,11 +2196,11 @@ struct proto tcpv6_prot = {
  .unhash   = inet_unhash,
  .get_port  = inet_csk_get_port,
  .enter_memory_pressure = tcp_enter_memory_pressure,
- .sockets_allocated = &tcp_sockets_allocated,
- .memory_allocated = &tcp_memory_allocated,
- .memory_pressure = &tcp_memory_pressure,
+ .sockets_allocated = sockets_allocated_tcp,
+ .memory_allocated = memory_allocated_tcp,
+ .memory_pressure = memory_pressure_tcp,
  .orphan_count  = &tcp_orphan_count,
- .sysctl_mem  = sysctl_tcp_mem,
+ .prot_mem  = tcp_sysctl_mem,
  .sysctl_wmem  = sysctl_tcp_wmem,
  .sysctl_rmem  = sysctl_tcp_rmem,
  .max_header  = MAX_TCP_HEADER,
diff --git a/net/ipv6/udp.c b/net/ipv6/udp.c
```

```
index bb95e8e..00611b8 100644
--- a/net/ipv6/udp.c
+++ b/net/ipv6/udp.c
@@ -1465,8 +1465,8 @@ struct proto udpv6_prot = {
  .unhash      = udp_lib_unhash,
  .rehash      = udp_v6_rehash,
  .get_port    = udp_v6_get_port,
- .memory_allocated  = &udp_memory_allocated,
- .sysctl_mem    = sysctl_udp_mem,
+ .memory_allocated  = memory_allocated_udp,
+ .prot_mem    = udp_sysctl_mem,
  .sysctl_wmem    = &sysctl_udp_wmem_min,
  .sysctl_rmem    = &sysctl_udp_rmem_min,
  .obj_size    = sizeof(struct udp6_sock),
diff --git a/net/sctp/socket.c b/net/sctp/socket.c
index 836aa63..62eb178 100644
--- a/net/sctp/socket.c
+++ b/net/sctp/socket.c
@@ -119,11 +119,30 @@ static int sctp_memory_pressure;
 static atomic_long_t sctp_memory_allocated;
 struct percpu_counter sctp_sockets_allocated;

+static long *sctp_sysctl_mem(struct mem_cgroup *memcg)
+{
+ return sysctl_sctp_mem;
+}
+
 static void sctp_enter_memory_pressure(struct sock *sk)
 {
  sctp_memory_pressure = 1;
 }

+static int *memory_pressure_sctp(struct mem_cgroup *memcg)
+{
+ return &sctp_memory_pressure;
+}
+
+static atomic_long_t *memory_allocated_sctp(struct mem_cgroup *memcg)
+{
+ return &sctp_memory_allocated;
+}
+
+static struct percpu_counter *sockets_allocated_sctp(struct mem_cgroup *memcg)
+{
+ return &sctp_sockets_allocated;
+}

 /* Get the sndbuf space available at the time on the association.  */
```

```
 static inline int sctp_wspace(struct sctp_association *asoc)
@@ -6831,13 +6850,13 @@ struct proto sctp_prot = {
  .unhash      = sctp_unhash,
  .get_port    = sctp_get_port,
  .obj_size    =  sizeof(struct sctp_sock),
- .sysctl_mem  =  sysctl_sctp_mem,
+ .prot_mem    =  sctp_sysctl_mem,
  .sysctl_rmem =  sysctl_sctp_rmem,
  .sysctl_wmem =  sysctl_sctp_wmem,
- .memory_pressure = &sctp_memory_pressure,
+ .memory_pressure = memory_pressure_sctp,
  .enter_memory_pressure = sctp_enter_memory_pressure,
- .memory_allocated = &sctp_memory_allocated,
- .sockets_allocated = &sctp_sockets_allocated,
+ .memory_allocated = memory_allocated_sctp,
+ .sockets_allocated = sockets_allocated_sctp,
 };

 #if defined(CONFIG_IPV6) || defined(CONFIG_IPV6_MODULE)
@@ -6863,12 +6882,12 @@ struct proto sctpv6_prot = {
  .unhash  = sctp_unhash,
  .get_port = sctp_get_port,
  .obj_size = sizeof(struct sctp6_sock),
- .sysctl_mem = sysctl_sctp_mem,
+ .prot_mem = sctp_sysctl_mem,
  .sysctl_rmem = sysctl_sctp_rmem,
  .sysctl_wmem = sysctl_sctp_wmem,
- .memory_pressure = &sctp_memory_pressure,
+ .memory_pressure = memory_pressure_sctp,
  .enter_memory_pressure = sctp_enter_memory_pressure,
- .memory_allocated = &sctp_memory_allocated,
- .sockets_allocated = &sctp_sockets_allocated,
+ .memory_allocated = memory_allocated_sctp,
+ .sockets_allocated = sockets_allocated_sctp,
 };
 #endif /* defined(CONFIG_IPV6) || defined(CONFIG_IPV6_MODULE) */
--
1.7.6
```

Subject: [PATCH v5 4/8] per-cgroup tcp buffers control
Posted by Glauber Costa on Tue, 04 Oct 2011 12:17:56 GMT
View Forum Message <> Reply to Message

With all the infrastructure in place, this patch implements
per-cgroup control for tcp memory pressure handling.

Signed-off-by: Glauber Costa <glommer@parallels.com>

```
Reviewed-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujtisu.com>
CC: David S. Miller <davem@davemloft.net>
CC: Eric W. Biederman <ebiederm@xmission.com>
---
 include/linux/memcontrol.h |    4 ++
 include/net/sock.h         |   15 ++++++-
 include/net/tcp.h          |   15 +++++++
 mm/memcontrol.c            |   97 ++++++++++++++++++++++++++++++++++++++++++++++
 net/core/sock.c            |   39 ++++++++++++++++-
 net/ipv4/tcp.c             |   44 +++++++++++++++----------
 net/ipv4/tcp_ipv4.c        |   12 +++++-
 net/ipv6/tcp_ipv6.c        |   10 ++++-
 8 files changed, 207 insertions(+), 29 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 2012060..50af61d 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -391,6 +391,10 @@ void memcg_sock_mem_alloc(struct mem_cgroup *memcg, struct proto
*prot,
 void memcg_sock_mem_free(struct mem_cgroup *memcg, struct proto *prot, int amt);
 void memcg_sockets_allocated_dec(struct mem_cgroup *memcg, struct proto *prot);
 void memcg_sockets_allocated_inc(struct mem_cgroup *memcg, struct proto *prot);
+int tcp_init_cgroup(struct proto *prot, struct cgroup *cgrp,
+      struct cgroup_subsys *ss);
+void tcp_destroy_cgroup(struct proto *prot, struct cgroup *cgrp,
+   struct cgroup_subsys *ss);
 #else
 /* memcontrol includes sockets.h, that includes memcontrol.h ... */
 static inline void memcg_sock_mem_alloc(struct mem_cgroup *memcg,
diff --git a/include/net/sock.h b/include/net/sock.h
index c6983cf..0625d79 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -64,6 +64,8 @@
 #include <net/dst.h>
 #include <net/checksum.h>

+int sockets_populate(struct cgroup *cgrp, struct cgroup_subsys *ss);
+void sockets_destroy(struct cgroup *cgrp, struct cgroup_subsys *ss);
 /*
  * This structure really needs to be cleaned up.
  * Most of it is for TCP, and not used by any of
@@ -814,7 +816,18 @@ struct proto {
  int   *(*memory_pressure)(struct mem_cgroup *memcg);
  /* Pointer to the per-cgroup version of the the sysctl_mem field */
  long   *(*prot_mem)(struct mem_cgroup *memcg);
-
```

```
+ /*
+  * cgroup specific init/deinit functions. Called once for all
+  * protocols that implement it, from cgroups populate function.
+  * This function has to setup any files the protocol want to
+  * appear in the kmem cgroup filesystem.
+  */
+ int   (*init_cgroup)(struct proto *prot,
+            struct cgroup *cgrp,
+            struct cgroup_subsys *ss);
+ void   (*destroy_cgroup)(struct proto *prot,
+         struct cgroup *cgrp,
+         struct cgroup_subsys *ss);
  int   *sysctl_wmem;
  int   *sysctl_rmem;
  int   max_header;
diff --git a/include/net/tcp.h b/include/net/tcp.h
index 2cbbc13..7ae7b4b 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -255,6 +255,20 @@ extern int sysctl_tcp_thin_linear_timeouts;
 extern int sysctl_tcp_thin_dupack;

 struct mem_cgroup;
+struct tcp_memcontrol {
+ /* per-cgroup tcp memory pressure knobs */
+ atomic_long_t tcp_memory_allocated;
+ struct percpu_counter tcp_sockets_allocated;
+ /* those two are read-mostly, leave them at the end */
+ long tcp_prot_mem[3];
+ int tcp_memory_pressure;
+};
+
+extern long *tcp_sysctl_mem_nocg(struct mem_cgroup *memcg);
+struct percpu_counter *sockets_allocated_tcp_nocg(struct mem_cgroup *memcg);
+int *memory_pressure_tcp_nocg(struct mem_cgroup *memcg);
+atomic_long_t *memory_allocated_tcp_nocg(struct mem_cgroup *memcg);
+
 extern long *tcp_sysctl_mem(struct mem_cgroup *memcg);
 struct percpu_counter *sockets_allocated_tcp(struct mem_cgroup *memcg);
 int *memory_pressure_tcp(struct mem_cgroup *memcg);
@@ -1022,6 +1036,7 @@ static inline void tcp_openreq_init(struct request_sock *req,
  ireq->loc_port = tcp_hdr(skb)->dest;
 }

+extern void tcp_enter_memory_pressure_nocg(struct sock *sk);
 extern void tcp_enter_memory_pressure(struct sock *sk);

 static inline int keepalive_intvl_when(const struct tcp_sock *tp)
```

```
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 60ab41d..9598e3e 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -49,6 +49,9 @@
 #include <linux/cpu.h>
 #include <linux/oom.h>
 #include "internal.h"
+#ifdef CONFIG_INET
+#include <net/tcp.h>
+#endif

 #include <asm/uaccess.h>

@@ -294,6 +297,10 @@ struct mem_cgroup {
   */
  struct mem_cgroup_stat_cpu nocpu_base;
  spinlock_t pcp_counter_lock;
+
+#ifdef CONFIG_INET
+ struct tcp_memcontrol tcp;
+#endif
 };

 static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg);
@@ -301,6 +308,7 @@ static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup
*memcg);
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
 #ifdef CONFIG_INET
 #include <net/sock.h>
+#include <net/ip.h>

 void sock_update_memcg(struct sock *sk)
 {
@@ -373,6 +381,80 @@ void memcg_sockets_allocated_inc(struct mem_cgroup *memcg, struct
proto *prot)
   percpu_counter_inc(prot->sockets_allocated(memcg));
 }
 EXPORT_SYMBOL(memcg_sockets_allocated_inc);
+
+static struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont);
+/*
+ * Pressure flag: try to collapse.
+ * Technical note: it is used by multiple contexts non atomically.
+ * All the __sk_mem_schedule() is of this nature: accounting
+ * is strict, actions are advisory and have some latency.
+ */
+void tcp_enter_memory_pressure(struct sock *sk)
```

```
+{
+ struct mem_cgroup *memcg = sk->sk_cgrp;
+ if (!memcg->tcp.tcp_memory_pressure) {
+  NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPMEMORYPRESSURES);
+  memcg->tcp.tcp_memory_pressure = 1;
+ }
+}
+EXPORT_SYMBOL(tcp_enter_memory_pressure);
+
+long *tcp_sysctl_mem(struct mem_cgroup *cg)
+{
+ return cg->tcp.tcp_prot_mem;
+}
+EXPORT_SYMBOL(tcp_sysctl_mem);
+
+atomic_long_t *memory_allocated_tcp(struct mem_cgroup *cg)
+{
+ return &(cg->tcp.tcp_memory_allocated);
+}
+EXPORT_SYMBOL(memory_allocated_tcp);
+
+int *memory_pressure_tcp(struct mem_cgroup *memcg)
+{
+ return &memcg->tcp.tcp_memory_pressure;
+}
+EXPORT_SYMBOL(memory_pressure_tcp);
+
+struct percpu_counter *sockets_allocated_tcp(struct mem_cgroup *memcg)
+{
+ return &memcg->tcp.tcp_sockets_allocated;
+}
+EXPORT_SYMBOL(sockets_allocated_tcp);
+
+static void tcp_create_cgroup(struct mem_cgroup *cg, struct cgroup_subsys *ss)
+{
+ cg->tcp.tcp_memory_pressure = 0;
+ atomic_long_set(&cg->tcp.tcp_memory_allocated, 0);
+ percpu_counter_init(&cg->tcp.tcp_sockets_allocated, 0);
+}
+
+int tcp_init_cgroup(struct proto *prot, struct cgroup *cgrp,
+       struct cgroup_subsys *ss)
+{
+ struct mem_cgroup *cg = mem_cgroup_from_cont(cgrp);
+ /*
+  * We need to initialize it at populate, not create time.
+  * This is because net sysctl tables are not up until much
+  * later
```

```
+  */
+ cg->tcp.tcp_prot_mem[0] = sysctl_tcp_mem[0];
+ cg->tcp.tcp_prot_mem[1] = sysctl_tcp_mem[1];
+ cg->tcp.tcp_prot_mem[2] = sysctl_tcp_mem[2];
+
+ return 0;
+}
+EXPORT_SYMBOL(tcp_init_cgroup);
+
+void tcp_destroy_cgroup(struct proto *prot, struct cgroup *cgrp,
+    struct cgroup_subsys *ss)
+{
+ struct mem_cgroup *cg = mem_cgroup_from_cont(cgrp);
+
+ percpu_counter_destroy(&cg->tcp.tcp_sockets_allocated);
+}
+EXPORT_SYMBOL(tcp_destroy_cgroup);
 #endif /* CONFIG_INET */
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

@@ -4864,9 +4946,18 @@ static int register_kmem_files(struct cgroup *cont, struct
cgroup_subsys *ss)
  if (!mem_cgroup_is_root(memcg))
   ret = cgroup_add_files(cont, ss, kmem_cgroup_files,
      ARRAY_SIZE(kmem_cgroup_files));
+
+ if (!ret)
+  ret = sockets_populate(cont, ss);
+
  return ret;
 };

+static void kmem_cgroup_destroy(struct cgroup_subsys *ss,
+     struct cgroup *cont)
+{
+ sockets_destroy(cont, ss);
+}
 #else
 static int register_kmem_files(struct cgroup *cont, struct cgroup_subsys *ss)
 {
@@ -5089,6 +5180,10 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
  mem->last_scanned_node = MAX_NUMNODES;
  INIT_LIST_HEAD(&mem->oom_notify);

+#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) && defined(CONFIG_INET)
+ tcp_create_cgroup(mem, ss);
+#endif
+
```

```
  if (parent)
   mem->swappiness = mem_cgroup_swappiness(parent);
  atomic_set(&mem->refcnt, 1);
@@ -5114,6 +5209,8 @@ static void mem_cgroup_destroy(struct cgroup_subsys *ss,
 {
  struct mem_cgroup *mem = mem_cgroup_from_cont(cont);

+ kmem_cgroup_destroy(ss, cont);
+
  mem_cgroup_put(mem);
 }

diff --git a/net/core/sock.c b/net/core/sock.c
index 6e3ace7..ef3a9a4 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -135,6 +135,42 @@
 #include <net/tcp.h>
 #endif

+static DEFINE_RWLOCK(proto_list_lock);
+static LIST_HEAD(proto_list);
+
+int sockets_populate(struct cgroup *cgrp, struct cgroup_subsys *ss)
+{
+ struct proto *proto;
+ int ret = 0;
+
+ read_lock(&proto_list_lock);
+ list_for_each_entry(proto, &proto_list, node) {
+  if (proto->init_cgroup)
+   ret = proto->init_cgroup(proto, cgrp, ss);
+   if (ret)
+    goto out;
+ }
+
+ read_unlock(&proto_list_lock);
+ return ret;
+out:
+ list_for_each_entry_continue_reverse(proto, &proto_list, node)
+  if (proto->destroy_cgroup)
+   proto->destroy_cgroup(proto, cgrp, ss);
+ read_unlock(&proto_list_lock);
+ return ret;
+}
+
+void sockets_destroy(struct cgroup *cgrp, struct cgroup_subsys *ss)
+{
```

```
+ struct proto *proto;
+ read_lock(&proto_list_lock);
+ list_for_each_entry_reverse(proto, &proto_list, node)
+  if (proto->destroy_cgroup)
+   proto->destroy_cgroup(proto, cgrp, ss);
+ read_unlock(&proto_list_lock);
+}
+
 /*
  * Each address family might have different locking rules, so we have
  * one slock key per address family:
@@ -2262,9 +2298,6 @@ void sk_common_release(struct sock *sk)
 }
 EXPORT_SYMBOL(sk_common_release);

-static DEFINE_RWLOCK(proto_list_lock);
-static LIST_HEAD(proto_list);
-
 #ifdef CONFIG_PROC_FS
 #define PROTO_INUSE_NR 64 /* should be enough for the first time */
 struct prot_inuse {
diff --git a/net/ipv4/tcp.c b/net/ipv4/tcp.c
index ca82b90..bbd3989 100644
--- a/net/ipv4/tcp.c
+++ b/net/ipv4/tcp.c
@@ -290,13 +290,6 @@ EXPORT_SYMBOL(sysctl_tcp_mem);
 EXPORT_SYMBOL(sysctl_tcp_rmem);
 EXPORT_SYMBOL(sysctl_tcp_wmem);

-atomic_long_t tcp_memory_allocated; /* Current allocated memory. */
-
-/*
- * Current number of TCP sockets.
- */
-struct percpu_counter tcp_sockets_allocated;
-
 /*
  * TCP splice context
  */
@@ -306,46 +299,49 @@ struct tcp_splice_state {
  unsigned int flags;
 };

-/*
- * Pressure flag: try to collapse.
- * Technical note: it is used by multiple contexts non atomically.
- * All the __sk_mem_schedule() is of this nature: accounting
- * is strict, actions are advisory and have some latency.
```

```
- */
+/* Current number of TCP sockets. */
+struct percpu_counter tcp_sockets_allocated;
+atomic_long_t tcp_memory_allocated; /* Current allocated memory. */
 int tcp_memory_pressure __read_mostly;

-int *memory_pressure_tcp(struct mem_cgroup *memcg)
+int *memory_pressure_tcp_nocg(struct mem_cgroup *memcg)
 {
  return &tcp_memory_pressure;
 }
-EXPORT_SYMBOL(memory_pressure_tcp);
+EXPORT_SYMBOL(memory_pressure_tcp_nocg);

-struct percpu_counter *sockets_allocated_tcp(struct mem_cgroup *memcg)
+struct percpu_counter *sockets_allocated_tcp_nocg(struct mem_cgroup *memcg)
 {
  return &tcp_sockets_allocated;
 }
-EXPORT_SYMBOL(sockets_allocated_tcp);
+EXPORT_SYMBOL(sockets_allocated_tcp_nocg);

-void tcp_enter_memory_pressure(struct sock *sk)
+/*
+ * Pressure flag: try to collapse.
+ * Technical note: it is used by multiple contexts non atomically.
+ * All the __sk_mem_schedule() is of this nature: accounting
+ * is strict, actions are advisory and have some latency.
+ */
+void tcp_enter_memory_pressure_nocg(struct sock *sk)
 {
  if (!tcp_memory_pressure) {
   NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPMEMORYPRESSURES);
   tcp_memory_pressure = 1;
  }
 }
-EXPORT_SYMBOL(tcp_enter_memory_pressure);
+EXPORT_SYMBOL(tcp_enter_memory_pressure_nocg);

-long *tcp_sysctl_mem(struct mem_cgroup *memcg)
+long *tcp_sysctl_mem_nocg(struct mem_cgroup *memcg)
 {
  return sysctl_tcp_mem;
 }
-EXPORT_SYMBOL(tcp_sysctl_mem);
+EXPORT_SYMBOL(tcp_sysctl_mem_nocg);

-atomic_long_t *memory_allocated_tcp(struct mem_cgroup *memcg)
```

```
+atomic_long_t *memory_allocated_tcp_nocg(struct mem_cgroup *memcg)
 {
  return &tcp_memory_allocated;
 }
-EXPORT_SYMBOL(memory_allocated_tcp);
+EXPORT_SYMBOL(memory_allocated_tcp_nocg);

 /* Convert seconds to retransmits based on initial and max timeout */
 static u8 secs_to_retrans(int seconds, int timeout, int rto_max)
@@ -3247,7 +3243,9 @@ void __init tcp_init(void)

  BUILD_BUG_ON(sizeof(struct tcp_skb_cb) > sizeof(skb->cb));

+#ifndef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
  percpu_counter_init(&tcp_sockets_allocated, 0);
+#endif
  percpu_counter_init(&tcp_orphan_count, 0);
  tcp_hashinfo.bind_bucket_cachep =
   kmem_cache_create("tcp_bind_bucket",
diff --git a/net/ipv4/tcp_ipv4.c b/net/ipv4/tcp_ipv4.c
index 3227f6a..0377af3 100644
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -2604,12 +2604,22 @@ struct proto tcp_prot = {
  .hash   = inet_hash,
  .unhash   = inet_unhash,
  .get_port  = inet_csk_get_port,
+ .orphan_count  = &tcp_orphan_count,
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ .init_cgroup  = tcp_init_cgroup,
+ .destroy_cgroup  = tcp_destroy_cgroup,
  .enter_memory_pressure = tcp_enter_memory_pressure,
  .memory_pressure = memory_pressure_tcp,
  .sockets_allocated = sockets_allocated_tcp,
- .orphan_count  = &tcp_orphan_count,
  .memory_allocated = memory_allocated_tcp,
  .prot_mem  = tcp_sysctl_mem,
+#else
+ .enter_memory_pressure = tcp_enter_memory_pressure_nocg,
+ .memory_pressure = memory_pressure_tcp_nocg,
+ .sockets_allocated = sockets_allocated_tcp_nocg,
+ .memory_allocated = memory_allocated_tcp_nocg,
+ .prot_mem  = tcp_sysctl_mem_nocg,
+#endif
  .sysctl_wmem  = sysctl_tcp_wmem,
  .sysctl_rmem  = sysctl_tcp_rmem,
  .max_header  = MAX_TCP_HEADER,
diff --git a/net/ipv6/tcp_ipv6.c b/net/ipv6/tcp_ipv6.c
```

```
index 962fb56..5597807 100644
--- a/net/ipv6/tcp_ipv6.c
+++ b/net/ipv6/tcp_ipv6.c
@@ -2195,12 +2195,20 @@ struct proto tcpv6_prot = {
  .hash    = tcp_v6_hash,
  .unhash   = inet_unhash,
  .get_port  = inet_csk_get_port,
+ .orphan_count  = &tcp_orphan_count,
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
  .enter_memory_pressure = tcp_enter_memory_pressure,
  .sockets_allocated = sockets_allocated_tcp,
  .memory_allocated = memory_allocated_tcp,
  .memory_pressure = memory_pressure_tcp,
- .orphan_count  = &tcp_orphan_count,
  .prot_mem  = tcp_sysctl_mem,
+#else
+ .enter_memory_pressure = tcp_enter_memory_pressure_nocg,
+ .sockets_allocated = sockets_allocated_tcp_nocg,
+ .memory_allocated = memory_allocated_tcp_nocg,
+ .memory_pressure = memory_pressure_tcp_nocg,
+ .prot_mem  = tcp_sysctl_mem_nocg,
+#endif
  .sysctl_wmem  = sysctl_tcp_wmem,
  .sysctl_rmem  = sysctl_tcp_rmem,
  .max_header  = MAX_TCP_HEADER,
--
1.7.6
```

Subject: [PATCH v5 5/8] per-netns ipv4 sysctl_tcp_mem
Posted by Glauber Costa on Tue, 04 Oct 2011 12:17:57 GMT
View Forum Message <> Reply to Message

This patch allows each namespace to independently set up
its levels for tcp memory pressure thresholds. This patch
alone does not buy much: we need to make this values
per group of process somehow. This is achieved in the
patches that follows in this patchset.

Signed-off-by: Glauber Costa <glommer@parallels.com>
Reviewed-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: David S. Miller <davem@davemloft.net>
CC: Eric W. Biederman <ebiederm@xmission.com>

```
 include/net/netns/ipv4.h   |    1 +
 include/net/tcp.h          |    1 -
 mm/memcontrol.c            |    8 ++++--
 net/ipv4/sysctl_net_ipv4.c |   51 ++++++++++++++++++++++++++++++++++++++++++++
```

```
 net/ipv4/tcp.c            |   13 ++--------
 5 files changed, 53 insertions(+), 21 deletions(-)

diff --git a/include/net/netns/ipv4.h b/include/net/netns/ipv4.h
index d786b4f..bbd023a 100644
--- a/include/net/netns/ipv4.h
+++ b/include/net/netns/ipv4.h
@@ -55,6 +55,7 @@ struct netns_ipv4 {
  int current_rt_cache_rebuild_count;

  unsigned int sysctl_ping_group_range[2];
+ long sysctl_tcp_mem[3];

  atomic_t rt_genid;
  atomic_t dev_addr_genid;
diff --git a/include/net/tcp.h b/include/net/tcp.h
index 7ae7b4b..04fedf7 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -231,7 +231,6 @@ extern int sysctl_tcp_fack;
 extern int sysctl_tcp_reordering;
 extern int sysctl_tcp_ecn;
 extern int sysctl_tcp_dsack;
-extern long sysctl_tcp_mem[3];
 extern int sysctl_tcp_wmem[3];
 extern int sysctl_tcp_rmem[3];
 extern int sysctl_tcp_app_win;
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 9598e3e..39e575e 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -309,6 +309,7 @@ static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup
*memcg);
 #ifdef CONFIG_INET
 #include <net/sock.h>
 #include <net/ip.h>
+#include <linux/nsproxy.h>

 void sock_update_memcg(struct sock *sk)
 {
@@ -434,14 +435,15 @@ int tcp_init_cgroup(struct proto *prot, struct cgroup *cgrp,
       struct cgroup_subsys *ss)
 {
  struct mem_cgroup *cg = mem_cgroup_from_cont(cgrp);
+ struct net *net = current->nsproxy->net_ns;
  /*
   * We need to initialize it at populate, not create time.
   * This is because net sysctl tables are not up until much
```

```
   * later
   */
- cg->tcp.tcp_prot_mem[0] = sysctl_tcp_mem[0];
- cg->tcp.tcp_prot_mem[1] = sysctl_tcp_mem[1];
- cg->tcp.tcp_prot_mem[2] = sysctl_tcp_mem[2];
+ cg->tcp.tcp_prot_mem[0] = net->ipv4.sysctl_tcp_mem[0];
+ cg->tcp.tcp_prot_mem[1] = net->ipv4.sysctl_tcp_mem[1];
+ cg->tcp.tcp_prot_mem[2] = net->ipv4.sysctl_tcp_mem[2];

  return 0;
 }
diff --git a/net/ipv4/sysctl_net_ipv4.c b/net/ipv4/sysctl_net_ipv4.c
index 69fd720..bbd67ab 100644
--- a/net/ipv4/sysctl_net_ipv4.c
+++ b/net/ipv4/sysctl_net_ipv4.c
@@ -14,6 +14,7 @@
 #include <linux/init.h>
 #include <linux/slab.h>
 #include <linux/nsproxy.h>
+#include <linux/swap.h>
 #include <net/snmp.h>
 #include <net/icmp.h>
 #include <net/ip.h>
@@ -174,6 +175,36 @@ static int proc_allowed_congestion_control(ctl_table *ctl,
  return ret;
 }

+static int ipv4_tcp_mem(ctl_table *ctl, int write,
+      void __user *buffer, size_t *lenp,
+      loff_t *ppos)
+{
+ int ret;
+ unsigned long vec[3];
+ struct net *net = current->nsproxy->net_ns;
+
+ ctl_table tmp = {
+  .data = &vec,
+  .maxlen = sizeof(vec),
+  .mode = ctl->mode,
+ };
+
+ if (!write) {
+  ctl->data = &net->ipv4.sysctl_tcp_mem;
+  return proc_doulongvec_minmax(ctl, write, buffer, lenp, ppos);
+ }
+
+ ret = proc_doulongvec_minmax(&tmp, write, buffer, lenp, ppos);
+ if (ret)
```

```
+  return ret;
+
+ net->ipv4.sysctl_tcp_mem[0] = vec[0];
+ net->ipv4.sysctl_tcp_mem[1] = vec[1];
+ net->ipv4.sysctl_tcp_mem[2] = vec[2];
+
+ return 0;
+}
+
 static struct ctl_table ipv4_table[] = {
  {
   .procname = "tcp_timestamps",
@@ -433,13 +464,6 @@ static struct ctl_table ipv4_table[] = {
   .proc_handler = proc_dointvec
  },
  {
-  .procname = "tcp_mem",
-  .data  = &sysctl_tcp_mem,
-  .maxlen  = sizeof(sysctl_tcp_mem),
-  .mode  = 0644,
-  .proc_handler = proc_doulongvec_minmax
- },
- {
   .procname = "tcp_wmem",
   .data  = &sysctl_tcp_wmem,
   .maxlen  = sizeof(sysctl_tcp_wmem),
@@ -721,6 +745,12 @@ static struct ctl_table ipv4_net_table[] = {
   .mode  = 0644,
   .proc_handler = ipv4_ping_group_range,
  },
+ {
+  .procname = "tcp_mem",
+  .maxlen  = sizeof(init_net.ipv4.sysctl_tcp_mem),
+  .mode  = 0644,
+  .proc_handler = ipv4_tcp_mem,
+ },
  { }
 };

@@ -734,6 +764,7 @@ EXPORT_SYMBOL_GPL(net_ipv4_ctl_path);
 static __net_init int ipv4_sysctl_init_net(struct net *net)
 {
  struct ctl_table *table;
+ unsigned long limit;

  table = ipv4_net_table;
  if (!net_eq(net, &init_net)) {
@@ -769,6 +800,12 @@ static __net_init int ipv4_sysctl_init_net(struct net *net)
```

```
  net->ipv4.sysctl_rt_cache_rebuild_count = 4;

+ limit = nr_free_buffer_pages() / 8;
+ limit = max(limit, 128UL);
+ net->ipv4.sysctl_tcp_mem[0] = limit / 4 * 3;
+ net->ipv4.sysctl_tcp_mem[1] = limit;
+ net->ipv4.sysctl_tcp_mem[2] = net->ipv4.sysctl_tcp_mem[0] * 2;
+
  net->ipv4.ipv4_hdr = register_net_sysctl_table(net,
    net_ipv4_ctl_path, table);
  if (net->ipv4.ipv4_hdr == NULL)
diff --git a/net/ipv4/tcp.c b/net/ipv4/tcp.c
index bbd3989..cd386df 100644
--- a/net/ipv4/tcp.c
+++ b/net/ipv4/tcp.c
@@ -282,11 +282,9 @@ int sysctl_tcp_fin_timeout __read_mostly = TCP_FIN_TIMEOUT;
 struct percpu_counter tcp_orphan_count;
 EXPORT_SYMBOL_GPL(tcp_orphan_count);

-long sysctl_tcp_mem[3] __read_mostly;
 int sysctl_tcp_wmem[3] __read_mostly;
 int sysctl_tcp_rmem[3] __read_mostly;

-EXPORT_SYMBOL(sysctl_tcp_mem);
 EXPORT_SYMBOL(sysctl_tcp_rmem);
 EXPORT_SYMBOL(sysctl_tcp_wmem);

@@ -333,7 +331,7 @@ EXPORT_SYMBOL(tcp_enter_memory_pressure_nocg);

 long *tcp_sysctl_mem_nocg(struct mem_cgroup *memcg)
 {
- return sysctl_tcp_mem;
+ return init_net.ipv4.sysctl_tcp_mem;
 }
 EXPORT_SYMBOL(tcp_sysctl_mem_nocg);

@@ -3296,14 +3294,9 @@ void __init tcp_init(void)
  sysctl_tcp_max_orphans = cnt / 2;
  sysctl_max_syn_backlog = max(128, cnt / 256);

- limit = nr_free_buffer_pages() / 8;
- limit = max(limit, 128UL);
- sysctl_tcp_mem[0] = limit / 4 * 3;
- sysctl_tcp_mem[1] = limit;
- sysctl_tcp_mem[2] = sysctl_tcp_mem[0] * 2;
-
  /* Set per-socket limits to no more than 1/128 the pressure threshold */
```

```
- limit = ((unsigned long)sysctl_tcp_mem[1]) << (PAGE_SHIFT - 7);
+ limit = ((unsigned long)init_net.ipv4.sysctl_tcp_mem[1])
+  << (PAGE_SHIFT - 7);
  max_share = min(4UL*1024*1024, limit);

  sysctl_tcp_wmem[0] = SK_MEM_QUANTUM;
--
1.7.6
```

Subject: [PATCH v5 6/8] tcp buffer limitation: per-cgroup limit
Posted by Glauber Costa on Tue, 04 Oct 2011 12:17:58 GMT
View Forum Message <> Reply to Message

This patch uses the "tcp_max_mem" field of the kmem_cgroup to
effectively control the amount of kernel memory pinned by a cgroup.

We have to make sure that none of the memory pressure thresholds
specified in the namespace are bigger than the current cgroup.

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>

```
 Documentation/cgroups/memory.txt |    1 +
 include/linux/memcontrol.h       |   10 +++++
 include/net/tcp.h                |    1 +
 mm/memcontrol.c                  |   80 ++++++++++++++++++++++++++++++++++++---
 net/ipv4/sysctl_net_ipv4.c       |   20 +++++++++
 5 files changed, 106 insertions(+), 6 deletions(-)

diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index bf00cd2..c1db134 100644
--- a/Documentation/cgroups/memory.txt
+++ b/Documentation/cgroups/memory.txt
@@ -78,6 +78,7 @@ Brief summary of control files.

  memory.independent_kmem_limit  # select whether or not kernel memory limits are
       independent of user limits
+ memory.kmem.tcp.limit_in_bytes  # set/show hard limit for tcp buf memory

 1. History

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 50af61d..6191ba1 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
```

```
@@ -395,6 +395,9 @@ int tcp_init_cgroup(struct proto *prot, struct cgroup *cgrp,
       struct cgroup_subsys *ss);
 void tcp_destroy_cgroup(struct proto *prot, struct cgroup *cgrp,
    struct cgroup_subsys *ss);
+
+unsigned long tcp_max_memory(struct mem_cgroup *cg);
+void tcp_prot_mem(struct mem_cgroup *cg, long val, int idx);
 #else
 /* memcontrol includes sockets.h, that includes memcontrol.h ... */
 static inline void memcg_sock_mem_alloc(struct mem_cgroup *memcg,
@@ -420,6 +423,13 @@ static inline void sock_update_memcg(struct sock *sk)
 static inline void sock_release_memcg(struct sock *sk)
 {
 }
+static inline unsigned long tcp_max_memory(struct mem_cgroup *cg)
+{
+ return 0;
+}
+static inline void tcp_prot_mem(struct mem_cgroup *cg, long val, int idx)
+{
+}
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
 #endif /* CONFIG_INET */
 #endif /* _LINUX_MEMCONTROL_H */
diff --git a/include/net/tcp.h b/include/net/tcp.h
index 04fedf7..716a42e 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -256,6 +256,7 @@ extern int sysctl_tcp_thin_dupack;
 struct mem_cgroup;
 struct tcp_memcontrol {
  /* per-cgroup tcp memory pressure knobs */
+ int tcp_max_memory;
  atomic_long_t tcp_memory_allocated;
  struct percpu_counter tcp_sockets_allocated;
  /* those two are read-mostly, leave them at the end */
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 39e575e..6fb14bb 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -304,6 +304,13 @@ struct mem_cgroup {
 };

 static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg);
+
+static inline bool mem_cgroup_is_root(struct mem_cgroup *memcg)
+{
+ return (memcg == root_mem_cgroup);
```

```
+}
+
+static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *mem);
 /* Writing them here to avoid exposing memcg's inner layout */
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
 #ifdef CONFIG_INET
@@ -424,6 +431,48 @@ struct percpu_counter *sockets_allocated_tcp(struct mem_cgroup
*memcg)
 }
 EXPORT_SYMBOL(sockets_allocated_tcp);

+static int tcp_write_limit(struct cgroup *cgrp, struct cftype *cft, u64 val)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
+ struct mem_cgroup *parent = parent_mem_cgroup(memcg);
+ struct net *net = current->nsproxy->net_ns;
+ int i;
+
+ if (parent && parent->use_hierarchy)
+  return -EINVAL;
+
+ /*
+  * We can't allow more memory than our parents. Since this
+  * will be tested for all calls, by induction, there is no need
+  * to test any parent other than our own
+  * */
+ val >>= PAGE_SHIFT;
+ if (parent && (val > parent->tcp.tcp_max_memory))
+  val = parent->tcp.tcp_max_memory;
+
+ memcg->tcp.tcp_max_memory = val;
+
+ for (i = 0; i < 3; i++)
+  memcg->tcp.tcp_prot_mem[i]  = min_t(long, val,
+         net->ipv4.sysctl_tcp_mem[i]);
+
+ return 0;
+}
+
+static u64 tcp_read_limit(struct cgroup *cgrp, struct cftype *cft)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
+ return memcg->tcp.tcp_max_memory << PAGE_SHIFT;
+}
+
+static struct cftype tcp_files[] = {
+ {
+  .name = "kmem.tcp.limit_in_bytes",
```

```
+  .write_u64 = tcp_write_limit,
+  .read_u64 = tcp_read_limit,
+ },
+};
+
 static void tcp_create_cgroup(struct mem_cgroup *cg, struct cgroup_subsys *ss)
 {
  cg->tcp.tcp_memory_pressure = 0;
@@ -435,6 +484,7 @@ int tcp_init_cgroup(struct proto *prot, struct cgroup *cgrp,
       struct cgroup_subsys *ss)
 {
  struct mem_cgroup *cg = mem_cgroup_from_cont(cgrp);
+ struct mem_cgroup *parent = parent_mem_cgroup(cg);
  struct net *net = current->nsproxy->net_ns;
  /*
   * We need to initialize it at populate, not create time.
@@ -445,6 +495,20 @@ int tcp_init_cgroup(struct proto *prot, struct cgroup *cgrp,
  cg->tcp.tcp_prot_mem[1] = net->ipv4.sysctl_tcp_mem[1];
  cg->tcp.tcp_prot_mem[2] = net->ipv4.sysctl_tcp_mem[2];

+
+ if (parent && parent->use_hierarchy)
+  cg->tcp.tcp_max_memory = parent->tcp.tcp_max_memory;
+ else {
+  unsigned long limit;
+  limit = nr_free_buffer_pages() / 8;
+  limit = max(limit, 128UL);
+  cg->tcp.tcp_max_memory = limit * 2;
+ }
+
+
+ if (!mem_cgroup_is_root(cg))
+  return cgroup_add_files(cgrp, ss, tcp_files,
+      ARRAY_SIZE(tcp_files));
  return 0;
 }
 EXPORT_SYMBOL(tcp_init_cgroup);
@@ -457,6 +521,16 @@ void tcp_destroy_cgroup(struct proto *prot, struct cgroup *cgrp,
  percpu_counter_destroy(&cg->tcp.tcp_sockets_allocated);
 }
 EXPORT_SYMBOL(tcp_destroy_cgroup);
+
+unsigned long tcp_max_memory(struct mem_cgroup *memcg)
+{
+ return memcg->tcp.tcp_max_memory;
+}
+
+void tcp_prot_mem(struct mem_cgroup *memcg, long val, int idx)
```

```
+{
+ memcg->tcp.tcp_prot_mem[idx] = val;
+}
 #endif /* CONFIG_INET */
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

@@ -1035,12 +1109,6 @@ static struct mem_cgroup *mem_cgroup_get_next(struct
mem_cgroup *iter,
 #define for_each_mem_cgroup_all(iter) \
  for_each_mem_cgroup_tree_cond(iter, NULL, true)

-
-static inline bool mem_cgroup_is_root(struct mem_cgroup *mem)
-{
- return (mem == root_mem_cgroup);
-}
-
 void mem_cgroup_count_vm_event(struct mm_struct *mm, enum vm_event_item idx)
 {
  struct mem_cgroup *mem;
diff --git a/net/ipv4/sysctl_net_ipv4.c b/net/ipv4/sysctl_net_ipv4.c
index bbd67ab..cdc35f6 100644
--- a/net/ipv4/sysctl_net_ipv4.c
+++ b/net/ipv4/sysctl_net_ipv4.c
@@ -14,6 +14,7 @@
 #include <linux/init.h>
 #include <linux/slab.h>
 #include <linux/nsproxy.h>
+#include <linux/memcontrol.h>
 #include <linux/swap.h>
 #include <net/snmp.h>
 #include <net/icmp.h>
@@ -182,6 +183,10 @@ static int ipv4_tcp_mem(ctl_table *ctl, int write,
  int ret;
  unsigned long vec[3];
  struct net *net = current->nsproxy->net_ns;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ int i;
+ struct mem_cgroup *cg;
+#endif

  ctl_table tmp = {
   .data = &vec,
@@ -198,6 +203,21 @@ static int ipv4_tcp_mem(ctl_table *ctl, int write,
  if (ret)
   return ret;

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
```

```
+ rcu_read_lock();
+ cg = mem_cgroup_from_task(current);
+ for (i = 0; i < 3; i++)
+  if (vec[i] > tcp_max_memory(cg)) {
+   rcu_read_unlock();
+   return -EINVAL;
+  }
+
+ tcp_prot_mem(cg, vec[0], 0);
+ tcp_prot_mem(cg, vec[1], 1);
+ tcp_prot_mem(cg, vec[2], 2);
+ rcu_read_unlock();
+#endif
+
  net->ipv4.sysctl_tcp_mem[0] = vec[0];
  net->ipv4.sysctl_tcp_mem[1] = vec[1];
  net->ipv4.sysctl_tcp_mem[2] = vec[2];
--
1.7.6
```

Subject: [PATCH v5 7/8] Display current tcp memory allocation in kmem cgroup
Posted by Glauber Costa on Tue, 04 Oct 2011 12:17:59 GMT
View Forum Message <> Reply to Message

This patch introduces kmem.tcp_current_memory file, living in the
kmem_cgroup filesystem. It is a simple read-only file that displays the
amount of kernel memory currently consumed by the cgroup.

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>

 Documentation/cgroups/memory.txt | 1 +
 mm/memcontrol.c | 11 ++++++++++
 2 files changed, 12 insertions(+), 0 deletions(-)

```
diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index c1db134..00f1a88 100644
--- a/Documentation/cgroups/memory.txt
+++ b/Documentation/cgroups/memory.txt
@@ -79,6 +79,7 @@ Brief summary of control files.
  memory.independent_kmem_limit  # select whether or not kernel memory limits are
       independent of user limits
  memory.kmem.tcp.limit_in_bytes  # set/show hard limit for tcp buf memory
+ memory.kmem.tcp.usage_in_bytes  # show current tcp buf memory allocation
```

1. History

```
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 6fb14bb..f178a64 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -465,12 +465,23 @@ static u64 tcp_read_limit(struct cgroup *cgrp, struct cftype *cft)
  return memcg->tcp.tcp_max_memory << PAGE_SHIFT;
 }

+static u64 tcp_usage_in_bytes(struct cgroup *cgrp, struct cftype *cft)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
+
+ return atomic_long_read(&memcg->tcp.tcp_memory_allocated) << PAGE_SHIFT;
+}
+
 static struct cftype tcp_files[] = {
  {
   .name = "kmem.tcp.limit_in_bytes",
   .write_u64 = tcp_write_limit,
   .read_u64 = tcp_read_limit,
  },
+ {
+  .name = "kmem.tcp.usage_in_bytes",
+  .read_u64 = tcp_usage_in_bytes,
+ },
 };

 static void tcp_create_cgroup(struct mem_cgroup *cg, struct cgroup_subsys *ss)
--
1.7.6
```

Subject: [PATCH v5 8/8] Disable task moving when using kernel memory accounting

Posted by Glauber Costa on Tue, 04 Oct 2011 12:18:00 GMT

View Forum Message <> Reply to Message

Since this code is still experimental, we are leaving the exact
details of how to move tasks between cgroups when kernel memory
accounting is used as future work.

For now, we simply disallow movement if there are any pending
accounted memory.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>

```
---
 include/net/tcp.h |    1 +
 mm/memcontrol.c   |   37 ++++++++++++++++++++++++++----------
 2 files changed, 23 insertions(+), 15 deletions(-)

diff --git a/include/net/tcp.h b/include/net/tcp.h
index 716a42e..62ad282 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -257,6 +257,7 @@ struct mem_cgroup;
 struct tcp_memcontrol {
  /* per-cgroup tcp memory pressure knobs */
  int tcp_max_memory;
+ atomic_t refcnt;
  atomic_long_t tcp_memory_allocated;
  struct percpu_counter tcp_sockets_allocated;
  /* those two are read-mostly, leave them at the end */
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index f178a64..db1f511 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -328,28 +328,17 @@ void sock_update_memcg(struct sock *sk)

  rcu_read_lock();
  sk->sk_cgrp = mem_cgroup_from_task(current);
-
- /*
-  * We don't need to protect against anything task-related, because
-  * we are basically stuck with the sock pointer that won't change,
-  * even if the task that originated the socket changes cgroups.
-  *
-  * What we do have to guarantee, is that the chain leading us to
-  * the top level won't change under our noses. Incrementing the
-  * reference count via cgroup_exclude_rmdir guarantees that.
-  */
- cgroup_exclude_rmdir(mem_cgroup_css(sk->sk_cgrp));
  rcu_read_unlock();
 }

 void sock_release_memcg(struct sock *sk)
 {
- cgroup_release_and_wakeup_rmdir(mem_cgroup_css(sk->sk_cgrp));
 }

 void memcg_sock_mem_alloc(struct mem_cgroup *memcg, struct proto *prot,
      int amt, int *parent_failure)
 {
+ atomic_inc(&memcg->tcp.refcnt);
```

```
  memcg = parent_mem_cgroup(memcg);
  for (; memcg != NULL; memcg = parent_mem_cgroup(memcg)) {
   long alloc;
@@ -368,9 +357,12 @@ EXPORT_SYMBOL(memcg_sock_mem_alloc);

 void memcg_sock_mem_free(struct mem_cgroup *memcg, struct proto *prot, int amt)
 {
- memcg = parent_mem_cgroup(memcg);
- for (; memcg != NULL; memcg = parent_mem_cgroup(memcg))
-  atomic_long_sub(amt, prot->memory_allocated(memcg));
+ struct mem_cgroup *parent;
+ parent = parent_mem_cgroup(memcg);
+ for (; parent != NULL; parent = parent_mem_cgroup(parent))
+  atomic_long_sub(amt, prot->memory_allocated(parent));
+
+ atomic_dec(&memcg->tcp.refcnt);
 }
 EXPORT_SYMBOL(memcg_sock_mem_free);

@@ -488,6 +480,7 @@ static void tcp_create_cgroup(struct mem_cgroup *cg, struct
cgroup_subsys *ss)
 {
  cg->tcp.tcp_memory_pressure = 0;
  atomic_long_set(&cg->tcp.tcp_memory_allocated, 0);
+ atomic_set(&cg->tcp.refcnt, 0);
  percpu_counter_init(&cg->tcp.tcp_sockets_allocated, 0);
 }

@@ -5633,6 +5626,12 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
  int ret = 0;
  struct mem_cgroup *mem = mem_cgroup_from_cont(cgroup);

+ if (atomic_read(&mem->tcp.refcnt)) {
+  printk(KERN_WARNING "Can't move tasks between cgroups: "
+   "Kernel memory held. task: %s\n", p->comm);
+  return 1;
+ }
+
  if (mem->move_charge_at_immigrate) {
   struct mm_struct *mm;
   struct mem_cgroup *from = mem_cgroup_from_task(p);
@@ -5803,6 +5802,14 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
     struct cgroup *cgroup,
     struct task_struct *p)
 {
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cgroup);
+
+ if (atomic_read(&mem->tcp.refcnt)) {
```

```
+  printk(KERN_WARNING "Can't move tasks between cgroups: "
+   "Kernel memory held. task: %s\n", p->comm);
+  return 1;
+ }
+
  return 0;
 }
 static void mem_cgroup_cancel_attach(struct cgroup_subsys *ss,
--
1.7.6
```

Subject: Re: [PATCH v5 6/8] tcp buffer limitation: per-cgroup limit
Posted by Eric Dumazet on Tue, 04 Oct 2011 12:48:55 GMT
View Forum Message <> Reply to Message

Le mardi 04 octobre 2011 à 16:17 +0400, Glauber Costa a écrit :
> This patch uses the "tcp_max_mem" field of the kmem_cgroup to
> effectively control the amount of kernel memory pinned by a cgroup.
>
> We have to make sure that none of the memory pressure thresholds
> specified in the namespace are bigger than the current cgroup.
>
> 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>
> ---

> --- a/include/net/tcp.h
> +++ b/include/net/tcp.h
> @@ -256,6 +256,7 @@ extern int sysctl_tcp_thin_dupack;
> struct mem_cgroup;
> struct tcp_memcontrol {
>   /* per-cgroup tcp memory pressure knobs */
> + int tcp_max_memory;
>   atomic_long_t tcp_memory_allocated;
>   struct percpu_counter tcp_sockets_allocated;
>   /* those two are read-mostly, leave them at the end */
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c

So tcp_max_memory is an "int".

> +static u64 tcp_read_limit(struct cgroup *cgrp, struct cftype *cft)
> +{
> + struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);

> + return memcg->tcp.tcp_max_memory << PAGE_SHIFT;
> +}

1) Typical integer overflow here.

You need :

return ((u64)memcg->tcp.tcp_max_memory) << PAGE_SHIFT;

2) Could you add const qualifiers when possible to your pointers ?

Subject: Re: [PATCH v5 0/8] per-cgroup tcp buffer pressure settings
Posted by KAMEZAWA Hiroyuki on Wed, 05 Oct 2011 00:29:54 GMT
View Forum Message <> Reply to Message

On Tue, 4 Oct 2011 16:17:52 +0400
Glauber Costa <glommer@parallels.com> wrote:

> [[v3: merge Kirill's suggestions, + a destroy-related bugfix]]
> [[v4: Fix a bug with non-mounted cgroups + disallow task movement]]
> [[v5: Compile bug with modular ipv6 + tcp files in bytes]]
>
> Kame, Kirill,
>
> I am submitting this again merging most of your comments. I've decided to
> leave some of them out:
> * I am not using res_counters for allocated_memory. Besides being more
> expensive than what we need, to make it work in a nice way, we'd have
> to change the !cgroup code, including other protocols than tcp. Also,
>
> * I am not using failcnt and max_usage_in_bytes for it. I believe the value
> of those lies more in the allocation than in the pressure control. Besides,
> fail conditions lie mostly outside of the memory cgroup's control. (Actually,
> a soft_limit makes a lot of sense, and I do plan to introduce it in a follow
> up series)
>
> If you agree with the above, and there are any other pressing issues, let me
> know and I will address them ASAP. Otherwise, let's discuss it. I'm always open.
>

I'm not familar with reuqirements of users. So, I appreciate your choices.
What I adivse you here is taking a deep breath. Making new version every day
is not good for reviewing process ;)
(It's now -rc8 and merge will not be so quick, anyway.)

At this stage, my concern is view of interfaces and documenation, and future plans.

Let me give a try explanation by myself. (Correct me ;)
I added some questions but I'm sorry you've already answered.

New interfaces are 5 files. All files exists only for non-root memory cgroup.

1. memory.independent_kmem_limit
2. memory.kmem.usage_in_bytes
3. memory.kmem.limit_in_bytes
4. memory.kmem.tcp.limit_in_bytes
5. memory.kmem.tcp.usage_in_bytes

* memory.independent_kmem_limit
 If 1, kmem_limit_in_bytes/kmem_usage_in_bytes works.
 If 0, kmem_limit_in_bytes/kmem_usage_in_bytes doesn't work and all kmem
    usages are controlled under memory.limit_in_bytes.

Question:
 - What happens when parent/chidlren cgroup has different indepedent_kmem_limit ?
 - What happens at creating a new cgroup with use_hierarchy==1.

* memory.kmem_limit_in_bytes/memory.kmem.tcp.limit_in_bytes

 Both files works independently for _Now_. And memory.kmem_usage_in_bytes and
 memory.kmem_tcp.usage_in_bytes has no relationships.

 In future plan, kmem.usage_in_bytes should includes tcp.kmem_usage_in_bytes.
 And kmem.limit_in_bytes should be the limiation of sum of all kmem.xxxx.limit_in_bytes.

Question:
 - Why this integration is difficult ?
   Can't tcp-limit-code borrows some amount of charges in batch from kmem_limit
   and use it ?

 - Don't you need a stat file to indicate "tcp memory pressure works!" ?
   It can be obtained already ?

Thanks,
-Kame

Subject: Re: [PATCH v5 0/8] per-cgroup tcp buffer pressure settings
Posted by Glauber Costa on Wed, 05 Oct 2011 07:25:50 GMT
View Forum Message <> Reply to Message

On 10/05/2011 04:29 AM, KAMEZAWA Hiroyuki wrote:
> On Tue, 4 Oct 2011 16:17:52 +0400
> Glauber Costa<glommer@parallels.com> wrote:

>
>> [[v3: merge Kirill's suggestions, + a destroy-related bugfix]]
>> [[v4: Fix a bug with non-mounted cgroups + disallow task movement]]
>> [[v5: Compile bug with modular ipv6 + tcp files in bytes]]
>>
>> Kame, Kirill,
>>
>> I am submitting this again merging most of your comments. I've decided to
>> leave some of them out:
>> * I am not using res_counters for allocated_memory. Besides being more
>>   expensive than what we need, to make it work in a nice way, we'd have
>>   to change the !cgroup code, including other protocols than tcp. Also,
>>
>> * I am not using failcnt and max_usage_in_bytes for it. I believe the value
>>   of those lies more in the allocation than in the pressure control. Besides,
>>   fail conditions lie mostly outside of the memory cgroup's control. (Actually,
>>   a soft_limit makes a lot of sense, and I do plan to introduce it in a follow
>>   up series)
>>
>> If you agree with the above, and there are any other pressing issues, let me
>> know and I will address them ASAP. Otherwise, let's discuss it. I'm always open.
>>
>
> I'm not familar with reuqirements of users. So, I appreciate your choices.
> What I adivse you here is taking a deep breath. Making new version every day
> is not good for reviewing process ;)
> (It's now -rc8 and merge will not be so quick, anyway.)

Kame,

Absolutely. I only did it this time because the difference between them
were really, really small.

> At this stage, my concern is view of interfaces and documenation, and future plans.

Okay. I will try to address them as well as I can.

> Let me give a try explanation by myself. (Correct me ;)
> I added some questions but I'm sorry you've already answered.
>
> New interfaces are 5 files. All files exists only for non-root memory cgroup.
>
> 1. memory.independent_kmem_limit
> 2. memory.kmem.usage_in_bytes
> 3. memory.kmem.limit_in_bytes
> 4. memory.kmem.tcp.limit_in_bytes
> 5. memory.kmem.tcp.usage_in_bytes

Correct so far. Note that the tcp memory pressure parameters right now
consists of 3 numbers, one of them being a soft limit. I plan to add the
soft limit file in a follow up patch, to avoid adding more and more
stuff for us to review here. (Unless of course you want to see it now)

> * memory.independent_kmem_limit
>   If 1, kmem_limit_in_bytes/kmem_usage_in_bytes works.
>   If 0, kmem_limit_in_bytes/kmem_usage_in_bytes doesn't work and all kmem
>   usages are controlled under memory.limit_in_bytes.

Correct. For the questions below, I won't even look at the code not to
get misguided. Let's settle on the desired behavior, and everything that
deviates from it, is a bug.

> Question:
>   - What happens when parent/chidlren cgroup has different indepedent_kmem_limit ?
I think it should be forbidden. It was raised by Kirill before, and
IIRC, he specifically requested it to be. (Okay: Saying it now, makes me
realizes that the child can have set it to 1 while parent was 1. But
then parent sets it to 0... I don't think I am handling this case).

>   - What happens at creating a new cgroup with use_hierarchy==1.
>
> * memory.kmem_limit_in_bytes/memory.kmem.tcp.limit_in_bytes
>
>   Both files works independently for _Now_. And memory.kmem_usage_in_bytes and
>   memory.kmem_tcp.usage_in_bytes has no relationships.

Correct.

>   In future plan, kmem.usage_in_bytes should includes tcp.kmem_usage_in_bytes.
>   And kmem.limit_in_bytes should be the limiation of sum of all kmem.xxxx.limit_in_bytes.

I am not sure there will be others xxx.limit_in_bytes. (see below)

>
> Question:
>   - Why this integration is difficult ?
It is not that it is difficult.
What happens is that there are two things taking place here:
One of them is allocation.
The other, is tcp-specific pressure thresholds. Bear with me with the
following example code: (from sk_stream_alloc_skb, net/ipv4/tcp.c)

```
1:      skb = alloc_skb_fclone(size + sk->sk_prot->max_header, gfp);
        if (skb) {
3:              if (sk_wmem_schedule(sk, skb->truesize)) {
                        /*
```

```
                 * Make sure that we have exactly size bytes
                 * available to the caller, no more, no less.
                 */
                skb_reserve(skb, skb_tailroom(skb) - size);
                return skb;
        }
        __kfree_skb(skb);
} else {
        sk->sk_prot->enter_memory_pressure(sk);
        sk_stream_moderate_sndbuf(sk);
}
```

In line 1, an allocation takes place. This allocs memory from the skbuff slab cache.
But then, pressure thresholds are applied in 3. If it fails, we drop the memory buffer even if the allocation succeeded.

So this patchset, as I've stated already, cares about pressure conditions only. It is enough to guarantee that no more memory will be pinned that we specified, because we'll free the allocation in case pressure is reached.

There is work in progress from guys at google (and I have my very own PoCs as well), to include all slab allocations in kmem.usage_in_bytes.

So what I really mean here with "will integrate later", is that I think that we'd be better off tracking the allocations themselves at the slab level.

> Can't tcp-limit-code borrows some amount of charges in batch from kmem_limit
> and use it ?
Sorry, I don't know what exactly do you mean. Can you clarify?

> - Don't you need a stat file to indicate "tcp memory pressure works!" ?
> It can be obtained already ?

Not 100 % clear as well. We can query the amount of buffer used, and the amount of buffer allowed. What else do we need?

Subject: Re: [PATCH v5 6/8] tcp buffer limitation: per-cgroup limit
Posted by Glauber Costa on Wed, 05 Oct 2011 08:08:04 GMT
View Forum Message <> Reply to Message

On 10/04/2011 04:48 PM, Eric Dumazet wrote:
> Le mardi 04 octobre 2011 à 16:17 +0400, Glauber Costa a écrit :
>> This patch uses the "tcp_max_mem" field of the kmem_cgroup to
>> effectively control the amount of kernel memory pinned by a cgroup.

>>
>> We have to make sure that none of the memory pressure thresholds
>> specified in the namespace are bigger than the current cgroup.
>>
>> 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>
>> ---
>
>
>> --- a/include/net/tcp.h
>> +++ b/include/net/tcp.h
>> @@ -256,6 +256,7 @@ extern int sysctl_tcp_thin_dupack;
>> struct mem_cgroup;
>> struct tcp_memcontrol {
>> /* per-cgroup tcp memory pressure knobs */
>> + int tcp_max_memory;
>> atomic_long_t tcp_memory_allocated;
>> struct percpu_counter tcp_sockets_allocated;
>> /* those two are read-mostly, leave them at the end */
>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
>
> So tcp_max_memory is an "int".
>
>
>> +static u64 tcp_read_limit(struct cgroup *cgrp, struct cftype *cft)
>> +{
>> + struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
>> + return memcg->tcp.tcp_max_memory<< PAGE_SHIFT;
>> +}
>
> 1) Typical integer overflow here.
>
> You need :
>
> return ((u64)memcg->tcp.tcp_max_memory)<< PAGE_SHIFT;

Thanks for spotting this, Eric.

>
> 2) Could you add const qualifiers when possible to your pointers ?

Well, I'll go over the patches again and see where I can add them.
Any specific place site you're concerned about?

Subject: Re: [PATCH v5 6/8] tcp buffer limitation: per-cgroup limit
Posted by Eric Dumazet on Wed, 05 Oct 2011 08:58:10 GMT
View Forum Message <> Reply to Message

Le mercredi 05 octobre 2011 à 12:08 +0400, Glauber Costa a écrit :
> On 10/04/2011 04:48 PM, Eric Dumazet wrote:

> > 2) Could you add const qualifiers when possible to your pointers ?
>
> Well, I'll go over the patches again and see where I can add them.
> Any specific place site you're concerned about?

Everywhere its possible :

It helps reader to instantly knows if a function is about to change some
part of the object or only read it, without reading function body.

Subject: Re: [PATCH v5 6/8] tcp buffer limitation: per-cgroup limit
Posted by Glauber Costa on Thu, 06 Oct 2011 08:38:59 GMT
View Forum Message <> Reply to Message

On 10/05/2011 12:58 PM, Eric Dumazet wrote:
> Le mercredi 05 octobre 2011 à 12:08 +0400, Glauber Costa a écrit :
>> On 10/04/2011 04:48 PM, Eric Dumazet wrote:
>
>>> 2) Could you add const qualifiers when possible to your pointers ?
>>
>> Well, I'll go over the patches again and see where I can add them.
>> Any specific place site you're concerned about?
>
> Everywhere its possible :
>
> It helps reader to instantly knows if a function is about to change some
> part of the object or only read it, without reading function body.
Sure it does.

So, give me your opinion on this:

most of the acessors inside struct sock do not modify the pointers,
but return an address of an element inside it (that can later on be
modified by the caller.

I think it is fine for the purpose of clarity, but to avoid warnings we
end up having to do stuff like this:

+#define CONSTCG(m) ((struct mem_cgroup *)(m))
+long *tcp_sysctl_mem(const struct mem_cgroup *memcg)

```
+{
+       return CONSTCG(memcg)->tcp.tcp_prot_mem;
+}
```

Is it acceptable?

Subject: Re: [PATCH v5 0/8] per-cgroup tcp buffer pressure settings
Posted by KAMEZAWA Hiroyuki on Fri, 07 Oct 2011 08:05:22 GMT
View Forum Message <> Reply to Message

Sorry for lazy answer.

On Wed, 5 Oct 2011 11:25:50 +0400
Glauber Costa <glommer@parallels.com> wrote:

> On 10/05/2011 04:29 AM, KAMEZAWA Hiroyuki wrote:
> > On Tue, 4 Oct 2011 16:17:52 +0400
> > Glauber Costa<glommer@parallels.com> wrote:
> >

> > At this stage, my concern is view of interfaces and documenation, and future plans.
>
> Okay. I will try to address them as well as I can.
>
> > * memory.independent_kmem_limit
> >   If 1, kmem_limit_in_bytes/kmem_usage_in_bytes works.
> >   If 0, kmem_limit_in_bytes/kmem_usage_in_bytes doesn't work and all kmem
> >   usages are controlled under memory.limit_in_bytes.
>
> Correct. For the questions below, I won't even look at the code not to
> get misguided. Let's settle on the desired behavior, and everything that
> deviates from it, is a bug.
>
> > Question:
> >   - What happens when parent/chidlren cgroup has different indepedent_kmem_limit ?
> I think it should be forbidden. It was raised by Kirill before, and
> IIRC, he specifically requested it to be. (Okay: Saying it now, makes me
> realizes that the child can have set it to 1 while parent was 1. But
> then parent sets it to 0... I don't think I am handling this case).
>

ok, please put it into TODO list ;)

> >   In future plan, kmem.usage_in_bytes should includes tcp.kmem_usage_in_bytes.
> >   And kmem.limit_in_bytes should be the limiation of sum of all kmem.xxxx.limit_in_bytes.

>
> I am not sure there will be others xxx.limit_in_bytes. (see below)
>

ok.

> >
> > Question:
> > - Why this integration is difficult ?
> It is not that it is difficult.
> What happens is that there are two things taking place here:
> One of them is allocation.
> The other, is tcp-specific pressure thresholds. Bear with me with the
> following example code: (from sk_stream_alloc_skb, net/ipv4/tcp.c)
>
> 1:   skb = alloc_skb_fclone(size + sk->sk_prot->max_header, gfp);
>   if (skb) {
> 3:   if (sk_wmem_schedule(sk, skb->truesize)) {
>   /*
>   * Make sure that we have exactly size bytes
>   * available to the caller, no more, no less.
>   */
>   skb_reserve(skb, skb_tailroom(skb) - size);
>   return skb;
>   }
>   __kfree_skb(skb);
>   } else {
>   sk->sk_prot->enter_memory_pressure(sk);
>   sk_stream_moderate_sndbuf(sk);
>   }
>
> In line 1, an allocation takes place. This allocs memory from the skbuff
> slab cache.
> But then, pressure thresholds are applied in 3. If it fails, we drop the
> memory buffer even if the allocation succeeded.
>

Sure.

> So this patchset, as I've stated already, cares about pressure
> conditions only. It is enough to guarantee that no more memory will be
> pinned that we specified, because we'll free the allocation in case
> pressure is reached.
>
> There is work in progress from guys at google (and I have my very own
> PoCs as well), to include all slab allocations in kmem.usage_in_bytes.

>

ok.

> So what I really mean here with "will integrate later", is that I think
> that we'd be better off tracking the allocations themselves at the slab
> level.
>
> > Can't tcp-limit-code borrows some amount of charges in batch from kmem_limit
> > and use it ?
> Sorry, I don't know what exactly do you mean. Can you clarify?
>
Now, tcp-usage is independent from kmem-usage.

My idea is

1. when you account tcp usage, charge kmem, too.

Now, your work is
a) tcp use new xxxx bytes.
b) account it to tcp.uage and check tcp limit

To ingegrate kmem,
a) tcp use new xxxx bytes.
b) account it to tcp.usage and check tcp limit
c) account it to kmem.usage

? 2 counters may be slow ?

> > - Don't you need a stat file to indicate "tcp memory pressure works!" ?
> > It can be obtained already ?
>
> Not 100 % clear as well. We can query the amount of buffer used, and the
> amount of buffer allowed. What else do we need?
>

IIUC, we can see the fact tcp.usage is near to tcp.limit but never can see it
got memory pressure and how many numbers of failure happens.
I'm sorry if I don't read codes correctly.

Thanks,
-Kame

Subject: Re: [PATCH v5 0/8] per-cgroup tcp buffer pressure settings

On 10/07/2011 12:05 PM, KAMEZAWA Hiroyuki wrote:
>
>
> Sorry for lazy answer.
Hi Kame,

Now matter how hard you try, you'll never be as lazy as I am. So that's
okay.

>
> On Wed, 5 Oct 2011 11:25:50 +0400
> Glauber Costa<glommer@parallels.com> wrote:
>
>> On 10/05/2011 04:29 AM, KAMEZAWA Hiroyuki wrote:
>>> On Tue, 4 Oct 2011 16:17:52 +0400
>>> Glauber Costa<glommer@parallels.com>   wrote:
>>>
>
>>> At this stage, my concern is view of interfaces and documenation, and future plans.
>>
>> Okay. I will try to address them as well as I can.
>>
>>> * memory.independent_kmem_limit
>>>   If 1, kmem_limit_in_bytes/kmem_usage_in_bytes works.
>>>   If 0, kmem_limit_in_bytes/kmem_usage_in_bytes doesn't work and all kmem
>>>   usages are controlled under memory.limit_in_bytes.
>>
>> Correct. For the questions below, I won't even look at the code not to
>> get misguided. Let's settle on the desired behavior, and everything that
>> deviates from it, is a bug.
>>
>>> Question:
>>>   - What happens when parent/chidlren cgroup has different indepedent_kmem_limit ?
>> I think it should be forbidden. It was raised by Kirill before, and
>> IIRC, he specifically requested it to be. (Okay: Saying it now, makes me
>> realizes that the child can have set it to 1 while parent was 1. But
>> then parent sets it to 0... I don't think I am handling this case).
>>
>
> ok, please put it into TODO list ;)

Done.

>
>
>>>   In future plan, kmem.usage_in_bytes should includes tcp.kmem_usage_in_bytes.

>>> And kmem.limit_in_bytes should be the limiation of sum of all kmem.xxxx.limit_in_bytes.
>>
>> I am not sure there will be others xxx.limit_in_bytes. (see below)
>>
>
> ok.
>
>
>>>
>>> Question:
>>> - Why this integration is difficult ?
>> It is not that it is difficult.
>> What happens is that there are two things taking place here:
>> One of them is allocation.
>> The other, is tcp-specific pressure thresholds. Bear with me with the
>> following example code: (from sk_stream_alloc_skb, net/ipv4/tcp.c)
>>
>> 1:   skb = alloc_skb_fclone(size + sk->sk_prot->max_header, gfp);
>>   if (skb) {
>> 3:   if (sk_wmem_schedule(sk, skb->truesize)) {
>>   /*
>>   * Make sure that we have exactly size bytes
>>   * available to the caller, no more, no less.
>>   */
>>   skb_reserve(skb, skb_tailroom(skb) - size);
>>   return skb;
>>   }
>>   __kfree_skb(skb);
>>   } else {
>>   sk->sk_prot->enter_memory_pressure(sk);
>>   sk_stream_moderate_sndbuf(sk);
>>   }
>>
>> In line 1, an allocation takes place. This allocs memory from the skbuff
>> slab cache.
>> But then, pressure thresholds are applied in 3. If it fails, we drop the
>> memory buffer even if the allocation succeeded.
>>
>
> Sure.
>
>
>> So this patchset, as I've stated already, cares about pressure
>> conditions only. It is enough to guarantee that no more memory will be
>> pinned that we specified, because we'll free the allocation in case
>> pressure is reached.
>>
>> There is work in progress from guys at google (and I have my very own

>> PoCs as well), to include all slab allocations in kmem.usage_in_bytes.
>>
>
> ok.
>
>
>> So what I really mean here with "will integrate later", is that I think
>> that we'd be better off tracking the allocations themselves at the slab
>> level.
>>
>>>> Can't tcp-limit-code borrows some amount of charges in batch from kmem_limit
>>>> and use it ?
>> Sorry, I don't know what exactly do you mean. Can you clarify?
>>
> Now, tcp-usage is independent from kmem-usage.
>
> My idea is
>
> 1. when you account tcp usage, charge kmem, too.

Absolutely.
> Now, your work is
> a) tcp use new xxxx bytes.
> b) account it to tcp.uage and check tcp limit
>
> To ingegrate kmem,
> a) tcp use new xxxx bytes.
> b) account it to tcp.usage and check tcp limit
> c) account it to kmem.usage
>
> ? 2 counters may be slow ?

Well, the way I see it, 1 counter is slow already =)
I honestly think we need some optimizations here. But
that is a side issue.

To begin with: The new patchset that I intend to spin
today or Monday, depending on my progress, uses res_counters,
as you and Kirill requested.

So what makes res_counters slow IMHO, is two things:

1) interrupts are always disabled.
2) All is done under a lock.

Now, we are starting to have resources that are billed to multiple
counters. One simple way to work around it, is to have child counters
that has to be accounted for as well everytime a resource is counted.

Like this:

1) tcp has kmem as child. When we bill to tcp, we bill to kmem as well.
   For protocols that do memory pressure, we then don't bill kmem from
   the slab.
2) When kmem_independent_account is set to 0, kmem has mem as child.

>
>
>>> - Don't you need a stat file to indicate "tcp memory pressure works!" ?
>>> It can be obtained already ?
>>
>> Not 100 % clear as well. We can query the amount of buffer used, and the
>> amount of buffer allowed. What else do we need?
>>
>
> IIUC, we can see the fact tcp.usage is near to tcp.limit but never can see it
> got memory pressure and how many numbers of failure happens.
> I'm sorry if I don't read codes correctly.

IIUC, With res_counters being used, we get at least failcnt for free, right?

Subject: Re: [PATCH v5 0/8] per-cgroup tcp buffer pressure settings
Posted by KAMEZAWA Hiroyuki on Fri, 07 Oct 2011 08:55:00 GMT
View Forum Message <> Reply to Message

On Fri, 7 Oct 2011 12:20:04 +0400
Glauber Costa <glommer@parallels.com> wrote:

> >
> >> So what I really mean here with "will integrate later", is that I think
> >> that we'd be better off tracking the allocations themselves at the slab
> >> level.
> >>
> >>> Can't tcp-limit-code borrows some amount of charges in batch from kmem_limit
> >>> and use it ?
> >> Sorry, I don't know what exactly do you mean. Can you clarify?
> >>
> > Now, tcp-usage is independent from kmem-usage.
> >
> > My idea is
> >
> > 1. when you account tcp usage, charge kmem, too.
>
> Absolutely.
> > Now, your work is

> >   a) tcp use new xxxx bytes.
> >   b) account it to tcp.uage and check tcp limit
> >
> >   To ingegrate kmem,
> >   a) tcp use new xxxx bytes.
> >   b) account it to tcp.usage and check tcp limit
> >   c) account it to kmem.usage
> >
> > ? 2 counters may be slow ?
>
> Well, the way I see it, 1 counter is slow already =)
> I honestly think we need some optimizations here. But
> that is a side issue.
>
> To begin with: The new patchset that I intend to spin
> today or Monday, depending on my progress, uses res_counters,
> as you and Kirill requested.
>
> So what makes res_counters slow IMHO, is two things:
>
> 1) interrupts are always disabled.
> 2) All is done under a lock.
>
> Now, we are starting to have resources that are billed to multiple
> counters. One simple way to work around it, is to have child counters
> that has to be accounted for as well everytime a resource is counted.
>
> Like this:
>
> 1) tcp has kmem as child. When we bill to tcp, we bill to kmem as well.
>   For protocols that do memory pressure, we then don't bill kmem from
>   the slab.
> 2) When kmem_independent_account is set to 0, kmem has mem as child.
>

Seems reasonable.

> >
> >
> > >>>   - Don't you need a stat file to indicate "tcp memory pressure works!" ?
> > >>>   It can be obtained already ?
> > >>
> > >> Not 100 % clear as well. We can query the amount of buffer used, and the
> > >> amount of buffer allowed. What else do we need?
> > >>
> > >
> > > IIUC, we can see the fact tcp.usage is near to tcp.limit but never can see it

> > got memory pressure and how many numbers of failure happens.
> > I'm sorry if I don't read codes correctly.
>
> IIUC, With res_counters being used, we get at least failcnt for free, right?
>

Right. you can get failcnt and max_usage and can have soft_limit base
implemenation at the same time.

Thank you.
-Kame
