Subject: [PATCH v5 03/10] socket: initial cgroup code.
Posted by Glauber Costa on Mon, 07 Nov 2011 15:26:28 GMT
View Forum Message <> Reply to Message

The goal of this work is to move the memory pressure tcp
controls to a cgroup, instead of just relying on global
conditions.

To avoid excessive overhead in the network fast paths,
the code that accounts allocated memory to a cgroup is
hidden inside a static_branch(). This branch is patched out
until the first non-root cgroup is created. So when nobody
is using cgroups, even if it is mounted, no significant performance
penalty should be seen.

This patch handles the generic part of the code, and has nothing
tcp-specific.

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>
CC: Eric Dumazet <eric.dumazet@gmail.com>
---
 include/linux/memcontrol.h | 27 ++++++++
 include/net/sock.h | 151 ++-
 mm/memcontrol.c | 85 ++++++++++++++++++++++++++--
 net/core/sock.c | 36 ++++++++---
 4 files changed, 281 insertions(+), 18 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index e9ff93a..994a06a 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -381,5 +381,32 @@ mem_cgroup_print_bad_page(struct page *page)
 }
 #endif

+#ifdef CONFIG_INET
+enum {
+ UNDER_LIMIT,
+ SOFT_LIMIT,
+ OVER_LIMIT,
+};
+
+struct sock;
+struct cg_proto;

```
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+void sock_update_memcg(struct sock *sk);
+void memcg_sockets_allocated_dec(struct mem_cgroup *memcg,
+     struct cg_proto *prot);
+void memcg_sockets_allocated_inc(struct mem_cgroup *memcg,
+     struct cg_proto *prot);
+void memcg_memory_allocated_add(struct mem_cgroup *memcg, struct cg_proto *prot,
+    unsigned long amt, int *parent_status);
+void memcg_memory_allocated_sub(struct mem_cgroup *memcg, struct cg_proto *prot,
+    unsigned long amt);
+u64 memcg_memory_allocated_read(struct mem_cgroup *memcg,
+    struct cg_proto *prot);
+#else
+static inline void sock_update_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 8959dcc..02d7cce 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -55,6 +55,7 @@
 #include <linux/slab.h>
 #include <linux/uaccess.h>
 #include <linux/cgroup.h>
+#include <linux/res_counter.h>

 #include <linux/filter.h>
 #include <linux/rculist_nulls.h>
@@ -64,6 +65,8 @@
 #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
@@ -231,6 +234,7 @@ struct mem_cgroup;
   * @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
@@ -342,6 +346,7 @@ 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);
@@ -733,6 +738,9 @@ struct timewait_sock_ops;
 struct inet_hashinfo;
 struct raw_hashinfo;

+
+struct cg_proto;
+
 /* Networking protocol blocks we attach to sockets.
  * socket layer -> transport layer interface
  * transport -> network interface is defined by struct inet_proto
@@ -835,9 +843,32 @@ struct proto {
 #ifdef SOCK_REFCNT_DEBUG
  atomic_t  socks;
 #endif
+ struct cg_proto  *cg_proto; /* This just makes proto replacement easier */
+};
+
+struct cg_proto {
+ /*
+  * 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 cgroup *cgrp,
+           struct cgroup_subsys *ss);
+ void   (*destroy_cgroup)(struct cgroup *cgrp,
+        struct cgroup_subsys *ss);
+ struct res_counter *(*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);
+
+ int   *(*memory_pressure)(struct mem_cgroup *memcg);
+ long   *(*prot_mem)(struct mem_cgroup *memcg);
+
+ struct list_head node; /* with a bit more effort, we could reuse proto's */
 };

 extern int proto_register(struct proto *prot, int alloc_slab);
```

```
+extern void cg_proto_register(struct cg_proto *prot, struct proto *proto);
 extern void proto_unregister(struct proto *prot);

 #ifdef SOCK_REFCNT_DEBUG
@@ -865,15 +896,40 @@ static inline void sk_refcnt_debug_release(const struct sock *sk)
 #define sk_refcnt_debug_release(sk) do { } while (0)
 #endif /* SOCK_REFCNT_DEBUG */

+extern struct jump_label_key cgroup_crap_enabled;
 #include <linux/memcontrol.h>
 static inline int *sk_memory_pressure(const struct sock *sk)
 {
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  int *ret = NULL;
+  struct cg_proto *cg_prot = sk->sk_prot->cg_proto;
+
+  if (!sk->sk_cgrp)
+   goto nocgroup;
+  if (cg_prot->memory_pressure)
+   ret = cg_prot->memory_pressure(sk->sk_cgrp);
+  return ret;
+ } else
+nocgroup:
+#endif
  return sk->sk_prot->memory_pressure;
 }

 static inline long sk_prot_mem(const struct sock *sk, int index)
 {
  long *prot = sk->sk_prot->sysctl_mem;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct mem_cgroup *cg = sk->sk_cgrp;
+  struct cg_proto *cg_prot = sk->sk_prot->cg_proto;
+  if (!cg) /* this handles the case with existing sockets */
+   goto nocgroup;
+
+  cg_prot->prot_mem(sk->sk_cgrp);
+ }
+nocgroup:
+#endif
  return prot[index];
 }

@@ -881,32 +937,93 @@ static inline long
 sk_memory_allocated(const struct sock *sk)
 {
```

```
  struct proto *prot = sk->sk_prot;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct mem_cgroup *cg = sk->sk_cgrp;
+  struct cg_proto *cg_prot = sk->sk_prot->cg_proto;
+  if (!cg) /* this handles the case with existing sockets */
+   goto nocgroup;
+
+  return memcg_memory_allocated_read(cg, cg_prot);
+ }
+nocgroup:
+#endif
  return atomic_long_read(prot->memory_allocated);
 }

 static inline long
-sk_memory_allocated_add(struct sock *sk, int amt)
+sk_memory_allocated_add(struct sock *sk, int amt, int *parent_status)
 {
  struct proto *prot = sk->sk_prot;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct mem_cgroup *cg = sk->sk_cgrp;
+  struct cg_proto *cg_prot = prot->cg_proto;
+
+  if (!cg)
+   goto nocgroup;
+
+  memcg_memory_allocated_add(cg, cg_prot, amt, parent_status);
+ }
+nocgroup:
+#endif
  return atomic_long_add_return(amt, prot->memory_allocated);
 }

 static inline void
-sk_memory_allocated_sub(struct sock *sk, int amt)
+sk_memory_allocated_sub(struct sock *sk, int amt, int parent_status)
 {
  struct proto *prot = sk->sk_prot;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct mem_cgroup *cg = sk->sk_cgrp;
+  struct cg_proto *cg_prot = prot->cg_proto;
+
+  if (!cg)
+   goto nocgroup;
+
```

```
+  /* Otherwise it was uncharged already */
+  if (parent_status != OVER_LIMIT)
+   memcg_memory_allocated_sub(cg, cg_prot, amt);
+ }
+nocgroup:
+#endif
  atomic_long_sub(amt, prot->memory_allocated);
 }

 static inline void sk_sockets_allocated_dec(struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct mem_cgroup *cg = sk->sk_cgrp;
+  struct cg_proto *cg_prot = prot->cg_proto;
+
+  if (!cg)
+   goto nocgroup;
+
+  memcg_sockets_allocated_dec(cg, cg_prot);
+ }
+nocgroup:
+#endif
  percpu_counter_dec(prot->sockets_allocated);
 }

 static inline void sk_sockets_allocated_inc(struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct mem_cgroup *cg = sk->sk_cgrp;
+  struct cg_proto *cg_prot = prot->cg_proto;
+
+  if (!cg)
+   goto nocgroup;
+
+  memcg_sockets_allocated_inc(cg, cg_prot);
+ }
+nocgroup:
+#endif
  percpu_counter_inc(prot->sockets_allocated);
 }

@@ -914,19 +1031,47 @@ static inline int
 sk_sockets_allocated_read_positive(struct sock *sk)
 {
```

```
  struct proto *prot = sk->sk_prot;
-
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct mem_cgroup *cg = sk->sk_cgrp;
+  struct cg_proto *cg_prot = prot->cg_proto;
+
+  if (!cg)
+   goto nocgroup;
+  return percpu_counter_sum_positive(cg_prot->sockets_allocated(cg));
+ }
+nocgroup:
+#endif
  return percpu_counter_sum_positive(prot->sockets_allocated);
 }

 static inline int
 kcg_sockets_allocated_sum_positive(struct proto *prot, struct mem_cgroup *cg)
 {
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct cg_proto *cg_prot = prot->cg_proto;
+  if (!cg)
+   goto nocgroup;
+  return percpu_counter_sum_positive(cg_prot->sockets_allocated(cg));
+ }
+nocgroup:
+#endif
  return percpu_counter_sum_positive(prot->sockets_allocated);
 }

 static inline long
 kcg_memory_allocated(struct proto *prot, struct mem_cgroup *cg)
 {
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct cg_proto *cg_prot = prot->cg_proto;
+  if (!cg)
+   goto nocgroup;
+  return memcg_memory_allocated_read(cg, cg_prot);
+ }
+nocgroup:
+#endif
  return atomic_long_read(prot->memory_allocated);
 }

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 3389d33..7d684d0 100644
```

```
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -376,6 +376,85 @@ enum mem_type {
 #define MEM_CGROUP_RECLAIM_SOFT_BIT 0x2
 #define MEM_CGROUP_RECLAIM_SOFT  (1 << MEM_CGROUP_RECLAIM_SOFT_BIT)

+static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg);
+
+static inline bool mem_cgroup_is_root(struct mem_cgroup *mem)
+{
+ return (mem == root_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;
+ }
+ if (static_branch(&cgroup_crap_enabled)) {
+  struct mem_cgroup *memcg;
+
+  BUG_ON(!sk->sk_prot->cg_proto);
+
+  rcu_read_lock();
+  memcg = mem_cgroup_from_task(current);
+  if (!mem_cgroup_is_root(memcg))
+   sk->sk_cgrp = memcg;
+  rcu_read_unlock();
+ }
+}
+
+void memcg_sockets_allocated_dec(struct mem_cgroup *memcg,
+     struct cg_proto *prot)
+{
+ 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 cg_proto *prot)
```

```
+{
+ for (; memcg; memcg = parent_mem_cgroup(memcg))
+  percpu_counter_inc(prot->sockets_allocated(memcg));
+}
+EXPORT_SYMBOL(memcg_sockets_allocated_inc);
+
+void memcg_memory_allocated_add(struct mem_cgroup *memcg, struct cg_proto *prot,
+    unsigned long amt, int *parent_status)
+{
+ struct res_counter *fail;
+ int ret;
+
+ ret = res_counter_charge(prot->memory_allocated(memcg),
+     amt << PAGE_SHIFT, &fail);
+
+ if (ret < 0)
+  *parent_status = OVER_LIMIT;
+}
+EXPORT_SYMBOL(memcg_memory_allocated_add);
+
+void memcg_memory_allocated_sub(struct mem_cgroup *memcg, struct cg_proto *prot,
+    unsigned long amt)
+{
+ res_counter_uncharge(prot->memory_allocated(memcg), amt << PAGE_SHIFT);
+}
+EXPORT_SYMBOL(memcg_memory_allocated_sub);
+
+u64 memcg_memory_allocated_read(struct mem_cgroup *memcg, struct cg_proto *prot)
+{
+ return res_counter_read_u64(prot->memory_allocated(memcg),
+        RES_USAGE) >> PAGE_SHIFT ;
+}
+EXPORT_SYMBOL(memcg_memory_allocated_read);
+#endif /* CONFIG_INET */
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+
+
 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);
@@ -872,12 +951,6 @@ 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/core/sock.c b/net/core/sock.c
index 26bdb1c..b64d36a 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -111,6 +111,7 @@
 #include <linux/init.h>
 #include <linux/highmem.h>
 #include <linux/user_namespace.h>
+#include <linux/jump_label.h>

 #include <asm/uaccess.h>
 #include <asm/system.h>
@@ -141,6 +142,9 @@
 static struct lock_class_key af_family_keys[AF_MAX];
 static struct lock_class_key af_family_slock_keys[AF_MAX];

+struct jump_label_key cgroup_crap_enabled;
+EXPORT_SYMBOL(cgroup_crap_enabled);
+
 /*
  * Make lock validator output more readable. (we pre-construct these
  * strings build-time, so that runtime initialization of socket
@@ -1678,26 +1682,27 @@ int __sk_mem_schedule(struct sock *sk, int size, int kind)
  int amt = sk_mem_pages(size);
  long allocated;
  int *memory_pressure;
+ int parent_status = UNDER_LIMIT;

  sk->sk_forward_alloc += amt * SK_MEM_QUANTUM;

  memory_pressure = sk_memory_pressure(sk);
- allocated = sk_memory_allocated_add(sk, amt);
+ allocated = sk_memory_allocated_add(sk, amt, &parent_status);
+
+ /* Over hard limit (we or our parents) */
+ if ((parent_status == OVER_LIMIT) || (allocated > sk_prot_mem(sk, 2)))
+  goto suppress_allocation;

  /* Under limit. */
  if (allocated <= sk_prot_mem(sk, 0))
   if (memory_pressure && *memory_pressure)
    *memory_pressure = 0;
```

```
- /* Under pressure. */
- if (allocated > sk_prot_mem(sk, 1))
+ /* Under pressure. (we or our parents) */
+ if ((parent_status == SOFT_LIMIT) || allocated > sk_prot_mem(sk, 1))
   if (prot->enter_memory_pressure)
    prot->enter_memory_pressure(sk);

- /* Over hard limit. */
- if (allocated > sk_prot_mem(sk, 2))
-  goto suppress_allocation;
-
  /* guarantee minimum buffer size under pressure */
  if (kind == SK_MEM_RECV) {
   if (atomic_read(&sk->sk_rmem_alloc) < prot->sysctl_rmem[0])
@@ -1742,7 +1747,7 @@ suppress_allocation:
  /* Alas. Undo changes. */
  sk->sk_forward_alloc -= amt * SK_MEM_QUANTUM;

- sk_memory_allocated_sub(sk, amt);
+ sk_memory_allocated_sub(sk, amt, parent_status);

  return 0;
 }
@@ -1757,7 +1762,7 @@ void __sk_mem_reclaim(struct sock *sk)
  int *memory_pressure = sk_memory_pressure(sk);

  sk_memory_allocated_sub(sk,
-    sk->sk_forward_alloc >> SK_MEM_QUANTUM_SHIFT);
+    sk->sk_forward_alloc >> SK_MEM_QUANTUM_SHIFT, 0);
  sk->sk_forward_alloc &= SK_MEM_QUANTUM - 1;

  if (memory_pressure && *memory_pressure &&
@@ -2257,6 +2262,9 @@ EXPORT_SYMBOL(sk_common_release);
 static DEFINE_RWLOCK(proto_list_lock);
 static LIST_HEAD(proto_list);

+static DEFINE_RWLOCK(cg_proto_list_lock);
+static LIST_HEAD(cg_proto_list);
+
 #ifdef CONFIG_PROC_FS
 #define PROTO_INUSE_NR 64 /* should be enough for the first time */
 struct prot_inuse {
@@ -2358,6 +2366,16 @@ static inline void release_proto_idx(struct proto *prot)
 }
 #endif

+void cg_proto_register(struct cg_proto *prot, struct proto *parent)
```

```
+{
+ write_lock(&cg_proto_list_lock);
+ list_add(&prot->node, &cg_proto_list);
+ write_unlock(&cg_proto_list_lock);
+
+ parent->cg_proto = prot;
+}
+EXPORT_SYMBOL(cg_proto_register);
+
 int proto_register(struct proto *prot, int alloc_slab)
 {
  if (alloc_slab) {
--
1.7.6.4
```