Subject: [PATCH v7 03/10] socket: initial cgroup code.
Posted by Glauber Costa on Tue, 29 Nov 2011 23:56:54 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>

 Documentation/cgroups/memory.txt | 4 +-
 include/linux/memcontrol.h | 16 +++
 include/net/sock.h | 210 ++++++++++++++++++++++++++++++++++++++
 mm/memcontrol.c | 40 ++++++-
 net/core/sock.c | 24 +++--
 5 files changed, 274 insertions(+), 20 deletions(-)

diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index 764781c..3cf9d96 100644
--- a/Documentation/cgroups/memory.txt
+++ b/Documentation/cgroups/memory.txt
@@ -295,7 +295,9 @@ and set kmem extension config option carefully.

 2.7.1 Current Kernel Memory resources accounted

-None
+* sockets memory pressure: some sockets protocols have memory pressure
+thresholds. The Memory Controller allows them to be controlled individually
+per cgroup, instead of globally.

 3. User Interface

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h

```
index b87068a..60964c3 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -381,5 +381,21 @@ mem_cgroup_print_bad_page(struct page *page)
 }
 #endif

+#ifdef CONFIG_INET
+enum {
+ UNDER_LIMIT,
+ SOFT_LIMIT,
+ OVER_LIMIT,
+};
+
+struct sock;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+void sock_update_memcg(struct sock *sk);
+#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 a71447b..49f0912 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -54,6 +54,7 @@
 #include <linux/slab.h>
 #include <linux/uaccess.h>
 #include <linux/memcontrol.h>
+#include <linux/res_counter.h>

 #include <linux/filter.h>
 #include <linux/rculist_nulls.h>
@@ -168,6 +169,7 @@ struct sock_common {
  /* public: */
 };

+struct cg_proto;
 /**
   * struct sock - network layer representation of sockets
   * @__sk_common: shared layout with inet_timewait_sock
@@ -228,6 +230,7 @@ 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 cgroup-specific proto data
   * @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 +342,7 @@ struct sock {
 #endif
  __u32   sk_mark;
  u32   sk_classid;
+ struct cg_proto  *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);
@@ -834,6 +838,28 @@ struct proto {
 #ifdef SOCK_REFCNT_DEBUG
  atomic_t  socks;
 #endif
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ /*
+  * 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 cg_proto  *(*proto_cgroup)(struct mem_cgroup *memcg);
+#endif
+};
+
+struct cg_proto {
+ void   (*enter_memory_pressure)(struct sock *sk);
+ struct res_counter *memory_allocated; /* Current allocated memory. */
+ struct percpu_counter *sockets_allocated; /* Current number of sockets. */
+ int   *memory_pressure;
+ long   *sysctl_mem;
+ struct cg_proto  *parent;
 };

 extern int proto_register(struct proto *prot, int alloc_slab);
@@ -864,6 +890,7 @@ 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 memcg_socket_limit_enabled;
 static inline bool sk_has_memory_pressure(const struct sock *sk)
```

```
 {
  return sk->sk_prot->memory_pressure != NULL;
@@ -873,6 +900,17 @@ static inline bool sk_under_memory_pressure(const struct sock *sk)
 {
  if (!sk->sk_prot->memory_pressure)
   return false;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;
+  return !!*cg_proto->memory_pressure;
+ } else
+nocgroup:
+#endif
+
  return !!*sk->sk_prot->memory_pressure;
 }

@@ -880,52 +918,176 @@ static inline void sk_leave_memory_pressure(struct sock *sk)
 {
  int *memory_pressure = sk->sk_prot->memory_pressure;

- if (memory_pressure && *memory_pressure)
+ if (!memory_pressure)
+  return;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;
+
+  for (; cg_proto; cg_proto = cg_proto->parent)
+   if (*cg_proto->memory_pressure)
+    *cg_proto->memory_pressure = 0;
+ }
+nocgroup:
+#endif
+ if (*memory_pressure)
   *memory_pressure = 0;
 }

 static inline void sk_enter_memory_pressure(struct sock *sk)
 {
- if (sk->sk_prot->enter_memory_pressure)
-  sk->sk_prot->enter_memory_pressure(sk);
```

```
+ if (!sk->sk_prot->enter_memory_pressure)
+  return;
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;
+
+  for (; cg_proto; cg_proto = cg_proto->parent)
+   cg_proto->enter_memory_pressure(sk);
+ }
+nocgroup:
+#endif
+ sk->sk_prot->enter_memory_pressure(sk);
 }

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

+static inline void memcg_memory_allocated_add(struct cg_proto *prot,
+           unsigned long amt,
+           int *parent_status)
+{
+ struct res_counter *fail;
+ int ret;
+
+ ret = res_counter_charge(prot->memory_allocated,
+     amt << PAGE_SHIFT, &fail);
+
+ if (ret < 0)
+  *parent_status = OVER_LIMIT;
+}
+
```

```
+static inline void memcg_memory_allocated_sub(struct cg_proto *prot,
+           unsigned long amt)
+{
+ res_counter_uncharge(prot->memory_allocated, amt << PAGE_SHIFT);
+}
+
+static inline u64 memcg_memory_allocated_read(struct cg_proto *prot)
+{
+ u64 ret;
+ ret = res_counter_read_u64(prot->memory_allocated, RES_USAGE);
+ return ret >> PAGE_SHIFT;
+}
+
 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(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+  if (!cg_proto) /* this handles the case with existing sockets */
+   goto nocgroup;
+
+  return memcg_memory_allocated_read(cg_proto);
+ }
+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(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;
+
+  memcg_memory_allocated_add(cg_proto, 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(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;
+
+  /* Otherwise it was uncharged already */
+  if (parent_status != OVER_LIMIT)
+   memcg_memory_allocated_sub(cg_proto, 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(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;
+
+  for (; cg_proto; cg_proto = cg_proto->parent)
+   percpu_counter_dec(cg_proto->sockets_allocated);
+ }
+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(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
```

```
+   goto nocgroup;
+
+  for (; cg_proto; cg_proto = cg_proto->parent)
+   percpu_counter_inc(cg_proto->sockets_allocated);
+ }
+nocgroup:
+#endif
  percpu_counter_inc(prot->sockets_allocated);
 }

@@ -933,19 +1095,57 @@ 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(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;

+  return percpu_counter_sum_positive(cg_proto->sockets_allocated);
+ }
+nocgroup:
+#endif
  return percpu_counter_sum_positive(prot->sockets_allocated);
 }

 static inline int
 memcg_sockets_allocated_sum_positive(struct proto *prot, struct mem_cgroup *memcg)
 {
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto;
+  if (!prot->proto_cgroup)
+   goto nocgroup;
+
+  cg_proto = prot->proto_cgroup(memcg);
+  if (!cg_proto)
+   goto nocgroup;
+
+  return percpu_counter_sum_positive(cg_proto->sockets_allocated);
+ }
+nocgroup:
+#endif
  return percpu_counter_sum_positive(prot->sockets_allocated);
 }
```

```
 static inline long
 memcg_memory_allocated(struct proto *prot, struct mem_cgroup *memcg)
 {
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto;
+  if (!prot->proto_cgroup)
+   goto nocgroup;
+
+  cg_proto = prot->proto_cgroup(memcg);
+  if (!cg_proto)
+   goto nocgroup;
+
+  return memcg_memory_allocated_read(cg_proto);
+ }
+nocgroup:
+#endif
  return atomic_long_read(prot->memory_allocated);
 }

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 3becb24..12a08bf 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -377,6 +377,40 @@ enum mem_type {
 #define MEM_CGROUP_RECLAIM_SOFT_BIT 0x2
 #define MEM_CGROUP_RECLAIM_SOFT  (1 << MEM_CGROUP_RECLAIM_SOFT_BIT)

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

```
+  BUG_ON(!sk->sk_prot->proto_cgroup);
+
+  rcu_read_lock();
+  memcg = mem_cgroup_from_task(current);
+  if (!mem_cgroup_is_root(memcg))
+   sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
+  rcu_read_unlock();
+ }
+}
+
+#endif /* CONFIG_INET */
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+
+
 static void mem_cgroup_get(struct mem_cgroup *memcg);
 static void mem_cgroup_put(struct mem_cgroup *memcg);
 static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg);
@@ -878,12 +912,6 @@ 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 *memcg)
-{
- return (memcg == root_mem_cgroup);
-}
-
 void mem_cgroup_count_vm_event(struct mm_struct *mm, enum vm_event_item idx)
 {
  struct mem_cgroup *memcg;
diff --git a/net/core/sock.c b/net/core/sock.c
index 44213ab..e7e0808 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 memcg_socket_limit_enabled;
```

```
+EXPORT_SYMBOL(memcg_socket_limit_enabled);
+
 /*
  * Make lock validator output more readable. (we pre-construct these
  * strings build-time, so that runtime initialization of socket
@@ -1677,21 +1681,25 @@ 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 parent_status = UNDER_LIMIT;

  sk->sk_forward_alloc += amt * SK_MEM_QUANTUM;

- allocated = sk_memory_allocated_add(sk, amt);
+ allocated = sk_memory_allocated_add(sk, amt, &parent_status);

  /* Under limit. */
- if (allocated <= sk_prot_mem_limits(sk, 0))
+ if (parent_status == UNDER_LIMIT &&
+    allocated <= sk_prot_mem_limits(sk, 0))
   sk_leave_memory_pressure(sk);

- /* Under pressure. */
- if (allocated > sk_prot_mem_limits(sk, 1))
+ /* Under pressure. (we or our parents) */
+ if ((parent_status > SOFT_LIMIT) ||
+    allocated > sk_prot_mem_limits(sk, 1))
   sk_enter_memory_pressure(sk);

- /* Over hard limit. */
- if (allocated > sk_prot_mem_limits(sk, 2))
+ /* Over hard limit (we or our parents) */
+ if ((parent_status == OVER_LIMIT) ||
+    (allocated > sk_prot_mem_limits(sk, 2)))
   goto suppress_allocation;

  /* guarantee minimum buffer size under pressure */
@@ -1738,7 +1746,7 @@ 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;
 }
@@ -1751,7 +1759,7 @@ EXPORT_SYMBOL(__sk_mem_schedule);
 void __sk_mem_reclaim(struct sock *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 (sk_under_memory_pressure(sk) &&
--
1.7.6.4
```