Subject: [PATCH v5 06/10] tcp buffer limitation: per-cgroup limit
Posted by Glauber Costa on Mon, 07 Nov 2011 15:26:31 GMT
View Forum Message <> Reply to Message

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

This value is ignored in the root cgroup, and in all others,
caps the value specified by the admin in the net namespaces'
view of tcp_sysctl_mem.

If namespaces are being used, the admin is allowed to set a
value bigger than cgroup's maximum, the same way it is allowed
to set pretty much unlimited values in a real box.

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 | 9 ++++
 mm/memcontrol.c | 76 ++++++++++++++++++++++++++++++++++++++
 net/ipv4/sysctl_net_ipv4.c | 14 +++++++
 4 files changed, 99 insertions(+), 1 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 994a06a..d025979 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -402,10 +402,19 @@ 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);
+unsigned long long tcp_max_memory(const struct mem_cgroup *memcg);

```
+void tcp_prot_mem(struct mem_cgroup *memcg, long val, int idx);
 #else
 static inline void sock_update_memcg(struct sock *sk)
 {
 }
+static inline unsigned long long tcp_max_memory(const struct mem_cgroup *memcg)
+{
+ return 0;
+}
+static inline void tcp_prot_mem(struct mem_cgroup *memcg, long val, int idx)
+{
+}
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
 #endif /* CONFIG_INET */
 #endif /* _LINUX_MEMCONTROL_H */
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 63360f8..ee122a6 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -365,6 +365,7 @@ enum mem_type {
  _MEMSWAP,
  _OOM_TYPE,
  _KMEM,
+ _KMEM_TCP,
 };

 #define MEMFILE_PRIVATE(x, val) (((x) << 16) | (val))
@@ -500,6 +501,35 @@ struct percpu_counter *sockets_allocated_tcp(struct mem_cgroup
*memcg)
 }
 EXPORT_SYMBOL(sockets_allocated_tcp);

+static void tcp_update_limit(struct mem_cgroup *memcg, u64 val)
+{
+ struct net *net = current->nsproxy->net_ns;
+ int i;
+
+ val >>= PAGE_SHIFT;
+
+ for (i = 0; i < 3; i++)
+  memcg->tcp.tcp_prot_mem[i]  = min_t(long, val,
+         net->ipv4.sysctl_tcp_mem[i]);
+}
+
+static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
+       const char *buffer);
+
+static u64 mem_cgroup_read(struct cgroup *cont, struct cftype *cft);
```

```
+/*
+ * We need those things internally in pages, so don't reuse
+ * mem_cgroup_{read,write}
+ */
+static struct cftype tcp_files[] = {
+ {
+  .name = "kmem.tcp.limit_in_bytes",
+  .write_string = mem_cgroup_write,
+  .read_u64 = mem_cgroup_read,
+  .private = MEMFILE_PRIVATE(_KMEM_TCP, RES_LIMIT),
+ },
+};
+
 static void tcp_create_cgroup(struct mem_cgroup *cg, struct cgroup_subsys *ss)
 {
  /*
@@ -527,6 +557,7 @@ static void tcp_create_cgroup(struct mem_cgroup *cg, struct
cgroup_subsys *ss)
 int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)
 {
  struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
+ struct mem_cgroup *parent = parent_mem_cgroup(memcg);
  struct net *net = current->nsproxy->net_ns;
  /*
   * We need to initialize it at populate, not create time.
@@ -537,7 +568,20 @@ int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)
  memcg->tcp.tcp_prot_mem[1] = net->ipv4.sysctl_tcp_mem[1];
  memcg->tcp.tcp_prot_mem[2] = net->ipv4.sysctl_tcp_mem[2];

- return 0;
+ /* Let root cgroup unlimited. All others, respect parent's if needed */
+ if (parent && !parent->use_hierarchy) {
+  unsigned long limit;
+  int ret;
+  limit = nr_free_buffer_pages() / 8;
+  limit = max(limit, 128UL);
+  ret = res_counter_set_limit(&memcg->tcp.tcp_memory_allocated,
+         limit * 2);
+  if (ret)
+   return ret;
+ }
+
+ return cgroup_add_files(cgrp, ss, tcp_files,
+    ARRAY_SIZE(tcp_files));
 }
 EXPORT_SYMBOL(tcp_init_cgroup);

@@ -548,6 +592,18 @@ void tcp_destroy_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)
```

```
 percpu_counter_destroy(&memcg->tcp.tcp_sockets_allocated);
}
EXPORT_SYMBOL(tcp_destroy_cgroup);
+
+unsigned long long tcp_max_memory(const struct mem_cgroup *memcg)
+{
+ struct mem_cgroup *cmemcg = (struct mem_cgroup *)memcg;
+ return res_counter_read_u64(&cmemcg->tcp.tcp_memory_allocated,
+        RES_LIMIT);
+}
+
+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 */

@@ -4067,6 +4123,15 @@ static u64 mem_cgroup_read(struct cgroup *cont, struct cftype *cft)
   val = res_counter_read_u64(&mem->kmem, name);
   break;

+#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) && defined(CONFIG_INET)
+ case _KMEM_TCP:
+  /* Be explicit: tcp root does not have a res_counter */
+  if (mem_cgroup_is_root(mem))
+   val = RESOURCE_MAX;
+  else
+   val = res_counter_read_u64(&mem->tcp.tcp_memory_allocated, name);
+  break;
+#endif
  default:
   BUG();
   break;
@@ -4099,6 +4164,15 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
    break;
   if (type == _MEM)
    ret = mem_cgroup_resize_limit(memcg, val);
+
+#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) && defined(CONFIG_INET)
+  else if (type == _KMEM_TCP) {
+   ret = res_counter_set_limit(&memcg->tcp.tcp_memory_allocated,
+           val);
+   if (!ret)
+    tcp_update_limit(memcg, val);
+  }
+#endif
   else
```

```
    ret = mem_cgroup_resize_memsw_limit(memcg, val);
   break;
diff --git a/net/ipv4/sysctl_net_ipv4.c b/net/ipv4/sysctl_net_ipv4.c
index bbd67ab..915e192 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 +183,9 @@ 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
+ struct mem_cgroup *cg;
+#endif

  ctl_table tmp = {
   .data = &vec,
@@ -198,6 +202,16 @@ 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);
+
+ 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];
--
1.7.6.4
```