Subject: [PATCH v6 06/10] tcp buffer limitation: per-cgroup limit
Posted by Glauber Costa on Fri, 25 Nov 2011 17:38:12 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/net/tcp_memcg.h | 3 +
 net/ipv4/sysctl_net_ipv4.c | 14 ++++
 net/ipv4/tcp_memcg.c | 138 ++++++++++++++++++++++++++++++++++++++-
 4 files changed, 154 insertions(+), 2 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/net/tcp_memcg.h b/include/net/tcp_memcg.h
index 5f5e158..2c8bb6b 100644
--- a/include/net/tcp_memcg.h
+++ b/include/net/tcp_memcg.h
@@ -14,4 +14,7 @@ struct tcp_memcontrol {
 struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg);
 int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss);
 void tcp_destroy_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss);
+unsigned long long tcp_max_memory(const struct mem_cgroup *memcg);
+void tcp_prot_mem(struct mem_cgroup *memcg, long val, int idx);

```
+int tcp_update_limit(struct mem_cgroup *memcg, u64 val);
 #endif /* _TCP_MEMCG_H */
diff --git a/net/ipv4/sysctl_net_ipv4.c b/net/ipv4/sysctl_net_ipv4.c
index bbd67ab..17aaa1b 100644
--- a/net/ipv4/sysctl_net_ipv4.c
+++ b/net/ipv4/sysctl_net_ipv4.c
@@ -24,6 +24,7 @@
 #include <net/cipso_ipv4.h>
 #include <net/inet_frag.h>
 #include <net/ping.h>
+#include <net/tcp_memcg.h>

 static int zero;
 static int tcp_retr1_max = 255;
@@ -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];
diff --git a/net/ipv4/tcp_memcg.c b/net/ipv4/tcp_memcg.c
index 1dbc0f3..b3721c3 100644
--- a/net/ipv4/tcp_memcg.c
+++ b/net/ipv4/tcp_memcg.c
@@ -5,6 +5,19 @@
 #include <linux/nsproxy.h>
 #include <linux/memcontrol.h>
```

```
+static u64 tcp_cgroup_read(struct cgroup *cont, struct cftype *cft);
+static int tcp_cgroup_write(struct cgroup *cont, struct cftype *cft,
+       const char *buffer);
+
+static struct cftype tcp_files[] = {
+ {
+  .name = "kmem.tcp.limit_in_bytes",
+  .write_string = tcp_cgroup_write,
+  .read_u64 = tcp_cgroup_read,
+  .private = RES_LIMIT,
+ },
+};
+
 static inline struct tcp_memcontrol *tcp_from_cgproto(struct cg_proto *cg_proto)
 {
  return container_of(cg_proto, struct tcp_memcontrol, cg_proto);
@@ -26,7 +39,7 @@ int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)

  cg_proto = tcp_prot.proto_cgroup(memcg);
  if (!cg_proto)
-  return 0;
+  goto create_files;

  tcp = tcp_from_cgproto(cg_proto);
  cg_proto->parent = tcp_prot.proto_cgroup(parent);
@@ -47,7 +60,9 @@ int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)
  cg_proto->memory_allocated = &tcp->tcp_memory_allocated;
  cg_proto->sockets_allocated = &tcp->tcp_sockets_allocated;

- return 0;
+create_files:
+ return cgroup_add_files(cgrp, ss, tcp_files,
+     ARRAY_SIZE(tcp_files));
 }
 EXPORT_SYMBOL(tcp_init_cgroup);

@@ -56,6 +71,7 @@ void tcp_destroy_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)
  struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
  struct cg_proto *cg_proto;
  struct tcp_memcontrol *tcp;
+ u64 val;

  cg_proto = tcp_prot.proto_cgroup(memcg);
  if (!cg_proto)
@@ -63,5 +79,123 @@ void tcp_destroy_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)

  tcp = tcp_from_cgproto(cg_proto);
```

```
  percpu_counter_destroy(&tcp->tcp_sockets_allocated);
+
+ val = res_counter_read_u64(&tcp->tcp_memory_allocated, RES_USAGE);
+
+ if (val != RESOURCE_MAX)
+  jump_label_dec(&memcg_socket_limit_enabled);
 }
 EXPORT_SYMBOL(tcp_destroy_cgroup);
+
+int tcp_update_limit(struct mem_cgroup *memcg, u64 val)
+{
+ struct net *net = current->nsproxy->net_ns;
+ struct tcp_memcontrol *tcp;
+ struct cg_proto *cg_proto;
+ int i;
+ int ret;
+
+ cg_proto = tcp_prot.proto_cgroup(memcg);
+ if (!cg_proto)
+  return -EINVAL;
+
+ tcp = tcp_from_cgproto(cg_proto);
+
+ ret = res_counter_set_limit(&tcp->tcp_memory_allocated, val);
+ if (ret)
+  return ret;
+
+ val >>= PAGE_SHIFT;
+
+ for (i = 0; i < 3; i++)
+  tcp->tcp_prot_mem[i] = min_t(long, val,
+          net->ipv4.sysctl_tcp_mem[i]);
+
+ if (val == RESOURCE_MAX)
+  jump_label_dec(&memcg_socket_limit_enabled);
+ else {
+  u64 old_lim;
+  old_lim = res_counter_read_u64(&tcp->tcp_memory_allocated,
+            RES_LIMIT);
+  if (old_lim == RESOURCE_MAX)
+   jump_label_inc(&memcg_socket_limit_enabled);
+ }
+ return 0;
+}
+
+static int tcp_cgroup_write(struct cgroup *cont, struct cftype *cft,
+       const char *buffer)
+{
```

```
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);
+ unsigned long long val;
+ int ret = 0;
+
+ switch (cft->private) {
+ case RES_LIMIT:
+  /* see memcontrol.c */
+  ret = res_counter_memparse_write_strategy(buffer, &val);
+  if (ret)
+   break;
+  ret = tcp_update_limit(memcg, val);
+  break;
+ default:
+  ret = -EINVAL;
+  break;
+ }
+ return ret;
+}
+
+static u64 tcp_read_stat(struct mem_cgroup *memcg, int type, u64 default_val)
+{
+ struct tcp_memcontrol *tcp;
+ struct cg_proto *cg_proto;
+
+ cg_proto = tcp_prot.proto_cgroup(memcg);
+ if (!cg_proto)
+  return default_val;
+
+ tcp = tcp_from_cgproto(cg_proto);
+ return res_counter_read_u64(&tcp->tcp_memory_allocated, type);
+}
+
+static u64 tcp_cgroup_read(struct cgroup *cont, struct cftype *cft)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);
+ u64 val;
+
+ switch (cft->private) {
+ case RES_LIMIT:
+  val = tcp_read_stat(memcg, RES_LIMIT, RESOURCE_MAX);
+  break;
+ default:
+  BUG();
+ }
+ return val;
+}
+
+unsigned long long tcp_max_memory(const struct mem_cgroup *memcg)
```

```
+{
+ struct tcp_memcontrol *tcp;
+ struct cg_proto *cg_proto;
+
+ cg_proto = tcp_prot.proto_cgroup((struct mem_cgroup *)memcg);
+ if (!cg_proto)
+  return 0;
+
+ tcp = tcp_from_cgproto(cg_proto);
+ return res_counter_read_u64(&tcp->tcp_memory_allocated, RES_LIMIT);
+}
+
+void tcp_prot_mem(struct mem_cgroup *memcg, long val, int idx)
+{
+ struct tcp_memcontrol *tcp;
+ struct cg_proto *cg_proto;
+
+ cg_proto = tcp_prot.proto_cgroup(memcg);
+ if (!cg_proto)
+  return;
+
+ tcp = tcp_from_cgproto(cg_proto);
+
+ tcp->tcp_prot_mem[idx] = val;
+}
--
1.7.6.4
```