Subject: [PATCH v5 04/10] per-cgroup tcp buffers control
Posted by Glauber Costa on Mon, 07 Nov 2011 15:26:29 GMT
View Forum Message <> Reply to Message

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

A resource conter is used to control allocated memory, except
for the root cgroup, that will keep using global counters.

This patch is the one that actually enables/disables the
jump labels controlling cgroup. To this point, they were always
disabled.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujtisu.com>
CC: David S. Miller <davem@davemloft.net>
CC: Eric W. Biederman <ebiederm@xmission.com>
CC: Eric Dumazet <eric.dumazet@gmail.com>
---
 include/net/tcp.h | 18 +++++++
 include/net/transp_v6.h | 1 +
 mm/memcontrol.c | 125 ++-
 net/core/sock.c | 46 ++++++++++++++---
 net/ipv4/af_inet.c | 3 +
 net/ipv4/tcp_ipv4.c | 12 +++++
 net/ipv6/af_inet6.c | 3 +
 net/ipv6/tcp_ipv6.c | 10 ++++
 8 files changed, 211 insertions(+), 7 deletions(-)

diff --git a/include/net/tcp.h b/include/net/tcp.h
index ccaa3b6..7301ca8 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -253,6 +253,22 @@ extern int sysctl_tcp_cookie_size;
 extern int sysctl_tcp_thin_linear_timeouts;
 extern int sysctl_tcp_thin_dupack;

+struct tcp_memcontrol {
+ /* per-cgroup tcp memory pressure knobs */
+ struct res_counter 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;
+};
+
+long *sysctl_mem_tcp(struct mem_cgroup *memcg);

```
+struct percpu_counter *sockets_allocated_tcp(struct mem_cgroup *memcg);
+int *memory_pressure_tcp(struct mem_cgroup *memcg);
+struct res_counter *memory_allocated_tcp(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);
+
 extern atomic_long_t tcp_memory_allocated;
 extern struct percpu_counter tcp_sockets_allocated;
 extern int tcp_memory_pressure;
@@ -305,6 +321,7 @@ static inline int tcp_synq_no_recent_overflow(const struct sock *sk)
 }

 extern struct proto tcp_prot;
+extern struct cg_proto tcp_cg_prot;

 #define TCP_INC_STATS(net, field) SNMP_INC_STATS((net)->mib.tcp_statistics, field)
 #define TCP_INC_STATS_BH(net, field) SNMP_INC_STATS_BH((net)->mib.tcp_statistics, field)
@@ -1022,6 +1039,7 @@ static inline void tcp_openreq_init(struct request_sock *req,
  ireq->loc_port = tcp_hdr(skb)->dest;
 }

+extern void tcp_enter_memory_pressure_cg(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/include/net/transp_v6.h b/include/net/transp_v6.h
index 498433d..1e18849 100644
--- a/include/net/transp_v6.h
+++ b/include/net/transp_v6.h
@@ -11,6 +11,7 @@ extern struct proto rawv6_prot;
 extern struct proto udpv6_prot;
 extern struct proto udplitev6_prot;
 extern struct proto tcpv6_prot;
+extern struct cg_proto tcpv6_cg_prot;

 struct flowi6;

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 7d684d0..f14d7d2 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
 };

 /* Stuffs for move charges at task migration. */
@@ -377,7 +384,7 @@ enum mem_type {
 #define MEM_CGROUP_RECLAIM_SOFT  (1 << MEM_CGROUP_RECLAIM_SOFT_BIT)

 static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg);
-
+static struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont);
 static inline bool mem_cgroup_is_root(struct mem_cgroup *mem)
 {
  return (mem == root_mem_cgroup);
@@ -387,6 +394,7 @@ static inline bool mem_cgroup_is_root(struct mem_cgroup *mem)
 #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)
 {
@@ -451,6 +459,93 @@ u64 memcg_memory_allocated_read(struct mem_cgroup *memcg,
struct cg_proto *prot)
        RES_USAGE) >> PAGE_SHIFT ;
 }
 EXPORT_SYMBOL(memcg_memory_allocated_read);
+/*
+ * 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_cg(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_cg);
+
+long *sysctl_mem_tcp(struct mem_cgroup *memcg)
+{
+ return memcg->tcp.tcp_prot_mem;
+}
+EXPORT_SYMBOL(sysctl_mem_tcp);
+
+struct res_counter *memory_allocated_tcp(struct mem_cgroup *memcg)
+{
+ return &memcg->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)
+{
+ /*
+  * The root cgroup does not use res_counters, but rather,
+  * rely on the data already collected by the network
+  * subsystem
+  */
+ if (!mem_cgroup_is_root(cg)) {
+  struct mem_cgroup *parent = parent_mem_cgroup(cg);
+  struct res_counter *res_parent = NULL;
+  cg->tcp.tcp_memory_pressure = 0;
+  percpu_counter_init(&cg->tcp.tcp_sockets_allocated, 0);
+
+  /*
+   * Because root is not using res_counter, we only need a parent
+   * if we're second in hierarchy.
+   */
+  if (!mem_cgroup_is_root(parent) && parent && parent->use_hierarchy)
+   res_parent = &parent->tcp.tcp_memory_allocated;
+
```

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

@@ -4867,17 +4962,39 @@ static struct cftype kmem_cgroup_files[] = {
 static int register_kmem_files(struct cgroup *cont, struct cgroup_subsys *ss)
 {
  int ret = 0;
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);

  ret = cgroup_add_files(cont, ss, kmem_cgroup_files,
          ARRAY_SIZE(kmem_cgroup_files));
+
+ if (!ret)
+  ret = sockets_populate(cont, ss);
+
+ if (!mem_cgroup_is_root(memcg))
+  jump_label_inc(&cgroup_crap_enabled);
+
  return ret;
 };
```

```
+static void kmem_cgroup_destroy(struct cgroup_subsys *ss,
+    struct cgroup *cont)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);
+ sockets_destroy(cont, ss);
+
+ if (!mem_cgroup_is_root(memcg))
+  jump_label_dec(&cgroup_crap_enabled);
+}
 #else
 static int register_kmem_files(struct cgroup *cont, struct cgroup_subsys *ss)
 {
  return 0;
 }
+
+static void kmem_cgroup_destroy(struct cgroup_subsys *ss,
+    struct cgroup *cont)
+{
+}
 #endif

 static int alloc_mem_cgroup_per_zone_info(struct mem_cgroup *mem, int node)
@@ -5095,6 +5212,10 @@ 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);
@@ -5120,6 +5241,8 @@ 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 b64d36a..c784173 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -135,6 +135,46 @@
 #include <net/tcp.h>
```

```
 #endif

+static DEFINE_RWLOCK(proto_list_lock);
+static LIST_HEAD(proto_list);
+
+static DEFINE_RWLOCK(cg_proto_list_lock);
+static LIST_HEAD(cg_proto_list);
+
+int sockets_populate(struct cgroup *cgrp, struct cgroup_subsys *ss)
+{
+ struct cg_proto *proto;
+ int ret = 0;
+
+ read_lock(&cg_proto_list_lock);
+ list_for_each_entry(proto, &cg_proto_list, node) {
+  if (proto->init_cgroup)
+   ret = proto->init_cgroup(cgrp, ss);
+   if (ret)
+    goto out;
+ }
+
+ read_unlock(&cg_proto_list_lock);
+ return ret;
+out:
+ list_for_each_entry_continue_reverse(proto, &cg_proto_list, node)
+  if (proto->destroy_cgroup)
+   proto->destroy_cgroup(cgrp, ss);
+ read_unlock(&cg_proto_list_lock);
+ return ret;
+}
+
+void sockets_destroy(struct cgroup *cgrp, struct cgroup_subsys *ss)
+{
+ struct cg_proto *proto;
+
+ read_lock(&cg_proto_list_lock);
+ list_for_each_entry_reverse(proto, &cg_proto_list, node)
+  if (proto->destroy_cgroup)
+   proto->destroy_cgroup(cgrp, ss);
+ read_unlock(&cg_proto_list_lock);
+}
+
 /*
  * Each address family might have different locking rules, so we have
  * one slock key per address family:
@@ -2259,12 +2299,6 @@ void sk_common_release(struct sock *sk)
 }
 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 {
diff --git a/net/ipv4/af_inet.c b/net/ipv4/af_inet.c
index 1b5096a..da19147 100644
--- a/net/ipv4/af_inet.c
+++ b/net/ipv4/af_inet.c
@@ -1661,6 +1661,9 @@ static int __init inet_init(void)
  if (rc)
   goto out_unregister_raw_proto;

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ cg_proto_register(&tcp_cg_prot, &tcp_prot);
+#endif
  /*
   * Tell SOCKET that we are alive...
   */
diff --git a/net/ipv4/tcp_ipv4.c b/net/ipv4/tcp_ipv4.c
index f124a4b..54f6b96 100644
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -1917,6 +1917,7 @@ static int tcp_v4_init_sock(struct sock *sk)
  sk_sockets_allocated_inc(sk);
  local_bh_enable();

+ sock_update_memcg(sk);
  return 0;
 }

@@ -2632,6 +2633,17 @@ struct proto tcp_prot = {
 };
 EXPORT_SYMBOL(tcp_prot);

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+struct cg_proto tcp_cg_prot = {
+ .memory_allocated = memory_allocated_tcp,
+ .memory_pressure = memory_pressure_tcp,
+ .sockets_allocated = sockets_allocated_tcp,
+ .prot_mem  = sysctl_mem_tcp,
+ .init_cgroup  = tcp_init_cgroup,
+ .destroy_cgroup  = tcp_destroy_cgroup,
```

```
+};
+EXPORT_SYMBOL(tcp_cg_prot);
+#endif

 static int __net_init tcp_sk_init(struct net *net)
 {
diff --git a/net/ipv6/af_inet6.c b/net/ipv6/af_inet6.c
index d27c797..51672f8 100644
--- a/net/ipv6/af_inet6.c
+++ b/net/ipv6/af_inet6.c
@@ -1103,6 +1103,9 @@ static int __init inet6_init(void)
  if (err)
   goto out_unregister_raw_proto;

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ cg_proto_register(&tcpv6_cg_prot, &tcpv6_prot);
+#endif
  /* Register the family here so that the init calls below will
   * be able to create sockets. (?? is this dangerous ??)
   */
diff --git a/net/ipv6/tcp_ipv6.c b/net/ipv6/tcp_ipv6.c
index 3a08fcd..3c13142 100644
--- a/net/ipv6/tcp_ipv6.c
+++ b/net/ipv6/tcp_ipv6.c
@@ -2224,6 +2224,16 @@ struct proto tcpv6_prot = {
 #endif
 };

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+struct cg_proto tcpv6_cg_prot = {
+ .memory_allocated = memory_allocated_tcp,
+ .memory_pressure = memory_pressure_tcp,
+ .sockets_allocated = sockets_allocated_tcp,
+ .prot_mem  = sysctl_mem_tcp,
+};
+EXPORT_SYMBOL(tcpv6_cg_prot);
+#endif
+
 static const struct inet6_protocol tcpv6_protocol = {
  .handler = tcp_v6_rcv,
  .err_handler = tcp_v6_err,
--
1.7.6.4
```