Subject: RE: [PATCH v5 04/10] per-cgroup tcp buffers control
Posted by Glauber Costa on Mon, 07 Nov 2011 17:28:50 GMT
View Forum Message <> Reply to Message

Ok, I forgot to change the temporary name I was using for the jump label. Shame on me :)

--- Mensagem Original ---

De: Glauber Costa <glommer@parallels.com>
Enviado: 7 de novembro de 2011 07/11/11
Para: linux-kernel@vger.kernel.org
Cc: paul@paulmenage.org, lizf@cn.fujitsu.com, kamezawa.hiroyu@jp.fujitsu.com, ebiederm@xmission.com, davem@davemloft.net, gthelen@google.com, netdev@vger.kernel.org, linux-mm@kvack.org, kirill@shutemov.name, Andrey Vagin <avagin@parallels.com>, devel@openvz.org, eric.dumazet@gmail.com, Glauber Costa <glommer@parallels.com>, KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujtisu.com>
Assunto: [PATCH v5 04/10] per-cgroup tcp buffers control

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-
```