Subject: [PATCH v5 4/8] per-cgroup tcp buffers control
Posted by Glauber Costa on Tue, 04 Oct 2011 12:17:56 GMT
View Forum Message <> Reply to Message

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

Signed-off-by: Glauber Costa <glommer@parallels.com>
Reviewed-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujtisu.com>
CC: David S. Miller <davem@davemloft.net>
CC: Eric W. Biederman <ebiederm@xmission.com>
---
 include/linux/memcontrol.h | 4 ++
 include/net/sock.h | 15 ++++++-
 include/net/tcp.h | 15 +++++++
 mm/memcontrol.c | 97 ++
 net/core/sock.c | 39 ++++++++++++++++-
 net/ipv4/tcp.c | 44 +++++++++++++++-----
 net/ipv4/tcp_ipv4.c | 12 +++++-
 net/ipv6/tcp_ipv6.c | 10 ++++-
 8 files changed, 207 insertions(+), 29 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 2012060..50af61d 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -391,6 +391,10 @@ void memcg_sock_mem_alloc(struct mem_cgroup *memcg, struct proto
*prot,
 void memcg_sock_mem_free(struct mem_cgroup *memcg, struct proto *prot, int amt);
 void memcg_sockets_allocated_dec(struct mem_cgroup *memcg, struct proto *prot);
 void memcg_sockets_allocated_inc(struct mem_cgroup *memcg, struct proto *prot);
+int tcp_init_cgroup(struct proto *prot, struct cgroup *cgrp,
+   struct cgroup_subsys *ss);
+void tcp_destroy_cgroup(struct proto *prot, struct cgroup *cgrp,
+   struct cgroup_subsys *ss);
 #else
 /* memcontrol includes sockets.h, that includes memcontrol.h ... */
 static inline void memcg_sock_mem_alloc(struct mem_cgroup *memcg,
diff --git a/include/net/sock.h b/include/net/sock.h
index c6983cf..0625d79 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -64,6 +64,8 @@
 #include <net/dst.h>
 #include <net/checksum.h>

+int sockets_populate(struct cgroup *cgrp, struct cgroup_subsys *ss);
+void sockets_destroy(struct cgroup *cgrp, struct cgroup_subsys *ss);

```
 /*
  * This structure really needs to be cleaned up.
  * Most of it is for TCP, and not used by any of
@@ -814,7 +816,18 @@ struct proto {
  int   *(*memory_pressure)(struct mem_cgroup *memcg);
  /* Pointer to the per-cgroup version of the the sysctl_mem field */
  long   *(*prot_mem)(struct mem_cgroup *memcg);
-
+ /*
+  * 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 proto *prot,
+            struct cgroup *cgrp,
+            struct cgroup_subsys *ss);
+ void   (*destroy_cgroup)(struct proto *prot,
+        struct cgroup *cgrp,
+        struct cgroup_subsys *ss);
  int   *sysctl_wmem;
  int   *sysctl_rmem;
  int   max_header;
diff --git a/include/net/tcp.h b/include/net/tcp.h
index 2cbbc13..7ae7b4b 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -255,6 +255,20 @@ extern int sysctl_tcp_thin_linear_timeouts;
 extern int sysctl_tcp_thin_dupack;

 struct mem_cgroup;
+struct tcp_memcontrol {
+ /* per-cgroup tcp memory pressure knobs */
+ atomic_long_t 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;
+};
+
+extern long *tcp_sysctl_mem_nocg(struct mem_cgroup *memcg);
+struct percpu_counter *sockets_allocated_tcp_nocg(struct mem_cgroup *memcg);
+int *memory_pressure_tcp_nocg(struct mem_cgroup *memcg);
+atomic_long_t *memory_allocated_tcp_nocg(struct mem_cgroup *memcg);
+
 extern long *tcp_sysctl_mem(struct mem_cgroup *memcg);
 struct percpu_counter *sockets_allocated_tcp(struct mem_cgroup *memcg);
 int *memory_pressure_tcp(struct mem_cgroup *memcg);
```

```
@@ -1022,6 +1036,7 @@ static inline void tcp_openreq_init(struct request_sock *req,
  ireq->loc_port = tcp_hdr(skb)->dest;
 }

+extern void tcp_enter_memory_pressure_nocg(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/mm/memcontrol.c b/mm/memcontrol.c
index 60ab41d..9598e3e 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
 };

 static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg);
@@ -301,6 +308,7 @@ static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup
*memcg);
 #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)
 {
@@ -373,6 +381,80 @@ void memcg_sockets_allocated_inc(struct mem_cgroup *memcg, struct
proto *prot)
   percpu_counter_inc(prot->sockets_allocated(memcg));
 }
 EXPORT_SYMBOL(memcg_sockets_allocated_inc);
+
```

```
+static struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont);
+/*
+ * 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(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);
+
+long *tcp_sysctl_mem(struct mem_cgroup *cg)
+{
+ return cg->tcp.tcp_prot_mem;
+}
+EXPORT_SYMBOL(tcp_sysctl_mem);
+
+atomic_long_t *memory_allocated_tcp(struct mem_cgroup *cg)
+{
+ return &(cg->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)
+{
+ cg->tcp.tcp_memory_pressure = 0;
+ atomic_long_set(&cg->tcp.tcp_memory_allocated, 0);
+ percpu_counter_init(&cg->tcp.tcp_sockets_allocated, 0);
+}
+
```

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

@@ -4864,9 +4946,18 @@ static int register_kmem_files(struct cgroup *cont, struct
cgroup_subsys *ss)
  if (!mem_cgroup_is_root(memcg))
   ret = cgroup_add_files(cont, ss, kmem_cgroup_files,
      ARRAY_SIZE(kmem_cgroup_files));
+
+ if (!ret)
+  ret = sockets_populate(cont, ss);
+
  return ret;
 };

+static void kmem_cgroup_destroy(struct cgroup_subsys *ss,
+    struct cgroup *cont)
+{
+ sockets_destroy(cont, ss);
+}
 #else
 static int register_kmem_files(struct cgroup *cont, struct cgroup_subsys *ss)
 {
```

```
@@ -5089,6 +5180,10 @@ 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);
@@ -5114,6 +5209,8 @@ 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 6e3ace7..ef3a9a4 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -135,6 +135,42 @@
 #include <net/tcp.h>
 #endif

+static DEFINE_RWLOCK(proto_list_lock);
+static LIST_HEAD(proto_list);
+
+int sockets_populate(struct cgroup *cgrp, struct cgroup_subsys *ss)
+{
+ struct proto *proto;
+ int ret = 0;
+
+ read_lock(&proto_list_lock);
+ list_for_each_entry(proto, &proto_list, node) {
+  if (proto->init_cgroup)
+   ret = proto->init_cgroup(proto, cgrp, ss);
+   if (ret)
+    goto out;
+ }
+
+ read_unlock(&proto_list_lock);
+ return ret;
+out:
+ list_for_each_entry_continue_reverse(proto, &proto_list, node)
```

```
+  if (proto->destroy_cgroup)
+   proto->destroy_cgroup(proto, cgrp, ss);
+ read_unlock(&proto_list_lock);
+ return ret;
+}
+
+void sockets_destroy(struct cgroup *cgrp, struct cgroup_subsys *ss)
+{
+ struct proto *proto;
+ read_lock(&proto_list_lock);
+ list_for_each_entry_reverse(proto, &proto_list, node)
+  if (proto->destroy_cgroup)
+   proto->destroy_cgroup(proto, cgrp, ss);
+ read_unlock(&proto_list_lock);
+}
+
 /*
  * Each address family might have different locking rules, so we have
  * one slock key per address family:
@@ -2262,9 +2298,6 @@ void sk_common_release(struct sock *sk)
 }
 EXPORT_SYMBOL(sk_common_release);

-static DEFINE_RWLOCK(proto_list_lock);
-static LIST_HEAD(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/tcp.c b/net/ipv4/tcp.c
index ca82b90..bbd3989 100644
--- a/net/ipv4/tcp.c
+++ b/net/ipv4/tcp.c
@@ -290,13 +290,6 @@ EXPORT_SYMBOL(sysctl_tcp_mem);
 EXPORT_SYMBOL(sysctl_tcp_rmem);
 EXPORT_SYMBOL(sysctl_tcp_wmem);

-atomic_long_t tcp_memory_allocated; /* Current allocated memory. */
-
-/*
- * Current number of TCP sockets.
- */
-struct percpu_counter tcp_sockets_allocated;
-
 /*
  * TCP splice context
  */
@@ -306,46 +299,49 @@ struct tcp_splice_state {
```

```
  unsigned int flags;
 };

-/*
- * 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.
- */
+/* Current number of TCP sockets. */
+struct percpu_counter tcp_sockets_allocated;
+atomic_long_t tcp_memory_allocated; /* Current allocated memory. */
 int tcp_memory_pressure __read_mostly;

-int *memory_pressure_tcp(struct mem_cgroup *memcg)
+int *memory_pressure_tcp_nocg(struct mem_cgroup *memcg)
 {
  return &tcp_memory_pressure;
 }
-EXPORT_SYMBOL(memory_pressure_tcp);
+EXPORT_SYMBOL(memory_pressure_tcp_nocg);

-struct percpu_counter *sockets_allocated_tcp(struct mem_cgroup *memcg)
+struct percpu_counter *sockets_allocated_tcp_nocg(struct mem_cgroup *memcg)
 {
  return &tcp_sockets_allocated;
 }
-EXPORT_SYMBOL(sockets_allocated_tcp);
+EXPORT_SYMBOL(sockets_allocated_tcp_nocg);

-void tcp_enter_memory_pressure(struct sock *sk)
+/*
+ * 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_nocg(struct sock *sk)
 {
  if (!tcp_memory_pressure) {
   NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPMEMORYPRESSURES);
   tcp_memory_pressure = 1;
  }
 }
-EXPORT_SYMBOL(tcp_enter_memory_pressure);
+EXPORT_SYMBOL(tcp_enter_memory_pressure_nocg);

-long *tcp_sysctl_mem(struct mem_cgroup *memcg)
```

```
+long *tcp_sysctl_mem_nocg(struct mem_cgroup *memcg)
 {
  return sysctl_tcp_mem;
 }
-EXPORT_SYMBOL(tcp_sysctl_mem);
+EXPORT_SYMBOL(tcp_sysctl_mem_nocg);

-atomic_long_t *memory_allocated_tcp(struct mem_cgroup *memcg)
+atomic_long_t *memory_allocated_tcp_nocg(struct mem_cgroup *memcg)
 {
  return &tcp_memory_allocated;
 }
-EXPORT_SYMBOL(memory_allocated_tcp);
+EXPORT_SYMBOL(memory_allocated_tcp_nocg);

 /* Convert seconds to retransmits based on initial and max timeout */
 static u8 secs_to_retrans(int seconds, int timeout, int rto_max)
@@ -3247,7 +3243,9 @@ void __init tcp_init(void)

  BUILD_BUG_ON(sizeof(struct tcp_skb_cb) > sizeof(skb->cb));

+#ifndef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
  percpu_counter_init(&tcp_sockets_allocated, 0);
+#endif
  percpu_counter_init(&tcp_orphan_count, 0);
  tcp_hashinfo.bind_bucket_cachep =
   kmem_cache_create("tcp_bind_bucket",
diff --git a/net/ipv4/tcp_ipv4.c b/net/ipv4/tcp_ipv4.c
index 3227f6a..0377af3 100644
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -2604,12 +2604,22 @@ struct proto tcp_prot = {
  .hash   = inet_hash,
  .unhash   = inet_unhash,
  .get_port  = inet_csk_get_port,
+ .orphan_count  = &tcp_orphan_count,
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ .init_cgroup  = tcp_init_cgroup,
+ .destroy_cgroup  = tcp_destroy_cgroup,
  .enter_memory_pressure = tcp_enter_memory_pressure,
  .memory_pressure = memory_pressure_tcp,
  .sockets_allocated = sockets_allocated_tcp,
- .orphan_count  = &tcp_orphan_count,
  .memory_allocated = memory_allocated_tcp,
  .prot_mem  = tcp_sysctl_mem,
+#else
+ .enter_memory_pressure = tcp_enter_memory_pressure_nocg,
+ .memory_pressure = memory_pressure_tcp_nocg,
```

```
+ .sockets_allocated = sockets_allocated_tcp_nocg,
+ .memory_allocated = memory_allocated_tcp_nocg,
+ .prot_mem  = tcp_sysctl_mem_nocg,
+#endif
  .sysctl_wmem  = sysctl_tcp_wmem,
  .sysctl_rmem  = sysctl_tcp_rmem,
  .max_header  = MAX_TCP_HEADER,
diff --git a/net/ipv6/tcp_ipv6.c b/net/ipv6/tcp_ipv6.c
index 962fb56..5597807 100644
--- a/net/ipv6/tcp_ipv6.c
+++ b/net/ipv6/tcp_ipv6.c
@@ -2195,12 +2195,20 @@ struct proto tcpv6_prot = {
  .hash   = tcp_v6_hash,
  .unhash   = inet_unhash,
  .get_port  = inet_csk_get_port,
+ .orphan_count  = &tcp_orphan_count,
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
  .enter_memory_pressure = tcp_enter_memory_pressure,
  .sockets_allocated = sockets_allocated_tcp,
  .memory_allocated = memory_allocated_tcp,
  .memory_pressure = memory_pressure_tcp,
- .orphan_count  = &tcp_orphan_count,
  .prot_mem  = tcp_sysctl_mem,
+#else
+ .enter_memory_pressure = tcp_enter_memory_pressure_nocg,
+ .sockets_allocated = sockets_allocated_tcp_nocg,
+ .memory_allocated = memory_allocated_tcp_nocg,
+ .memory_pressure = memory_pressure_tcp_nocg,
+ .prot_mem  = tcp_sysctl_mem_nocg,
+#endif
  .sysctl_wmem  = sysctl_tcp_wmem,
  .sysctl_rmem  = sysctl_tcp_rmem,
  .max_header  = MAX_TCP_HEADER,
--
1.7.6
```