
Subject: [PATCH v8 4/9] tcp memory pressure controls
Posted by [Glauber Costa](#) on Mon, 05 Dec 2011 21:34:58 GMT
[View Forum Message](#) <> [Reply to Message](#)

This patch introduces memory pressure controls for the tcp protocol. It uses the generic socket memory pressure code introduced in earlier patches, and fills in the necessary data in cg_proto struct.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Eric W. Biederman <ebiederm@xmission.com>

```
Documentation/cgroups/memory.txt | 2 +
include/linux/memcontrol.h       | 1 +
include/net/sock.h               | 2 +
include/net/tcp_memcontrol.h     | 17 ++++++++
mm/memcontrol.c                  | 40 ++++++++
net/core/sock.c                  | 43 ++++++++
net/ipv4/Makefile                 | 1 +
net/ipv4/tcp_ipv4.c              | 9 +++-
net/ipv4/tcp_memcontrol.c        | 74 ++++++++
net/ipv6/tcp_ipv6.c              | 5 +++
10 files changed, 189 insertions(+), 5 deletions(-)
create mode 100644 include/net/tcp_memcontrol.h
create mode 100644 net/ipv4/tcp_memcontrol.c
```

```
diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index 23a8dc5..687dea5 100644
--- a/Documentation/cgroups/memory.txt
+++ b/Documentation/cgroups/memory.txt
@@ -293,6 +293,8 @@ to trigger slab reclaim when those limits are reached.
 thresholds. The Memory Controller allows them to be controlled individually
 per cgroup, instead of globally.
```

+* tcp memory pressure: sockets memory pressure for the tcp protocol.

+

3. User Interface

0. Configuration

```
diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index f15021b..1513994 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -86,6 +86,7 @@ extern struct mem_cgroup *mem_cgroup_from_task(struct task_struct *p);
extern struct mem_cgroup *try_get_mem_cgroup_from_mm(struct mm_struct *mm);

extern struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg);
```

```

+extern struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont);

static inline
int mm_match_cgroup(const struct mm_struct *mm, const struct mem_cgroup *cgroup)
diff --git a/include/net/sock.h b/include/net/sock.h
index b8a63f8..910cb0b 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -64,6 +64,8 @@
#include <net/dst.h>
#include <net/checksum.h>

+int mem_cgroup_sockets_init(struct cgroup *cgrp, struct cgroup_subsys *ss);
+void mem_cgroup_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
diff --git a/include/net/tcp_memcontrol.h b/include/net/tcp_memcontrol.h
new file mode 100644
index 0000000..5f5e158
--- /dev/null
+++ b/include/net/tcp_memcontrol.h
@@ -0,0 +1,17 @@
+#ifndef _TCP_MEMCG_H
+#define _TCP_MEMCG_H
+
+struct tcp_memcontrol {
+ struct cg_proto cg_proto;
+ /* 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;
+};
+
+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);
+#endif /* _TCP_MEMCG_H */
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index beedff3..b121127 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -50,6 +50,8 @@
#include <linux/cpu.h>
#include <linux/oom.h>
#include "internal.h"

```

```

#include <net/sock.h>
#include <net/tcp_memcontrol.h>

#include <asm/uaccess.h>

@@ -295,6 +297,10 @@ struct mem_cgroup {
    */
    struct mem_cgroup_stat_cpu nocpu_base;
    spinlock_t pcp_counter_lock;
+
#ifdef CONFIG_INET
+ struct tcp_memcontrol tcp_mem;
#endif
};

/* Stuffs for move charges at task migration. */
@@ -384,6 +390,7 @@ static void mem_cgroup_put(struct mem_cgroup *memcg);
#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
#ifdef CONFIG_INET
#include <net/sock.h>
#include <net/ip.h>

static bool mem_cgroup_is_root(struct mem_cgroup *memcg);
void sock_update_memcg(struct sock *sk)
@@ -418,6 +425,15 @@ void sock_release_memcg(struct sock *sk)
    mem_cgroup_put(memcg);
}
}
+
+struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
+{
+ if (!memcg || mem_cgroup_is_root(memcg))
+ return NULL;
+
+ return &memcg->tcp_mem.cg_proto;
+}
+EXPORT_SYMBOL(tcp_proto_cgroup);
#endif /* CONFIG_INET */
#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

@@ -800,7 +816,7 @@ static void memcg_check_events(struct mem_cgroup *memcg, struct
page *page)
    preempt_enable();
}

-static struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont)
+struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont)
{

```

```

return container_of(cgroup_subsys_state(cont,
    mem_cgroup_subsys_id), struct mem_cgroup,
@@ -4730,14 +4746,34 @@ static int register_kmem_files(struct cgroup *cont, struct
cgroup_subsys *ss)

```

```

ret = cgroup_add_files(cont, ss, kmem_cgroup_files,
    ARRAY_SIZE(kmem_cgroup_files));
+
+ /*
+  * Part of this would be better living in a separate allocation
+  * function, leaving us with just the cgroup tree population work.
+  * We, however, depend on state such as network's proto_list that
+  * is only initialized after cgroup creation. I found the less
+  * cumbersome way to deal with it to defer it all to populate time
+  */
+ if (!ret)
+ ret = mem_cgroup_sockets_init(cont, ss);
return ret;
};

```

```

+static void kmem_cgroup_destroy(struct cgroup_subsys *ss,
+ struct cgroup *cont)
+{
+ mem_cgroup_sockets_destroy(cont, ss);
+}
+
+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 struct cftype mem_cgroup_files[] = {
@@ -5096,6 +5132,8 @@ static void mem_cgroup_destroy(struct cgroup_subsys *ss,
{
struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);

+ kmem_cgroup_destroy(ss, cont);
+
mem_cgroup_put(memcg);
}

```

```

diff --git a/net/core/sock.c b/net/core/sock.c

```

index 39e5d01..3d6e370 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);

+

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM

+int mem_cgroup_sockets_init(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(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(cgrp, ss);

+ read_unlock(&proto_list_lock);

+ return ret;

+}

+

+void mem_cgroup_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(cgrp, ss);

+ read_unlock(&proto_list_lock);

+}

+#endif

+

/*

```

* Each address family might have different locking rules, so we have
* one slock key per address family:
@@ -2256,9 +2296,6 @@ 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/Makefile b/net/ipv4/Makefile
index f2dc69c..dc67a99 100644
--- a/net/ipv4/Makefile
+++ b/net/ipv4/Makefile
@@ -47,6 +47,7 @@ obj-$(CONFIG_TCP_CONG_SCALABLE) += tcp_scalable.o
obj-$(CONFIG_TCP_CONG_LP) += tcp_lp.o
obj-$(CONFIG_TCP_CONG_YEAH) += tcp_yeah.o
obj-$(CONFIG_TCP_CONG_ILLINOIS) += tcp_illinois.o
+obj-$(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) += tcp_memcontrol.o
obj-$(CONFIG_NETLABEL) += cipso_ipv4.o

obj-$(CONFIG_XFRM) += xfrm4_policy.o xfrm4_state.o xfrm4_input.o \
diff --git a/net/ipv4/tcp_ipv4.c b/net/ipv4/tcp_ipv4.c
index d1f4bf8..f70923e 100644
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -73,6 +73,7 @@
#include <net/xfrm.h>
#include <net/netdma.h>
#include <net/secure_seq.h>
+#include <net/tcp_memcontrol.h>

#include <linux/inet.h>
#include <linux/ipv6.h>
@@ -1915,6 +1916,7 @@ static int tcp_v4_init_sock(struct sock *sk)
sk->sk_rcvbuf = sysctl_tcp_rmem[1];

local_bh_disable();
+ sock_update_memcg(sk);
sk_sockets_allocated_inc(sk);
local_bh_enable();

@@ -1972,6 +1974,7 @@ void tcp_v4_destroy_sock(struct sock *sk)
}

sk_sockets_allocated_dec(sk);

```

```

+ sock_release_memcg(sk);
}
EXPORT_SYMBOL(tcp_v4_destroy_sock);

@@ -2632,10 +2635,14 @@ struct proto tcp_prot = {
    .compat_setsockopt = compat_tcp_setsockopt,
    .compat_getsockopt = compat_tcp_getsockopt,
#ifdef
#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ .init_cgroup = tcp_init_cgroup,
+ .destroy_cgroup = tcp_destroy_cgroup,
+ .proto_cgroup = tcp_proto_cgroup,
#endif
};
EXPORT_SYMBOL(tcp_prot);

-
static int __net_init tcp_sk_init(struct net *net)
{
    return inet_ctl_sock_create(&net->ipv4.tcp_sock,
diff --git a/net/ipv4/tcp_memcontrol.c b/net/ipv4/tcp_memcontrol.c
new file mode 100644
index 0000000..4a68d2c
--- /dev/null
+++ b/net/ipv4/tcp_memcontrol.c
@@ -0,0 +1,74 @@
#include <net/tcp.h>
#include <net/tcp_memcontrol.h>
#include <net/sock.h>
#include <linux/memcontrol.h>
#include <linux/module.h>
+
+static inline struct tcp_memcontrol *tcp_from_cgproto(struct cg_proto *cg_proto)
+{
+ return container_of(cg_proto, struct tcp_memcontrol, cg_proto);
+}
+
+static void memcg_tcp_enter_memory_pressure(struct sock *sk)
+{
+ if (!sk->sk_cgrp->memory_pressure)
+ *sk->sk_cgrp->memory_pressure = 1;
+}
+EXPORT_SYMBOL(memcg_tcp_enter_memory_pressure);
+
+int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)
+{
+ /*
+ * The root cgroup does not use res_counters, but rather,

```

```

+ * rely on the data already collected by the network
+ * subsystem
+ */
+ struct res_counter *res_parent = NULL;
+ struct cg_proto *cg_proto, *parent_cg;
+ struct tcp_memcontrol *tcp;
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
+ struct mem_cgroup *parent = parent_mem_cgroup(memcg);
+
+ cg_proto = tcp_prot.proto_cgroup(memcg);
+ if (!cg_proto)
+ return 0;
+
+ tcp = tcp_from_cgproto(cg_proto);
+
+ tcp->tcp_prot_mem[0] = sysctl_tcp_mem[0];
+ tcp->tcp_prot_mem[1] = sysctl_tcp_mem[1];
+ tcp->tcp_prot_mem[2] = sysctl_tcp_mem[2];
+ tcp->tcp_memory_pressure = 0;
+
+ parent_cg = tcp_prot.proto_cgroup(parent);
+ if (parent_cg)
+ res_parent = parent_cg->memory_allocated;
+
+ res_counter_init(&tcp->tcp_memory_allocated, res_parent);
+ percpu_counter_init(&tcp->tcp_sockets_allocated, 0);
+
+ cg_proto->enter_memory_pressure = memcg_tcp_enter_memory_pressure;
+ cg_proto->memory_pressure = &tcp->tcp_memory_pressure;
+ cg_proto->sysctl_mem = tcp->tcp_prot_mem;
+ cg_proto->memory_allocated = &tcp->tcp_memory_allocated;
+ cg_proto->sockets_allocated = &tcp->tcp_sockets_allocated;
+ cg_proto->memcg = memcg;
+
+ 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);
+ struct cg_proto *cg_proto;
+ struct tcp_memcontrol *tcp;
+
+ cg_proto = tcp_prot.proto_cgroup(memcg);
+ if (!cg_proto)
+ return;
+
+

```



```

+ tcp = tcp_from_cgproto(cg_proto);
+ percpu_counter_destroy(&tcp->tcp_sockets_allocated);
+}
+EXPORT_SYMBOL(tcp_destroy_cgroup);
diff --git a/net/ipv6/tcp_ipv6.c b/net/ipv6/tcp_ipv6.c
index e666768..820ae82 100644
--- a/net/ipv6/tcp_ipv6.c
+++ b/net/ipv6/tcp_ipv6.c
@@ -62,6 +62,7 @@
#include <net/netdma.h>
#include <net/inet_common.h>
#include <net/secure_seq.h>
#include <net/tcp_memcontrol.h>

#include <asm/uaccess.h>

@@ -1995,6 +1996,7 @@ static int tcp_v6_init_sock(struct sock *sk)
    sk->sk_rcvbuf = sysctl_tcp_rmem[1];

    local_bh_disable();
+ sock_update_memcg(sk);
    sk_sockets_allocated_inc(sk);
    local_bh_enable();

@@ -2228,6 +2230,9 @@ struct proto tcpv6_prot = {
    .compat_setsockopt = compat_tcp_setsockopt,
    .compat_getsockopt = compat_tcp_getsockopt,
    #endif
+ #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ .proto_cgroup = tcp_proto_cgroup,
+ #endif
    };

    static const struct inet6_protocol tcpv6_protocol = {
--
1.7.6.4

```
