
Subject: [PATCH v5 6/8] tcp buffer limitation: per-cgroup limit
Posted by [Glauber Costa](#) on Tue, 04 Oct 2011 12:17:58 GMT
[View Forum Message](#) <> [Reply to Message](#)

This patch uses the "tcp_max_mem" field of the kmem_cgroup to effectively control the amount of kernel memory pinned by a cgroup.

We have to make sure that none of the memory pressure thresholds specified in the namespace are bigger than the current cgroup.

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/linux/memcontrol.h       | 10 +++++
include/net/tcp.h                 | 1 +
mm/memcontrol.c                  | 80 ++++++-----
net/ipv4/sysctl_net_ipv4.c       | 20 ++++++++
5 files changed, 106 insertions(+), 6 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/linux/memcontrol.h b/include/linux/memcontrol.h
index 50af61d..6191ba1 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -395,6 +395,9 @@ @@ 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);
+
+unsigned long tcp_max_memory(struct mem_cgroup *cg);
+void tcp_prot_mem(struct mem_cgroup *cg, long val, int idx);
#else
/* memcontrol includes sockets.h, that includes memcontrol.h ... */
static inline void memcg_sock_mem_alloc(struct mem_cgroup *memcg,
```

```

@@ -420,6 +423,13 @@ static inline void sock_update_memcg(struct sock *sk)
static inline void sock_release_memcg(struct sock *sk)
{
}
+static inline unsigned long tcp_max_memory(struct mem_cgroup *cg)
+{
+ return 0;
+}
+static inline void tcp_prot_mem(struct mem_cgroup *cg, long val, int idx)
+{
+}
#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
#endif /* CONFIG_INET */
#endif /* _LINUX_MEMCONTROL_H */
diff --git a/include/net/tcp.h b/include/net/tcp.h
index 04fedf7..716a42e 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -256,6 +256,7 @@ extern int sysctl_tcp_thin_dupack;
struct mem_cgroup;
struct tcp_memcontrol {
/* per-cgroup tcp memory pressure knobs */
+ int tcp_max_memory;
atomic_long_t tcp_memory_allocated;
struct percpu_counter tcp_sockets_allocated;
/* those two are read-mostly, leave them at the end */
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 39e575e..6fb14bb 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -304,6 +304,13 @@ struct mem_cgroup {
};

static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg);
+
+static inline bool mem_cgroup_is_root(struct mem_cgroup *memcg)
+{
+ return (memcg == root_mem_cgroup);
+}
+
+static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *mem);
/* Writing them here to avoid exposing memcg's inner layout */
#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
#ifdef CONFIG_INET
@@ -424,6 +431,48 @@ struct percpu_counter *sockets_allocated_tcp(struct mem_cgroup
*memcg)
}
EXPORT_SYMBOL(sockets_allocated_tcp);

```

```

+static int tcp_write_limit(struct cgroup *cgrp, struct cftype *cft, u64 val)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
+ struct mem_cgroup *parent = parent_mem_cgroup(memcg);
+ struct net *net = current->nsproxy->net_ns;
+ int i;
+
+ if (parent && parent->use_hierarchy)
+ return -EINVAL;
+
+ /*
+ * We can't allow more memory than our parents. Since this
+ * will be tested for all calls, by induction, there is no need
+ * to test any parent other than our own
+ */
+ val >>= PAGE_SHIFT;
+ if (parent && (val > parent->tcp.tcp_max_memory))
+ val = parent->tcp.tcp_max_memory;
+
+ memcg->tcp.tcp_max_memory = val;
+
+ for (i = 0; i < 3; i++)
+ memcg->tcp.tcp_prot_mem[i] = min_t(long, val,
+ net->ipv4.sysctl_tcp_mem[i]);
+
+ return 0;
+}
+
+static u64 tcp_read_limit(struct cgroup *cgrp, struct cftype *cft)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
+ return memcg->tcp.tcp_max_memory << PAGE_SHIFT;
+}
+
+static struct cftype tcp_files[] = {
+ {
+ .name = "kmem.tcp.limit_in_bytes",
+ .write_u64 = tcp_write_limit,
+ .read_u64 = tcp_read_limit,
+ },
+};
+
+static void tcp_create_cgroup(struct mem_cgroup *cg, struct cgroup_subsys *ss)
+{
+ cg->tcp.memory_pressure = 0;
+@@ -435,6 +484,7 @@ int tcp_init_cgroup(struct proto *prot, struct cgroup *cgrp,
+ struct cgroup_subsys *ss)

```

```

{
    struct mem_cgroup *cg = mem_cgroup_from_cont(cgrp);
+ struct mem_cgroup *parent = parent_mem_cgroup(cg);
    struct net *net = current->nsproxy->net_ns;
    /*
     * We need to initialize it at populate, not create time.
@@ -445,6 +495,20 @@ int tcp_init_cgroup(struct proto *prot, struct cgroup *cgrp,
    cg->tcp.tcp_prot_mem[1] = net->ipv4.sysctl_tcp_mem[1];
    cg->tcp.tcp_prot_mem[2] = net->ipv4.sysctl_tcp_mem[2];

+
+ if (parent && parent->use_hierarchy)
+   cg->tcp.tcp_max_memory = parent->tcp.tcp_max_memory;
+ else {
+   unsigned long limit;
+   limit = nr_free_buffer_pages() / 8;
+   limit = max(limit, 128UL);
+   cg->tcp.tcp_max_memory = limit * 2;
+ }
+
+
+ if (!mem_cgroup_is_root(cg))
+   return cgroup_add_files(cgrp, ss, tcp_files,
+   ARRAY_SIZE(tcp_files));
    return 0;
}
EXPORT_SYMBOL(tcp_init_cgroup);
@@ -457,6 +521,16 @@ void tcp_destroy_cgroup(struct proto *prot, struct cgroup *cgrp,
    percpu_counter_destroy(&cg->tcp.tcp_sockets_allocated);
}
EXPORT_SYMBOL(tcp_destroy_cgroup);
+
+unsigned long tcp_max_memory(struct mem_cgroup *memcg)
+{
+   return memcg->tcp.tcp_max_memory;
+}
+
+void tcp_prot_mem(struct mem_cgroup *memcg, long val, int idx)
+{
+   memcg->tcp.tcp_prot_mem[idx] = val;
+}
#endif /* CONFIG_INET */
#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

@@ -1035,12 +1109,6 @@ static struct mem_cgroup *mem_cgroup_get_next(struct
mem_cgroup *iter,
#define for_each_mem_cgroup_all(iter) \
    for_each_mem_cgroup_tree_cond(iter, NULL, true)

```

```

-
-static inline bool mem_cgroup_is_root(struct mem_cgroup *mem)
-{
- return (mem == root_mem_cgroup);
-}
-
void mem_cgroup_count_vm_event(struct mm_struct *mm, enum vm_event_item idx)
{
    struct mem_cgroup *mem;
diff --git a/net/ipv4/sysctl_net_ipv4.c b/net/ipv4/sysctl_net_ipv4.c
index bbd67ab..cdc35f6 100644
--- a/net/ipv4/sysctl_net_ipv4.c
+++ b/net/ipv4/sysctl_net_ipv4.c
@@ -14,6 +14,7 @@
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/nsproxy.h>
+#include <linux/memcontrol.h>
#include <linux/swap.h>
#include <net/snmp.h>
#include <net/icmp.h>
@@ -182,6 +183,10 @@ 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
+ int i;
+ struct mem_cgroup *cg;
+#endif

    ctl_table tmp = {
        .data = &vec,
@@ -198,6 +203,21 @@ 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);
+ for (i = 0; i < 3; i++)
+ if (vec[i] > tcp_max_memory(cg)) {
+ rcu_read_unlock();
+ return -EINVAL;
+ }
+
+ 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];
--
```

1.7.6
