Subject: [PATCH v7 00/10] Request for Inclusion: per-cgroup tcp memory pressure
Posted by Glauber Costa on Tue, 29 Nov 2011 23:56:51 GMT
View Forum Message <> Reply to Message

Hi,

This patchset implements per-cgroup tcp memory pressure controls. It did not change
significantly since last submission: rather, it just merges the comments Kame had.
Most of them are style-related and/or Documentation, but there are two real bugs he
managed to spot (thanks)

Please let me know if there is anything else I should address.

Glauber Costa (10):
  Basic kernel memory functionality for the Memory Controller
  foundations of per-cgroup memory pressure controlling.
  socket: initial cgroup code.
  tcp memory pressure controls
  per-netns ipv4 sysctl_tcp_mem
  tcp buffer limitation: per-cgroup limit
  Display current tcp memory allocation in kmem cgroup
  Display current tcp failcnt in kmem cgroup
  Display maximum tcp memory allocation in kmem cgroup
  Disable task moving when using kernel memory accounting

 Documentation/cgroups/memory.txt | 52 +++++++-
 include/linux/memcontrol.h | 19 +++
 include/net/netns/ipv4.h | 1 +
 include/net/sock.h | 298 ++-
 include/net/tcp.h | 4 +-
 include/net/tcp_memcontrol.h | 19 +++
 init/Kconfig | 14 ++
 mm/memcontrol.c | 202 +++++++++++++++++++++++++++++--
 net/core/sock.c | 120 +++++++++++++-----
 net/ipv4/Makefile | 1 +
 net/ipv4/af_inet.c | 2 +
 net/ipv4/proc.c | 7 +-
 net/ipv4/sysctl_net_ipv4.c | 65 ++++++++-
 net/ipv4/tcp.c | 11 +-
 net/ipv4/tcp_input.c | 12 +-
 net/ipv4/tcp_ipv4.c | 13 +-
 net/ipv4/tcp_memcontrol.c | 268 ++++++++++++++++++++++++++++++++++++++
 net/ipv4/tcp_output.c | 2 +-
 net/ipv4/tcp_timer.c | 2 +-
 net/ipv6/af_inet6.c | 2 +
 net/ipv6/tcp_ipv6.c | 7 +-
 21 files changed, 1036 insertions(+), 85 deletions(-)

create mode 100644 include/net/tcp_memcontrol.h
create mode 100644 net/ipv4/tcp_memcontrol.c

--
1.7.6.4

Subject: [PATCH v7 01/10] Basic kernel memory functionality for the Memory Controller
Posted by Glauber Costa on Tue, 29 Nov 2011 23:56:52 GMT
View Forum Message <> Reply to Message

This patch lays down the foundation for the kernel memory component
of the Memory Controller.

As of today, I am only laying down the following files:

 * memory.independent_kmem_limit
 * memory.kmem.limit_in_bytes (currently ignored)
 * memory.kmem.usage_in_bytes (always zero)

Signed-off-by: Glauber Costa <glommer@parallels.com>
Reviewed-by: Kirill A. Shutemov <kirill@shutemov.name>
CC: Paul Menage <paul@paulmenage.org>
CC: Greg Thelen <gthelen@google.com>

 Documentation/cgroups/memory.txt | 46 ++++++++++++++++++-
 init/Kconfig | 14 +++++
 mm/memcontrol.c | 103 ++--
 3 files changed, 156 insertions(+), 7 deletions(-)

diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index cc0ebc5..764781c 100644
--- a/Documentation/cgroups/memory.txt
+++ b/Documentation/cgroups/memory.txt
@@ -44,8 +44,9 @@ Features:
  - oom-killer disable knob and oom-notifier
  - Root cgroup has no limit controls.

- Kernel memory and Hugepages are not under control yet. We just manage
- pages on LRU. To add more controls, we have to take care of performance.
+ Hugepages is not under control yet. We just manage pages on LRU. To add more
+ controls, we have to take care of performance. Kernel memory support is work
+ in progress, and the current version provides basically functionality.

 Brief summary of control files.

@@ -56,8 +57,11 @@ Brief summary of control files.

```
      (See 5.5 for details)
  memory.memsw.usage_in_bytes  # show current res_counter usage for memory+Swap
      (See 5.5 for details)
+ memory.kmem.usage_in_bytes  # show current res_counter usage for kmem only.
+     (See 2.7 for details)
  memory.limit_in_bytes   # set/show limit of memory usage
  memory.memsw.limit_in_bytes  # set/show limit of memory+Swap usage
+ memory.kmem.limit_in_bytes  # if allowed, set/show limit of kernel memory
  memory.failcnt    # show the number of memory usage hits limits
  memory.memsw.failcnt   # show the number of memory+Swap hits limits
  memory.max_usage_in_bytes  # show max memory usage recorded
@@ -72,6 +76,9 @@ Brief summary of control files.
  memory.oom_control   # set/show oom controls.
  memory.numa_stat   # show the number of memory usage per numa node

+ memory.independent_kmem_limit  # select whether or not kernel memory limits are
+       independent of user limits
+
 1. History

 The memory controller has a long history. A request for comments for the memory
@@ -255,6 +262,41 @@ When oom event notifier is registered, event will be delivered.
   per-zone-per-cgroup LRU (cgroup's private LRU) is just guarded by
   zone->lru_lock, it has no lock of its own.

+2.7 Kernel Memory Extension (CONFIG_CGROUP_MEM_RES_CTLR_KMEM)
+
+With the Kernel memory extension, the Memory Controller is able to limit
+the amount of kernel memory used by the system. Kernel memory is fundamentally
+different than user memory, since it can't be swapped out, which makes it
+possible to DoS the system by consuming too much of this precious resource.
+
+Some kernel memory resources may be accounted and limited separately from the
+main "kmem" resource. For instance, a slab cache that is considered important
+enough to be limited separately may have its own knobs.
+
+Kernel memory limits are not imposed for the root cgroup. Usage for the root
+cgroup may or may not be accounted.
+
+Memory limits as specified by the standard Memory Controller may or may not
+take kernel memory into consideration. This is achieved through the file
+memory.independent_kmem_limit. A Value different than 0 will allow for kernel
+memory to be controlled separately.
+
+When kernel memory limits are not independent, the limit values set in
+memory.kmem files are ignored.
+
+Currently no soft limit is implemented for kernel memory. It is future work
```

```
+to trigger slab reclaim when those limits are reached.
+
+CAUTION: As of this writing, the kmem extension may prevent tasks from moving
+among cgroups. If a task has kmem accounting in a cgroup, the task cannot be
+moved until the kmem resource is released. Also, until the resource is fully
+released, the cgroup cannot be destroyed. So, please consider your use cases
+and set kmem extension config option carefully.
+
+2.7.1 Current Kernel Memory resources accounted
+
+None
+
 3. User Interface

 0. Configuration
diff --git a/init/Kconfig b/init/Kconfig
index 43298f9..56558a2 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -689,6 +689,20 @@ config CGROUP_MEM_RES_CTLR_SWAP_ENABLED
    For those who want to have the feature enabled by default should
    select this option (if, for some reason, they need to disable it
    then swapaccount=0 does the trick).
+config CGROUP_MEM_RES_CTLR_KMEM
+ bool "Memory Resource Controller Kernel Memory accounting (EXPERIMENTAL)"
+ depends on CGROUP_MEM_RES_CTLR && EXPERIMENTAL
+ default n
+ help
+   The Kernel Memory extension for Memory Resource Controller can limit
+   the amount of memory used by kernel objects in the system. Those are
+   fundamentally different from the entities handled by the standard
+   Memory Controller, which are page-based, and can be swapped. Users of
+   the kmem extension can use it to guarantee that no group of processes
+   will ever exhaust kernel resources alone.
+
+   WARNING: The current experimental implementation does not allow a
+   task to move among different cgroups with a kmem resource being held.

 config CGROUP_PERF
   bool "Enable perf_event per-cpu per-container group (cgroup) monitoring"
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 6aff93c..3becb24 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -227,6 +227,10 @@ struct mem_cgroup {
   */
  struct res_counter memsw;
  /*
```

```
+  * the counter to account for kmem usage.
+  */
+ struct res_counter kmem;
+ /*
   * Per cgroup active and inactive list, similar to the
   * per zone LRU lists.
   */
@@ -277,6 +281,11 @@ struct mem_cgroup {
   */
  unsigned long  move_charge_at_immigrate;
  /*
+  * Should kernel memory limits be stabilished independently
+  * from user memory ?
+  */
+ int  kmem_independent_accounting;
+ /*
   * percpu counter.
   */
  struct mem_cgroup_stat_cpu *stat;
@@ -344,9 +353,14 @@ enum charge_type {
 };

 /* for encoding cft->private value on file */
-#define _MEM   (0)
-#define _MEMSWAP  (1)
-#define _OOM_TYPE  (2)
+
+enum mem_type {
+ _MEM = 0,
+ _MEMSWAP,
+ _OOM_TYPE,
+ _KMEM,
+};
+
 #define MEMFILE_PRIVATE(x, val) (((x) << 16) | (val))
 #define MEMFILE_TYPE(val) (((val) >> 16) & 0xffff)
 #define MEMFILE_ATTR(val) ((val) & 0xffff)
@@ -3848,10 +3862,17 @@ static inline u64 mem_cgroup_usage(struct mem_cgroup *memcg,
bool swap)
  u64 val;

  if (!mem_cgroup_is_root(memcg)) {
+  val = 0;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+  if (!memcg->kmem_independent_accounting)
+   val = res_counter_read_u64(&memcg->kmem, RES_USAGE);
+#endif
   if (!swap)
```

```
-   return res_counter_read_u64(&memcg->res, RES_USAGE);
+   val += res_counter_read_u64(&memcg->res, RES_USAGE);
    else
-   return res_counter_read_u64(&memcg->memsw, RES_USAGE);
+   val += res_counter_read_u64(&memcg->memsw, RES_USAGE);
+
+  return val;
   }

   val = mem_cgroup_recursive_stat(memcg, MEM_CGROUP_STAT_CACHE);
@@ -3884,6 +3905,11 @@ static u64 mem_cgroup_read(struct cgroup *cont, struct cftype *cft)
    else
     val = res_counter_read_u64(&memcg->memsw, name);
    break;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ case _KMEM:
+  val = res_counter_read_u64(&memcg->kmem, name);
+  break;
+#endif
   default:
    BUG();
    break;
@@ -4612,6 +4638,67 @@ static int mem_control_numa_stat_open(struct inode *unused, struct
file *file)
 }
 #endif /* CONFIG_NUMA */

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+static u64 kmem_limit_independent_read(struct cgroup *cgroup, struct cftype *cft)
+{
+ return mem_cgroup_from_cont(cgroup)->kmem_independent_accounting;
+}
+
+static int kmem_limit_independent_write(struct cgroup *cgroup, struct cftype *cft,
+     u64 val)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgroup);
+ struct mem_cgroup *parent = parent_mem_cgroup(memcg);
+
+ val = !!val;
+
+ if (parent && parent->use_hierarchy &&
+     (val != parent->kmem_independent_accounting))
+  return -EINVAL;
+ /*
+  * TODO: We need to handle the case in which we are doing
+  * independent kmem accounting as authorized by our parent,
+  * but then our parent changes its parameter.
```

```
+  */
+ cgroup_lock();
+ memcg->kmem_independent_accounting = val;
+ cgroup_unlock();
+ return 0;
+}
+static struct cftype kmem_cgroup_files[] = {
+ {
+  .name = "independent_kmem_limit",
+  .read_u64 = kmem_limit_independent_read,
+  .write_u64 = kmem_limit_independent_write,
+ },
+ {
+  .name = "kmem.usage_in_bytes",
+  .private = MEMFILE_PRIVATE(_KMEM, RES_USAGE),
+  .read_u64 = mem_cgroup_read,
+ },
+ {
+  .name = "kmem.limit_in_bytes",
+  .private = MEMFILE_PRIVATE(_KMEM, RES_LIMIT),
+  .read_u64 = mem_cgroup_read,
+ },
+};
+
+static int register_kmem_files(struct cgroup *cont, struct cgroup_subsys *ss)
+{
+ int ret = 0;
+
+ ret = cgroup_add_files(cont, ss, kmem_cgroup_files,
+          ARRAY_SIZE(kmem_cgroup_files));
+ return ret;
+};
+
+#else
+static int register_kmem_files(struct cgroup *cont, struct cgroup_subsys *ss)
+{
+ return 0;
+}
+#endif
+
 static struct cftype mem_cgroup_files[] = {
  {
   .name = "usage_in_bytes",
@@ -4925,6 +5012,7 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
  if (parent && parent->use_hierarchy) {
   res_counter_init(&memcg->res, &parent->res);
   res_counter_init(&memcg->memsw, &parent->memsw);
+  res_counter_init(&memcg->kmem, &parent->kmem);
```

```
   /*
    * We increment refcnt of the parent to ensure that we can
    * safely access it on res_counter_charge/uncharge.
@@ -4935,6 +5023,7 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
  } else {
   res_counter_init(&memcg->res, NULL);
   res_counter_init(&memcg->memsw, NULL);
+  res_counter_init(&memcg->kmem, NULL);
  }
  memcg->last_scanned_child = 0;
  memcg->last_scanned_node = MAX_NUMNODES;
@@ -4978,6 +5067,10 @@ static int mem_cgroup_populate(struct cgroup_subsys *ss,

  if (!ret)
   ret = register_memsw_files(cont, ss);
+
+ if (!ret)
+  ret = register_kmem_files(cont, ss);
+
  return ret;
 }

--
1.7.6.4
```

Subject: [PATCH v7 02/10] foundations of per-cgroup memory pressure controlling.
Posted by Glauber Costa on Tue, 29 Nov 2011 23:56:53 GMT
View Forum Message <> Reply to Message

This patch replaces all uses of struct sock fields' memory_pressure,
memory_allocated, sockets_allocated, and sysctl_mem to acessor
macros. Those macros can either receive a socket argument, or a mem_cgroup
argument, depending on the context they live in.

Since we're only doing a macro wrapping here, no performance impact at all is
expected in the case where we don't have cgroups disabled.

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>
CC: Eric Dumazet <eric.dumazet@gmail.com>

```
 include/net/sock.h    |   96 ++++++++++++++++++++++++++++++++++++++++++++++++-
 include/net/tcp.h     |    3 +-
 net/core/sock.c       |   62 ++++++++++++++++++++++++++-----------
 net/ipv4/proc.c       |    7 ++--
```

```
 net/ipv4/tcp_input.c  |   12 +++---
 net/ipv4/tcp_ipv4.c   |    4 +-
 net/ipv4/tcp_output.c |    2 +-
 net/ipv4/tcp_timer.c  |    2 +-
 net/ipv6/tcp_ipv6.c   |    2 +-
 9 files changed, 149 insertions(+), 41 deletions(-)

diff --git a/include/net/sock.h b/include/net/sock.h
index abb6e0f..a71447b 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -53,6 +53,7 @@
 #include <linux/security.h>
 #include <linux/slab.h>
 #include <linux/uaccess.h>
+#include <linux/memcontrol.h>

 #include <linux/filter.h>
 #include <linux/rculist_nulls.h>
@@ -863,6 +864,99 @@ static inline void sk_refcnt_debug_release(const struct sock *sk)
 #define sk_refcnt_debug_release(sk) do { } while (0)
 #endif /* SOCK_REFCNT_DEBUG */

+static inline bool sk_has_memory_pressure(const struct sock *sk)
+{
+ return sk->sk_prot->memory_pressure != NULL;
+}
+
+static inline bool sk_under_memory_pressure(const struct sock *sk)
+{
+ if (!sk->sk_prot->memory_pressure)
+  return false;
+ return !!*sk->sk_prot->memory_pressure;
+}
+
+static inline void sk_leave_memory_pressure(struct sock *sk)
+{
+ int *memory_pressure = sk->sk_prot->memory_pressure;
+
+ if (memory_pressure && *memory_pressure)
+  *memory_pressure = 0;
+}
+
+static inline void sk_enter_memory_pressure(struct sock *sk)
+{
+ if (sk->sk_prot->enter_memory_pressure)
+  sk->sk_prot->enter_memory_pressure(sk);
+}
```

```
+
+static inline long sk_prot_mem_limits(const struct sock *sk, int index)
+{
+ long *prot = sk->sk_prot->sysctl_mem;
+ return prot[index];
+}
+
+static inline long
+sk_memory_allocated(const struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+ return atomic_long_read(prot->memory_allocated);
+}
+
+static inline long
+sk_memory_allocated_add(struct sock *sk, int amt)
+{
+ struct proto *prot = sk->sk_prot;
+ return atomic_long_add_return(amt, prot->memory_allocated);
+}
+
+static inline void
+sk_memory_allocated_sub(struct sock *sk, int amt)
+{
+ struct proto *prot = sk->sk_prot;
+ atomic_long_sub(amt, prot->memory_allocated);
+}
+
+static inline void sk_sockets_allocated_dec(struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+ percpu_counter_dec(prot->sockets_allocated);
+}
+
+static inline void sk_sockets_allocated_inc(struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+ percpu_counter_inc(prot->sockets_allocated);
+}
+
+static inline int
+sk_sockets_allocated_read_positive(struct sock *sk)
+{
+ struct proto *prot = sk->sk_prot;
+
+ return percpu_counter_sum_positive(prot->sockets_allocated);
+}
+
```

```
+static inline int
+memcg_sockets_allocated_sum_positive(struct proto *prot, struct mem_cgroup *memcg)
+{
+ return percpu_counter_sum_positive(prot->sockets_allocated);
+}
+
+static inline long
+memcg_memory_allocated(struct proto *prot, struct mem_cgroup *memcg)
+{
+ return atomic_long_read(prot->memory_allocated);
+}
+
+static inline bool
+memcg_memory_pressure(struct proto *prot, struct mem_cgroup *memcg)
+{
+ if (!prot->memory_pressure)
+  return false;
+ return !!prot->memory_pressure;
+}
+

 #ifdef CONFIG_PROC_FS
 /* Called with local bh disabled */
@@ -1670,7 +1764,7 @@ static inline struct page *sk_stream_alloc_page(struct sock *sk)

  page = alloc_pages(sk->sk_allocation, 0);
  if (!page) {
-  sk->sk_prot->enter_memory_pressure(sk);
+  sk_enter_memory_pressure(sk);
   sk_stream_moderate_sndbuf(sk);
  }
  return page;
diff --git a/include/net/tcp.h b/include/net/tcp.h
index bb18c4d..f080e0b 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -44,6 +44,7 @@
 #include <net/dst.h>

 #include <linux/seq_file.h>
+#include <linux/memcontrol.h>

 extern struct inet_hashinfo tcp_hashinfo;

@@ -285,7 +286,7 @@ static inline bool tcp_too_many_orphans(struct sock *sk, int shift)
  }

  if (sk->sk_wmem_queued > SOCK_MIN_SNDBUF &&
```

```
-     atomic_long_read(&tcp_memory_allocated) > sysctl_tcp_mem[2])
+     sk_memory_allocated(sk) > sk_prot_mem_limits(sk, 2))
    return true;
   return false;
 }
diff --git a/net/core/sock.c b/net/core/sock.c
index 4ed7b1d..44213ab 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -1288,7 +1288,7 @@ struct sock *sk_clone(const struct sock *sk, const gfp_t priority)
    newsk->sk_wq = NULL;

    if (newsk->sk_prot->sockets_allocated)
-    percpu_counter_inc(newsk->sk_prot->sockets_allocated);
+    sk_sockets_allocated_inc(newsk);

    if (sock_flag(newsk, SOCK_TIMESTAMP) ||
        sock_flag(newsk, SOCK_TIMESTAMPING_RX_SOFTWARE))
@@ -1679,28 +1679,26 @@ int __sk_mem_schedule(struct sock *sk, int size, int kind)
  long allocated;

  sk->sk_forward_alloc += amt * SK_MEM_QUANTUM;
- allocated = atomic_long_add_return(amt, prot->memory_allocated);
+
+ allocated = sk_memory_allocated_add(sk, amt);

  /* Under limit. */
- if (allocated <= prot->sysctl_mem[0]) {
-  if (prot->memory_pressure && *prot->memory_pressure)
-   *prot->memory_pressure = 0;
-  return 1;
- }
+ if (allocated <= sk_prot_mem_limits(sk, 0))
+  sk_leave_memory_pressure(sk);

  /* Under pressure. */
- if (allocated > prot->sysctl_mem[1])
-  if (prot->enter_memory_pressure)
-   prot->enter_memory_pressure(sk);
+ if (allocated > sk_prot_mem_limits(sk, 1))
+  sk_enter_memory_pressure(sk);

  /* Over hard limit. */
- if (allocated > prot->sysctl_mem[2])
+ if (allocated > sk_prot_mem_limits(sk, 2))
    goto suppress_allocation;

  /* guarantee minimum buffer size under pressure */
```

```
  if (kind == SK_MEM_RECV) {
   if (atomic_read(&sk->sk_rmem_alloc) < prot->sysctl_rmem[0])
    return 1;
+
  } else { /* SK_MEM_SEND */
   if (sk->sk_type == SOCK_STREAM) {
    if (sk->sk_wmem_queued < prot->sysctl_wmem[0])
@@ -1710,13 +1708,13 @@ int __sk_mem_schedule(struct sock *sk, int size, int kind)
     return 1;
  }

- if (prot->memory_pressure) {
+ if (sk_has_memory_pressure(sk)) {
   int alloc;

-  if (!*prot->memory_pressure)
+  if (!sk_under_memory_pressure(sk))
    return 1;
-  alloc = percpu_counter_read_positive(prot->sockets_allocated);
-  if (prot->sysctl_mem[2] > alloc *
+  alloc = sk_sockets_allocated_read_positive(sk);
+  if (sk_prot_mem_limits(sk, 2) > alloc *
       sk_mem_pages(sk->sk_wmem_queued +
      atomic_read(&sk->sk_rmem_alloc) +
      sk->sk_forward_alloc))
@@ -1739,7 +1737,9 @@ suppress_allocation:

  /* Alas. Undo changes. */
  sk->sk_forward_alloc -= amt * SK_MEM_QUANTUM;
- atomic_long_sub(amt, prot->memory_allocated);
+
+ sk_memory_allocated_sub(sk, amt);
+
  return 0;
 }
 EXPORT_SYMBOL(__sk_mem_schedule);
@@ -1750,15 +1750,13 @@ EXPORT_SYMBOL(__sk_mem_schedule);
  */
 void __sk_mem_reclaim(struct sock *sk)
 {
- struct proto *prot = sk->sk_prot;
-
- atomic_long_sub(sk->sk_forward_alloc >> SK_MEM_QUANTUM_SHIFT,
-     prot->memory_allocated);
+ sk_memory_allocated_sub(sk,
+     sk->sk_forward_alloc >> SK_MEM_QUANTUM_SHIFT);
  sk->sk_forward_alloc &= SK_MEM_QUANTUM - 1;
```

```
- if (prot->memory_pressure && *prot->memory_pressure &&
-     (atomic_long_read(prot->memory_allocated) < prot->sysctl_mem[0]))
-  *prot->memory_pressure = 0;
+ if (sk_under_memory_pressure(sk) &&
+     (sk_memory_allocated(sk) < sk_prot_mem_limits(sk, 0)))
+  sk_leave_memory_pressure(sk);
 }
 EXPORT_SYMBOL(__sk_mem_reclaim);

@@ -2474,16 +2472,30 @@ static char proto_method_implemented(const void *method)
 {
  return method == NULL ? 'n' : 'y';
 }
+static long sock_prot_memory_allocated(struct proto *proto,
+           struct mem_cgroup *memcg)
+{
+ return proto->memory_allocated != NULL ? memcg_memory_allocated(proto, memcg): -1L;
+}
+
+static char *sock_prot_memory_pressure(struct proto *proto,
+           struct mem_cgroup *memcg)
+{
+ return proto->memory_pressure != NULL ?
+ memcg_memory_pressure(proto, memcg) ? "yes" : "no" : "NI";
+}

 static void proto_seq_printf(struct seq_file *seq, struct proto *proto)
 {
+ struct mem_cgroup *memcg = mem_cgroup_from_task(current);
+
  seq_printf(seq, "%-9s %4u %6d  %6ld   %-3s %6u   %-3s  %-10s "
    "%2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c
%2c\n",
      proto->name,
      proto->obj_size,
      sock_prot_inuse_get(seq_file_net(seq), proto),
-     proto->memory_allocated != NULL ? atomic_long_read(proto->memory_allocated) : -1L,
-     proto->memory_pressure != NULL ? *proto->memory_pressure ? "yes" : "no" : "NI",
+     sock_prot_memory_allocated(proto, memcg),
+     sock_prot_memory_pressure(proto, memcg),
      proto->max_header,
      proto->slab == NULL ? "no" : "yes",
      module_name(proto->owner),
diff --git a/net/ipv4/proc.c b/net/ipv4/proc.c
index 466ea8b..969172b 100644
--- a/net/ipv4/proc.c
+++ b/net/ipv4/proc.c
@@ -53,20 +53,21 @@ static int sockstat_seq_show(struct seq_file *seq, void *v)
```

```
 {
  struct net *net = seq->private;
  int orphans, sockets;
+ struct mem_cgroup *memcg = mem_cgroup_from_task(current);

  local_bh_disable();
  orphans = percpu_counter_sum_positive(&tcp_orphan_count);
- sockets = percpu_counter_sum_positive(&tcp_sockets_allocated);
+ sockets = memcg_sockets_allocated_sum_positive(&tcp_prot, memcg);
  local_bh_enable();

  socket_seq_show(seq);
  seq_printf(seq, "TCP: inuse %d orphan %d tw %d alloc %d mem %ld\n",
      sock_prot_inuse_get(net, &tcp_prot), orphans,
      tcp_death_row.tw_count, sockets,
-     atomic_long_read(&tcp_memory_allocated));
+     memcg_memory_allocated(&tcp_prot, memcg));
  seq_printf(seq, "UDP: inuse %d mem %ld\n",
      sock_prot_inuse_get(net, &udp_prot),
-     atomic_long_read(&udp_memory_allocated));
+     memcg_memory_allocated(&udp_prot, memcg));
  seq_printf(seq, "UDPLITE: inuse %d\n",
      sock_prot_inuse_get(net, &udplite_prot));
  seq_printf(seq, "RAW: inuse %d\n",
diff --git a/net/ipv4/tcp_input.c b/net/ipv4/tcp_input.c
index 52b5c2d..b64b5e8 100644
--- a/net/ipv4/tcp_input.c
+++ b/net/ipv4/tcp_input.c
@@ -322,7 +322,7 @@ static void tcp_grow_window(struct sock *sk, const struct sk_buff *skb)
  /* Check #1 */
  if (tp->rcv_ssthresh < tp->window_clamp &&
      (int)tp->rcv_ssthresh < tcp_space(sk) &&
-     !tcp_memory_pressure) {
+     !sk_under_memory_pressure(sk)) {
   int incr;

   /* Check #2. Increase window, if skb with such overhead
@@ -411,8 +411,8 @@ static void tcp_clamp_window(struct sock *sk)

  if (sk->sk_rcvbuf < sysctl_tcp_rmem[2] &&
      !(sk->sk_userlocks & SOCK_RCVBUF_LOCK) &&
-     !tcp_memory_pressure &&
-     atomic_long_read(&tcp_memory_allocated) < sysctl_tcp_mem[0]) {
+     !sk_under_memory_pressure(sk) &&
+     sk_memory_allocated(sk) < sk_prot_mem_limits(sk, 0)) {
   sk->sk_rcvbuf = min(atomic_read(&sk->sk_rmem_alloc),
       sysctl_tcp_rmem[2]);
  }
```

```
@@ -4864,7 +4864,7 @@ static int tcp_prune_queue(struct sock *sk)

  if (atomic_read(&sk->sk_rmem_alloc) >= sk->sk_rcvbuf)
   tcp_clamp_window(sk);
- else if (tcp_memory_pressure)
+ else if (sk_under_memory_pressure(sk))
   tp->rcv_ssthresh = min(tp->rcv_ssthresh, 4U * tp->advmss);

  tcp_collapse_ofo_queue(sk);
@@ -4930,11 +4930,11 @@ static int tcp_should_expand_sndbuf(const struct sock *sk)
   return 0;

  /* If we are under global TCP memory pressure, do not expand.  */
- if (tcp_memory_pressure)
+ if (sk_under_memory_pressure(sk))
   return 0;

  /* If we are under soft global TCP memory pressure, do not expand.  */
- if (atomic_long_read(&tcp_memory_allocated) >= sysctl_tcp_mem[0])
+ if (sk_memory_allocated(sk) >= sk_prot_mem_limits(sk, 0))
   return 0;

  /* If we filled the congestion window, do not expand.  */
diff --git a/net/ipv4/tcp_ipv4.c b/net/ipv4/tcp_ipv4.c
index a744315..d1f4bf8 100644
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -1915,7 +1915,7 @@ static int tcp_v4_init_sock(struct sock *sk)
  sk->sk_rcvbuf = sysctl_tcp_rmem[1];

  local_bh_disable();
- percpu_counter_inc(&tcp_sockets_allocated);
+ sk_sockets_allocated_inc(sk);
  local_bh_enable();

  return 0;
@@ -1971,7 +1971,7 @@ void tcp_v4_destroy_sock(struct sock *sk)
   tp->cookie_values = NULL;
  }

- percpu_counter_dec(&tcp_sockets_allocated);
+ sk_sockets_allocated_dec(sk);
 }
 EXPORT_SYMBOL(tcp_v4_destroy_sock);

diff --git a/net/ipv4/tcp_output.c b/net/ipv4/tcp_output.c
index 980b98f..b378490 100644
--- a/net/ipv4/tcp_output.c
```

```
+++ b/net/ipv4/tcp_output.c
@@ -1919,7 +1919,7 @@ u32 __tcp_select_window(struct sock *sk)
  if (free_space < (full_space >> 1)) {
   icsk->icsk_ack.quick = 0;

-  if (tcp_memory_pressure)
+  if (sk_under_memory_pressure(sk))
    tp->rcv_ssthresh = min(tp->rcv_ssthresh,
           4U * tp->advmss);

diff --git a/net/ipv4/tcp_timer.c b/net/ipv4/tcp_timer.c
index 2e0f0af..d6ddacb 100644
--- a/net/ipv4/tcp_timer.c
+++ b/net/ipv4/tcp_timer.c
@@ -261,7 +261,7 @@ static void tcp_delack_timer(unsigned long data)
  }

 out:
- if (tcp_memory_pressure)
+ if (sk_under_memory_pressure(sk))
   sk_mem_reclaim(sk);
 out_unlock:
  bh_unlock_sock(sk);
diff --git a/net/ipv6/tcp_ipv6.c b/net/ipv6/tcp_ipv6.c
index 36131d1..e666768 100644
--- a/net/ipv6/tcp_ipv6.c
+++ b/net/ipv6/tcp_ipv6.c
@@ -1995,7 +1995,7 @@ static int tcp_v6_init_sock(struct sock *sk)
  sk->sk_rcvbuf = sysctl_tcp_rmem[1];

  local_bh_disable();
- percpu_counter_inc(&tcp_sockets_allocated);
+ sk_sockets_allocated_inc(sk);
  local_bh_enable();

  return 0;
--
1.7.6.4
```

Subject: [PATCH v7 03/10] socket: initial cgroup code.
Posted by Glauber Costa on Tue, 29 Nov 2011 23:56:54 GMT
View Forum Message <> Reply to Message

The goal of this work is to move the memory pressure tcp
controls to a cgroup, instead of just relying on global
conditions.

To avoid excessive overhead in the network fast paths,
the code that accounts allocated memory to a cgroup is
hidden inside a static_branch(). This branch is patched out
until the first non-root cgroup is created. So when nobody
is using cgroups, even if it is mounted, no significant performance
penalty should be seen.

This patch handles the generic part of the code, and has nothing
tcp-specific.

Signed-off-by: Glauber Costa <glommer@parallels.com>
Acked-by: Kirill A. Shutemov<kirill@shutemov.name>
Reviewed-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujtsu.com>
CC: David S. Miller <davem@davemloft.net>
CC: Eric W. Biederman <ebiederm@xmission.com>
CC: Eric Dumazet <eric.dumazet@gmail.com>

 Documentation/cgroups/memory.txt | 4 +-
 include/linux/memcontrol.h | 16 +++
 include/net/sock.h | 210 ++++++++++++++++++++++++++++++++++++++-
 mm/memcontrol.c | 40 ++++++-
 net/core/sock.c | 24 +++--
 5 files changed, 274 insertions(+), 20 deletions(-)

diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index 764781c..3cf9d96 100644
--- a/Documentation/cgroups/memory.txt
+++ b/Documentation/cgroups/memory.txt
@@ -295,7 +295,9 @@ and set kmem extension config option carefully.

 2.7.1 Current Kernel Memory resources accounted

-None
+* sockets memory pressure: some sockets protocols have memory pressure
+thresholds. The Memory Controller allows them to be controlled individually
+per cgroup, instead of globally.

 3. User Interface

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index b87068a..60964c3 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -381,5 +381,21 @@ mem_cgroup_print_bad_page(struct page *page)
 }
 #endif

+#ifdef CONFIG_INET

```
+enum {
+ UNDER_LIMIT,
+ SOFT_LIMIT,
+ OVER_LIMIT,
+};
+
+struct sock;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+void sock_update_memcg(struct sock *sk);
+#else
+static inline void sock_update_memcg(struct sock *sk)
+{
+}
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+#endif /* CONFIG_INET */
 #endif /* _LINUX_MEMCONTROL_H */

diff --git a/include/net/sock.h b/include/net/sock.h
index a71447b..49f0912 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -54,6 +54,7 @@
 #include <linux/slab.h>
 #include <linux/uaccess.h>
 #include <linux/memcontrol.h>
+#include <linux/res_counter.h>

 #include <linux/filter.h>
 #include <linux/rculist_nulls.h>
@@ -168,6 +169,7 @@ struct sock_common {
  /* public: */
 };

+struct cg_proto;
 /**
   * struct sock - network layer representation of sockets
   * @__sk_common: shared layout with inet_timewait_sock
@@ -228,6 +230,7 @@ struct sock_common {
   * @sk_security: used by security modules
   * @sk_mark: generic packet mark
   * @sk_classid: this socket's cgroup classid
+  * @sk_cgrp: this socket's cgroup-specific proto data
   * @sk_write_pending: a write to stream socket waits to start
   * @sk_state_change: callback to indicate change in the state of the sock
   * @sk_data_ready: callback to indicate there is data to be processed
@@ -339,6 +342,7 @@ struct sock {
 #endif
  __u32   sk_mark;
```

```
  u32   sk_classid;
+ struct cg_proto  *sk_cgrp;
  void   (*sk_state_change)(struct sock *sk);
  void   (*sk_data_ready)(struct sock *sk, int bytes);
  void   (*sk_write_space)(struct sock *sk);
@@ -834,6 +838,28 @@ struct proto {
 #ifdef SOCK_REFCNT_DEBUG
  atomic_t  socks;
 #endif
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ /*
+  * 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 cgroup *cgrp,
+            struct cgroup_subsys *ss);
+ void   (*destroy_cgroup)(struct cgroup *cgrp,
+        struct cgroup_subsys *ss);
+ struct cg_proto  *(*proto_cgroup)(struct mem_cgroup *memcg);
+#endif
+};
+
+struct cg_proto {
+ void   (*enter_memory_pressure)(struct sock *sk);
+ struct res_counter *memory_allocated; /* Current allocated memory. */
+ struct percpu_counter *sockets_allocated; /* Current number of sockets. */
+ int   *memory_pressure;
+ long   *sysctl_mem;
+ struct cg_proto  *parent;
 };

 extern int proto_register(struct proto *prot, int alloc_slab);
@@ -864,6 +890,7 @@ static inline void sk_refcnt_debug_release(const struct sock *sk)
 #define sk_refcnt_debug_release(sk) do { } while (0)
 #endif /* SOCK_REFCNT_DEBUG */

+extern struct jump_label_key memcg_socket_limit_enabled;
 static inline bool sk_has_memory_pressure(const struct sock *sk)
 {
  return sk->sk_prot->memory_pressure != NULL;
@@ -873,6 +900,17 @@ static inline bool sk_under_memory_pressure(const struct sock *sk)
 {
  if (!sk->sk_prot->memory_pressure)
   return false;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
```

```
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;
+  return !!*cg_proto->memory_pressure;
+ } else
+nocgroup:
+#endif
+
  return !!*sk->sk_prot->memory_pressure;
 }

@@ -880,52 +918,176 @@ static inline void sk_leave_memory_pressure(struct sock *sk)
 {
  int *memory_pressure = sk->sk_prot->memory_pressure;

- if (memory_pressure && *memory_pressure)
+ if (!memory_pressure)
+  return;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;
+
+  for (; cg_proto; cg_proto = cg_proto->parent)
+   if (*cg_proto->memory_pressure)
+    *cg_proto->memory_pressure = 0;
+ }
+nocgroup:
+#endif
+ if (*memory_pressure)
   *memory_pressure = 0;
 }

 static inline void sk_enter_memory_pressure(struct sock *sk)
 {
- if (sk->sk_prot->enter_memory_pressure)
-  sk->sk_prot->enter_memory_pressure(sk);
+ if (!sk->sk_prot->enter_memory_pressure)
+  return;
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
```

```
+   goto nocgroup;
+
+  for (; cg_proto; cg_proto = cg_proto->parent)
+   cg_proto->enter_memory_pressure(sk);
+ }
+nocgroup:
+#endif
+ sk->sk_prot->enter_memory_pressure(sk);
 }

 static inline long sk_prot_mem_limits(const struct sock *sk, int index)
 {
  long *prot = sk->sk_prot->sysctl_mem;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+  if (!cg_proto) /* this handles the case with existing sockets */
+   goto nocgroup;
+
+  prot = cg_proto->sysctl_mem;
+ }
+nocgroup:
+#endif
  return prot[index];
 }

+static inline void memcg_memory_allocated_add(struct cg_proto *prot,
+           unsigned long amt,
+           int *parent_status)
+{
+ struct res_counter *fail;
+ int ret;
+
+ ret = res_counter_charge(prot->memory_allocated,
+     amt << PAGE_SHIFT, &fail);
+
+ if (ret < 0)
+  *parent_status = OVER_LIMIT;
+}
+
+static inline void memcg_memory_allocated_sub(struct cg_proto *prot,
+           unsigned long amt)
+{
+ res_counter_uncharge(prot->memory_allocated, amt << PAGE_SHIFT);
+}
+
+static inline u64 memcg_memory_allocated_read(struct cg_proto *prot)
+{
```

```
+ u64 ret;
+ ret = res_counter_read_u64(prot->memory_allocated, RES_USAGE);
+ return ret >> PAGE_SHIFT;
+}
+
 static inline long
 sk_memory_allocated(const struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+  if (!cg_proto) /* this handles the case with existing sockets */
+   goto nocgroup;
+
+  return memcg_memory_allocated_read(cg_proto);
+ }
+nocgroup:
+#endif
  return atomic_long_read(prot->memory_allocated);
 }

 static inline long
-sk_memory_allocated_add(struct sock *sk, int amt)
+sk_memory_allocated_add(struct sock *sk, int amt, int *parent_status)
 {
  struct proto *prot = sk->sk_prot;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;
+
+  memcg_memory_allocated_add(cg_proto, amt, parent_status);
+ }
+nocgroup:
+#endif
  return atomic_long_add_return(amt, prot->memory_allocated);
 }

 static inline void
-sk_memory_allocated_sub(struct sock *sk, int amt)
+sk_memory_allocated_sub(struct sock *sk, int amt, int parent_status)
 {
  struct proto *prot = sk->sk_prot;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
```

```
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;
+
+  /* Otherwise it was uncharged already */
+  if (parent_status != OVER_LIMIT)
+   memcg_memory_allocated_sub(cg_proto, amt);
+ }
+nocgroup:
+#endif
  atomic_long_sub(amt, prot->memory_allocated);
 }

 static inline void sk_sockets_allocated_dec(struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;
+
+  for (; cg_proto; cg_proto = cg_proto->parent)
+   percpu_counter_dec(cg_proto->sockets_allocated);
+ }
+nocgroup:
+#endif
  percpu_counter_dec(prot->sockets_allocated);
 }

 static inline void sk_sockets_allocated_inc(struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;
+
+  for (; cg_proto; cg_proto = cg_proto->parent)
+   percpu_counter_inc(cg_proto->sockets_allocated);
+ }
+nocgroup:
+#endif
  percpu_counter_inc(prot->sockets_allocated);
```

```
 }

@@ -933,19 +1095,57 @@ static inline int
 sk_sockets_allocated_read_positive(struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;

+  return percpu_counter_sum_positive(cg_proto->sockets_allocated);
+ }
+nocgroup:
+#endif
  return percpu_counter_sum_positive(prot->sockets_allocated);
 }

 static inline int
 memcg_sockets_allocated_sum_positive(struct proto *prot, struct mem_cgroup *memcg)
 {
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto;
+  if (!prot->proto_cgroup)
+   goto nocgroup;
+
+  cg_proto = prot->proto_cgroup(memcg);
+  if (!cg_proto)
+   goto nocgroup;
+
+  return percpu_counter_sum_positive(cg_proto->sockets_allocated);
+ }
+nocgroup:
+#endif
  return percpu_counter_sum_positive(prot->sockets_allocated);
 }

 static inline long
 memcg_memory_allocated(struct proto *prot, struct mem_cgroup *memcg)
 {
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto;
+  if (!prot->proto_cgroup)
+   goto nocgroup;
```

```
+
+  cg_proto = prot->proto_cgroup(memcg);
+  if (!cg_proto)
+   goto nocgroup;
+
+  return memcg_memory_allocated_read(cg_proto);
+ }
+nocgroup:
+#endif
  return atomic_long_read(prot->memory_allocated);
 }

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 3becb24..12a08bf 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -377,6 +377,40 @@ enum mem_type {
 #define MEM_CGROUP_RECLAIM_SOFT_BIT 0x2
 #define MEM_CGROUP_RECLAIM_SOFT  (1 << MEM_CGROUP_RECLAIM_SOFT_BIT)

+static inline bool mem_cgroup_is_root(struct mem_cgroup *memcg)
+{
+ return (memcg == root_mem_cgroup);
+}
+
+/* Writing them here to avoid exposing memcg's inner layout */
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+#ifdef CONFIG_INET
+#include <net/sock.h>
+
+void sock_update_memcg(struct sock *sk)
+{
+ /* right now a socket spends its whole life in the same cgroup */
+ if (sk->sk_cgrp) {
+  WARN_ON(1);
+  return;
+ }
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct mem_cgroup *memcg;
+
+  BUG_ON(!sk->sk_prot->proto_cgroup);
+
+  rcu_read_lock();
+  memcg = mem_cgroup_from_task(current);
+  if (!mem_cgroup_is_root(memcg))
+   sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
+  rcu_read_unlock();
+ }
```

```
+}
+
+#endif /* CONFIG_INET */
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+
+
 static void mem_cgroup_get(struct mem_cgroup *memcg);
 static void mem_cgroup_put(struct mem_cgroup *memcg);
 static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg);
@@ -878,12 +912,6 @@ 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 *memcg)
-{
- return (memcg == root_mem_cgroup);
-}
-
 void mem_cgroup_count_vm_event(struct mm_struct *mm, enum vm_event_item idx)
 {
  struct mem_cgroup *memcg;
diff --git a/net/core/sock.c b/net/core/sock.c
index 44213ab..e7e0808 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -111,6 +111,7 @@
 #include <linux/init.h>
 #include <linux/highmem.h>
 #include <linux/user_namespace.h>
+#include <linux/jump_label.h>

 #include <asm/uaccess.h>
 #include <asm/system.h>
@@ -141,6 +142,9 @@
 static struct lock_class_key af_family_keys[AF_MAX];
 static struct lock_class_key af_family_slock_keys[AF_MAX];

+struct jump_label_key memcg_socket_limit_enabled;
+EXPORT_SYMBOL(memcg_socket_limit_enabled);
+
 /*
  * Make lock validator output more readable. (we pre-construct these
  * strings build-time, so that runtime initialization of socket
@@ -1677,21 +1681,25 @@ int __sk_mem_schedule(struct sock *sk, int size, int kind)
  struct proto *prot = sk->sk_prot;
  int amt = sk_mem_pages(size);
```

```
  long allocated;
+ int parent_status = UNDER_LIMIT;

  sk->sk_forward_alloc += amt * SK_MEM_QUANTUM;

- allocated = sk_memory_allocated_add(sk, amt);
+ allocated = sk_memory_allocated_add(sk, amt, &parent_status);

  /* Under limit. */
- if (allocated <= sk_prot_mem_limits(sk, 0))
+ if (parent_status == UNDER_LIMIT &&
+   allocated <= sk_prot_mem_limits(sk, 0))
   sk_leave_memory_pressure(sk);

- /* Under pressure. */
- if (allocated > sk_prot_mem_limits(sk, 1))
+ /* Under pressure. (we or our parents) */
+ if ((parent_status > SOFT_LIMIT) ||
+   allocated > sk_prot_mem_limits(sk, 1))
   sk_enter_memory_pressure(sk);

- /* Over hard limit. */
- if (allocated > sk_prot_mem_limits(sk, 2))
+ /* Over hard limit (we or our parents) */
+ if ((parent_status == OVER_LIMIT) ||
+   (allocated > sk_prot_mem_limits(sk, 2)))
   goto suppress_allocation;

  /* guarantee minimum buffer size under pressure */
@@ -1738,7 +1746,7 @@ suppress_allocation:
  /* Alas. Undo changes. */
  sk->sk_forward_alloc -= amt * SK_MEM_QUANTUM;

- sk_memory_allocated_sub(sk, amt);
+ sk_memory_allocated_sub(sk, amt, parent_status);

  return 0;
 }
@@ -1751,7 +1759,7 @@ EXPORT_SYMBOL(__sk_mem_schedule);
 void __sk_mem_reclaim(struct sock *sk)
 {
  sk_memory_allocated_sub(sk,
-    sk->sk_forward_alloc >> SK_MEM_QUANTUM_SHIFT);
+    sk->sk_forward_alloc >> SK_MEM_QUANTUM_SHIFT, 0);
  sk->sk_forward_alloc &= SK_MEM_QUANTUM - 1;

  if (sk_under_memory_pressure(sk) &&
--
```

Subject: [PATCH v7 04/10] tcp memory pressure controls
Posted by Glauber Costa on Tue, 29 Nov 2011 23:56:55 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.fujtisu.com>
CC: Eric W. Biederman <ebiederm@xmission.com>

 Documentation/cgroups/memory.txt | 2 +
 include/linux/memcontrol.h | 3 ++
 include/net/sock.h | 2 +
 include/net/tcp_memcontrol.h | 17 +++++++++
 mm/memcontrol.c | 36 ++++++++++++++++++--
 net/core/sock.c | 42 ++++++++++++++++++++--
 net/ipv4/Makefile | 1 +
 net/ipv4/tcp_ipv4.c | 8 ++++-
 net/ipv4/tcp_memcontrol.c | 73 ++
 net/ipv6/tcp_ipv6.c | 4 ++
 10 files changed, 181 insertions(+), 7 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 3cf9d96..1e43da4 100644
--- a/Documentation/cgroups/memory.txt
+++ b/Documentation/cgroups/memory.txt
@@ -299,6 +299,8 @@ and set kmem extension config option carefully.
 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 60964c3..fa2482a 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h

```
@@ -85,6 +85,9 @@ extern struct mem_cgroup *try_get_mem_cgroup_from_page(struct page
*page);
 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 *mem_cgroup_from_cont(struct cgroup *cont);
+extern struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *mem);
+
 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 49f0912..2e94346 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 12a08bf..a31a278 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. */
@@ -386,6 +392,7 @@ static inline bool mem_cgroup_is_root(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)
 {
@@ -407,13 +414,21 @@ void sock_update_memcg(struct sock *sk)
  }
 }

+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 */
```

```
 static void mem_cgroup_get(struct mem_cgroup *memcg);
 static void mem_cgroup_put(struct mem_cgroup *memcg);
-static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg);
 static void drain_all_stock_async(struct mem_cgroup *memcg);

 static struct mem_cgroup_per_zone *
@@ -793,7 +808,7 @@ 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,
@@ -4717,14 +4732,27 @@ 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));
+
+ 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[] = {
@@ -4958,7 +4986,7 @@ static void mem_cgroup_put(struct mem_cgroup *memcg)
 /*
  * Returns the parent mem_cgroup in memcgroup hierarchy with hierarchy enabled.
```

```
 */
-static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg)
+struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *memcg)
 {
  if (!memcg->res.parent)
   return NULL;
@@ -5082,6 +5110,8 @@ 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 e7e0808..7309fc3 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -135,6 +135,45 @@
 #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 +2295,6 @@ 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..7fa08c5 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>
@@ -1918,6 +1919,7 @@ static int tcp_v4_init_sock(struct sock *sk)
  sk_sockets_allocated_inc(sk);
  local_bh_enable();

+ sock_update_memcg(sk);
  return 0;
 }

@@ -2632,10 +2634,14 @@ struct proto tcp_prot = {
  .compat_setsockopt = compat_tcp_setsockopt,
  .compat_getsockopt = compat_tcp_getsockopt,
 #endif
+#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..261e22c
--- /dev/null
+++ b/net/ipv4/tcp_memcontrol.c
@@ -0,0 +1,73 @@
+#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;
+ 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);
+ cg_proto->parent = tcp_prot.proto_cgroup(parent);
+
+ 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;
+
+ if (cg_proto->parent)
+  res_parent = cg_proto->parent->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;
+
+ 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..ca8e8a6 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>

@@ -2228,6 +2229,9 @@ 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
```

Subject: [PATCH v7 05/10] per-netns ipv4 sysctl_tcp_mem
Posted by Glauber Costa on Tue, 29 Nov 2011 23:56:56 GMT
View Forum Message <> Reply to Message

This patch allows each namespace to independently set up
its levels for tcp memory pressure thresholds. This patch
alone does not buy much: we need to make this values
per group of process somehow. This is achieved in the
patches that follows in this patchset.

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

```
CC: David S. Miller <davem@davemloft.net>
CC: Eric W. Biederman <ebiederm@xmission.com>
---
 include/net/netns/ipv4.h   |    1 +
 include/net/tcp.h          |    1 -
 net/ipv4/af_inet.c         |    2 +
 net/ipv4/sysctl_net_ipv4.c |   51 ++++++++++++++++++++++++++++++++++++------
 net/ipv4/tcp.c             |   11 +-------
 net/ipv4/tcp_ipv4.c        |    1 -
 net/ipv4/tcp_memcontrol.c  |    9 +++++--
 net/ipv6/af_inet6.c        |    2 +
 net/ipv6/tcp_ipv6.c        |    1 -
 9 files changed, 57 insertions(+), 22 deletions(-)

diff --git a/include/net/netns/ipv4.h b/include/net/netns/ipv4.h
index d786b4f..bbd023a 100644
--- a/include/net/netns/ipv4.h
+++ b/include/net/netns/ipv4.h
@@ -55,6 +55,7 @@ struct netns_ipv4 {
  int current_rt_cache_rebuild_count;

  unsigned int sysctl_ping_group_range[2];
+ long sysctl_tcp_mem[3];

  atomic_t rt_genid;
  atomic_t dev_addr_genid;
diff --git a/include/net/tcp.h b/include/net/tcp.h
index f080e0b..61c7e76 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
@@ -230,7 +230,6 @@ extern int sysctl_tcp_fack;
 extern int sysctl_tcp_reordering;
 extern int sysctl_tcp_ecn;
 extern int sysctl_tcp_dsack;
-extern long sysctl_tcp_mem[3];
 extern int sysctl_tcp_wmem[3];
 extern int sysctl_tcp_rmem[3];
 extern int sysctl_tcp_app_win;
diff --git a/net/ipv4/af_inet.c b/net/ipv4/af_inet.c
index 1b5096a..a8bbcff 100644
--- a/net/ipv4/af_inet.c
+++ b/net/ipv4/af_inet.c
@@ -1671,6 +1671,8 @@ static int __init inet_init(void)
  ip_static_sysctl_init();
 #endif

+ tcp_prot.sysctl_mem = init_net.ipv4.sysctl_tcp_mem;
+
```

```
  /*
   * Add all the base protocols.
   */
diff --git a/net/ipv4/sysctl_net_ipv4.c b/net/ipv4/sysctl_net_ipv4.c
index 69fd720..bbd67ab 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/swap.h>
 #include <net/snmp.h>
 #include <net/icmp.h>
 #include <net/ip.h>
@@ -174,6 +175,36 @@ static int proc_allowed_congestion_control(ctl_table *ctl,
  return ret;
 }

+static int ipv4_tcp_mem(ctl_table *ctl, int write,
+       void __user *buffer, size_t *lenp,
+       loff_t *ppos)
+{
+ int ret;
+ unsigned long vec[3];
+ struct net *net = current->nsproxy->net_ns;
+
+ ctl_table tmp = {
+  .data = &vec,
+  .maxlen = sizeof(vec),
+  .mode = ctl->mode,
+ };
+
+ if (!write) {
+  ctl->data = &net->ipv4.sysctl_tcp_mem;
+  return proc_doulongvec_minmax(ctl, write, buffer, lenp, ppos);
+ }
+
+ ret = proc_doulongvec_minmax(&tmp, write, buffer, lenp, ppos);
+ if (ret)
+  return ret;
+
+ net->ipv4.sysctl_tcp_mem[0] = vec[0];
+ net->ipv4.sysctl_tcp_mem[1] = vec[1];
+ net->ipv4.sysctl_tcp_mem[2] = vec[2];
+
+ return 0;
+}
```

```
+
 static struct ctl_table ipv4_table[] = {
  {
   .procname = "tcp_timestamps",
@@ -433,13 +464,6 @@ static struct ctl_table ipv4_table[] = {
   .proc_handler = proc_dointvec
  },
  {
-  .procname = "tcp_mem",
-  .data  = &sysctl_tcp_mem,
-  .maxlen  = sizeof(sysctl_tcp_mem),
-  .mode  = 0644,
-  .proc_handler = proc_doulongvec_minmax
- },
- {
   .procname = "tcp_wmem",
   .data  = &sysctl_tcp_wmem,
   .maxlen  = sizeof(sysctl_tcp_wmem),
@@ -721,6 +745,12 @@ static struct ctl_table ipv4_net_table[] = {
   .mode  = 0644,
   .proc_handler = ipv4_ping_group_range,
  },
+ {
+  .procname = "tcp_mem",
+  .maxlen  = sizeof(init_net.ipv4.sysctl_tcp_mem),
+  .mode  = 0644,
+  .proc_handler = ipv4_tcp_mem,
+ },
  { }
 };

@@ -734,6 +764,7 @@ EXPORT_SYMBOL_GPL(net_ipv4_ctl_path);
 static __net_init int ipv4_sysctl_init_net(struct net *net)
 {
  struct ctl_table *table;
+ unsigned long limit;

  table = ipv4_net_table;
  if (!net_eq(net, &init_net)) {
@@ -769,6 +800,12 @@ static __net_init int ipv4_sysctl_init_net(struct net *net)

  net->ipv4.sysctl_rt_cache_rebuild_count = 4;

+ limit = nr_free_buffer_pages() / 8;
+ limit = max(limit, 128UL);
+ net->ipv4.sysctl_tcp_mem[0] = limit / 4 * 3;
+ net->ipv4.sysctl_tcp_mem[1] = limit;
+ net->ipv4.sysctl_tcp_mem[2] = net->ipv4.sysctl_tcp_mem[0] * 2;
```

```
+
  net->ipv4.ipv4_hdr = register_net_sysctl_table(net,
    net_ipv4_ctl_path, table);
  if (net->ipv4.ipv4_hdr == NULL)
diff --git a/net/ipv4/tcp.c b/net/ipv4/tcp.c
index 34f5db1..5f618d1 100644
--- a/net/ipv4/tcp.c
+++ b/net/ipv4/tcp.c
@@ -282,11 +282,9 @@ int sysctl_tcp_fin_timeout __read_mostly = TCP_FIN_TIMEOUT;
 struct percpu_counter tcp_orphan_count;
 EXPORT_SYMBOL_GPL(tcp_orphan_count);

-long sysctl_tcp_mem[3] __read_mostly;
 int sysctl_tcp_wmem[3] __read_mostly;
 int sysctl_tcp_rmem[3] __read_mostly;

-EXPORT_SYMBOL(sysctl_tcp_mem);
 EXPORT_SYMBOL(sysctl_tcp_rmem);
 EXPORT_SYMBOL(sysctl_tcp_wmem);

@@ -3272,14 +3270,9 @@ void __init tcp_init(void)
  sysctl_tcp_max_orphans = cnt / 2;
  sysctl_max_syn_backlog = max(128, cnt / 256);

- limit = nr_free_buffer_pages() / 8;
- limit = max(limit, 128UL);
- sysctl_tcp_mem[0] = limit / 4 * 3;
- sysctl_tcp_mem[1] = limit;
- sysctl_tcp_mem[2] = sysctl_tcp_mem[0] * 2;
-
  /* Set per-socket limits to no more than 1/128 the pressure threshold */
- limit = ((unsigned long)sysctl_tcp_mem[1]) << (PAGE_SHIFT - 7);
+ limit = ((unsigned long)init_net.ipv4.sysctl_tcp_mem[1])
+  << (PAGE_SHIFT - 7);
  max_share = min(4UL*1024*1024, limit);

  sysctl_tcp_wmem[0] = SK_MEM_QUANTUM;
diff --git a/net/ipv4/tcp_ipv4.c b/net/ipv4/tcp_ipv4.c
index 7fa08c5..559f1ff 100644
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -2620,7 +2620,6 @@ struct proto tcp_prot = {
  .orphan_count  = &tcp_orphan_count,
  .memory_allocated = &tcp_memory_allocated,
  .memory_pressure = &tcp_memory_pressure,
- .sysctl_mem  = sysctl_tcp_mem,
  .sysctl_wmem  = sysctl_tcp_wmem,
  .sysctl_rmem  = sysctl_tcp_rmem,
```

```
  .max_header  = MAX_TCP_HEADER,
diff --git a/net/ipv4/tcp_memcontrol.c b/net/ipv4/tcp_memcontrol.c
index 261e22c..6dd6528 100644
--- a/net/ipv4/tcp_memcontrol.c
+++ b/net/ipv4/tcp_memcontrol.c
@@ -1,6 +1,8 @@
 #include <net/tcp.h>
 #include <net/tcp_memcontrol.h>
 #include <net/sock.h>
+#include <net/ip.h>
+#include <linux/nsproxy.h>
 #include <linux/memcontrol.h>
 #include <linux/module.h>

@@ -28,6 +30,7 @@ int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)
  struct tcp_memcontrol *tcp;
  struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
  struct mem_cgroup *parent = parent_mem_cgroup(memcg);
+ struct net *net = current->nsproxy->net_ns;

  cg_proto = tcp_prot.proto_cgroup(memcg);
  if (!cg_proto)
@@ -36,9 +39,9 @@ int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)
  tcp = tcp_from_cgproto(cg_proto);
  cg_proto->parent = tcp_prot.proto_cgroup(parent);

- 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_prot_mem[0] = net->ipv4.sysctl_tcp_mem[0];
+ tcp->tcp_prot_mem[1] = net->ipv4.sysctl_tcp_mem[1];
+ tcp->tcp_prot_mem[2] = net->ipv4.sysctl_tcp_mem[2];
  tcp->tcp_memory_pressure = 0;

  if (cg_proto->parent)
diff --git a/net/ipv6/af_inet6.c b/net/ipv6/af_inet6.c
index d27c797..49b2145 100644
--- a/net/ipv6/af_inet6.c
+++ b/net/ipv6/af_inet6.c
@@ -1115,6 +1115,8 @@ static int __init inet6_init(void)
  if (err)
   goto static_sysctl_fail;
 #endif
+ tcpv6_prot.sysctl_mem = init_net.ipv4.sysctl_tcp_mem;
+
  /*
   * ipngwg API draft makes clear that the correct semantics
   * for TCP and UDP is to consider one TCP and UDP instance
```

```
diff --git a/net/ipv6/tcp_ipv6.c b/net/ipv6/tcp_ipv6.c
index ca8e8a6..4973ccf 100644
--- a/net/ipv6/tcp_ipv6.c
+++ b/net/ipv6/tcp_ipv6.c
@@ -2215,7 +2215,6 @@ struct proto tcpv6_prot = {
  .memory_allocated = &tcp_memory_allocated,
  .memory_pressure = &tcp_memory_pressure,
  .orphan_count  = &tcp_orphan_count,
- .sysctl_mem  = sysctl_tcp_mem,
  .sysctl_wmem  = sysctl_tcp_wmem,
  .sysctl_rmem  = sysctl_tcp_rmem,
  .max_header  = MAX_TCP_HEADER,
--
1.7.6.4
```

Subject: [PATCH v7 06/10] tcp buffer limitation: per-cgroup limit
Posted by Glauber Costa on Tue, 29 Nov 2011 23:56:57 GMT
View Forum Message <> Reply to Message

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

This value is ignored in the root cgroup, and in all others,
caps the value specified by the admin in the net namespaces'
view of tcp_sysctl_mem.

If namespaces are being used, the admin is allowed to set a
value bigger than cgroup's maximum, the same way it is allowed
to set pretty much unlimited values in a real box.

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/net/tcp_memcontrol.h     |    2 +
 net/ipv4/sysctl_net_ipv4.c       |   14 ++++
 net/ipv4/tcp_memcontrol.c        |  134 ++++++++++++++++++++++++++++++++++++++-
 4 files changed, 149 insertions(+), 2 deletions(-)

diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index 1e43da4..4d9ed1f 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/net/tcp_memcontrol.h b/include/net/tcp_memcontrol.h
index 5f5e158..3512082 100644
--- a/include/net/tcp_memcontrol.h
+++ b/include/net/tcp_memcontrol.h
@@ -14,4 +14,6 @@ struct tcp_memcontrol {
 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);
+unsigned long long tcp_max_memory(const struct mem_cgroup *memcg);
+void tcp_prot_mem(struct mem_cgroup *memcg, long val, int idx);
 #endif /* _TCP_MEMCG_H */
diff --git a/net/ipv4/sysctl_net_ipv4.c b/net/ipv4/sysctl_net_ipv4.c
index bbd67ab..fe9bf91 100644
--- a/net/ipv4/sysctl_net_ipv4.c
+++ b/net/ipv4/sysctl_net_ipv4.c
@@ -24,6 +24,7 @@
 #include <net/cipso_ipv4.h>
 #include <net/inet_frag.h>
 #include <net/ping.h>
+#include <net/tcp_memcontrol.h>

 static int zero;
 static int tcp_retr1_max = 255;
@@ -182,6 +183,9 @@ 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
+ struct mem_cgroup *memcg;
+#endif

  ctl_table tmp = {
   .data = &vec,
@@ -198,6 +202,16 @@ static int ipv4_tcp_mem(ctl_table *ctl, int write,
  if (ret)
   return ret;

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ rcu_read_lock();
+ memcg = mem_cgroup_from_task(current);
+
+ tcp_prot_mem(memcg, vec[0], 0);
```

```
+ tcp_prot_mem(memcg, vec[1], 1);
+ tcp_prot_mem(memcg, 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];
diff --git a/net/ipv4/tcp_memcontrol.c b/net/ipv4/tcp_memcontrol.c
index 6dd6528..3edcf5f 100644
--- a/net/ipv4/tcp_memcontrol.c
+++ b/net/ipv4/tcp_memcontrol.c
@@ -6,6 +6,19 @@
 #include <linux/memcontrol.h>
 #include <linux/module.h>

+static u64 tcp_cgroup_read(struct cgroup *cont, struct cftype *cft);
+static int tcp_cgroup_write(struct cgroup *cont, struct cftype *cft,
+       const char *buffer);
+
+static struct cftype tcp_files[] = {
+ {
+  .name = "kmem.tcp.limit_in_bytes",
+  .write_string = tcp_cgroup_write,
+  .read_u64 = tcp_cgroup_read,
+  .private = RES_LIMIT,
+ },
+};
+
 static inline struct tcp_memcontrol *tcp_from_cgproto(struct cg_proto *cg_proto)
 {
  return container_of(cg_proto, struct tcp_memcontrol, cg_proto);
@@ -34,7 +47,7 @@ int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)

  cg_proto = tcp_prot.proto_cgroup(memcg);
  if (!cg_proto)
-  return 0;
+  goto create_files;

  tcp = tcp_from_cgproto(cg_proto);
  cg_proto->parent = tcp_prot.proto_cgroup(parent);
@@ -56,7 +69,9 @@ int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)
  cg_proto->memory_allocated = &tcp->tcp_memory_allocated;
  cg_proto->sockets_allocated = &tcp->tcp_sockets_allocated;

- return 0;
+create_files:
+ return cgroup_add_files(cgrp, ss, tcp_files,
```

```
+    ARRAY_SIZE(tcp_files));
 }
 EXPORT_SYMBOL(tcp_init_cgroup);

@@ -65,6 +80,7 @@ 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;
+ u64 val;

  cg_proto = tcp_prot.proto_cgroup(memcg);
  if (!cg_proto)
@@ -72,5 +88,119 @@ void tcp_destroy_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)

  tcp = tcp_from_cgproto(cg_proto);
  percpu_counter_destroy(&tcp->tcp_sockets_allocated);
+
+ val = res_counter_read_u64(&tcp->tcp_memory_allocated, RES_USAGE);
+
+ if (val != RESOURCE_MAX)
+  jump_label_dec(&memcg_socket_limit_enabled);
 }
 EXPORT_SYMBOL(tcp_destroy_cgroup);
+
+static int tcp_update_limit(struct mem_cgroup *memcg, u64 val)
+{
+ struct net *net = current->nsproxy->net_ns;
+ struct tcp_memcontrol *tcp;
+ struct cg_proto *cg_proto;
+ u64 old_lim;
+ int i;
+ int ret;
+
+ cg_proto = tcp_prot.proto_cgroup(memcg);
+ if (!cg_proto)
+  return -EINVAL;
+
+ tcp = tcp_from_cgproto(cg_proto);
+
+ old_lim = res_counter_read_u64(&tcp->tcp_memory_allocated, RES_LIMIT);
+ ret = res_counter_set_limit(&tcp->tcp_memory_allocated, val);
+ if (ret)
+  return ret;
+
+ for (i = 0; i < 3; i++)
+  tcp->tcp_prot_mem[i] = min_t(long, val >> PAGE_SHIFT,
+          net->ipv4.sysctl_tcp_mem[i]);
+
```

```
+ if (val == RESOURCE_MAX)
+  jump_label_dec(&memcg_socket_limit_enabled);
+ else if (old_lim == RESOURCE_MAX)
+  jump_label_inc(&memcg_socket_limit_enabled);
+
+ return 0;
+}
+
+static int tcp_cgroup_write(struct cgroup *cont, struct cftype *cft,
+       const char *buffer)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);
+ unsigned long long val;
+ int ret = 0;
+
+ switch (cft->private) {
+ case RES_LIMIT:
+  /* see memcontrol.c */
+  ret = res_counter_memparse_write_strategy(buffer, &val);
+  if (ret)
+   break;
+  ret = tcp_update_limit(memcg, val);
+  break;
+ default:
+  ret = -EINVAL;
+  break;
+ }
+ return ret;
+}
+
+static u64 tcp_read_stat(struct mem_cgroup *memcg, int type, u64 default_val)
+{
+ struct tcp_memcontrol *tcp;
+ struct cg_proto *cg_proto;
+
+ cg_proto = tcp_prot.proto_cgroup(memcg);
+ if (!cg_proto)
+  return default_val;
+
+ tcp = tcp_from_cgproto(cg_proto);
+ return res_counter_read_u64(&tcp->tcp_memory_allocated, type);
+}
+
+static u64 tcp_cgroup_read(struct cgroup *cont, struct cftype *cft)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);
+ u64 val;
+
```

```
+ switch (cft->private) {
+ case RES_LIMIT:
+  val = tcp_read_stat(memcg, RES_LIMIT, RESOURCE_MAX);
+  break;
+ default:
+  BUG();
+ }
+ return val;
+}
+
+unsigned long long tcp_max_memory(const struct mem_cgroup *memcg)
+{
+ struct tcp_memcontrol *tcp;
+ struct cg_proto *cg_proto;
+
+ cg_proto = tcp_prot.proto_cgroup((struct mem_cgroup *)memcg);
+ if (!cg_proto)
+  return 0;
+
+ tcp = tcp_from_cgproto(cg_proto);
+ return res_counter_read_u64(&tcp->tcp_memory_allocated, RES_LIMIT);
+}
+
+void tcp_prot_mem(struct mem_cgroup *memcg, long val, int idx)
+{
+ struct tcp_memcontrol *tcp;
+ struct cg_proto *cg_proto;
+
+ cg_proto = tcp_prot.proto_cgroup(memcg);
+ if (!cg_proto)
+  return;
+
+ tcp = tcp_from_cgproto(cg_proto);
+
+ tcp->tcp_prot_mem[idx] = val;
+}
--
1.7.6.4
```

Subject: [PATCH v7 07/10] Display current tcp memory allocation in kmem cgroup
Posted by Glauber Costa on Tue, 29 Nov 2011 23:56:58 GMT
View Forum Message <> Reply to Message

This patch introduces kmem.tcp.usage_in_bytes file, living in the kmem_cgroup filesystem. It is a simple read-only file that displays the amount of kernel memory currently consumed by the 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 +
 net/ipv4/tcp_memcontrol.c        |   21 ++++++++++++++++++++
 2 files changed, 22 insertions(+), 0 deletions(-)

diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index 4d9ed1f..09fcc82 100644
--- a/Documentation/cgroups/memory.txt
+++ b/Documentation/cgroups/memory.txt
@@ -79,6 +79,7 @@ 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
+ memory.kmem.tcp.usage_in_bytes  # show current tcp buf memory allocation

 1. History

diff --git a/net/ipv4/tcp_memcontrol.c b/net/ipv4/tcp_memcontrol.c
index 3edcf5f..bc232fc 100644
--- a/net/ipv4/tcp_memcontrol.c
+++ b/net/ipv4/tcp_memcontrol.c
@@ -17,6 +17,11 @@ static struct cftype tcp_files[] = {
   .read_u64 = tcp_cgroup_read,
   .private = RES_LIMIT,
  },
+ {
+  .name = "kmem.tcp.usage_in_bytes",
+  .read_u64 = tcp_cgroup_read,
+  .private = RES_USAGE,
+ },
 };

 static inline struct tcp_memcontrol *tcp_from_cgproto(struct cg_proto *cg_proto)
@@ -163,6 +168,19 @@ static u64 tcp_read_stat(struct mem_cgroup *memcg, int type, u64
default_val)
  return res_counter_read_u64(&tcp->tcp_memory_allocated, type);
 }

+static u64 tcp_read_usage(struct mem_cgroup *memcg)
+{
+ struct tcp_memcontrol *tcp;
+ struct cg_proto *cg_proto;
+
+ cg_proto = tcp_prot.proto_cgroup(memcg);
```

```
+ if (!cg_proto)
+  return atomic_long_read(&tcp_memory_allocated) << PAGE_SHIFT;
+
+ tcp = tcp_from_cgproto(cg_proto);
+ return res_counter_read_u64(&tcp->tcp_memory_allocated, RES_USAGE);
+}
+
 static u64 tcp_cgroup_read(struct cgroup *cont, struct cftype *cft)
 {
  struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);
@@ -172,6 +190,9 @@ static u64 tcp_cgroup_read(struct cgroup *cont, struct cftype *cft)
  case RES_LIMIT:
   val = tcp_read_stat(memcg, RES_LIMIT, RESOURCE_MAX);
   break;
+ case RES_USAGE:
+  val = tcp_read_usage(memcg);
+  break;
  default:
   BUG();
  }
--
1.7.6.4
```

Subject: [PATCH v7 08/10] Display current tcp failcnt in kmem cgroup
Posted by Glauber Costa on Tue, 29 Nov 2011 23:56:59 GMT
View Forum Message <> Reply to Message

This patch introduces kmem.tcp.failcnt file, living in the
kmem_cgroup filesystem. Following the pattern in the other
memcg resources, this files keeps a counter of how many times
allocation failed due to limits being hit in this cgroup.
The root cgroup will always show a failcnt of 0.

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>
---
 net/ipv4/tcp_memcontrol.c | 31 +++++++++++++++++++++++++++++++
 1 files changed, 31 insertions(+), 0 deletions(-)

```
diff --git a/net/ipv4/tcp_memcontrol.c b/net/ipv4/tcp_memcontrol.c
index bc232fc..f7350fa1 100644
--- a/net/ipv4/tcp_memcontrol.c
+++ b/net/ipv4/tcp_memcontrol.c
@@ -9,6 +9,7 @@
 static u64 tcp_cgroup_read(struct cgroup *cont, struct cftype *cft);
```

```
 static int tcp_cgroup_write(struct cgroup *cont, struct cftype *cft,
        const char *buffer);
+static int tcp_cgroup_reset(struct cgroup *cont, unsigned int event);

 static struct cftype tcp_files[] = {
  {
@@ -22,6 +23,12 @@ static struct cftype tcp_files[] = {
    .read_u64 = tcp_cgroup_read,
    .private = RES_USAGE,
  },
+ {
+  .name = "kmem.tcp.failcnt",
+  .private = RES_FAILCNT,
+  .trigger = tcp_cgroup_reset,
+  .read_u64 = tcp_cgroup_read,
+ },
 };

 static inline struct tcp_memcontrol *tcp_from_cgproto(struct cg_proto *cg_proto)
@@ -193,12 +200,36 @@ static u64 tcp_cgroup_read(struct cgroup *cont, struct cftype *cft)
  case RES_USAGE:
   val = tcp_read_usage(memcg);
   break;
+ case RES_FAILCNT:
+  val = tcp_read_stat(memcg, RES_FAILCNT, 0);
+  break;
  default:
   BUG();
  }
  return val;
 }

+static int tcp_cgroup_reset(struct cgroup *cont, unsigned int event)
+{
+ struct mem_cgroup *memcg;
+ struct tcp_memcontrol *tcp;
+ struct cg_proto *cg_proto;
+
+ memcg = mem_cgroup_from_cont(cont);
+ cg_proto = tcp_prot.proto_cgroup(memcg);
+ if (!cg_proto)
+  return 0;
+ tcp = tcp_from_cgproto(cg_proto);
+
+ switch (event) {
+ case RES_FAILCNT:
+  res_counter_reset_failcnt(&tcp->tcp_memory_allocated);
+  break;
```

```
+ }
+
+ return 0;
+}
+
 unsigned long long tcp_max_memory(const struct mem_cgroup *memcg)
 {
  struct tcp_memcontrol *tcp;
--
1.7.6.4
```

Subject: [PATCH v7 09/10] Display maximum tcp memory allocation in kmem cgroup
Posted by Glauber Costa on Tue, 29 Nov 2011 23:57:00 GMT
View Forum Message <> Reply to Message

This patch introduces kmem.tcp.max_usage_in_bytes file, living in the kmem_cgroup filesystem. The root cgroup will display a value equal to RESOURCE_MAX. This is to avoid introducing any locking schemes in the network paths when cgroups are not being actively used.

All others, will see the maximum memory ever used by this 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>
---
 net/ipv4/tcp_memcontrol.c | 12 ++++++++++-
 1 files changed, 11 insertions(+), 1 deletions(-)

```
diff --git a/net/ipv4/tcp_memcontrol.c b/net/ipv4/tcp_memcontrol.c
index f7350fa1..6ec90de 100644
--- a/net/ipv4/tcp_memcontrol.c
+++ b/net/ipv4/tcp_memcontrol.c
@@ -29,6 +29,12 @@ static struct cftype tcp_files[] = {
   .trigger = tcp_cgroup_reset,
   .read_u64 = tcp_cgroup_read,
  },
+ {
+  .name = "kmem.tcp.max_usage_in_bytes",
+  .private = RES_MAX_USAGE,
+  .trigger = tcp_cgroup_reset,
+  .read_u64 = tcp_cgroup_read,
+ },
 };
```

```
 static inline struct tcp_memcontrol *tcp_from_cgproto(struct cg_proto *cg_proto)
@@ -201,7 +207,8 @@ static u64 tcp_cgroup_read(struct cgroup *cont, struct cftype *cft)
   val = tcp_read_usage(memcg);
   break;
  case RES_FAILCNT:
-  val = tcp_read_stat(memcg, RES_FAILCNT, 0);
+ case RES_MAX_USAGE:
+  val = tcp_read_stat(memcg, cft->private, 0);
   break;
  default:
   BUG();
@@ -222,6 +229,9 @@ static int tcp_cgroup_reset(struct cgroup *cont, unsigned int event)
  tcp = tcp_from_cgproto(cg_proto);

  switch (event) {
+ case RES_MAX_USAGE:
+  res_counter_reset_max(&tcp->tcp_memory_allocated);
+  break;
  case RES_FAILCNT:
   res_counter_reset_failcnt(&tcp->tcp_memory_allocated);
   break;
--
1.7.6.4
```

Subject: [PATCH v7 10/10] Disable task moving when using kernel memory accounting

Posted by Glauber Costa on Tue, 29 Nov 2011 23:57:01 GMT

View Forum Message <> Reply to Message

Since this code is still experimental, we are leaving the exact
details of how to move tasks between cgroups when kernel memory
accounting is used as future work.

For now, we simply disallow movement if there are any pending
accounted memory.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
---
 mm/memcontrol.c | 23 ++++++++++++++++++++++-
 1 files changed, 22 insertions(+), 1 deletions(-)

```
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index a31a278..dd9a6d9 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -5453,10 +5453,19 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
```

```
 {
  int ret = 0;
  struct mem_cgroup *memcg = mem_cgroup_from_cont(cgroup);
+ struct mem_cgroup *from = mem_cgroup_from_task(p);
+
+#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) && defined(CONFIG_INET)
+ if (from != memcg && !mem_cgroup_is_root(from) &&
+     res_counter_read_u64(&from->tcp_mem.tcp_memory_allocated, RES_USAGE)) {
+  printk(KERN_WARNING "Can't move tasks between cgroups: "
+   "Kernel memory held.\n");
+  return 1;
+ }
+#endif

  if (memcg->move_charge_at_immigrate) {
   struct mm_struct *mm;
-  struct mem_cgroup *from = mem_cgroup_from_task(p);

   VM_BUG_ON(from == memcg);

@@ -5624,6 +5633,18 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
     struct cgroup *cgroup,
     struct task_struct *p)
 {
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgroup);
+ struct mem_cgroup *from = mem_cgroup_from_task(p);
+
+#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) && defined(CONFIG_INET)
+ if (from != memcg && !mem_cgroup_is_root(from) &&
+     res_counter_read_u64(&from->tcp_mem.tcp_memory_allocated, RES_USAGE)) {
+  printk(KERN_WARNING "Can't move tasks between cgroups: "
+   "Kernel memory held.\n");
+  return 1;
+ }
+#endif
+
  return 0;
 }
 static void mem_cgroup_cancel_attach(struct cgroup_subsys *ss,
--
1.7.6.4
```

Subject: Re: [PATCH v7 02/10] foundations of per-cgroup memory pressure controlling.
Posted by KAMEZAWA Hiroyuki on Wed, 30 Nov 2011 00:43:05 GMT
View Forum Message <> Reply to Message

On Tue, 29 Nov 2011 21:56:53 -0200
Glauber Costa <glommer@parallels.com> wrote:

> This patch replaces all uses of struct sock fields' memory_pressure,
> memory_allocated, sockets_allocated, and sysctl_mem to acessor
> macros. Those macros can either receive a socket argument, or a mem_cgroup
> argument, depending on the context they live in.
>
> Since we're only doing a macro wrapping here, no performance impact at all is
> expected in the case where we don't have cgroups disabled.
>
> 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>
> CC: Eric Dumazet <eric.dumazet@gmail.com>
<snip>

> +static inline bool
> +memcg_memory_pressure(struct proto *prot, struct mem_cgroup *memcg)
> +{
> + if (!prot->memory_pressure)
> + return false;
> + return !!prot->memory_pressure;
> +}

I think you should take a deep breath and write patech relaxedly, and do enough test.

This should be

 return !!*prot->memory_pressure;

BTW, I don't like to receive tons of everyday-update even if you're in hurry.

> static void proto_seq_printf(struct seq_file *seq, struct proto *proto)
> {
> + struct mem_cgroup *memcg = mem_cgroup_from_task(current);
> +
>   seq_printf(seq, "%-9s %4u %6d %6ld   %-3s %6u   %-3s %-10s "
>   "%2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c
%2c\n",
>   proto->name,
>   proto->obj_size,
>   sock_prot_inuse_get(seq_file_net(seq), proto),

> -   proto->memory_allocated != NULL ? atomic_long_read(proto->memory_allocated) : -1L,
> -   proto->memory_pressure != NULL ? *proto->memory_pressure ? "yes" : "no" : "NI",
> +   sock_prot_memory_allocated(proto, memcg),
> +   sock_prot_memory_pressure(proto, memcg),

I wonder I should say NO, here. (Networking guys are ok ??)

IIUC, this means there is no way to see aggregated sockstat of all system.
And the result depends on the cgroup which the caller is under control.

I think you should show aggregated sockstat(global + per-memcg) here and
show per-memcg ones via /cgroup interface or add private_sockstat to show
per cgroup summary.

Thanks,
-Kame

Subject: Re: [PATCH v7 01/10] Basic kernel memory functionality for the Memory Controller
Posted by KAMEZAWA Hiroyuki on Wed, 30 Nov 2011 00:48:51 GMT
View Forum Message <> Reply to Message

On Tue, 29 Nov 2011 21:56:52 -0200
Glauber Costa <glommer@parallels.com> wrote:

> This patch lays down the foundation for the kernel memory component
> of the Memory Controller.
>
> As of today, I am only laying down the following files:
>
> * memory.independent_kmem_limit
> * memory.kmem.limit_in_bytes (currently ignored)
> * memory.kmem.usage_in_bytes (always zero)
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> Reviewed-by: Kirill A. Shutemov <kirill@shutemov.name>
> CC: Paul Menage <paul@paulmenage.org>
> CC: Greg Thelen <gthelen@google.com>

Now, there are new memcg maintainers as

Johannes Weiner <hannes@cmpxchg.org>
Michal Hocko <mhocko@suse.cz>
KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
Balbir Singh <bsingharora@gmail.com>

So, please CC them.

I may not be able to make a quick/good reply.

Hmm, my concern is

+ memory.independent_kmem_limit # select whether or not kernel memory limits are
+   independent of user limits

I'll be okay with this. other ones ?

Acked-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

Subject: Re: [PATCH v7 03/10] socket: initial cgroup code.
Posted by KAMEZAWA Hiroyuki on Wed, 30 Nov 2011 01:07:38 GMT
View Forum Message <> Reply to Message

On Tue, 29 Nov 2011 21:56:54 -0200
Glauber Costa <glommer@parallels.com> wrote:

> The goal of this work is to move the memory pressure tcp
> controls to a cgroup, instead of just relying on global
> conditions.
>
> To avoid excessive overhead in the network fast paths,
> the code that accounts allocated memory to a cgroup is
> hidden inside a static_branch(). This branch is patched out
> until the first non-root cgroup is created. So when nobody
> is using cgroups, even if it is mounted, no significant performance
> penalty should be seen.
>
> This patch handles the generic part of the code, and has nothing
> tcp-specific.
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> Acked-by: Kirill A. Shutemov<kirill@shutemov.name>
> Reviewed-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujtsu.com>
> CC: David S. Miller <davem@davemloft.net>
> CC: Eric W. Biederman <ebiederm@xmission.com>
> CC: Eric Dumazet <eric.dumazet@gmail.com>

<snip>

> +extern struct jump_label_key memcg_socket_limit_enabled;
> static inline bool sk_has_memory_pressure(const struct sock *sk)
> {
>   return sk->sk_prot->memory_pressure != NULL;
> @@ -873,6 +900,17 @@ static inline bool sk_under_memory_pressure(const struct sock *sk)
> {

>   if (!sk->sk_prot->memory_pressure)
>   return false;
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> + if (static_branch(&memcg_socket_limit_enabled)) {
> + struct cg_proto *cg_proto = sk->sk_cgrp;
> +
> + if (!cg_proto)
> +   goto nocgroup;
> + return !!*cg_proto->memory_pressure;
> + } else

What is dangling 'else' for ?

> +nocgroup:
> +#endif
> +
>   return !!*sk->sk_prot->memory_pressure;
> }
>
> @@ -880,52 +918,176 @@ static inline void sk_leave_memory_pressure(struct sock *sk)
> {
>   int *memory_pressure = sk->sk_prot->memory_pressure;
>
> - if (memory_pressure && *memory_pressure)
> + if (!memory_pressure)
> + return;
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> + if (static_branch(&memcg_socket_limit_enabled)) {
> + struct cg_proto *cg_proto = sk->sk_cgrp;
> +
> + if (!cg_proto)
> +   goto nocgroup;
> +
> + for (; cg_proto; cg_proto = cg_proto->parent)
> +   if (*cg_proto->memory_pressure)
> +   *cg_proto->memory_pressure = 0;
> + }
> +nocgroup:
> +#endif

Hmm..can't we have a good way for avoiding this #ifdef ?

I guess... as NUMA_BUILD macro in page_alloc.c, you can define

if (HAS_KMEM_LIMIT && static_branch(&.....)).

For example,

==
```
#include <stdio.h>

#define HAS_SPECIAL     0

int main(int argc, char *argv[])
{
        if (HAS_SPECIAL)
                call();

        printf("Hey!");
}
```
==

This can be compiled.

So. I guess...

```
#ifdef CONFIG_CGROUP_MEM_RES_CTLR
#define do_memcg_kmem_account static_branch(&memcg_socket_limit_enabled)
#else
#define do_memcg_kmem_account 0
#endif
```

maybe good.(not tested.)

BTW, I don't think 'goto nocgroup' is good.

```
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 3becb24..12a08bf 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -377,6 +377,40 @@ enum mem_type {
>  #define MEM_CGROUP_RECLAIM_SOFT_BIT 0x2
>  #define MEM_CGROUP_RECLAIM_SOFT  (1 << MEM_CGROUP_RECLAIM_SOFT_BIT)
>
> +static inline bool mem_cgroup_is_root(struct mem_cgroup *memcg)
> +{
> + return (memcg == root_mem_cgroup);
> +}
> +
```

Why do you need this move of definition ?

Thanks,
-Kame

Subject: Re: [PATCH v7 04/10] tcp memory pressure controls
Posted by KAMEZAWA Hiroyuki on Wed, 30 Nov 2011 01:49:43 GMT
View Forum Message <> Reply to Message

On Tue, 29 Nov 2011 21:56:55 -0200
Glauber Costa <glommer@parallels.com> wrote:

> 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.fujtisu.com>
> CC: Eric W. Biederman <ebiederm@xmission.com>

some comments.

> ---
> Documentation/cgroups/memory.txt | 2 +
> include/linux/memcontrol.h | 3 ++
> include/net/sock.h | 2 +
> include/net/tcp_memcontrol.h | 17 +++++++++
> mm/memcontrol.c | 36 ++++++++++++++++++--
> net/core/sock.c | 42 ++++++++++++++++++++++--
> net/ipv4/Makefile | 1 +
> net/ipv4/tcp_ipv4.c | 8 ++++-
> net/ipv4/tcp_memcontrol.c | 73 ++++++++++++++++++++++++++++++++++++++
> net/ipv6/tcp_ipv6.c | 4 ++
> 10 files changed, 181 insertions(+), 7 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 3cf9d96..1e43da4 100644
> --- a/Documentation/cgroups/memory.txt
> +++ b/Documentation/cgroups/memory.txt
> @@ -299,6 +299,8 @@ and set kmem extension config option carefully.
> 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 60964c3..fa2482a 100644
> --- a/include/linux/memcontrol.h
> +++ b/include/linux/memcontrol.h
> @@ -85,6 +85,9 @@ extern struct mem_cgroup *try_get_mem_cgroup_from_page(struct page
*page);
> 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 *mem_cgroup_from_cont(struct cgroup *cont);
> +extern struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *mem);
> +

use 'memcg' please.

> -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,
> @@ -4717,14 +4732,27 @@ 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));
> +
> + if (!ret)
> + ret = mem_cgroup_sockets_init(cont, ss);
>   return ret;
> };

You does initizalication here. The reason what I think is
1. 'proto_list' is not available at createion of root cgroup and
    you need to delay set up until mounting.

If so, please add comment or find another way.
This seems not very clean to me.

> +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;
> + }
```

seems indent is bad or {} is missing.

```
> +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;
> + 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);
> + cg_proto->parent = tcp_prot.proto_cgroup(parent);
> +
> + 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;
```

Question:

Is this value will be updated when an admin chages sysctl ?

I guess, this value is set at system init script or some which may
happen later than mounting cgroup.
I don't like to write a guideline 'please set sysctl val before
mounting cgroup'

Thanks,
-Kame

Subject: Re: [PATCH v7 06/10] tcp buffer limitation: per-cgroup limit
Posted by KAMEZAWA Hiroyuki on Wed, 30 Nov 2011 02:00:29 GMT
View Forum Message <> Reply to Message

On Tue, 29 Nov 2011 21:56:57 -0200
Glauber Costa <glommer@parallels.com> wrote:

> This patch uses the "tcp.limit_in_bytes" field of the kmem_cgroup to
> effectively control the amount of kernel memory pinned by a cgroup.
>
> This value is ignored in the root cgroup, and in all others,
> caps the value specified by the admin in the net namespaces'
> view of tcp_sysctl_mem.
>
> If namespaces are being used, the admin is allowed to set a
> value bigger than cgroup's maximum, the same way it is allowed
> to set pretty much unlimited values in a real box.
>
> 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>

you need one more fix.
(please add changelog.)

> +static int tcp_update_limit(struct mem_cgroup *memcg, u64 val)
> +{
> + struct net *net = current->nsproxy->net_ns;
> + struct tcp_memcontrol *tcp;
> + struct cg_proto *cg_proto;
> + u64 old_lim;
> + int i;
> + int ret;
> +

> + cg_proto = tcp_prot.proto_cgroup(memcg);
> + if (!cg_proto)
> + return -EINVAL;
> +
> + tcp = tcp_from_cgproto(cg_proto);
> +
> + old_lim = res_counter_read_u64(&tcp->tcp_memory_allocated, RES_LIMIT);
> + ret = res_counter_set_limit(&tcp->tcp_memory_allocated, val);
> + if (ret)
> + return ret;
> +
> + for (i = 0; i < 3; i++)
> + tcp->tcp_prot_mem[i] = min_t(long, val >> PAGE_SHIFT,
> + net->ipv4.sysctl_tcp_mem[i]);
> +
> + if (val == RESOURCE_MAX)
> + jump_label_dec(&memcg_socket_limit_enabled);

if (val == RESOUCE_MAX && old_lim != RESOUCE_MAX)

Thanks,
-Kame

Subject: Re: [PATCH v7 08/10] Display current tcp failcnt in kmem cgroup
Posted by KAMEZAWA Hiroyuki on Wed, 30 Nov 2011 02:01:31 GMT
View Forum Message <> Reply to Message

On Tue, 29 Nov 2011 21:56:59 -0200
Glauber Costa <glommer@parallels.com> wrote:

> This patch introduces kmem.tcp.failcnt file, living in the
> kmem_cgroup filesystem. Following the pattern in the other
> memcg resources, this files keeps a counter of how many times
> allocation failed due to limits being hit in this cgroup.
> The root cgroup will always show a failcnt of 0.
>
> 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>

Reviewed-by : KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

Subject: Re: [PATCH v7 09/10] Display maximum tcp memory allocation in kmem cgroup

Posted by KAMEZAWA Hiroyuki on Wed, 30 Nov 2011 02:02:20 GMT
View Forum Message <> Reply to Message

On Tue, 29 Nov 2011 21:57:00 -0200
Glauber Costa <glommer@parallels.com> wrote:

> This patch introduces kmem.tcp.max_usage_in_bytes file, living in the
> kmem_cgroup filesystem. The root cgroup will display a value equal
> to RESOURCE_MAX. This is to avoid introducing any locking schemes in
> the network paths when cgroups are not being actively used.
>
> All others, will see the maximum memory ever used by this 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>

Reviewed-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

Subject: Re: [PATCH v7 00/10] Request for Inclusion: per-cgroup tcp memory pressure
Posted by KAMEZAWA Hiroyuki on Wed, 30 Nov 2011 02:11:52 GMT
View Forum Message <> Reply to Message

On Tue, 29 Nov 2011 21:56:51 -0200
Glauber Costa <glommer@parallels.com> wrote:

> Hi,
>
> This patchset implements per-cgroup tcp memory pressure controls. It did not change
> significantly since last submission: rather, it just merges the comments Kame had.
> Most of them are style-related and/or Documentation, but there are two real bugs he
> managed to spot (thanks)
>
> Please let me know if there is anything else I should address.
>

After reading all codes again, I feel some strange. Could you clarify ?

Here.
==
+void sock_update_memcg(struct sock *sk)
+{
+ /* right now a socket spends its whole life in the same cgroup */
+ if (sk->sk_cgrp) {
+ WARN_ON(1);

```
+  return;
+ }
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct mem_cgroup *memcg;
+
+  BUG_ON(!sk->sk_prot->proto_cgroup);
+
+  rcu_read_lock();
+  memcg = mem_cgroup_from_task(current);
+  if (!mem_cgroup_is_root(memcg))
+   sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
+  rcu_read_unlock();
==
```

sk->sk_cgrp is set to a memcg without any reference count.

Then, no check for preventing rmdir() and freeing memcgroup.

Is there some css_get() or mem_cgroup_get() somewhere ?

Thanks,
-Kame

Subject: Re: [PATCH v7 10/10] Disable task moving when using kernel memory accounting
Posted by KAMEZAWA Hiroyuki on Wed, 30 Nov 2011 02:22:10 GMT
View Forum Message <> Reply to Message

On Tue, 29 Nov 2011 21:57:01 -0200
Glauber Costa <glommer@parallels.com> wrote:

> Since this code is still experimental, we are leaving the exact
> details of how to move tasks between cgroups when kernel memory
> accounting is used as future work.
>
> For now, we simply disallow movement if there are any pending
> accounted memory.
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
> ---
> mm/memcontrol.c | 23 ++++++++++++++++++++++-
> 1 files changed, 22 insertions(+), 1 deletions(-)
>
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index a31a278..dd9a6d9 100644
> --- a/mm/memcontrol.c

> +++ b/mm/memcontrol.c
> @@ -5453,10 +5453,19 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
> {
>   int ret = 0;
>   struct mem_cgroup *memcg = mem_cgroup_from_cont(cgroup);
> + struct mem_cgroup *from = mem_cgroup_from_task(p);
> +
> +#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) && defined(CONFIG_INET)
> + if (from != memcg && !mem_cgroup_is_root(from) &&
> +   res_counter_read_u64(&from->tcp_mem.tcp_memory_allocated, RES_USAGE)) {
> + printk(KERN_WARNING "Can't move tasks between cgroups: "
> +   "Kernel memory held.\n");
> + return 1;
> + }
> +#endif

I wonder....reading all codes again, this is incorrect check.

Hm, let me cralify. IIUC, in old code, "prevent moving" is because you hold
reference count of cgroup, which can cause trouble at rmdir() as leaking refcnt.

BTW, because socket is a shared resource between cgroup, changes in mm->owner
may cause task cgroup moving implicitly. So, if you allow leak of resource
here, I guess... you can take mem_cgroup_get() refcnt which is memcg-local and
allow rmdir(). Then, this limitation may disappear.

Then, users will be happy but admins will have unseen kernel resource usage in
not populated(by rmdir) memcg. Hm, big trouble ?

Thanks,
-Kame

Subject: Re: [PATCH v7 02/10] foundations of per-cgroup memory pressure controlling.
Posted by Glauber Costa on Fri, 02 Dec 2011 17:46:46 GMT

View Forum Message <> Reply to Message

>>   static void proto_seq_printf(struct seq_file *seq, struct proto *proto)
>>   {
>> + struct mem_cgroup *memcg = mem_cgroup_from_task(current);
>> +
>>   seq_printf(seq, "%-9s %4u %6d %6ld   %-3s %6u   %-3s %-10s "
>>   "%2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c\n",
>>   proto->name,
>>   proto->obj_size,
>>   sock_prot_inuse_get(seq_file_net(seq), proto),

>> -   proto->memory_allocated != NULL ? atomic_long_read(proto->memory_allocated) : -1L,
>> -   proto->memory_pressure != NULL ? *proto->memory_pressure ? "yes" : "no" : "NI",
>> +   sock_prot_memory_allocated(proto, memcg),
>> +   sock_prot_memory_pressure(proto, memcg),
>
> I wonder I should say NO, here. (Networking guys are ok ??)
>
> IIUC, this means there is no way to see aggregated sockstat of all system.
> And the result depends on the cgroup which the caller is under control.
>
> I think you should show aggregated sockstat(global + per-memcg) here and
> show per-memcg ones via /cgroup interface or add private_sockstat to show
> per cgroup summary.
>

Hi Kame,

Yes, the statistics displayed depends on which cgroup you live.
Also, note that the parent cgroup here is always updated (even when
use_hierarchy is set to 0). So it is always possible to grab global
statistics, by being in the root cgroup.

For the others, I believe it to be a question of naturalization. Any
tool that is fetching these values is likely interested in the amount of
resources available/used. When you are on a cgroup, the amount of
resources available/used changes, so that's what you should see.

Also brings the point of resource isolation: if you shouldn't interfere
with other set of process' resources, there is no reason for you to see
them in the first place.

So given all that, I believe that whenever we talk about resources in a
cgroup, we should talk about cgroup-local ones.

Subject: Re: [PATCH v7 04/10] tcp memory pressure controls
Posted by Glauber Costa on Fri, 02 Dec 2011 17:57:28 GMT
View Forum Message <> Reply to Message

On 11/29/2011 11:49 PM, KAMEZAWA Hiroyuki wrote:
>
>> -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,
>> @@ -4717,14 +4732,27 @@ 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));
>> +
>> + if (!ret)
>> + ret = mem_cgroup_sockets_init(cont, ss);
>>   return ret;
>>   };
>
> You does initizalication here. The reason what I think is
> 1. 'proto_list' is not available at createion of root cgroup and
>   you need to delay set up until mounting.
>
> If so, please add comment or find another way.
> This seems not very clean to me.

Yes, we do can run into some ordering issues. A part of the
initialization can be done earlier. But I preferred to move it all later
instead of creating two functions for it. But I can change that if you
want, no big deal.

>
>
>
>
>> +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;
>> + }
>
> seems indent is bad or {} is missing.
>
Thanks. I'll rewrite it, since I did miss {} around the first if. But no
test could possibly catch it, since what I wanted to write, and what I
wrote by mistake end up being equivalent.

```
>> +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;
>> + 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);
>> + cg_proto->parent = tcp_prot.proto_cgroup(parent);
>> +
>> + 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;
>
> Question:
>
> Is this value will be updated when an admin chages sysctl ?
```

yes.

> I guess, this value is set at system init script or some which may
> happen later than mounting cgroup.
> I don't like to write a guideline 'please set sysctl val before
> mounting cgroup'

Agreed.

This code is in patch 6 (together with the limiting):

```
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+       rcu_read_lock();
+       memcg = mem_cgroup_from_task(current);
+
+       tcp_prot_mem(memcg, vec[0], 0);
+       tcp_prot_mem(memcg, vec[1], 1);
```

```
+       tcp_prot_mem(memcg, vec[2], 2);
+       rcu_read_unlock();
+#endif
```

tcp_prot_mem is just a wrapper around the assignment so we can access
memcg's inner fields.

Subject: Re: [PATCH v7 00/10] Request for Inclusion: per-cgroup tcp memory pressure
Posted by Glauber Costa on Fri, 02 Dec 2011 18:04:08 GMT
View Forum Message <> Reply to Message

On 11/30/2011 12:11 AM, KAMEZAWA Hiroyuki wrote:
> On Tue, 29 Nov 2011 21:56:51 -0200
> Glauber Costa<glommer@parallels.com> wrote:
>
>> Hi,
>>
>> This patchset implements per-cgroup tcp memory pressure controls. It did not change
>> significantly since last submission: rather, it just merges the comments Kame had.
>> Most of them are style-related and/or Documentation, but there are two real bugs he
>> managed to spot (thanks)
>>
>> Please let me know if there is anything else I should address.
>>
>
> After reading all codes again, I feel some strange. Could you clarify ?
>
> Here.
> ==
> +void sock_update_memcg(struct sock *sk)
> +{
> + /* right now a socket spends its whole life in the same cgroup */
> + if (sk->sk_cgrp) {
> + WARN_ON(1);
> + return;
> + }
> + if (static_branch(&memcg_socket_limit_enabled)) {
> + struct mem_cgroup *memcg;
> +
> + BUG_ON(!sk->sk_prot->proto_cgroup);
> +
> + rcu_read_lock();
> + memcg = mem_cgroup_from_task(current);
> + if (!mem_cgroup_is_root(memcg))
> +   sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
> + rcu_read_unlock();

> ==
>
> sk->sk_cgrp is set to a memcg without any reference count.
>
> Then, no check for preventing rmdir() and freeing memcgroup.
>
> Is there some css_get() or mem_cgroup_get() somewhere ?
>

There were a css_get in the first version of this patchset. It was
removed, however, because it was deemed anti-intuitive to prevent rmdir,
since we can't know which sockets are blocking it, or do anything about
it. Or did I misunderstand something ?

Subject: Re: [PATCH v7 10/10] Disable task moving when using kernel memory accounting
Posted by Glauber Costa on Fri, 02 Dec 2011 18:11:56 GMT
View Forum Message <> Reply to Message

On 11/30/2011 12:22 AM, KAMEZAWA Hiroyuki wrote:
> On Tue, 29 Nov 2011 21:57:01 -0200
> Glauber Costa<glommer@parallels.com> wrote:
>
>> Since this code is still experimental, we are leaving the exact
>> details of how to move tasks between cgroups when kernel memory
>> accounting is used as future work.
>>
>> For now, we simply disallow movement if there are any pending
>> accounted memory.
>>
>> Signed-off-by: Glauber Costa<glommer@parallels.com>
>> CC: Hiroyouki Kamezawa<kamezawa.hiroyu@jp.fujitsu.com>
>> ---
>>   mm/memcontrol.c | 23 ++++++++++++++++++++++-
>>   1 files changed, 22 insertions(+), 1 deletions(-)
>>
>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
>> index a31a278..dd9a6d9 100644
>> --- a/mm/memcontrol.c
>> +++ b/mm/memcontrol.c
>> @@ -5453,10 +5453,19 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
>>   {
>>   int ret = 0;
>>   struct mem_cgroup *memcg = mem_cgroup_from_cont(cgroup);
>> + struct mem_cgroup *from = mem_cgroup_from_task(p);
>> +
>> +#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM)&& defined(CONFIG_INET)

>> + if (from != memcg&& !mem_cgroup_is_root(from)&&
>> +   res_counter_read_u64(&from->tcp_mem.tcp_memory_allocated, RES_USAGE)) {
>> + printk(KERN_WARNING "Can't move tasks between cgroups: "
>> +   "Kernel memory held.\n");
>> + return 1;
>> + }
>> +#endif
>
> I wonder....reading all codes again, this is incorrect check.
>
> Hm, let me cralify. IIUC, in old code, "prevent moving" is because you hold
> reference count of cgroup, which can cause trouble at rmdir() as leaking refcnt.
right.

> BTW, because socket is a shared resource between cgroup, changes in mm->owner
> may cause task cgroup moving implicitly. So, if you allow leak of resource
> here, I guess... you can take mem_cgroup_get() refcnt which is memcg-local and
> allow rmdir(). Then, this limitation may disappear.

Sorry, I didn't fully understand. Can you clarify further?
If the task is implicitly moved, it will end up calling can_attach as
well, right?

>
> Then, users will be happy but admins will have unseen kernel resource usage in
> not populated(by rmdir) memcg. Hm, big trouble ?
>
> Thanks,
> -Kame
>

Subject: Re: [PATCH v7 02/10] foundations of per-cgroup memory pressure controlling.
Posted by KAMEZAWA Hiroyuki on Mon, 05 Dec 2011 01:59:16 GMT
View Forum Message <> Reply to Message

On Fri, 2 Dec 2011 15:46:46 -0200
Glauber Costa <glommer@parallels.com> wrote:

>
> >>   static void proto_seq_printf(struct seq_file *seq, struct proto *proto)
> >>   {
> >> + struct mem_cgroup *memcg = mem_cgroup_from_task(current);
> >> +
> >>   seq_printf(seq, "%-9s %4u %6d %6ld   %-3s %6u   %-3s %-10s "
> >>   "%2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c

%2c %2c\n",
> >> proto->name,
> >> proto->obj_size,
> >> sock_prot_inuse_get(seq_file_net(seq), proto),
> >> - proto->memory_allocated != NULL ? atomic_long_read(proto->memory_allocated) : -1L,
> >> - proto->memory_pressure != NULL ? *proto->memory_pressure ? "yes" : "no" : "NI",
> >> + sock_prot_memory_allocated(proto, memcg),
> >> + sock_prot_memory_pressure(proto, memcg),
> >
> > I wonder I should say NO, here. (Networking guys are ok ??)
> >
> > IIUC, this means there is no way to see aggregated sockstat of all system.
> > And the result depends on the cgroup which the caller is under control.
> >
> > I think you should show aggregated sockstat(global + per-memcg) here and
> > show per-memcg ones via /cgroup interface or add private_sockstat to show
> > per cgroup summary.
> >
>
> Hi Kame,
>
> Yes, the statistics displayed depends on which cgroup you live.
> Also, note that the parent cgroup here is always updated (even when
> use_hierarchy is set to 0). So it is always possible to grab global
> statistics, by being in the root cgroup.
>
> For the others, I believe it to be a question of naturalization. Any
> tool that is fetching these values is likely interested in the amount of
> resources available/used. When you are on a cgroup, the amount of
> resources available/used changes, so that's what you should see.
>
> Also brings the point of resource isolation: if you shouldn't interfere
> with other set of process' resources, there is no reason for you to see
> them in the first place.
>
> So given all that, I believe that whenever we talk about resources in a
> cgroup, we should talk about cgroup-local ones.

But you changes /proc/ information without any arguments with other guys.
If you go this way, you should move this patch as independent add-on patch
and discuss what this should be. For example, /proc/meminfo doesn't reflect
memcg's information (for now). And scheduler statiscits in /proc/stat doesn't
reflect cgroup's information.

So, please discuss the problem in open way. This issue is not only related to
this patch but also to other cgroups. Sneaking this kind of _big_ change in
a middle of complicated patch series isn't good.

In short, could you divide this patch into a independent patch and discuss
again ? If we agree the general diection should go this way, other guys will
post patches for cpu, memory, blkio, etc.

Thanks,
-Kame

Subject: Re: [PATCH v7 04/10] tcp memory pressure controls
Posted by KAMEZAWA Hiroyuki on Mon, 05 Dec 2011 02:01:58 GMT
View Forum Message <> Reply to Message

On Fri, 2 Dec 2011 15:57:28 -0200
Glauber Costa <glommer@parallels.com> wrote:

> On 11/29/2011 11:49 PM, KAMEZAWA Hiroyuki wrote:
> >
> >> -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,
> >> @@ -4717,14 +4732,27 @@ 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));
> >> +
> >> + if (!ret)
> >> + ret = mem_cgroup_sockets_init(cont, ss);
> >>   return ret;
> >> };
> >
> > You does initizalication here. The reason what I think is
> > 1. 'proto_list' is not available at createion of root cgroup and
> >   you need to delay set up until mounting.
> >
> > If so, please add comment or find another way.
> > This seems not very clean to me.
>
> Yes, we do can run into some ordering issues. A part of the
> initialization can be done earlier. But I preferred to move it all later
> instead of creating two functions for it. But I can change that if you
> want, no big deal.
>

Hmm. please add comments about the 'issue'. It will help readers.

> >> + 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;
> >
> > Question:
> >
> > Is this value will be updated when an admin chages sysctl ?
>
> yes.
>
> > I guess, this value is set at system init script or some which may
> > happen later than mounting cgroup.
> > I don't like to write a guideline 'please set sysctl val before
> > mounting cgroup'
>
> Agreed.
>
> This code is in patch 6 (together with the limiting):
>
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> +	rcu_read_lock();
> +	memcg = mem_cgroup_from_task(current);
> +
> +	tcp_prot_mem(memcg, vec[0], 0);
> +	tcp_prot_mem(memcg, vec[1], 1);
> +	tcp_prot_mem(memcg, vec[2], 2);
> +	rcu_read_unlock();
> +#endif
>
> tcp_prot_mem is just a wrapper around the assignment so we can access
> memcg's inner fields.
>

Ok. sysctl and cgroup are updated at the same time.
thank you.
-Kame

Subject: Re: [PATCH v7 00/10] Request for Inclusion: per-cgroup tcp memory pressure
Posted by KAMEZAWA Hiroyuki on Mon, 05 Dec 2011 02:06:19 GMT
View Forum Message <> Reply to Message

On Fri, 2 Dec 2011 16:04:08 -0200

Glauber Costa <glommer@parallels.com> wrote:

> On 11/30/2011 12:11 AM, KAMEZAWA Hiroyuki wrote:
> > On Tue, 29 Nov 2011 21:56:51 -0200
> > Glauber Costa<glommer@parallels.com> wrote:
> >
> >> Hi,
> >>
> >> This patchset implements per-cgroup tcp memory pressure controls. It did not change
> >> significantly since last submission: rather, it just merges the comments Kame had.
> >> Most of them are style-related and/or Documentation, but there are two real bugs he
> >> managed to spot (thanks)
> >>
> >> Please let me know if there is anything else I should address.
> >>
> >
> > After reading all codes again, I feel some strange. Could you clarify ?
> >
> > Here.
> > ==
> > +void sock_update_memcg(struct sock *sk)
> > +{
> > + /* right now a socket spends its whole life in the same cgroup */
> > + if (sk->sk_cgrp) {
> > + WARN_ON(1);
> > + return;
> > + }
> > + if (static_branch(&memcg_socket_limit_enabled)) {
> > + struct mem_cgroup *memcg;
> > +
> > + BUG_ON(!sk->sk_prot->proto_cgroup);
> > +
> > + rcu_read_lock();
> > + memcg = mem_cgroup_from_task(current);
> > + if (!mem_cgroup_is_root(memcg))
> > +   sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
> > + rcu_read_unlock();
> > ==
> >
> > sk->sk_cgrp is set to a memcg without any reference count.
> >
> > Then, no check for preventing rmdir() and freeing memcgroup.
> >
> > Is there some css_get() or mem_cgroup_get() somewhere ?
> >
>
> There were a css_get in the first version of this patchset. It was
> removed, however, because it was deemed anti-intuitive to prevent rmdir,

> since we can't know which sockets are blocking it, or do anything about
> it. Or did I misunderstand something ?
>

Maybe I misuderstood. Thank you. Ok, there is no css_get/put and
rmdir() is allowed. But, hmm....what's guarding threads from stale
pointer access ?

Does a memory cgroup which is pointed by sk->sk_cgrp always exist ?

-Kame

Subject: Re: [PATCH v7 10/10] Disable task moving when using kernel memory accounting
Posted by KAMEZAWA Hiroyuki on Mon, 05 Dec 2011 02:18:35 GMT
View Forum Message <> Reply to Message

On Fri, 2 Dec 2011 16:11:56 -0200
Glauber Costa <glommer@parallels.com> wrote:

> On 11/30/2011 12:22 AM, KAMEZAWA Hiroyuki wrote:
> > On Tue, 29 Nov 2011 21:57:01 -0200
> > Glauber Costa<glommer@parallels.com> wrote:
> >
> >> Since this code is still experimental, we are leaving the exact
> >> details of how to move tasks between cgroups when kernel memory
> >> accounting is used as future work.
> >>
> >> For now, we simply disallow movement if there are any pending
> >> accounted memory.
> >>
> >> Signed-off-by: Glauber Costa<glommer@parallels.com>
> >> CC: Hiroyouki Kamezawa<kamezawa.hiroyu@jp.fujitsu.com>
> >> ---
> >>   mm/memcontrol.c | 23 ++++++++++++++++++++++-
> >>   1 files changed, 22 insertions(+), 1 deletions(-)
> >>
> >> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> >> index a31a278..dd9a6d9 100644
> >> --- a/mm/memcontrol.c
> >> +++ b/mm/memcontrol.c
> >> @@ -5453,10 +5453,19 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
> >>   {
> >>   int ret = 0;
> >>   struct mem_cgroup *memcg = mem_cgroup_from_cont(cgroup);
> >> + struct mem_cgroup *from = mem_cgroup_from_task(p);
> >> +

> >> +#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM)&& defined(CONFIG_INET)
> >> + if (from != memcg&& !mem_cgroup_is_root(from)&&
> >> +   res_counter_read_u64(&from->tcp_mem.tcp_memory_allocated, RES_USAGE)) {
> >> + printk(KERN_WARNING "Can't move tasks between cgroups: "
> >> +   "Kernel memory held.\n");
> >> + return 1;
> >> + }
> >> +#endif
> >
> > I wonder....reading all codes again, this is incorrect check.
> >
> > Hm, let me cralify. IIUC, in old code, "prevent moving" is because you hold
> > reference count of cgroup, which can cause trouble at rmdir() as leaking refcnt.
> right.
>
> > BTW, because socket is a shared resource between cgroup, changes in mm->owner
> > may cause task cgroup moving implicitly. So, if you allow leak of resource
> > here, I guess... you can take mem_cgroup_get() refcnt which is memcg-local and
> > allow rmdir(). Then, this limitation may disappear.
>
> Sorry, I didn't fully understand. Can you clarify further?
> If the task is implicitly moved, it will end up calling can_attach as
> well, right?
>
I'm sorry that my explanation is bad.

You can take memory cgroup itself's reference count by mem_cgroup_put/get.
By getting this, memory cgroup object will continue to exist even after
its struct cgroup* is freed by rmdir().

So, assume you do mem_cgroup_get()/put at socket attaching/detatching.

0) A task has a tcp socekts in memcg0.

task(memcg0)
 +- socket0 --> memcg0,usage=4096

1) move this task to memcg1

task(memcg1)
 +- socket0 --> memcg0,usage=4096

2) The task create a new socket.

task(memcg1)
 +- socekt0 --> memcg0,usage=4096
 +- socket1 --> memcg1,usage=xxxx

Here, the task will hold 4096bytes of usage in memcg0 implicitly.

3) an admin removes memcg0
task(memcg1)
 +- socket0 -->memcg0, usage=4096 <-----(*)
 +- socket1 -->memcg1, usage=xxxx

(*) is invisible to users....but this will not be very big problem.

Thanks,
-Kame

Thanks,
-Kame

Subject: Re: [PATCH v7 02/10] foundations of per-cgroup memory pressure controlling.
Posted by Glauber Costa on Mon, 05 Dec 2011 09:06:20 GMT
View Forum Message <> Reply to Message

On 12/04/2011 11:59 PM, KAMEZAWA Hiroyuki wrote:
> On Fri, 2 Dec 2011 15:46:46 -0200
> Glauber Costa<glommer@parallels.com> wrote:
>
>>
>>>>>   static void proto_seq_printf(struct seq_file *seq, struct proto *proto)
>>>>>   {
>>>> + struct mem_cgroup *memcg = mem_cgroup_from_task(current);
>>>> +
>>>>>   seq_printf(seq, "%-9s %4u %6d %6ld   %-3s %6u   %-3s %-10s "
>>>>>   "%2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c %2c\n",
>>>>>   proto->name,
>>>>>   proto->obj_size,
>>>>>   sock_prot_inuse_get(seq_file_net(seq), proto),
>>>> -   proto->memory_allocated != NULL ? atomic_long_read(proto->memory_allocated) : -1L,
>>>> -   proto->memory_pressure != NULL ? *proto->memory_pressure ? "yes" : "no" : "NI",

>>>> +	sock_prot_memory_allocated(proto, memcg),
>>>> +	sock_prot_memory_pressure(proto, memcg),
>>>
>>> I wonder I should say NO, here. (Networking guys are ok ??)
>>>
>>> IIUC, this means there is no way to see aggregated sockstat of all system.
>>> And the result depends on the cgroup which the caller is under control.
>>>
>>> I think you should show aggregated sockstat(global + per-memcg) here and
>>> show per-memcg ones via /cgroup interface or add private_sockstat to show
>>> per cgroup summary.
>>>
>>
>> Hi Kame,
>>
>> Yes, the statistics displayed depends on which cgroup you live.
>> Also, note that the parent cgroup here is always updated (even when
>> use_hierarchy is set to 0). So it is always possible to grab global
>> statistics, by being in the root cgroup.
>>
>> For the others, I believe it to be a question of naturalization. Any
>> tool that is fetching these values is likely interested in the amount of
>> resources available/used. When you are on a cgroup, the amount of
>> resources available/used changes, so that's what you should see.
>>
>> Also brings the point of resource isolation: if you shouldn't interfere
>> with other set of process' resources, there is no reason for you to see
>> them in the first place.
>>
>> So given all that, I believe that whenever we talk about resources in a
>> cgroup, we should talk about cgroup-local ones.
>
> But you changes /proc/ information without any arguments with other guys.
> If you go this way, you should move this patch as independent add-on patch
> and discuss what this should be. For example, /proc/meminfo doesn't reflect
> memcg's information (for now). And scheduler statiscits in /proc/stat doesn't
> reflect cgroup's information.

No, I do not.
I may not have discussed it with everybody, but I did send some mails
about it a while ago:

https://lkml.org/lkml/2011/10/3/60 (I sent it to containers as well
once, but I now realize it was during the time the ML was down).

At the time, *I* was probably the only one, arguing not to do it. I've
changed my mind since then.

> So, please discuss the problem in open way. This issue is not only related to
> this patch but also to other cgroups. Sneaking this kind of _big_ change in
> a middle of complicated patch series isn't good.

Absolutely. I can even remove this entirely and queue it for a following
patchset if you prefer.

> In short, could you divide this patch into a independent patch and discuss
> again ? If we agree the general diection should go this way, other guys will
> post patches for cpu, memory, blkio, etc.

Yes I can.

I am expanding the CC list here so other people that cares for other
controllers can chime in. You are welcome to give your opinion as the
memcg maintainer as well.

Subject: Re: [PATCH v7 00/10] Request for Inclusion: per-cgroup tcp memory pressure
Posted by Glauber Costa on Mon, 05 Dec 2011 09:09:51 GMT
View Forum Message <> Reply to Message

On 12/05/2011 12:06 AM, KAMEZAWA Hiroyuki wrote:
> On Fri, 2 Dec 2011 16:04:08 -0200
> Glauber Costa<glommer@parallels.com> wrote:
>
>> On 11/30/2011 12:11 AM, KAMEZAWA Hiroyuki wrote:
>>> On Tue, 29 Nov 2011 21:56:51 -0200
>>> Glauber Costa<glommer@parallels.com> wrote:
>>>
>>>> Hi,
>>>>
>>>> This patchset implements per-cgroup tcp memory pressure controls. It did not change
>>>> significantly since last submission: rather, it just merges the comments Kame had.
>>>> Most of them are style-related and/or Documentation, but there are two real bugs he
>>>> managed to spot (thanks)
>>>>
>>>> Please let me know if there is anything else I should address.
>>>>
>>>
>>> After reading all codes again, I feel some strange. Could you clarify ?
>>>
>>> Here.
>>> ==
>>> +void sock_update_memcg(struct sock *sk)
>>> +{
>>> + /* right now a socket spends its whole life in the same cgroup */

>>> + if (sk->sk_cgrp) {
>>> + WARN_ON(1);
>>> + return;
>>> + }
>>> + if (static_branch(&memcg_socket_limit_enabled)) {
>>> + struct mem_cgroup *memcg;
>>> +
>>> + BUG_ON(!sk->sk_prot->proto_cgroup);
>>> +
>>> + rcu_read_lock();
>>> + memcg = mem_cgroup_from_task(current);
>>> + if (!mem_cgroup_is_root(memcg))
>>> +   sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
>>> + rcu_read_unlock();
>>> ==
>>>
>>> sk->sk_cgrp is set to a memcg without any reference count.
>>>
>>> Then, no check for preventing rmdir() and freeing memcgroup.
>>>
>>> Is there some css_get() or mem_cgroup_get() somewhere ?
>>>
>>
>> There were a css_get in the first version of this patchset. It was
>> removed, however, because it was deemed anti-intuitive to prevent rmdir,
>> since we can't know which sockets are blocking it, or do anything about
>> it. Or did I misunderstand something ?
>>
>
> Maybe I misuderstood. Thank you. Ok, there is no css_get/put and
> rmdir() is allowed. But, hmm....what's guarding threads from stale
> pointer access ?
>
> Does a memory cgroup which is pointed by sk->sk_cgrp always exist ?
>
If I am not mistaken, yes, it will. (Ok, right now it won't)

Reason is a cgroup can't be removed if it is empty.
To make it empty, you need to move the tasks away.

So the sockets will be moved away as well when you do it. So right now
they are not, so it would then probably be better to increase a
reference count with a comment saying that it is temporary.

Subject: Re: [PATCH v7 10/10] Disable task moving when using kernel memory accounting

On 12/05/2011 12:18 AM, KAMEZAWA Hiroyuki wrote:
> On Fri, 2 Dec 2011 16:11:56 -0200
> Glauber Costa<glommer@parallels.com> wrote:
>
>> On 11/30/2011 12:22 AM, KAMEZAWA Hiroyuki wrote:
>>> On Tue, 29 Nov 2011 21:57:01 -0200
>>> Glauber Costa<glommer@parallels.com>   wrote:
>>>
>>>> Since this code is still experimental, we are leaving the exact
>>>> details of how to move tasks between cgroups when kernel memory
>>>> accounting is used as future work.
>>>>
>>>> For now, we simply disallow movement if there are any pending
>>>> accounted memory.
>>>>
>>>> Signed-off-by: Glauber Costa<glommer@parallels.com>
>>>> CC: Hiroyouki Kamezawa<kamezawa.hiroyu@jp.fujitsu.com>
>>>> ---
>>>>   mm/memcontrol.c | 23 ++++++++++++++++++++++-
>>>>   1 files changed, 22 insertions(+), 1 deletions(-)
>>>>
>>>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
>>>> index a31a278..dd9a6d9 100644
>>>> --- a/mm/memcontrol.c
>>>> +++ b/mm/memcontrol.c
>>>> @@ -5453,10 +5453,19 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
>>>>   {
>>>>   int ret = 0;
>>>>   struct mem_cgroup *memcg = mem_cgroup_from_cont(cgroup);
>>>> + struct mem_cgroup *from = mem_cgroup_from_task(p);
>>>> +
>>>> +#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM)&&   defined(CONFIG_INET)
>>>> + if (from != memcg&&   !mem_cgroup_is_root(from)&&
>>>> +   res_counter_read_u64(&from->tcp_mem.tcp_memory_allocated, RES_USAGE)) {
>>>> + printk(KERN_WARNING "Can't move tasks between cgroups: "
>>>> +   "Kernel memory held.\n");
>>>> + return 1;
>>>> + }
>>>> +#endif
>>>
>>> I wonder....reading all codes again, this is incorrect check.
>>>
>>> Hm, let me cralify. IIUC, in old code, "prevent moving" is because you hold
>>> reference count of cgroup, which can cause trouble at rmdir() as leaking refcnt.
>> right.
>>

>>> BTW, because socket is a shared resource between cgroup, changes in mm->owner
>>> may cause task cgroup moving implicitly. So, if you allow leak of resource
>>> here, I guess... you can take mem_cgroup_get() refcnt which is memcg-local and
>>> allow rmdir(). Then, this limitation may disappear.
>>
>> Sorry, I didn't fully understand. Can you clarify further?
>> If the task is implicitly moved, it will end up calling can_attach as
>> well, right?
>>
> I'm sorry that my explanation is bad.
>
> You can take memory cgroup itself's reference count by mem_cgroup_put/get.
> By getting this, memory cgroup object will continue to exist even after
> its struct cgroup* is freed by rmdir().
>
> So, assume you do mem_cgroup_get()/put at socket attaching/detatching.
>
> 0) A task has a tcp socekts in memcg0.
>
> task(memcg0)
>   +- socket0 --> memcg0,usage=4096
>
> 1) move this task to memcg1
>
> task(memcg1)
>   +- socket0 --> memcg0,usage=4096
>
> 2) The task create a new socket.
>
> task(memcg1)
>   +- socekt0 --> memcg0,usage=4096
>   +- socket1 --> memcg1,usage=xxxx
>
> Here, the task will hold 4096bytes of usage in memcg0 implicitly.
>
> 3) an admin removes memcg0
> task(memcg1)
>   +- socket0 -->memcg0, usage=4096<-----(*)
>   +- socket1 -->memcg1, usage=xxxx
>
> (*) is invisible to users....but this will not be very big problem.
>
Hi Kame,

Thanks for the explanation.

Hummm, Do you think that by doing it, we get rid of the need of moving
sockets to another memcg when the task is moved? So in my original

patchset, if you recall, I wanted to keep a socket forever in the same cgroup. I didn't, because then rmdir would be blocked.

By using this memcg reference trick, both can be achieved. What do you think ?

Subject: Re: [PATCH v7 00/10] Request for Inclusion: per-cgroup tcp memory pressure
Posted by KAMEZAWA Hiroyuki on Mon, 05 Dec 2011 09:51:08 GMT
View Forum Message <> Reply to Message

On Mon, 5 Dec 2011 07:09:51 -0200
Glauber Costa <glommer@parallels.com> wrote:

> On 12/05/2011 12:06 AM, KAMEZAWA Hiroyuki wrote:
> > On Fri, 2 Dec 2011 16:04:08 -0200
> > Glauber Costa<glommer@parallels.com> wrote:
> >
> >> On 11/30/2011 12:11 AM, KAMEZAWA Hiroyuki wrote:
> >>> On Tue, 29 Nov 2011 21:56:51 -0200
> >>> Glauber Costa<glommer@parallels.com>   wrote:
> >>>
> >>>> Hi,
> >>>>
> >>>> This patchset implements per-cgroup tcp memory pressure controls. It did not change
> >>>> significantly since last submission: rather, it just merges the comments Kame had.
> >>>> Most of them are style-related and/or Documentation, but there are two real bugs he
> >>>> managed to spot (thanks)
> >>>>
> >>>> Please let me know if there is anything else I should address.
> >>>>
> >>>
> >>> After reading all codes again, I feel some strange. Could you clarify ?
> >>>
> >>> Here.
> >>> ==
> >>> +void sock_update_memcg(struct sock *sk)
> >>> +{
> >>> + /* right now a socket spends its whole life in the same cgroup */
> >>> + if (sk->sk_cgrp) {
> >>> + WARN_ON(1);
> >>> + return;
> >>> + }
> >>> + if (static_branch(&memcg_socket_limit_enabled)) {
> >>> + struct mem_cgroup *memcg;
> >>> +
> >>> + BUG_ON(!sk->sk_prot->proto_cgroup);

> >>> +
> >>> + rcu_read_lock();
> >>> + memcg = mem_cgroup_from_task(current);
> >>> + if (!mem_cgroup_is_root(memcg))
> >>> +   sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
> >>> + rcu_read_unlock();
> >>> ==
> >>>
> >>> sk->sk_cgrp is set to a memcg without any reference count.
> >>>
> >>> Then, no check for preventing rmdir() and freeing memcgroup.
> >>>
> >>> Is there some css_get() or mem_cgroup_get() somewhere ?
> >>>
> >>
> >> There were a css_get in the first version of this patchset. It was
> >> removed, however, because it was deemed anti-intuitive to prevent rmdir,
> >> since we can't know which sockets are blocking it, or do anything about
> >> it. Or did I misunderstand something ?
> >>
> >
> > Maybe I misuderstood. Thank you. Ok, there is no css_get/put and
> > rmdir() is allowed. But, hmm....what's guarding threads from stale
> > pointer access ?
> >
> > Does a memory cgroup which is pointed by sk->sk_cgrp always exist ?
> >
> If I am not mistaken, yes, it will. (Ok, right now it won't)
>
> Reason is a cgroup can't be removed if it is empty.
> To make it empty, you need to move the tasks away.
>
> So the sockets will be moved away as well when you do it. So right now
> they are not, so it would then probably be better to increase a
> reference count with a comment saying that it is temporary.
>

I'm sorry if I misunderstand.

At task exit, __fput() will be called against file descriptors, yes.
__fput() calles f_op->release() => inet_release() => tcp_close().

But TCP socket may be alive after task exit until it gets down to
protocol close. For example, until the all message in send buffer
is acked, socket and tcp connection will not be disappear.

In short, socket's lifetime is different from it's task's.
So, there may be sockets which are not belongs to any task.

Thanks,
-Kame

Subject: Re: [PATCH v7 00/10] Request for Inclusion: per-cgroup tcp memory pressure
Posted by Glauber Costa on Mon, 05 Dec 2011 10:28:56 GMT
View Forum Message <> Reply to Message

On 12/05/2011 07:51 AM, KAMEZAWA Hiroyuki wrote:
> On Mon, 5 Dec 2011 07:09:51 -0200
> Glauber Costa<glommer@parallels.com> wrote:
>
>> On 12/05/2011 12:06 AM, KAMEZAWA Hiroyuki wrote:
>>> On Fri, 2 Dec 2011 16:04:08 -0200
>>> Glauber Costa<glommer@parallels.com> wrote:
>>>
>>>> On 11/30/2011 12:11 AM, KAMEZAWA Hiroyuki wrote:
>>>>> On Tue, 29 Nov 2011 21:56:51 -0200
>>>>> Glauber Costa<glommer@parallels.com> wrote:
>>>>>
>>>>>> Hi,
>>>>>>
>>>>>> This patchset implements per-cgroup tcp memory pressure controls. It did not change
>>>>>> significantly since last submission: rather, it just merges the comments Kame had.
>>>>>> Most of them are style-related and/or Documentation, but there are two real bugs he
>>>>>> managed to spot (thanks)
>>>>>>
>>>>>> Please let me know if there is anything else I should address.
>>>>>>
>>>>>
>>>>> After reading all codes again, I feel some strange. Could you clarify ?
>>>>>
>>>>> Here.
>>>>> ==
>>>>> +void sock_update_memcg(struct sock *sk)
>>>>> +{
>>>>> + /* right now a socket spends its whole life in the same cgroup */
>>>>> + if (sk->sk_cgrp) {
>>>>> + WARN_ON(1);
>>>>> + return;
>>>>> + }
>>>>> + if (static_branch(&memcg_socket_limit_enabled)) {
>>>>> + struct mem_cgroup *memcg;
>>>>> +

>>>>> + BUG_ON(!sk->sk_prot->proto_cgroup);
>>>>> +
>>>>> + rcu_read_lock();
>>>>> + memcg = mem_cgroup_from_task(current);
>>>>> + if (!mem_cgroup_is_root(memcg))
>>>>> +   sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
>>>>> + rcu_read_unlock();
>>>>> ==
>>>>>
>>>>> sk->sk_cgrp is set to a memcg without any reference count.
>>>>>
>>>>> Then, no check for preventing rmdir() and freeing memcgroup.
>>>>>
>>>>> Is there some css_get() or mem_cgroup_get() somewhere ?
>>>>>
>>>>
>>>> There were a css_get in the first version of this patchset. It was
>>>> removed, however, because it was deemed anti-intuitive to prevent rmdir,
>>>> since we can't know which sockets are blocking it, or do anything about
>>>> it. Or did I misunderstand something ?
>>>>
>>>
>>> Maybe I misuderstood. Thank you. Ok, there is no css_get/put and
>>> rmdir() is allowed. But, hmm....what's guarding threads from stale
>>> pointer access ?
>>>
>>> Does a memory cgroup which is pointed by sk->sk_cgrp always exist ?
>>>
>> If I am not mistaken, yes, it will. (Ok, right now it won't)
>>
>> Reason is a cgroup can't be removed if it is empty.
>> To make it empty, you need to move the tasks away.
>>
>> So the sockets will be moved away as well when you do it. So right now
>> they are not, so it would then probably be better to increase a
>> reference count with a comment saying that it is temporary.
>>
>
> I'm sorry if I misunderstand.
>
> At task exit, __fput() will be called against file descriptors, yes.
> __fput() calles f_op->release() => inet_release() => tcp_close().
>
> But TCP socket may be alive after task exit until it gets down to
> protocol close. For example, until the all message in send buffer
> is acked, socket and tcp connection will not be disappear.
>
> In short, socket's lifetime is different from it's task's.

> So, there may be sockets which are not belongs to any task.
>

Yeah, you're right. I guess this is one more reason for us to just keep
the memcg reference around.

Subject: Re: [PATCH v7 10/10] Disable task moving when using kernel memory accounting
Posted by KAMEZAWA Hiroyuki on Tue, 06 Dec 2011 00:07:26 GMT
View Forum Message <> Reply to Message

On Mon, 5 Dec 2011 07:18:37 -0200
Glauber Costa <glommer@parallels.com> wrote:

> On 12/05/2011 12:18 AM, KAMEZAWA Hiroyuki wrote:
> > On Fri, 2 Dec 2011 16:11:56 -0200
> > Glauber Costa<glommer@parallels.com> wrote:
> >
> >> On 11/30/2011 12:22 AM, KAMEZAWA Hiroyuki wrote:
> >>> On Tue, 29 Nov 2011 21:57:01 -0200
> >>> Glauber Costa<glommer@parallels.com>   wrote:
> >>>
> >>>> Since this code is still experimental, we are leaving the exact
> >>>> details of how to move tasks between cgroups when kernel memory
> >>>> accounting is used as future work.
> >>>>
> >>>> For now, we simply disallow movement if there are any pending
> >>>> accounted memory.
> >>>>
> >>>> Signed-off-by: Glauber Costa<glommer@parallels.com>
> >>>> CC: Hiroyouki Kamezawa<kamezawa.hiroyu@jp.fujitsu.com>
> >>>> ---
> >>>>   mm/memcontrol.c | 23 ++++++++++++++++++++++-
> >>>>   1 files changed, 22 insertions(+), 1 deletions(-)
> >>>>
> >>>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> >>>> index a31a278..dd9a6d9 100644
> >>>> --- a/mm/memcontrol.c
> >>>> +++ b/mm/memcontrol.c
> >>>> @@ -5453,10 +5453,19 @@ static int mem_cgroup_can_attach(struct cgroup_subsys
*ss,
> >>>>   {
> >>>>   int ret = 0;
> >>>>   struct mem_cgroup *memcg = mem_cgroup_from_cont(cgroup);
> >>>> + struct mem_cgroup *from = mem_cgroup_from_task(p);
> >>>> +
> >>>> +#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM)&&   defined(CONFIG_INET)

> >>>> + if (from != memcg&&   !mem_cgroup_is_root(from)&&
> >>>> +   res_counter_read_u64(&from->tcp_mem.tcp_memory_allocated, RES_USAGE)) {
> >>>> + printk(KERN_WARNING "Can't move tasks between cgroups: "
> >>>> +   "Kernel memory held.\n");
> >>>> + return 1;
> >>>> + }
> >>>> +#endif
> >>>
> >>> I wonder....reading all codes again, this is incorrect check.
> >>>
> >>> Hm, let me cralify. IIUC, in old code, "prevent moving" is because you hold
> >>> reference count of cgroup, which can cause trouble at rmdir() as leaking refcnt.
> >> right.
> >>
> >>> BTW, because socket is a shared resource between cgroup, changes in mm->owner
> >>> may cause task cgroup moving implicitly. So, if you allow leak of resource
> >>> here, I guess... you can take mem_cgroup_get() refcnt which is memcg-local and
> >>> allow rmdir(). Then, this limitation may disappear.
> >>
> >> Sorry, I didn't fully understand. Can you clarify further?
> >> If the task is implicitly moved, it will end up calling can_attach as
> >> well, right?
> >>
> > I'm sorry that my explanation is bad.
> >
> > You can take memory cgroup itself's reference count by mem_cgroup_put/get.
> > By getting this, memory cgroup object will continue to exist even after
> > its struct cgroup* is freed by rmdir().
> >
> > So, assume you do mem_cgroup_get()/put at socket attaching/detatching.
> >
> > 0) A task has a tcp socekts in memcg0.
> >
> > task(memcg0)
> >   +- socket0 --> memcg0,usage=4096
> >
> > 1) move this task to memcg1
> >
> > task(memcg1)
> >   +- socket0 --> memcg0,usage=4096
> >
> > 2) The task create a new socket.
> >
> > task(memcg1)
> >   +- socekt0 --> memcg0,usage=4096
> >   +- socket1 --> memcg1,usage=xxxx
> >
> > Here, the task will hold 4096bytes of usage in memcg0 implicitly.

> >
> > 3) an admin removes memcg0
> > task(memcg1)
> >   +- socket0 -->memcg0, usage=4096<-----(*)
> >   +- socket1 -->memcg1, usage=xxxx
> >
> > (*) is invisible to users....but this will not be very big problem.
> >
> Hi Kame,
>
> Thanks for the explanation.
>
> Hummm, Do you think that by doing it, we get rid of the need of moving
> sockets to another memcg when the task is moved? So in my original
> patchset, if you recall, I wanted to keep a socket forever in the same
> cgroup. I didn't, because then rmdir would be blocked.
>
> By using this memcg reference trick, both can be achieved. What do you
> think ?

I think so. Using mem_cgroup_put/get is a way. Could you try ?

Thanks,
-Kame
