Subject: [PATCH v9 0/9] Request for inclusion: per-cgroup tcp memory pressure controls
Posted by Glauber Costa on Mon, 12 Dec 2011 07:47:00 GMT
View Forum Message <> Reply to Message

Hi,

This series fixes all the few comments raised in the last round,
and seem to have acquired consensus from the memcg side.

Dave, do you think it is acceptable now from the networking PoV?
In case positive, would you prefer merging this trough your tree,
or acking this so a cgroup maintainer can do it?

Thanks

Glauber Costa (9):
  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

 Documentation/cgroups/memory.txt | 46 ++++++-
 include/linux/memcontrol.h | 23 ++++
 include/net/netns/ipv4.h | 1 +
 include/net/sock.h | 244 ++++++++++++++++++++++++++++++++++++-
 include/net/tcp.h | 4 +-
 include/net/tcp_memcontrol.h | 19 +++
 init/Kconfig | 11 ++
 mm/memcontrol.c | 191 ++++++++++++++++++++++++++++++
 net/core/sock.c | 112 ++++++++++++++----
 net/ipv4/Makefile | 1 +
 net/ipv4/af_inet.c | 2 +
 net/ipv4/proc.c | 6 +-
 net/ipv4/sysctl_net_ipv4.c | 65 ++++++++++-
 net/ipv4/tcp.c | 11 +--
 net/ipv4/tcp_input.c | 12 +-
 net/ipv4/tcp_ipv4.c | 14 ++-
 net/ipv4/tcp_memcontrol.c | 272 ++
 net/ipv4/tcp_output.c | 2 +-
 net/ipv4/tcp_timer.c | 2 +-
 net/ipv6/af_inet6.c | 2 +

```
 net/ipv6/tcp_ipv6.c            |    8 +-
 21 files changed, 973 insertions(+), 75 deletions(-)
 create mode 100644 include/net/tcp_memcontrol.h
 create mode 100644 net/ipv4/tcp_memcontrol.c

--
1.7.6.4
```

Subject: [PATCH v9 1/9] Basic kernel memory functionality for the Memory Controller
Posted by Glauber Costa on Mon, 12 Dec 2011 07:47:01 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>
CC: Kirill A. Shutemov <kirill@shutemov.name>
CC: Paul Menage <paul@paulmenage.org>
CC: Greg Thelen <gthelen@google.com>
CC: Johannes Weiner <jweiner@redhat.com>
CC: Michal Hocko <mhocko@suse.cz>

```
 Documentation/cgroups/memory.txt |   40 +++++++++++++++-
 init/Kconfig                     |   11 ++++
 mm/memcontrol.c                  |  105 ++++++++++++++++++++++++++++++++++++++++--
 3 files changed, 149 insertions(+), 7 deletions(-)
```

```
diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index cc0ebc5..f245324 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 +262,35 @@ 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.
+
+2.7.1 Current Kernel Memory resources accounted
+
+None
+
 3. User Interface

 0. Configuration
diff --git a/init/Kconfig b/init/Kconfig
index 43298f9..b8930d5 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -689,6 +689,17 @@ 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.

 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..9fbcff7 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 +4638,69 @@ 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;
+
+ /*
+  * This follows the same hierarchy restrictions than
+  * mem_cgroup_hierarchy_write()
+  */
+ if (!parent || !parent->use_hierarchy) {
+  if (list_empty(&cgroup->children))
+   memcg->kmem_independent_accounting = val;
+  else
+   return -EBUSY;
+ }
+ else
+  return -EINVAL;
```

```
+
+ 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 +5014,7 @@ 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 +5025,7 @@ 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 +5069,10 @@ 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 v9 2/9] foundations of per-cgroup memory pressure controlling.
Posted by Glauber Costa on Mon, 12 Dec 2011 07:47:02 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>
Reviewed-by: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
CC: David S. Miller <davem@davemloft.net>
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       |   57 ++++++++++++++++++++-----------
 net/ipv4/proc.c       |    6 ++--
 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, 145 insertions(+), 39 deletions(-)

diff --git a/include/net/sock.h b/include/net/sock.h
index abb6e0f..5f43fd9 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
+proto_sockets_allocated_sum_positive(struct proto *prot)
+{
```

```
+ return percpu_counter_sum_positive(prot->sockets_allocated);
+}
+
+static inline long
+proto_memory_allocated(struct proto *prot)
+{
+ return atomic_long_read(prot->memory_allocated);
+}
+
+static inline bool
+proto_memory_pressure(struct proto *prot)
+{
+ 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..c441d37 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 +1679,28 @@ 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;
+ if (allocated <= sk_prot_mem_limits(sk, 0)) {
+  sk_leave_memory_pressure(sk);
   return 1;
  }

  /* 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 +1710,13 @@ 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 +1739,9 @@ 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 +1752,13 @@ 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 +2474,27 @@ static char proto_method_implemented(const void *method)
 {
  return method == NULL ? 'n' : 'y';
 }
+static long sock_prot_memory_allocated(struct proto *proto)
+{
+ return proto->memory_allocated != NULL ? proto_memory_allocated(proto): -1L;
+}
+
+static char *sock_prot_memory_pressure(struct proto *proto)
+{
+ return proto->memory_pressure != NULL ?
+ proto_memory_pressure(proto) ? "yes" : "no" : "NI";
+}

 static void proto_seq_printf(struct seq_file *seq, struct proto *proto)
 {
+
  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),
+     sock_prot_memory_pressure(proto),
      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..91be152 100644
--- a/net/ipv4/proc.c
+++ b/net/ipv4/proc.c
@@ -56,17 +56,17 @@ static int sockstat_seq_show(struct seq_file *seq, void *v)

  local_bh_disable();
  orphans = percpu_counter_sum_positive(&tcp_orphan_count);
- sockets = percpu_counter_sum_positive(&tcp_sockets_allocated);
+ sockets = proto_sockets_allocated_sum_positive(&tcp_prot);
  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));
+     proto_memory_allocated(&tcp_prot));
  seq_printf(seq, "UDP: inuse %d mem %ld\n",
      sock_prot_inuse_get(net, &udp_prot),
-     atomic_long_read(&udp_memory_allocated));
+     proto_memory_allocated(&udp_prot));
  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 v9 3/9] socket: initial cgroup code.
Posted by Glauber Costa on Mon, 12 Dec 2011 07:47:03 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>
Reviewed-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujtsu.com>
CC: Kirill A. Shutemov <kirill@shutemov.name>
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 | 22 ++++++
 include/net/sock.h | 156 ++++++++++++++++++++++++++++++++++++--
 mm/memcontrol.c | 46 +++++++++++-
 net/core/sock.c | 24 ++++--
 5 files changed, 235 insertions(+), 17 deletions(-)

diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index f245324..23a8dc5 100644
--- a/Documentation/cgroups/memory.txt
+++ b/Documentation/cgroups/memory.txt
@@ -289,7 +289,9 @@ to trigger slab reclaim when those limits are reached.

 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..f15021b 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -85,6 +85,8 @@ 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 *parent_mem_cgroup(struct mem_cgroup *memcg);
+
 static inline
 int mm_match_cgroup(const struct mm_struct *mm, const struct mem_cgroup *cgroup)
 {
@@ -381,5 +383,25 @@ 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);
+void sock_release_memcg(struct sock *sk);
+#else
+static inline void sock_update_memcg(struct sock *sk)
+{
+}
+static inline void sock_release_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 5f43fd9..6cbee80 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 +838,37 @@ 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;
+ /*
+  * memcg field is used to find which memcg we belong directly
+  * Each memcg struct can hold more than one cg_proto, so container_of
+  * won't really cut.
+  *
+  * The elegant solution would be having an inverse function to
+  * proto_cgroup in struct proto, but that means polluting the structure
+  * for everybody, instead of just for memcg users.
+  */
+ struct mem_cgroup *memcg;
 };

 extern int proto_register(struct proto *prot, int alloc_slab);
@@ -852,7 +887,7 @@ static inline void sk_refcnt_debug_dec(struct sock *sk)
```

```
        sk->sk_prot->name, sk, atomic_read(&sk->sk_prot->socks));
 }

-static inline void sk_refcnt_debug_release(const struct sock *sk)
+inline void sk_refcnt_debug_release(const struct sock *sk)
 {
  if (atomic_read(&sk->sk_refcnt) != 1)
   printk(KERN_DEBUG "Destruction of the %s socket %p delayed, refcnt=%d\n",
@@ -864,6 +899,24 @@ static inline void sk_refcnt_debug_release(const struct sock *sk)
 #define sk_refcnt_debug_release(sk) do { } while (0)
 #endif /* SOCK_REFCNT_DEBUG */

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+extern struct jump_label_key memcg_socket_limit_enabled;
+static inline struct cg_proto *parent_cg_proto(struct proto *proto,
+           struct cg_proto *cg_proto)
+{
+ return proto->proto_cgroup(parent_mem_cgroup(cg_proto->memcg));
+}
+#define mem_cgroup_sockets_enabled static_branch(&memcg_socket_limit_enabled)
+#else
+#define mem_cgroup_sockets_enabled 0
+static inline struct cg_proto *parent_cg_proto(struct proto *proto,
+           struct cg_proto *cg_proto)
+{
+ return NULL;
+}
+#endif
+
+
 static inline bool sk_has_memory_pressure(const struct sock *sk)
 {
  return sk->sk_prot->memory_pressure != NULL;
@@ -873,6 +926,10 @@ static inline bool sk_under_memory_pressure(const struct sock *sk)
 {
  if (!sk->sk_prot->memory_pressure)
   return false;
+
+ if (mem_cgroup_sockets_enabled && sk->sk_cgrp)
+  return !!*sk->sk_cgrp->memory_pressure;
+
  return !!*sk->sk_prot->memory_pressure;
 }

@@ -880,52 +937,136 @@ 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;
+
+ if (*memory_pressure)
   *memory_pressure = 0;
+
+ if (mem_cgroup_sockets_enabled && sk->sk_cgrp) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+  struct proto *prot = sk->sk_prot;
+
+  for (; cg_proto; cg_proto = parent_cg_proto(prot, cg_proto))
+   if (*cg_proto->memory_pressure)
+    *cg_proto->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;
+
+ if (mem_cgroup_sockets_enabled && sk->sk_cgrp) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+  struct proto *prot = sk->sk_prot;
+
+  for (; cg_proto; cg_proto = parent_cg_proto(prot, cg_proto))
+   cg_proto->enter_memory_pressure(sk);
+ }
+
+ 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;
+ if (mem_cgroup_sockets_enabled && sk->sk_cgrp)
+  prot = sk->sk_cgrp->sysctl_mem;
  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;
+ if (mem_cgroup_sockets_enabled && sk->sk_cgrp)
+   return memcg_memory_allocated_read(sk->sk_cgrp);
+
  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;
+
+ if (mem_cgroup_sockets_enabled && sk->sk_cgrp) {
+   memcg_memory_allocated_add(sk->sk_cgrp, amt, parent_status);
+   /* update the root cgroup regardless */
+   atomic_long_add_return(amt, prot->memory_allocated);
+   return memcg_memory_allocated_read(sk->sk_cgrp);
+ }
+
  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;
+
+ if (mem_cgroup_sockets_enabled && sk->sk_cgrp &&
+     parent_status != OVER_LIMIT) /* Otherwise was uncharged already */
+  memcg_memory_allocated_sub(sk->sk_cgrp, amt);
+
  atomic_long_sub(amt, prot->memory_allocated);
 }

 static inline void sk_sockets_allocated_dec(struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;
+
+ if (mem_cgroup_sockets_enabled && sk->sk_cgrp) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  for (; cg_proto; cg_proto = parent_cg_proto(prot, cg_proto))
+   percpu_counter_dec(cg_proto->sockets_allocated);
+ }
+
  percpu_counter_dec(prot->sockets_allocated);
 }

 static inline void sk_sockets_allocated_inc(struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;
+
+ if (mem_cgroup_sockets_enabled && sk->sk_cgrp) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  for (; cg_proto; cg_proto = parent_cg_proto(prot, cg_proto))
+   percpu_counter_inc(cg_proto->sockets_allocated);
+ }
+
  percpu_counter_inc(prot->sockets_allocated);
 }

@@ -934,6 +1075,9 @@ sk_sockets_allocated_read_positive(struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;

+ if (mem_cgroup_sockets_enabled && sk->sk_cgrp)
+  return percpu_counter_sum_positive(sk->sk_cgrp->sockets_allocated);
```

```
+
  return percpu_counter_sum_positive(prot->sockets_allocated);
 }

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 9fbcff7..3de3901 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -379,7 +379,48 @@ enum mem_type {

 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);
+
+/* Writing them here to avoid exposing memcg's inner layout */
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+#ifdef CONFIG_INET
+#include <net/sock.h>
+
+static bool mem_cgroup_is_root(struct mem_cgroup *memcg);
+void sock_update_memcg(struct sock *sk)
+{
+ /* 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)) {
+   mem_cgroup_get(memcg);
+   sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
+  }
+  rcu_read_unlock();
+ }
+}
+EXPORT_SYMBOL(sock_update_memcg);
+
+void sock_release_memcg(struct sock *sk)
+{
+ if (static_branch(&memcg_socket_limit_enabled) && sk->sk_cgrp) {
+  struct mem_cgroup *memcg;
+  WARN_ON(!sk->sk_cgrp->memcg);
```

```
+  memcg = sk->sk_cgrp->memcg;
+  mem_cgroup_put(memcg);
+ }
+}
+#endif /* CONFIG_INET */
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+
 static void drain_all_stock_async(struct mem_cgroup *memcg);

 static struct mem_cgroup_per_zone *
@@ -4932,12 +4973,13 @@ 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;
  return mem_cgroup_from_res_counter(memcg->res.parent, res);
 }
+EXPORT_SYMBOL(parent_mem_cgroup);

 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_SWAP
 static void __init enable_swap_cgroup(void)
diff --git a/net/core/sock.c b/net/core/sock.c
index c441d37..02f32be 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
@@ -1677,23 +1681,27 @@ 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);
   return 1;
  }

- /* 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 */
@@ -1740,7 +1748,7 @@ 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;
 }
@@ -1753,7 +1761,7 @@ 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) &&
--
1.7.6.4
```

Subject: [PATCH v9 4/9] tcp memory pressure controls
Posted by Glauber Costa on Mon, 12 Dec 2011 07:47:04 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>
Reviewed-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujtisu.com>
CC: Eric W. Biederman <ebiederm@xmission.com>

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

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

+* tcp memory pressure: sockets memory pressure for the tcp protocol.
+
 3. User Interface

 0. Configuration
diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h

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

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

 static inline
 int mm_match_cgroup(const struct mm_struct *mm, const struct mem_cgroup *cgroup)
diff --git a/include/net/sock.h b/include/net/sock.h
index 6cbee80..1df44e2 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 3de3901..7266202 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -50,6 +50,8 @@
 #include <linux/cpu.h>
 #include <linux/oom.h>
 #include "internal.h"
+#include <net/sock.h>
+#include <net/tcp_memcontrol.h>

 #include <asm/uaccess.h>

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

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

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

@@ -800,7 +816,7 @@ static void memcg_check_events(struct mem_cgroup *memcg, struct
```

```
page *page)
  preempt_enable();
 }

-static struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont)
+struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont)
 {
  return container_of(cgroup_subsys_state(cont,
      mem_cgroup_subsys_id), struct mem_cgroup,
@@ -4732,14 +4748,34 @@ static int register_kmem_files(struct cgroup *cont, struct
cgroup_subsys *ss)

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

+static void kmem_cgroup_destroy(struct cgroup_subsys *ss,
+     struct cgroup *cont)
+{
+ mem_cgroup_sockets_destroy(cont, ss);
+}
 #else
 static int register_kmem_files(struct cgroup *cont, struct cgroup_subsys *ss)
 {
  return 0;
 }
+
+static void kmem_cgroup_destroy(struct cgroup_subsys *ss,
+     struct cgroup *cont)
+{
+}
 #endif

 static struct cftype mem_cgroup_files[] = {
@@ -5098,6 +5134,8 @@ 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 02f32be..5de62d3 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -135,6 +135,46 @@
 #include <net/tcp.h>
 #endif

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

```
+  if (proto->destroy_cgroup)
+   proto->destroy_cgroup(cgrp, ss);
+ read_unlock(&proto_list_lock);
+}
+#endif
+
 /*
  * Each address family might have different locking rules, so we have
  * one slock key per address family:
@@ -2258,9 +2298,6 @@ void sk_common_release(struct sock *sk)
 }
 EXPORT_SYMBOL(sk_common_release);

-static DEFINE_RWLOCK(proto_list_lock);
-static LIST_HEAD(proto_list);
-
 #ifdef CONFIG_PROC_FS
 #define PROTO_INUSE_NR 64 /* should be enough for the first time */
 struct prot_inuse {
diff --git a/net/ipv4/Makefile b/net/ipv4/Makefile
index f2dc69c..dc67a99 100644
--- a/net/ipv4/Makefile
+++ b/net/ipv4/Makefile
@@ -47,6 +47,7 @@ obj-$(CONFIG_TCP_CONG_SCALABLE) += tcp_scalable.o
 obj-$(CONFIG_TCP_CONG_LP) += tcp_lp.o
 obj-$(CONFIG_TCP_CONG_YEAH) += tcp_yeah.o
 obj-$(CONFIG_TCP_CONG_ILLINOIS) += tcp_illinois.o
+obj-$(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) += tcp_memcontrol.o
 obj-$(CONFIG_NETLABEL) += cipso_ipv4.o

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

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

  local_bh_disable();
+ sock_update_memcg(sk);
```

```
  sk_sockets_allocated_inc(sk);
  local_bh_enable();

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

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

@@ -2632,10 +2635,14 @@ struct proto tcp_prot = {
  .compat_setsockopt = compat_tcp_setsockopt,
  .compat_getsockopt = compat_tcp_getsockopt,
 #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..4a68d2c
--- /dev/null
+++ b/net/ipv4/tcp_memcontrol.c
@@ -0,0 +1,74 @@
+#include <net/tcp.h>
+#include <net/tcp_memcontrol.h>
+#include <net/sock.h>
+#include <linux/memcontrol.h>
+#include <linux/module.h>
+
+static inline struct tcp_memcontrol *tcp_from_cgproto(struct cg_proto *cg_proto)
+{
+ return container_of(cg_proto, struct tcp_memcontrol, cg_proto);
+}
+
+static void memcg_tcp_enter_memory_pressure(struct sock *sk)
+{
+ if (!sk->sk_cgrp->memory_pressure)
+  *sk->sk_cgrp->memory_pressure = 1;
```

```
+}
+EXPORT_SYMBOL(memcg_tcp_enter_memory_pressure);
+
+int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)
+{
+ /*
+  * The root cgroup does not use res_counters, but rather,
+  * rely on the data already collected by the network
+  * subsystem
+  */
+ struct res_counter *res_parent = NULL;
+ struct cg_proto *cg_proto, *parent_cg;
+ struct tcp_memcontrol *tcp;
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
+ struct mem_cgroup *parent = parent_mem_cgroup(memcg);
+
+ cg_proto = tcp_prot.proto_cgroup(memcg);
+ if (!cg_proto)
+  return 0;
+
+ tcp = tcp_from_cgproto(cg_proto);
+
+ tcp->tcp_prot_mem[0] = sysctl_tcp_mem[0];
+ tcp->tcp_prot_mem[1] = sysctl_tcp_mem[1];
+ tcp->tcp_prot_mem[2] = sysctl_tcp_mem[2];
+ tcp->tcp_memory_pressure = 0;
+
+ parent_cg = tcp_prot.proto_cgroup(parent);
+ if (parent_cg)
+  res_parent = parent_cg->memory_allocated;
+
+ res_counter_init(&tcp->tcp_memory_allocated, res_parent);
+ percpu_counter_init(&tcp->tcp_sockets_allocated, 0);
+
+ cg_proto->enter_memory_pressure = memcg_tcp_enter_memory_pressure;
+ cg_proto->memory_pressure = &tcp->tcp_memory_pressure;
+ cg_proto->sysctl_mem = tcp->tcp_prot_mem;
+ cg_proto->memory_allocated = &tcp->tcp_memory_allocated;
+ cg_proto->sockets_allocated = &tcp->tcp_sockets_allocated;
+ cg_proto->memcg = memcg;
+
+ return 0;
+}
+EXPORT_SYMBOL(tcp_init_cgroup);
+
+void tcp_destroy_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);
```

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

 #include <asm/uaccess.h>

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

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

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

 static const struct inet6_protocol tcpv6_protocol = {
--
1.7.6.4
```

Subject: [PATCH v9 5/9] per-netns ipv4 sysctl_tcp_mem
Posted by Glauber Costa on Mon, 12 Dec 2011 07:47:05 GMT

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>
Reviewed-by: 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 f70923e..cbba5aa 100644
```

```
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -2621,7 +2621,6 @@ 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 4a68d2c..bfb0c2b 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)
@@ -35,9 +38,9 @@ int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)

  tcp = tcp_from_cgproto(cg_proto);

- tcp->tcp_prot_mem[0] = sysctl_tcp_mem[0];
- tcp->tcp_prot_mem[1] = sysctl_tcp_mem[1];
- tcp->tcp_prot_mem[2] = sysctl_tcp_mem[2];
+ tcp->tcp_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;

  parent_cg = tcp_prot.proto_cgroup(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 820ae82..51bbfb0 100644
--- a/net/ipv6/tcp_ipv6.c
+++ b/net/ipv6/tcp_ipv6.c
@@ -2216,7 +2216,6 @@ 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 v9 6/9] tcp buffer limitation: per-cgroup limit
Posted by Glauber Costa on Mon, 12 Dec 2011 07:47:06 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>
Reviewed-by: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
CC: David S. Miller <davem@davemloft.net>
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       |  137 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-
 4 files changed, 152 insertions(+), 2 deletions(-)

diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index 687dea5..1c9779a 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 bfb0c2b..e353390 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);

@@ -57,7 +70,9 @@ int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)
  cg_proto->sockets_allocated = &tcp->tcp_sockets_allocated;
  cg_proto->memcg = memcg;

- return 0;
+create_files:
+ return cgroup_add_files(cgrp, ss, tcp_files,
+     ARRAY_SIZE(tcp_files));
 }
 EXPORT_SYMBOL(tcp_init_cgroup);

@@ -66,6 +81,7 @@ 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)
@@ -73,5 +89,122 @@ 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;
+
+ if (val > RESOURCE_MAX)
+  val = RESOURCE_MAX;
```

```
+
+ 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 && old_lim != RESOURCE_MAX)
+  jump_label_dec(&memcg_socket_limit_enabled);
+ else if (old_lim == RESOURCE_MAX && val != 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 v9 7/9] Display current tcp memory allocation in kmem cgroup
Posted by Glauber Costa on Mon, 12 Dec 2011 07:47:07 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>
Reviewed-by: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
CC: David S. Miller <davem@davemloft.net>
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 1c9779a..6922b6c 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 e353390..9481f23 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)
@@ -167,6 +172,19 @@ 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);
@@ -176,6 +194,9 @@ 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 v9 8/9] Display current tcp failcnt in kmem cgroup
Posted by Glauber Costa on Mon, 12 Dec 2011 07:47:08 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>
Reviewed-by: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>

```
CC: David S. Miller <davem@davemloft.net>
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 9481f23..d438fba 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)
@@ -197,12 +204,36 @@ 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 v9 9/9] Display maximum tcp memory allocation in kmem cgroup
Posted by Glauber Costa on Mon, 12 Dec 2011 07:47:09 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>
Reviewed-by: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
CC: David S. Miller <davem@davemloft.net>
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 d438fba..171d7b6 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)
@@ -205,7 +211,8 @@ 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();
@@ -226,6 +233,9 @@ 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: Re: [PATCH v9 0/9] Request for inclusion: per-cgroup tcp memory pressure controls

Posted by davem on Tue, 13 Dec 2011 00:07:34 GMT

View Forum Message <> Reply to Message

From: Glauber Costa <glommer@parallels.com>
Date: Mon, 12 Dec 2011 11:47:00 +0400

> This series fixes all the few comments raised in the last round,
> and seem to have acquired consensus from the memcg side.
>
> Dave, do you think it is acceptable now from the networking PoV?

> In case positive, would you prefer merging this trough your tree,
> or acking this so a cgroup maintainer can do it?

All applied to net-next, thanks.

Subject: Re: [PATCH v9 0/9] Request for inclusion: per-cgroup tcp memory pressure controls
Posted by Christoph Paasch on Tue, 13 Dec 2011 13:49:11 GMT
View Forum Message <> Reply to Message

Hi,

On 12/13/2011 01:07 AM, David Miller wrote:
> From: Glauber Costa <glommer@parallels.com>
> Date: Mon, 12 Dec 2011 11:47:00 +0400
>
>> This series fixes all the few comments raised in the last round,
>> and seem to have acquired consensus from the memcg side.
>>
>> Dave, do you think it is acceptable now from the networking PoV?
>> In case positive, would you prefer merging this trough your tree,
>> or acking this so a cgroup maintainer can do it?
>
> All applied to net-next, thanks.

now there are plenty of compiler-warnings when CONFIG_CGROUPS is not set:

In file included from include/linux/tcp.h:211:0,
                 from include/linux/ipv6.h:221,
                 from include/net/ip_vs.h:23,
                 from kernel/sysctl_binary.c:6:
include/net/sock.h:67:57: warning: ‘struct cgroup_subsys’ declared
inside parameter list [enabled by default]
include/net/sock.h:67:57: warning: its scope is only this definition or
declaration, which is probably not what you want [enabled by default]
include/net/sock.h:67:57: warning: ‘struct cgroup’ declared inside
parameter list [enabled by default]
include/net/sock.h:68:61: warning: ‘struct cgroup_subsys’ declared
inside parameter list [enabled by default]
include/net/sock.h:68:61: warning: ‘struct cgroup’ declared inside
parameter list [enabled by default]

Because struct cgroup is only declared if CONFIG_CGROUPS is enabled.
(cfr. linux/cgroup.h)

Christoph

--
Christoph Paasch
PhD Student

IP Networking Lab --- http://inl.info.ucl.ac.be
MultiPath TCP in the Linux Kernel --- http://mptcp.info.ucl.ac.be
Université Catholique de Louvain
--

Subject: Re: [PATCH v9 0/9] Request for inclusion: per-cgroup tcp memory pressure controls
Posted by Eric Dumazet on Tue, 13 Dec 2011 13:59:08 GMT
View Forum Message <> Reply to Message

Le mardi 13 décembre 2011 à 14:49 +0100, Christoph Paasch a écrit :

> now there are plenty of compiler-warnings when CONFIG_CGROUPS is not set:
>
> In file included from include/linux/tcp.h:211:0,
> from include/linux/ipv6.h:221,
> from include/net/ip_vs.h:23,
> from kernel/sysctl_binary.c:6:
> include/net/sock.h:67:57: warning: ‘struct cgroup_subsys’ declared
> inside parameter list [enabled by default]
> include/net/sock.h:67:57: warning: its scope is only this definition or
> declaration, which is probably not what you want [enabled by default]
> include/net/sock.h:67:57: warning: ‘struct cgroup’ declared inside
> parameter list [enabled by default]
> include/net/sock.h:68:61: warning: ‘struct cgroup_subsys’ declared
> inside parameter list [enabled by default]
> include/net/sock.h:68:61: warning: ‘struct cgroup’ declared inside
> parameter list [enabled by default]
>
>
> Because struct cgroup is only declared if CONFIG_CGROUPS is enabled.
> (cfr. linux/cgroup.h)
>

Yes, we probably need forward reference like this :

Thanks !

[PATCH net-next] net: fix build error if CONFIG_CGROUPS=n

Reported-by: Christoph Paasch <christoph.paasch@uclouvain.be>

Signed-off-by: Eric Dumazet <eric.dumazet@gmail.com>
---
 include/net/sock.h | 2 ++
 1 file changed, 2 insertions(+)

diff --git a/include/net/sock.h b/include/net/sock.h
index 18ecc99..6fe0dae 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -64,6 +64,8 @@
 #include <net/dst.h>
 #include <net/checksum.h>

+struct cgroup;
+struct cgroup_subsys;
 int mem_cgroup_sockets_init(struct cgroup *cgrp, struct cgroup_subsys *ss);
 void mem_cgroup_sockets_destroy(struct cgroup *cgrp, struct cgroup_subsys *ss);
 /*

Subject: Re: [PATCH v9 0/9] Request for inclusion: per-cgroup tcp memory pressure controls
Posted by davem on Tue, 13 Dec 2011 18:45:44 GMT
View Forum Message <> Reply to Message

From: Eric Dumazet <eric.dumazet@gmail.com>
Date: Tue, 13 Dec 2011 14:59:08 +0100

> [PATCH net-next] net: fix build error if CONFIG_CGROUPS=n
>
> Reported-by: Christoph Paasch <christoph.paasch@uclouvain.be>
> Signed-off-by: Eric Dumazet <eric.dumazet@gmail.com>

Applied, thanks.

Subject: Re: [PATCH v9 0/9] Request for inclusion: per-cgroup tcp memory pressure controls
Posted by Glauber Costa on Tue, 13 Dec 2011 20:11:00 GMT
View Forum Message <> Reply to Message

On 12/13/2011 05:59 PM, Eric Dumazet wrote:
> Le mardi 13 décembre 2011 à 14:49 +0100, Christoph Paasch a écrit :
>
>> now there are plenty of compiler-warnings when CONFIG_CGROUPS is not set:
>>
>> In file included from include/linux/tcp.h:211:0,

>> from include/linux/ipv6.h:221,
>> from include/net/ip_vs.h:23,
>> from kernel/sysctl_binary.c:6:
>> include/net/sock.h:67:57: warning: ‘struct cgroup_subsys’ declared
>> inside parameter list [enabled by default]
>> include/net/sock.h:67:57: warning: its scope is only this definition or
>> declaration, which is probably not what you want [enabled by default]
>> include/net/sock.h:67:57: warning: ‘struct cgroup’ declared inside
>> parameter list [enabled by default]
>> include/net/sock.h:68:61: warning: ‘struct cgroup_subsys’ declared
>> inside parameter list [enabled by default]
>> include/net/sock.h:68:61: warning: ‘struct cgroup’ declared inside
>> parameter list [enabled by default]
>>
>>
>> Because struct cgroup is only declared if CONFIG_CGROUPS is enabled.
>> (cfr. linux/cgroup.h)
>>
>
> Yes, we probably need forward reference like this :
>
> Thanks !
>
> [PATCH net-next] net: fix build error if CONFIG_CGROUPS=n
>
> Reported-by: Christoph Paasch<christoph.paasch@uclouvain.be>
> Signed-off-by: Eric Dumazet<eric.dumazet@gmail.com>
I am deeply sorry about that.
I was pretty sure I tested this case. But now that I looked into it, it
occurs to me that I may have tested it only with the Memory Cgroup
disabled, not with the master flag off.

Thanks for spotting this

Subject: Re: [PATCH v9 1/9] Basic kernel memory functionality for the Memory Controller
Posted by Michal Hocko on Wed, 14 Dec 2011 17:04:47 GMT
View Forum Message <> Reply to Message

[Now with the current patch version, I hope]

On Mon 12-12-11 11:47:01, Glauber Costa 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

Maybe has been already discussed but the name is rather awkward and it
would deserve more clarification. It is independent in the way that it
doesn't add up to the standard (user) allocations or it enables/disables
accounting?

> * memory.kmem.limit_in_bytes (currently ignored)

What happens if we reach the limit? Are all kernel allocations
considered or only selected caches? How do I find out which are those?

AFAIU you have implemented it for network buffers at this stage but I
guess that dentries will follow...

> * memory.kmem.usage_in_bytes (always zero)
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: Kirill A. Shutemov <kirill@shutemov.name>
> CC: Paul Menage <paul@paulmenage.org>
> CC: Greg Thelen <gthelen@google.com>
> CC: Johannes Weiner <jweiner@redhat.com>
> CC: Michal Hocko <mhocko@suse.cz>
> ---
> Documentation/cgroups/memory.txt | 40 +++++++++++++++-
> init/Kconfig | 11 ++++
> mm/memcontrol.c | 105 ++++++++++++++++++++++++++++++++++++++--
> 3 files changed, 149 insertions(+), 7 deletions(-)
>
> diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
> index cc0ebc5..f245324 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

Hugepages are not
Anyway this sounds outdated as we track both THP and hugetlb, right?

> + controls, we have to take care of performance. Kernel memory support is work
> + in progress, and the current version provides basically functionality.

s/basically/basic/

>
> 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
> +

It is not clear what happens in enabled/disabled cases. Let's say they
are not independent. Does it form a single limit with user charges or it
toggles kmem charging on/off.

> 1. History
>
> The memory controller has a long history. A request for comments for the memory
> @@ -255,6 +262,35 @@ 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.

How do you tell which are those that are accounted to the "main kmem"?

> +
> +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.

Separately from user space allocations, right?
What happens if we reach the limit in both cases?

> @@ -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,
> +};
> +

Probably in a separate (cleanup) patch?

> #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;
>   }

So you report kmem+user but we do not consider kmem during charge so one
can easily end up with usage_in_bytes over limit but no reclaim is going
on. Not good, I would say.

OK, so to sum it up. The biggest problem I see is the (non)independent
accounting. We simply cannot mix user+kernel limits otherwise we would
see issues (like kernel resource hog would force memcg-oom and innocent
members would die because their rss is much bigger).
It is also not clear to me what should happen when we hit the kmem
limit. I guess it will be kmem cache dependent.
--
Michal Hocko
SUSE Labs
SUSE LINUX s.r.o.
Lihovarska 1060/12
190 00 Praha 9
Czech Republic

Subject: Re: [PATCH v9 0/9] Request for inclusion: per-cgroup tcp memory pressure controls
Posted by KAMEZAWA Hiroyuki on Thu, 15 Dec 2011 05:40:19 GMT
View Forum Message <> Reply to Message

On Mon, 12 Dec 2011 11:47:00 +0400
Glauber Costa <glommer@parallels.com> wrote:

> Hi,
>
> This series fixes all the few comments raised in the last round,
> and seem to have acquired consensus from the memcg side.
>
> Dave, do you think it is acceptable now from the networking PoV?
> In case positive, would you prefer merging this trough your tree,
> or acking this so a cgroup maintainer can do it?
>

I met this bug at _1st_ run. Please enable _all_ debug options!.

>From this log, your mem_cgroup_sockets_init() is totally buggy because
tcp init routine will call percpu_alloc() under read_lock.

please fix. I'll turn off the config for today's linux-next.

Regards,
-Kame

==
Dec 15 14:47:15 bluextal kernel: [228.386054] BUG: sleeping function called from invalid context at k
ernel/mutex.c:271
Dec 15 14:47:15 bluextal kernel: [228.386286] in_atomic(): 1, irqs_disabled(): 0, pid: 2692, name: cg
create
Dec 15 14:47:15 bluextal kernel: [228.386438] 4 locks held by cgcreate/2692:
Dec 15 14:47:15 bluextal kernel: [228.386580] #0: (&sb->s_type->i_mutex_key#15/1){+.+.+.}, at: [<ff
ffffff8118717e>] kern_path_create+0x8e/0x150
Dec 15 14:47:15 bluextal kernel: [228.387238] #1: (cgroup_mutex){+.+.+.}, at: [<ffffffff810c2907>]
cgroup_mkdir+0x77/0x3f0
Dec 15 14:47:15 bluextal kernel: [228.387755] #2: (&sb->s_type->i_mutex_key#15/2){+.+.+.}, at: [<ff
ffffff810be168>] cgroup_create_file+0xc8/0xf0
Dec 15 14:47:15 bluextal kernel: [228.388401] #3: (proto_list_lock){++++.+}, at: [<ffffffff814e7fe4
>] mem_cgroup_sockets_init+0x24/0xf0
Dec 15 14:47:15 bluextal kernel: [228.388922] Pid: 2692, comm: cgcreate Not tainted 3.2.0-rc5-next-20
111214+ #34
Dec 15 14:47:15 bluextal kernel: [228.389154] Call Trace:
Dec 15 14:47:15 bluextal kernel: [228.389296] [<ffffffff81080f07>] __might_sleep+0x107/0x140
Dec 15 14:47:15 bluextal kernel: [228.389446] [<ffffffff815ce15f>] mutex_lock_nested+0x2f/0x50
Dec 15 14:47:15 bluextal kernel: [228.389597] [<ffffffff811354fd>] pcpu_alloc+0x4d/0xa20
Dec 15 14:47:15 bluextal kernel: [228.389745] [<ffffffff810a5759>] ? debug_check_no_locks_freed+0xa9/0x180
Dec 15 14:47:15 bluextal kernel: [228.389897] [<ffffffff810a5615>] ? trace_hardirqs_on_caller+0x105/0x190
Dec 15 14:47:15 bluextal kernel: [228.390057] [<ffffffff810a56ad>] ? trace_hardirqs_on+0xd/0x10
Dec 15 14:47:15 bluextal kernel: [228.390206] [<ffffffff810a3600>] ? lockdep_init_map+0x50/0x140
Dec 15 14:47:15 bluextal kernel: [228.390356] [<ffffffff81135f00>] __alloc_percpu+0x10/0x20
Dec 15 14:47:15 bluextal kernel: [228.390506] [<ffffffff812fda18>] __percpu_counter_init+0x58/0xc0
Dec 15 14:47:15 bluextal kernel: [228.390658] [<ffffffff8158fb26>] tcp_init_cgroup+0xd6/0x140
Dec 15 14:47:15 bluextal kernel: [228.390808] [<ffffffff814e8014>] mem_cgroup_sockets_init+0x54/0xf0
Dec 15 14:47:15 bluextal kernel: [228.390961] [<ffffffff8116b47b>]

mem_cgroup_populate+0xab/0xc0
Dec 15 14:47:15 bluextal kernel: [228.391120] [<ffffffff810c053a>]
cgroup_populate_dir+0x7a/0x110
Dec 15 14:47:15 bluextal kernel: [228.391271] [<ffffffff810c2b16>] cgroup_mkdir+0x286/0x3f0
Dec 15 14:47:15 bluextal kernel: [228.391420] [<ffffffff811836b4>] vfs_mkdir+0xa4/0xe0
Dec 15 14:47:15 bluextal kernel: [228.391566] [<ffffffff8118747d>] sys_mkdirat+0xcd/0xe0
Dec 15 14:47:15 bluextal kernel: [228.391714] [<ffffffff811874a8>] sys_mkdir+0x18/0x20
Dec 15 14:47:15 bluextal kernel: [228.391862] [<ffffffff815d86cf>] tracesys+0xdd/0xe2
==

Subject: Re: [PATCH v9 0/9] Request for inclusion: per-cgroup tcp memory pressure controls
Posted by davem on Thu, 15 Dec 2011 05:48:36 GMT
View Forum Message <> Reply to Message

From: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
Date: Thu, 15 Dec 2011 14:40:19 +0900

> I met this bug at _1st_ run. Please enable _all_ debug options!.

Plus the CONFIG_NET=n and other build failures.

This patch series was seriously rushed, and very poorly handled.

Yet I kept getting so much pressure to review, comment upon, and
ultimately apply these patches. Never, ever, do this to me ever
again.

If I don't feel your patches are high priority enough or ready enough
for me to review, then TOO BAD. Don't ask people to pressure me or
get my attention. Instead, ask others for help and do testing before
wasting MY time and crapping up MY tree.

I should have noticed a red flag when I have James Bottomly asking me
to look at these patches, I should have pushed back. Instead, I
relented, and now I'm very seriously regretting it.

All the regressions in the net-next tree over the past several days
have been due to this patch set, and this patch set alone.

This code wasn't ready and needed, at a minimum, several more weeks of
work before being put in.

Instead, we're going to bandaid patch it up after the fact, rather
than just letting these changes mature naturally during the review
process.

Subject: Re: [PATCH v9 0/9] Request for inclusion: per-cgroup tcp memory pressure controls
Posted by Glauber Costa on Thu, 15 Dec 2011 06:48:26 GMT
View Forum Message <> Reply to Message

On 12/15/2011 09:48 AM, David Miller wrote:
> From: KAMEZAWA Hiroyuki<kamezawa.hiroyu@jp.fujitsu.com>
> Date: Thu, 15 Dec 2011 14:40:19 +0900
>
>> I met this bug at _1st_ run. Please enable _all_ debug options!.
>
> Plus the CONFIG_NET=n and other build failures.
>
> This patch series was seriously rushed, and very poorly handled.
>
> Yet I kept getting so much pressure to review, comment upon, and
> ultimately apply these patches. Never, ever, do this to me ever
> again.
>
> If I don't feel your patches are high priority enough or ready enough
> for me to review, then TOO BAD. Don't ask people to pressure me or
> get my attention. Instead, ask others for help and do testing before
> wasting MY time and crapping up MY tree.
>
> I should have noticed a red flag when I have James Bottomly asking me
> to look at these patches, I should have pushed back. Instead, I
> relented, and now I'm very seriously regretting it.
>
> All the regressions in the net-next tree over the past several days
> have been due to this patch set, and this patch set alone.
>
> This code wasn't ready and needed, at a minimum, several more weeks of
> work before being put in.
>
> Instead, we're going to bandaid patch it up after the fact, rather
> than just letting these changes mature naturally during the review
> process.
Hi Dave,

You are right about all points. I will admit to it, face it, and
apologize it.
I guess the best I can do right now is fix whatever you guys point me to
and not repeat it in the future.

Thanks

Subject: Re: [PATCH v9 1/9] Basic kernel memory functionality for the Memory

On 12/14/2011 09:04 PM, Michal Hocko wrote:
> [Now with the current patch version, I hope]
>
> On Mon 12-12-11 11:47:01, Glauber Costa 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
>
> Maybe has been already discussed but the name is rather awkward and it
> would deserve more clarification. It is independent in the way that it
> doesn't add up to the standard (user) allocations or it enables/disables
> accounting?

If turned on, it doesn't add up to the user allocations.
As for the name, this is marked experimental, so I don't think anyone
will be relying on it for a while. We can change it, if you have a
better suggestion.

>> * memory.kmem.limit_in_bytes (currently ignored)
>
> What happens if we reach the limit? Are all kernel allocations
> considered or only selected caches? How do I find out which are those?
>
> AFAIU you have implemented it for network buffers at this stage but I
> guess that dentries will follow...

Further allocations should fail.

About other caches, tcp is a bit different because we are concerned with
conditions that applies after the allocation already took place. It is
not clear to me if we will treat the other caches as a single entity, or
separate them.

>> * memory.kmem.usage_in_bytes (always zero)
>>
>> Signed-off-by: Glauber Costa<glommer@parallels.com>
>> CC: Kirill A. Shutemov<kirill@shutemov.name>
>> CC: Paul Menage<paul@paulmenage.org>
>> CC: Greg Thelen<gthelen@google.com>
>> CC: Johannes Weiner<jweiner@redhat.com>
>> CC: Michal Hocko<mhocko@suse.cz>
>> ---

>> Documentation/cgroups/memory.txt | 40 +++++++++++++-
>> init/Kconfig | 11 ++++
>> mm/memcontrol.c | 105 +++++++++++++++++++++++++++++++++++++++--
>> 3 files changed, 149 insertions(+), 7 deletions(-)
>>
>> diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
>> index cc0ebc5..f245324 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
>
> Hugepages are not
> Anyway this sounds outdated as we track both THP and hugetlb, right?
>
>> + controls, we have to take care of performance. Kernel memory support is work
>> + in progress, and the current version provides basically functionality.
>
> s/basically/basic/
>
>>
>> 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
>> +
>

> It is not clear what happens in enabled/disabled cases. Let's say they
> are not independent. Does it form a single limit with user charges or it
> toggles kmem charging on/off.
>
>>   1. History
>>
>>   The memory controller has a long history. A request for comments for the memory
>> @@ -255,6 +262,35 @@ 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.
>
> How do you tell which are those that are accounted to the "main kmem"?

Besides being in this list, they should have they own files, like tcp.
>
>> +
>> +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.
>
> Separately from user space allocations, right?
Yes.
> What happens if we reach the limit in both cases?
For kernel memory, further allocations should fail.

>
>> @@ -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,
>> +};
>> +
>
> Probably in a separate (cleanup) patch?
>
>> #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;
>> }
>
> So you report kmem+user but we do not consider kmem during charge so one
> can easily end up with usage_in_bytes over limit but no reclaim is going
> on. Not good, I would say.
>
> OK, so to sum it up. The biggest problem I see is the (non)independent
> accounting. We simply cannot mix user+kernel limits otherwise we would
> see issues (like kernel resource hog would force memcg-oom and innocent
> members would die because their rss is much bigger).
> It is also not clear to me what should happen when we hit the kmem
> limit. I guess it will be kmem cache dependent.

So right now, tcp is completely independent, since it is not accounted
to kmem. In summary, we still never do non-independent accounting. When
we start doing it for the other caches, We will have to add a test at

charge time as well.

We still need to keep it separate though, in case the independent flag
is turned on/off

Subject: Re: [PATCH v9 1/9] Basic kernel memory functionality for the Memory Controller
Posted by Greg Thelen on Fri, 16 Dec 2011 06:20:52 GMT
View Forum Message <> Reply to Message

On Sun, Dec 11, 2011 at 11:47 PM, Glauber Costa <glommer@parallels.com> wrote:
> +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

s/Value/value/

It is probably worth documenting the default value for
memory.independent_kmem_limit? I figure it would be zero at root and
and inherited from parents. But I think the implementation differs.

```
> @@ -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;
```

I have no serious objection, but a full int seems like overkill for a
boolean value.

```
> +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;
```

If you want to this function could be condensed down to:
  return cgroup_add_files(...);

Subject: Re: [PATCH v9 1/9] Basic kernel memory functionality for the Memory

On Thu 15-12-11 16:29:18, Glauber Costa wrote:
> On 12/14/2011 09:04 PM, Michal Hocko wrote:
> >[Now with the current patch version, I hope]
> >On Mon 12-12-11 11:47:01, Glauber Costa wrote:
[...]
> >>@@ -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;
> >> }
> >
> >So you report kmem+user but we do not consider kmem during charge so one
> >can easily end up with usage_in_bytes over limit but no reclaim is going
> >on. Not good, I would say.

I find this a problem and one of the reason I do not like !independent
accounting.

> >
> >OK, so to sum it up. The biggest problem I see is the (non)independent
> >accounting. We simply cannot mix user+kernel limits otherwise we would
> >see issues (like kernel resource hog would force memcg-oom and innocent
> >members would die because their rss is much bigger).
> >It is also not clear to me what should happen when we hit the kmem
> >limit. I guess it will be kmem cache dependent.
>
> So right now, tcp is completely independent, since it is not
> accounted to kmem.

So why do we need kmem accounting when tcp (the only user at the moment)
doesn't use it?

> In summary, we still never do non-independent accounting. When we
> start doing it for the other caches, We will have to add a test at
> charge time as well.

So we shouldn't do it as a part of this patchset because the further usage is not clear and I think there will be some real issues with user+kmem accounting (e.g. a proper memcg-oom implementation). Can you just drop this patch?

> We still need to keep it separate though, in case the independent
> flag is turned on/off

I don't mind to have kmem.tcp.* knobs.

--
Michal Hocko
SUSE Labs
SUSE LINUX s.r.o.
Lihovarska 1060/12
190 00 Praha 9
Czech Republic

Subject: Re: [PATCH v9 1/9] Basic kernel memory functionality for the Memory Controller
Posted by Glauber Costa on Fri, 16 Dec 2011 13:02:51 GMT
View Forum Message <> Reply to Message

On 12/16/2011 04:32 PM, Michal Hocko wrote:
> On Thu 15-12-11 16:29:18, Glauber Costa wrote:
>> On 12/14/2011 09:04 PM, Michal Hocko wrote:
>>> [Now with the current patch version, I hope]
>>> On Mon 12-12-11 11:47:01, Glauber Costa wrote:
> [...]
>>>> @@ -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;
>>>>   }
>>>
>>> So you report kmem+user but we do not consider kmem during charge so one
>>> can easily end up with usage_in_bytes over limit but no reclaim is going
>>> on. Not good, I would say.
>
> I find this a problem and one of the reason I do not like !independent
> accounting.
>
>>>
>>> OK, so to sum it up. The biggest problem I see is the (non)independent
>>> accounting. We simply cannot mix user+kernel limits otherwise we would
>>> see issues (like kernel resource hog would force memcg-oom and innocent
>>> members would die because their rss is much bigger).
>>> It is also not clear to me what should happen when we hit the kmem
>>> limit. I guess it will be kmem cache dependent.
>>
>> So right now, tcp is completely independent, since it is not
>> accounted to kmem.
>
> So why do we need kmem accounting when tcp (the only user at the moment)
> doesn't use it?

Well, a bit historical. I needed a basic placeholder for it, since it
tcp is officially kmem. As the time passed, I took most of the stuff out
of this patch to leave just the basics I would need for tcp.
Turns out I ended up focusing on the rest, and some of the stuff was
left here.

At one point I merged tcp data into kmem, but then reverted this
behavior. the kmem counter stayed.

I agree deferring the whole behavior would be better.

>> In summary, we still never do non-independent accounting. When we
>> start doing it for the other caches, We will have to add a test at
>> charge time as well.
>
> So we shouldn't do it as a part of this patchset because the further
> usage is not clear and I think there will be some real issues with
> user+kmem accounting (e.g. a proper memcg-oom implementation).
> Can you just drop this patch?

Yes, but the whole set is in the net tree already. (All other patches
are tcp-related but this) Would you mind if I'd send a follow up patch
removing the kmem files, and leaving just the registration functions and
basic documentation? (And sorry for that as well in advance)

>> We still need to keep it separate though, in case the independent
>> flag is turned on/off
>
> I don't mind to have kmem.tcp.* knobs.
>

Subject: Re: [PATCH v9 1/9] Basic kernel memory functionality for the Memory Controller
Posted by Michal Hocko on Fri, 16 Dec 2011 13:30:49 GMT
View Forum Message <> Reply to Message

On Fri 16-12-11 17:02:51, Glauber Costa wrote:
> On 12/16/2011 04:32 PM, Michal Hocko wrote:
[...]
> > >So why do we need kmem accounting when tcp (the only user at the moment)
> > >doesn't use it?
>
> Well, a bit historical. I needed a basic placeholder for it, since
> it tcp is officially kmem. As the time passed, I took most of the
> stuff out of this patch to leave just the basics I would need for
> tcp.
> Turns out I ended up focusing on the rest, and some of the stuff was
> left here.
>
> At one point I merged tcp data into kmem, but then reverted this
> behavior. the kmem counter stayed.
>
> I agree deferring the whole behavior would be better.
>
> >>In summary, we still never do non-independent accounting. When we
> >>start doing it for the other caches, We will have to add a test at
> >>charge time as well.
> >
> >So we shouldn't do it as a part of this patchset because the further
> >usage is not clear and I think there will be some real issues with
> >user+kmem accounting (e.g. a proper memcg-oom implementation).
> >Can you just drop this patch?
>
> Yes, but the whole set is in the net tree already.

Isn't it only in some for-next branch? Can that one be updated?

> (All other patches are tcp-related but this) Would you mind if I'd
> send a follow up patch removing the kmem files, and leaving just the
> registration functions and basic documentation? (And sorry for that as
> well in advance)

Yes a followup patch would work as well.

--
Michal Hocko
SUSE Labs
SUSE LINUX s.r.o.
Lihovarska 1060/12
190 00 Praha 9
Czech Republic

Subject: Re: [PATCH v9 3/9] socket: initial cgroup code.
Posted by Jason Baron on Thu, 22 Dec 2011 21:10:29 GMT
View Forum Message <> Reply to Message

On Mon, Dec 12, 2011 at 11:47:03AM +0400, Glauber Costa wrote:
> +
> +static bool mem_cgroup_is_root(struct mem_cgroup *memcg);
> +void sock_update_memcg(struct sock *sk)
> +{
> + /* 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)) {
> + mem_cgroup_get(memcg);
> + sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
> + }
> + rcu_read_unlock();
> + }
> +}
> +EXPORT_SYMBOL(sock_update_memcg);
> +
> +void sock_release_memcg(struct sock *sk)
> +{

> + if (static_branch(&memcg_socket_limit_enabled) && sk->sk_cgrp) {
> + struct mem_cgroup *memcg;
> + WARN_ON(!sk->sk_cgrp->memcg);
> + memcg = sk->sk_cgrp->memcg;
> + mem_cgroup_put(memcg);
> + }
> +}

Hi Glauber,

I think for 'sock_release_memcg()', you want:

static inline sock_release_memcg(sk)
{
 if (static_branch())
  __sock_release_memcg();
}

And then re-define the current sock_release_memcg -> __sock_release_memcg().
In that way the straight line path is a single no-op. As currently
written, there is function call and then an immediate return.

Thanks,

-Jason

Subject: Re: [PATCH v9 3/9] socket: initial cgroup code.
Posted by Glauber Costa on Fri, 23 Dec 2011 08:57:49 GMT
View Forum Message <> Reply to Message

On 12/23/2011 01:10 AM, Jason Baron wrote:
> On Mon, Dec 12, 2011 at 11:47:03AM +0400, Glauber Costa wrote:
>> +
>> +static bool mem_cgroup_is_root(struct mem_cgroup *memcg);
>> +void sock_update_memcg(struct sock *sk)
>> +{
>> + /* 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)) {
>> +   mem_cgroup_get(memcg);
>> +   sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
>> +  }
>> +  rcu_read_unlock();
>> + }
>> +}
>> +EXPORT_SYMBOL(sock_update_memcg);
>> +
>> +void sock_release_memcg(struct sock *sk)
>> +{
>> + if (static_branch(&memcg_socket_limit_enabled)&&  sk->sk_cgrp) {
>> +  struct mem_cgroup *memcg;
>> +  WARN_ON(!sk->sk_cgrp->memcg);
>> +  memcg = sk->sk_cgrp->memcg;
>> +  mem_cgroup_put(memcg);
>> + }
>> +}
>
> Hi Glauber,
>
> I think for 'sock_release_memcg()', you want:
>
> static inline sock_release_memcg(sk)
> {
>  if (static_branch())
>   __sock_release_memcg();
> }
>
> And then re-define the current sock_release_memcg ->  __sock_release_memcg().
> In that way the straight line path is a single no-op. As currently
> written, there is function call and then an immediate return.
>
```

Hello Jason,

Thanks for the tip. I may be wrong here, but I don't think that the release performance matters to that level. But your suggestion seems good nevertheless. Since this is already sitting on a tree, would you like to send a patch for that?