Subject: [PATCH v6 00/10] Request for inclusion: per-cgroup tcp memory pressure controls
Posted by Glauber Costa on Fri, 25 Nov 2011 17:38:06 GMT
View Forum Message <> Reply to Message

Hi Dave,

I hope the following series is in an acceptable state: I modified the tests in __sk_mem_schedule() in a way that we should still leave the function pretty soon under no pressure conditions.

Also, I managed to remove almost everything tcp related from memcontrol.c: the only thing left is a simple function to calculate the address of the tcp sub-structure in the main memcg structure - I hope this is acceptable, since besides being simple, it is not related to the protocol itself.

I also believed that by now, all other comments so far were addressed. Let me know if there are any blocking concerns to this, and I'll address them as soon as I can.

Thanks

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

 Documentation/cgroups/memory.txt | 38 +++++-
 include/linux/memcontrol.h | 19 +++
 include/net/netns/ipv4.h | 1 +
 include/net/sock.h | 232 ++++++++++++++++++++++++++++++++++++++
 include/net/tcp.h | 4 +-
 include/net/tcp_memcg.h | 20 +++
 init/Kconfig | 14 ++
 mm/memcontrol.c | 209 +++++++++++++++++++++++++++++++++--
 net/core/sock.c | 106 +++++++++++++-----
 net/ipv4/Makefile | 1 +
 net/ipv4/af_inet.c | 2 +
 net/ipv4/proc.c | 7 +-
 net/ipv4/sysctl_net_ipv4.c | 65 +++++++++-
 net/ipv4/tcp.c | 17 +--
 net/ipv4/tcp_input.c | 12 +-

```
 net/ipv4/tcp_ipv4.c           |   13 ++-
 net/ipv4/tcp_memcg.c          |  263 ++++++++++++++++++++++++++++++++++++++++
 net/ipv4/tcp_output.c         |    2 +-
 net/ipv4/tcp_timer.c          |    2 +-
 net/ipv6/af_inet6.c           |    2 +
 net/ipv6/tcp_ipv6.c           |    7 +-
 21 files changed, 955 insertions(+), 81 deletions(-)
 create mode 100644 include/net/tcp_memcg.h
 create mode 100644 net/ipv4/tcp_memcg.c

--
1.7.6.4
```

Subject: [PATCH v6 01/10] Basic kernel memory functionality for the Memory Controller

Posted by Glauber Costa on Fri, 25 Nov 2011 17:38:07 GMT

View Forum Message <> Reply to Message

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

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

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

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

```
 Documentation/cgroups/memory.txt |   36 ++++++++++++-
 init/Kconfig                     |   14 +++++
 mm/memcontrol.c                  |  107 ++++++++++++++++++++++++++++++++++++--
 3 files changed, 150 insertions(+), 7 deletions(-)
```

```
diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index 06eb6d9..bf00cd2 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 +262,31 @@ 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.
+Kernel memory limits are not imposed for the root cgroup.
+
+Memory limits as specified by the standard Memory Controller may or may not
+take kernel memory into consideration. This is achieved through the file
+memory.independent_kmem_limit. A Value different than 0 will allow for kernel
+memory to be controlled separately.
+
+When kernel memory limits are not independent, the limit values set in
+memory.kmem files are ignored.
+
```

```
+Currently no soft limit is implemented for kernel memory. It is future work
+to trigger slab reclaim when those limits are reached.
+
+CAUTION: As of this writing, the kmem extention may prevent tasks from moving
+among cgroups. If a task has kmem accounting in a cgroup, the task cannot be
+moved until the kmem resource is released. Also, until the resource is fully
+released, the cgroup cannot be destroyed. So, please consider your use cases
+and set kmem extention config option carefully.
+
 3. User Interface

 0. Configuration
diff --git a/init/Kconfig b/init/Kconfig
index 31ba0fd..e4b6246 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -689,6 +689,20 @@ config CGROUP_MEM_RES_CTLR_SWAP_ENABLED
    For those who want to have the feature enabled by default should
    select this option (if, for some reason, they need to disable it
    then swapaccount=0 does the trick).
+config CGROUP_MEM_RES_CTLR_KMEM
+ bool "Memory Resource Controller Kernel Memory accounting (EXPERIMENTAL)"
+ depends on CGROUP_MEM_RES_CTLR && EXPERIMENTAL
+ default n
+ help
+   The Kernel Memory extension for Memory Resource Controller can limit
+   the amount of memory used by kernel objects in the system. Those are
+   fundamentally different from the entities handled by the standard
+   Memory Controller, which are page-based, and can be swapped. Users of
+   the kmem extension can use it to guarantee that no group of processes
+   will ever exhaust kernel resources alone.
+
+   WARNING: The current experimental implementation does not allow a
+   task to move among different cgroups with a kmem resource being held.

 config CGROUP_PERF
   bool "Enable perf_event per-cpu per-container group (cgroup) monitoring"
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 2d57555..1cb7daa 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -226,6 +226,10 @@ 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.
   */
@@ -276,6 +280,11 @@ 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;
@@ -343,9 +352,14 @@ 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)
@@ -3838,10 +3852,17 @@ static inline u64 mem_cgroup_usage(struct mem_cgroup *mem,
bool swap)
  u64 val;

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

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

   val = mem_cgroup_recursive_stat(mem, MEM_CGROUP_STAT_CACHE);
@@ -3874,6 +3895,11 @@ static u64 mem_cgroup_read(struct cgroup *cont, struct cftype *cft)
    else
     val = res_counter_read_u64(&mem->memsw, name);
    break;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ case _KMEM:
+   val = res_counter_read_u64(&mem->kmem, name);
+   break;
+#endif
   default:
    BUG();
    break;
@@ -4604,6 +4630,35 @@ static int mem_control_numa_stat_open(struct inode *unused, struct
file *file)
 }
 #endif /* CONFIG_NUMA */

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

```
+ cgroup_unlock();
+ return 0;
+}
+#endif
+
 static struct cftype mem_cgroup_files[] = {
  {
   .name = "usage_in_bytes",
@@ -4719,6 +4774,42 @@ static int register_memsw_files(struct cgroup *cont, struct
cgroup_subsys *ss)
 }
 #endif

+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+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 int alloc_mem_cgroup_per_zone_info(struct mem_cgroup *mem, int node)
 {
  struct mem_cgroup_per_node *pn;
@@ -4917,6 +5008,7 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
  if (parent && parent->use_hierarchy) {
   res_counter_init(&mem->res, &parent->res);
   res_counter_init(&mem->memsw, &parent->memsw);
+  res_counter_init(&mem->kmem, &parent->kmem);
   /*
    * We increment refcnt of the parent to ensure that we can
    * safely access it on res_counter_charge/uncharge.
@@ -4927,6 +5019,7 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
  } else {
   res_counter_init(&mem->res, NULL);
   res_counter_init(&mem->memsw, NULL);
+  res_counter_init(&mem->kmem, NULL);
  }
  mem->last_scanned_child = 0;
  mem->last_scanned_node = MAX_NUMNODES;
@@ -4970,6 +5063,10 @@ 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 v6 02/10] foundations of per-cgroup memory pressure controlling.
Posted by Glauber Costa on Fri, 25 Nov 2011 17:38:08 GMT

View Forum Message <> Reply to Message

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

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

Signed-off-by: Glauber Costa <glommer@parallels.com>

```
CC: David S. Miller <davem@davemloft.net>
CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
CC: Eric W. Biederman <ebiederm@xmission.com>
CC: Eric Dumazet <eric.dumazet@gmail.com>
---
 include/net/sock.h    |   64 ++++++++++++++++++++++++++++++++++++++++++++++++++
 include/net/tcp.h     |    3 +-
 net/core/sock.c       |   55 ++++++++++++++++++++++++++++++++----------------
 net/ipv4/proc.c       |    7 +++--
 net/ipv4/tcp.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 +-
 10 files changed, 118 insertions(+), 39 deletions(-)

diff --git a/include/net/sock.h b/include/net/sock.h
index c6658be..0d054e0 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -54,6 +54,7 @@
 #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 +864,69 @@ 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 int *sk_memory_pressure(const struct sock *sk)
+{
+ return sk->sk_prot->memory_pressure;
+}
+
+static inline long sk_prot_mem(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
+kcg_sockets_allocated_sum_positive(struct proto *prot, struct mem_cgroup *cg)
+{
+ return percpu_counter_sum_positive(prot->sockets_allocated);
+}
+
+static inline long
+kcg_memory_allocated(struct proto *prot, struct mem_cgroup *cg)
+{
+ return atomic_long_read(prot->memory_allocated);
+}
```

```
 #ifdef CONFIG_PROC_FS
 /* Called with local bh disabled */
diff --git a/include/net/tcp.h b/include/net/tcp.h
index e147f42..ccaa3b6 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(sk, 2))
   return true;
  return false;
 }
diff --git a/net/core/sock.c b/net/core/sock.c
index 4ed7b1d..26bdb1c 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))
@@ -1677,30 +1677,32 @@ 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 *memory_pressure;

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

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

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

  /* Over hard limit. */
- if (allocated > prot->sysctl_mem[2])
+ if (allocated > sk_prot_mem(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 +1712,13 @@ int __sk_mem_schedule(struct sock *sk, int size, int kind)
     return 1;
  }

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

-  if (!*prot->memory_pressure)
+  if (!*memory_pressure)
    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(sk, 2) > alloc *
       sk_mem_pages(sk->sk_wmem_queued +
      atomic_read(&sk->sk_rmem_alloc) +
      sk->sk_forward_alloc))
```

```
@@ -1739,7 +1741,9 @@ 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 +1754,15 @@ EXPORT_SYMBOL(__sk_mem_schedule);
  */
 void __sk_mem_reclaim(struct sock *sk)
 {
- struct proto *prot = sk->sk_prot;
+ int *memory_pressure = sk_memory_pressure(sk);

- 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 (memory_pressure && *memory_pressure &&
+     (sk_memory_allocated(sk) < sk_prot_mem(sk, 0)))
+  *memory_pressure = 0;
 }
 EXPORT_SYMBOL(__sk_mem_reclaim);

@@ -2477,13 +2481,20 @@ static char proto_method_implemented(const void *method)

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

```
       sock_prot_inuse_get(seq_file_net(seq), proto),
-      proto->memory_allocated != NULL ? atomic_long_read(proto->memory_allocated) : -1L,
-      proto->memory_pressure != NULL ? *proto->memory_pressure ? "yes" : "no" : "NI",
+      proto->memory_allocated != NULL ?
+    kcg_memory_allocated(proto, cg) : -1L,
+      memory_pressure != NULL ? *memory_pressure ? "yes" : "no" : "NI",
       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 4bfad5d..535456d 100644
--- a/net/ipv4/proc.c
+++ b/net/ipv4/proc.c
@@ -52,20 +52,21 @@ static int sockstat_seq_show(struct seq_file *seq, void *v)
 {
  struct net *net = seq->private;
  int orphans, sockets;
+ struct mem_cgroup *cg = mem_cgroup_from_task(current);

  local_bh_disable();
  orphans = percpu_counter_sum_positive(&tcp_orphan_count);
- sockets = percpu_counter_sum_positive(&tcp_sockets_allocated);
+ sockets = kcg_sockets_allocated_sum_positive(&tcp_prot, cg);
  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));
+      kcg_memory_allocated(&tcp_prot, cg));
  seq_printf(seq, "UDP: inuse %d mem %ld\n",
       sock_prot_inuse_get(net, &udp_prot),
-      atomic_long_read(&udp_memory_allocated));
+      kcg_memory_allocated(&udp_prot, cg));
  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.c b/net/ipv4/tcp.c
index 34f5db1..89a2bfe 100644
--- a/net/ipv4/tcp.c
+++ b/net/ipv4/tcp.c
@@ -319,9 +319,11 @@ EXPORT_SYMBOL(tcp_memory_pressure);

 void tcp_enter_memory_pressure(struct sock *sk)
 {
- if (!tcp_memory_pressure) {
+ int *memory_pressure = sk_memory_pressure(sk);
```

```
+
+ if (!*memory_pressure) {
   NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPMEMORYPRESSURES);
-  tcp_memory_pressure = 1;
+  *memory_pressure = 1;
  }
 }
 EXPORT_SYMBOL(tcp_enter_memory_pressure);
diff --git a/net/ipv4/tcp_input.c b/net/ipv4/tcp_input.c
index 52b5c2d..3df862d 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_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_memory_pressure(sk) &&
+     sk_memory_allocated(sk) < sk_prot_mem(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_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_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(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 0ea10ee..f124a4b 100644
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -1914,7 +1914,7 @@ 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;
@@ -1970,7 +1970,7 @@ 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..04e229b 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_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..c9f830c 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_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 10b2b31..3a08fcd 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 v6 03/10] socket: initial cgroup code.
Posted by Glauber Costa on Fri, 25 Nov 2011 17:38:09 GMT
View Forum Message <> Reply to Message

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

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

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

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

```
---
 include/linux/memcontrol.h |   16 ++++
 include/net/sock.h         |  169 ++++++++++++++++++++++++++++++++++++++++++++-
 mm/memcontrol.c            |   40 +++++++++--
 net/core/sock.c            |   21 ++++--
 4 files changed, 230 insertions(+), 16 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index ac797fa..6644d90 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -377,5 +377,21 @@ mem_cgroup_print_bad_page(struct page *page)
 }
 #endif

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

diff --git a/include/net/sock.h b/include/net/sock.h
index 0d054e0..d802761 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -55,6 +55,7 @@
 #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>
@@ -169,6 +170,7 @@ struct sock_common {
   /* public: */
 };
```

```
+struct cg_proto;
 /**
   * struct sock - network layer representation of sockets
   * @__sk_common: shared layout with inet_timewait_sock
@@ -229,6 +231,7 @@ 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
@@ -340,6 +343,7 @@ 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 +838,27 @@ 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 {
+ struct res_counter *memory_allocated; /* Current allocated memory. */
+ struct percpu_counter *sockets_allocated; /* Current number of sockets. */
+ int   *memory_pressure;
+ long   *sysctl_mem;
+ struct cg_proto  *parent;
 };
```

```
 extern int proto_register(struct proto *prot, int alloc_slab);
@@ -864,47 +889,149 @@ static inline void sk_refcnt_debug_release(const struct sock *sk)
 #define sk_refcnt_debug_release(sk) do { } while (0)
 #endif /* SOCK_REFCNT_DEBUG */

+extern struct jump_label_key memcg_socket_limit_enabled;
 static inline int *sk_memory_pressure(const struct sock *sk)
 {
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  int *ret = NULL;
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;
+  if (cg_proto->memory_pressure)
+   ret = cg_proto->memory_pressure;
+  return ret;
+ } else
+nocgroup:
+#endif
+
  return sk->sk_prot->memory_pressure;
 }

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

+static inline void memcg_memory_allocated_add(struct cg_proto *prot,
+           unsigned long amt,
+           int *parent_status)
+{
+ struct res_counter *fail;
+ int ret;
+
```

```
+ ret = res_counter_charge(prot->memory_allocated,
+     amt << PAGE_SHIFT, &fail);
+
+ if (ret < 0)
+  *parent_status = OVER_LIMIT;
+}
+
+static inline void memcg_memory_allocated_sub(struct cg_proto *prot,
+           unsigned long amt)
+{
+ res_counter_uncharge(prot->memory_allocated, amt << PAGE_SHIFT);
+}
+
+static inline u64 memcg_memory_allocated_read(struct cg_proto *prot)
+{
+ u64 ret;
+ ret = res_counter_read_u64(prot->memory_allocated, RES_USAGE);
+ return ret >> PAGE_SHIFT;
+}
+
 static inline long
 sk_memory_allocated(const struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+  if (!cg_proto) /* this handles the case with existing sockets */
+   goto nocgroup;
+
+  return memcg_memory_allocated_read(cg_proto);
+ }
+nocgroup:
+#endif
  return atomic_long_read(prot->memory_allocated);
 }

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

```
+
+  memcg_memory_allocated_add(cg_proto, amt, parent_status);
+ }
+nocgroup:
+#endif
  return atomic_long_add_return(amt, prot->memory_allocated);
 }

 static inline void
-sk_memory_allocated_sub(struct sock *sk, int amt)
+sk_memory_allocated_sub(struct sock *sk, int amt, int parent_status)
 {
  struct proto *prot = sk->sk_prot;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  if (!cg_proto)
+   goto nocgroup;
+
+  /* Otherwise it was uncharged already */
+  if (parent_status != OVER_LIMIT)
+   memcg_memory_allocated_sub(cg_proto, amt);
+ }
+nocgroup:
+#endif
  atomic_long_sub(amt, prot->memory_allocated);
 }

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

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

```
+  struct cg_proto *cg_proto = sk->sk_cgrp;
+
+  for (; cg_proto; cg_proto = cg_proto->parent)
+   percpu_counter_inc(cg_proto->sockets_allocated);
+ }
+#endif
  percpu_counter_inc(prot->sockets_allocated);
 }

@@ -912,19 +1039,57 @@ static inline int
 sk_sockets_allocated_read_positive(struct sock *sk)
 {
  struct proto *prot = sk->sk_prot;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto = sk->sk_cgrp;

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

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

 static inline long
```

```
 kcg_memory_allocated(struct proto *prot, struct mem_cgroup *cg)
 {
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+ if (static_branch(&memcg_socket_limit_enabled)) {
+  struct cg_proto *cg_proto;
+  if (!prot->proto_cgroup)
+   goto nocgroup;
+
+  cg_proto = prot->proto_cgroup(cg);
+  if (!cg_proto)
+   goto nocgroup;
+
+  return memcg_memory_allocated_read(cg_proto);
+ }
+nocgroup:
+#endif
  return atomic_long_read(prot->memory_allocated);
 }

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 1cb7daa..5f29194 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -376,6 +376,40 @@ enum mem_type {
 #define MEM_CGROUP_RECLAIM_SOFT_BIT 0x2
 #define MEM_CGROUP_RECLAIM_SOFT  (1 << MEM_CGROUP_RECLAIM_SOFT_BIT)

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

```
+
+  rcu_read_lock();
+  memcg = mem_cgroup_from_task(current);
+  if (!mem_cgroup_is_root(memcg))
+   sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
+  rcu_read_unlock();
+ }
+}
+
+#endif /* CONFIG_INET */
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
+
+
 static void mem_cgroup_get(struct mem_cgroup *mem);
 static void mem_cgroup_put(struct mem_cgroup *mem);
 static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *mem);
@@ -872,12 +906,6 @@ static struct mem_cgroup *mem_cgroup_get_next(struct mem_cgroup
*iter,
 #define for_each_mem_cgroup_all(iter) \
  for_each_mem_cgroup_tree_cond(iter, NULL, true)

-
-static inline bool mem_cgroup_is_root(struct mem_cgroup *mem)
-{
- return (mem == root_mem_cgroup);
-}
-
 void mem_cgroup_count_vm_event(struct mm_struct *mm, enum vm_event_item idx)
 {
  struct mem_cgroup *mem;
diff --git a/net/core/sock.c b/net/core/sock.c
index 26bdb1c..8382c80 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
@@ -1678,24 +1682,25 @@ int __sk_mem_schedule(struct sock *sk, int size, int kind)
  int amt = sk_mem_pages(size);
  long allocated;
  int *memory_pressure;
+ int parent_status = UNDER_LIMIT;

  sk->sk_forward_alloc += amt * SK_MEM_QUANTUM;

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

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

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

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

  /* guarantee minimum buffer size under pressure */
@@ -1742,7 +1747,7 @@ 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;
 }
@@ -1757,7 +1762,7 @@ void __sk_mem_reclaim(struct sock *sk)
  int *memory_pressure = sk_memory_pressure(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 (memory_pressure && *memory_pressure &&
--
1.7.6.4
```

Subject: [PATCH v6 04/10] Account tcp memory as kernel memory
Posted by Glauber Costa on Fri, 25 Nov 2011 17:38:10 GMT
View Forum Message <> Reply to Message

Now that we account and control tcp memory buffers memory for pressure
controlling purposes, display this information as part of the normal memcg
files and other usages.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujtisu.com>
CC: Eric W. Biederman <ebiederm@xmission.com>
---
 include/linux/memcontrol.h | 3 ++
 include/net/sock.h | 3 ++
 include/net/tcp_memcg.h | 17 ++++++++++
 mm/memcontrol.c | 39 ++++++++++++++++++++++++++--
 net/core/sock.c | 42 ++++++++++++++++++++++++++++--
 net/ipv4/Makefile | 1 +
 net/ipv4/tcp_ipv4.c | 8 +++++-
 net/ipv4/tcp_memcg.c | 64 ++
 net/ipv6/tcp_ipv6.c | 4 +++
 9 files changed, 174 insertions(+), 7 deletions(-)
 create mode 100644 include/net/tcp_memcg.h
 create mode 100644 net/ipv4/tcp_memcg.c

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 6644d90..1aff2f6 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -85,6 +85,9 @@ extern struct mem_cgroup *try_get_mem_cgroup_from_page(struct page
*page);
 extern struct mem_cgroup *mem_cgroup_from_task(struct task_struct *p);
 extern struct mem_cgroup *try_get_mem_cgroup_from_mm(struct mm_struct *mm);

+extern struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont);
+extern struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *mem);
+
 static inline

```
 int mm_match_cgroup(const struct mm_struct *mm, const struct mem_cgroup *cgroup)
 {
diff --git a/include/net/sock.h b/include/net/sock.h
index d802761..da38de2 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -65,6 +65,9 @@
 #include <net/dst.h>
 #include <net/checksum.h>

+int sockets_populate(struct cgroup *cgrp, struct cgroup_subsys *ss);
+void sockets_destroy(struct cgroup *cgrp, struct cgroup_subsys *ss);
+
 /*
  * This structure really needs to be cleaned up.
  * Most of it is for TCP, and not used by any of
diff --git a/include/net/tcp_memcg.h b/include/net/tcp_memcg.h
new file mode 100644
index 0000000..5f5e158
--- /dev/null
+++ b/include/net/tcp_memcg.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 5f29194..2df5d3c 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -49,6 +49,8 @@
 #include <linux/cpu.h>
 #include <linux/oom.h>
 #include "internal.h"
+#include <net/sock.h>
```

```
+#include <net/tcp_memcg.h>

 #include <asm/uaccess.h>

@@ -294,6 +296,10 @@ 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. */
@@ -385,6 +391,7 @@ static inline bool mem_cgroup_is_root(struct mem_cgroup *mem)
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
 #ifdef CONFIG_INET
 #include <net/sock.h>
+#include <net/ip.h>

 void sock_update_memcg(struct sock *sk)
 {
@@ -406,13 +413,21 @@ void sock_update_memcg(struct sock *sk)
  }
 }

+struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
+{
+ if (!memcg || mem_cgroup_is_root(memcg))
+  return NULL;
+
+ return &memcg->tcp_mem.cg_proto;
+}
+EXPORT_SYMBOL(tcp_proto_cgroup);
+
 #endif /* CONFIG_INET */
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */


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

 static struct mem_cgroup_per_zone *
@@ -787,7 +802,7 @@ static void memcg_check_events(struct mem_cgroup *mem, struct page
*page)
```

```
 }
}

-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,
@@ -4828,14 +4843,28 @@ static int register_kmem_files(struct cgroup *cont, struct
cgroup_subsys *ss)

 ret = cgroup_add_files(cont, ss, kmem_cgroup_files,
         ARRAY_SIZE(kmem_cgroup_files));
+
+ if (!ret)
+  ret = sockets_populate(cont, ss);
+
 return ret;
};

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

static int alloc_mem_cgroup_per_zone_info(struct mem_cgroup *mem, int node)
@@ -4954,7 +4983,7 @@ static void mem_cgroup_put(struct mem_cgroup *mem)
/*
 * Returns the parent mem_cgroup in memcgroup hierarchy with hierarchy enabled.
 */
-static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *mem)
+struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *mem)
{
 if (!mem->res.parent)
  return NULL;
@@ -5037,6 +5066,7 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
```

```
   res_counter_init(&mem->res, &parent->res);
   res_counter_init(&mem->memsw, &parent->memsw);
   res_counter_init(&mem->kmem, &parent->kmem);
+
   /*
    * We increment refcnt of the parent to ensure that we can
    * safely access it on res_counter_charge/uncharge.
@@ -5053,6 +5083,7 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
  mem->last_scanned_node = MAX_NUMNODES;
  INIT_LIST_HEAD(&mem->oom_notify);

+
  if (parent)
   mem->swappiness = mem_cgroup_swappiness(parent);
  atomic_set(&mem->refcnt, 1);
@@ -5078,6 +5109,8 @@ static void mem_cgroup_destroy(struct cgroup_subsys *ss,
 {
  struct mem_cgroup *mem = mem_cgroup_from_cont(cont);

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

diff --git a/net/core/sock.c b/net/core/sock.c
index 8382c80..399e566 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -135,6 +135,45 @@
 #include <net/tcp.h>
 #endif

+static DEFINE_RWLOCK(proto_list_lock);
+static LIST_HEAD(proto_list);
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
+int sockets_populate(struct cgroup *cgrp, struct cgroup_subsys *ss)
+{
+ struct proto *proto;
+ int ret = 0;
+
+ read_lock(&proto_list_lock);
+ list_for_each_entry(proto, &proto_list, node) {
+  if (proto->init_cgroup)
+   ret = proto->init_cgroup(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 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:
@@ -2259,9 +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..393e0af 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-$(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) += tcp_memcg.o
 obj-$(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 f124a4b..c517d04 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_memcg.h>

 #include <linux/inet.h>
 #include <linux/ipv6.h>
@@ -1917,6 +1918,7 @@ static int tcp_v4_init_sock(struct sock *sk)
  sk_sockets_allocated_inc(sk);
  local_bh_enable();

+ sock_update_memcg(sk);
  return 0;
 }

@@ -2629,10 +2631,14 @@ 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_memcg.c b/net/ipv4/tcp_memcg.c
new file mode 100644
index 0000000..cc91918
--- /dev/null
+++ b/net/ipv4/tcp_memcg.c
@@ -0,0 +1,64 @@
+#include <net/tcp.h>
+#include <net/tcp_memcg.h>
+#include <net/sock.h>
+#include <linux/memcontrol.h>
+
+static inline struct tcp_memcontrol *tcp_from_cgproto(struct cg_proto *cg_proto)
```

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

```
+ struct tcp_memcontrol *tcp;
+
+ cg_proto = tcp_prot.proto_cgroup(memcg);
+ if (!cg_proto)
+  return;
+
+ tcp = tcp_from_cgproto(cg_proto);
+ percpu_counter_destroy(&tcp->tcp_sockets_allocated);
+}
+EXPORT_SYMBOL(tcp_destroy_cgroup);
diff --git a/net/ipv6/tcp_ipv6.c b/net/ipv6/tcp_ipv6.c
index 3a08fcd..d57d7a7 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_memcg.h>

 #include <asm/uaccess.h>

@@ -2222,6 +2223,9 @@ 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 v6 05/10] per-netns ipv4 sysctl_tcp_mem
Posted by Glauber Costa on Fri, 25 Nov 2011 17:38:11 GMT
View Forum Message <> Reply to Message

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

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

```
CC: David S. Miller <davem@davemloft.net>
CC: Eric W. Biederman <ebiederm@xmission.com>
---
 include/net/netns/ipv4.h   |    1 +
 include/net/tcp.h          |    1 -
 net/ipv4/af_inet.c         |    2 +
 net/ipv4/sysctl_net_ipv4.c |   51 ++++++++++++++++++++++++++++++++++++------
 net/ipv4/tcp.c             |   11 +-------
 net/ipv4/tcp_ipv4.c        |    1 -
 net/ipv4/tcp_memcg.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 ccaa3b6..f3cc395 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 89a2bfe..631e6b3 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);

@@ -3274,14 +3272,9 @@ 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 c517d04..8920f98 100644
--- a/net/ipv4/tcp_ipv4.c
+++ b/net/ipv4/tcp_ipv4.c
@@ -2617,7 +2617,6 @@ 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_memcg.c b/net/ipv4/tcp_memcg.c
index cc91918..1dbc0f3 100644
--- a/net/ipv4/tcp_memcg.c
+++ b/net/ipv4/tcp_memcg.c
@@ -1,6 +1,8 @@
 #include <net/tcp.h>
 #include <net/tcp_memcg.h>
 #include <net/sock.h>
+#include <net/ip.h>
+#include <linux/nsproxy.h>
 #include <linux/memcontrol.h>

 static inline struct tcp_memcontrol *tcp_from_cgproto(struct cg_proto *cg_proto)
@@ -20,6 +22,7 @@ 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)
@@ -28,9 +31,9 @@ int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)
  tcp = tcp_from_cgproto(cg_proto);
  cg_proto->parent = tcp_prot.proto_cgroup(parent);

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

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

```
diff --git a/net/ipv6/tcp_ipv6.c b/net/ipv6/tcp_ipv6.c
index d57d7a7..b6451f2 100644
--- a/net/ipv6/tcp_ipv6.c
+++ b/net/ipv6/tcp_ipv6.c
@@ -2209,7 +2209,6 @@ 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 v6 06/10] tcp buffer limitation: per-cgroup limit
Posted by Glauber Costa on Fri, 25 Nov 2011 17:38:12 GMT
View Forum Message <> Reply to Message

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

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

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

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: David S. Miller <davem@davemloft.net>
CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
CC: Eric W. Biederman <ebiederm@xmission.com>

```
 Documentation/cgroups/memory.txt |    1 +
 include/net/tcp_memcg.h          |    3 +
 net/ipv4/sysctl_net_ipv4.c       |   14 ++++
 net/ipv4/tcp_memcg.c             |  138 ++++++++++++++++++++++++++++++++++++++-
 4 files changed, 154 insertions(+), 2 deletions(-)

diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index bf00cd2..c1db134 100644
--- a/Documentation/cgroups/memory.txt
+++ b/Documentation/cgroups/memory.txt
@@ -78,6 +78,7 @@ Brief summary of control files.
```

```
  memory.independent_kmem_limit  # select whether or not kernel memory limits are
       independent of user limits
+ memory.kmem.tcp.limit_in_bytes  # set/show hard limit for tcp buf memory
```

1. History

```
diff --git a/include/net/tcp_memcg.h b/include/net/tcp_memcg.h
index 5f5e158..2c8bb6b 100644
--- a/include/net/tcp_memcg.h
+++ b/include/net/tcp_memcg.h
@@ -14,4 +14,7 @@ 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);
+int tcp_update_limit(struct mem_cgroup *memcg, u64 val);
 #endif /* _TCP_MEMCG_H */
diff --git a/net/ipv4/sysctl_net_ipv4.c b/net/ipv4/sysctl_net_ipv4.c
index bbd67ab..17aaa1b 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_memcg.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 *cg;
+#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();
+ cg = mem_cgroup_from_task(current);
+
```

```
+ tcp_prot_mem(cg, vec[0], 0);
+ tcp_prot_mem(cg, vec[1], 1);
+ tcp_prot_mem(cg, vec[2], 2);
+ rcu_read_unlock();
+#endif
+
  net->ipv4.sysctl_tcp_mem[0] = vec[0];
  net->ipv4.sysctl_tcp_mem[1] = vec[1];
  net->ipv4.sysctl_tcp_mem[2] = vec[2];
diff --git a/net/ipv4/tcp_memcg.c b/net/ipv4/tcp_memcg.c
index 1dbc0f3..b3721c3 100644
--- a/net/ipv4/tcp_memcg.c
+++ b/net/ipv4/tcp_memcg.c
@@ -5,6 +5,19 @@
 #include <linux/nsproxy.h>
 #include <linux/memcontrol.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);
@@ -26,7 +39,7 @@ int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)

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

  tcp = tcp_from_cgproto(cg_proto);
  cg_proto->parent = tcp_prot.proto_cgroup(parent);
@@ -47,7 +60,9 @@ int tcp_init_cgroup(struct cgroup *cgrp, struct cgroup_subsys *ss)
  cg_proto->memory_allocated = &tcp->tcp_memory_allocated;
  cg_proto->sockets_allocated = &tcp->tcp_sockets_allocated;

- return 0;
+create_files:
```

```
+ return cgroup_add_files(cgrp, ss, tcp_files,
+    ARRAY_SIZE(tcp_files));
 }
 EXPORT_SYMBOL(tcp_init_cgroup);

@@ -56,6 +71,7 @@ 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)
@@ -63,5 +79,123 @@ 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);
+
+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;
+ int i;
+ int ret;
+
+ cg_proto = tcp_prot.proto_cgroup(memcg);
+ if (!cg_proto)
+  return -EINVAL;
+
+ tcp = tcp_from_cgproto(cg_proto);
+
+ ret = res_counter_set_limit(&tcp->tcp_memory_allocated, val);
+ if (ret)
+  return ret;
+
+ val >>= PAGE_SHIFT;
+
+ for (i = 0; i < 3; i++)
+  tcp->tcp_prot_mem[i] = min_t(long, val,
+          net->ipv4.sysctl_tcp_mem[i]);
```

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

```
+static u64 tcp_cgroup_read(struct cgroup *cont, struct cftype *cft)
+{
+ struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);
+ u64 val;
+
+ switch (cft->private) {
+ case RES_LIMIT:
+  val = tcp_read_stat(memcg, RES_LIMIT, RESOURCE_MAX);
+  break;
+ default:
+  BUG();
+ }
+ return val;
+}
+
+unsigned long long tcp_max_memory(const struct mem_cgroup *memcg)
+{
+ struct tcp_memcontrol *tcp;
+ struct cg_proto *cg_proto;
+
+ cg_proto = tcp_prot.proto_cgroup((struct mem_cgroup *)memcg);
+ if (!cg_proto)
+  return 0;
+
+ tcp = tcp_from_cgproto(cg_proto);
+ return res_counter_read_u64(&tcp->tcp_memory_allocated, RES_LIMIT);
+}
+
+void tcp_prot_mem(struct mem_cgroup *memcg, long val, int idx)
+{
+ struct tcp_memcontrol *tcp;
+ struct cg_proto *cg_proto;
+
+ cg_proto = tcp_prot.proto_cgroup(memcg);
+ if (!cg_proto)
+  return;
+
+ tcp = tcp_from_cgproto(cg_proto);
+
+ tcp->tcp_prot_mem[idx] = val;
+}
--
1.7.6.4
```

Subject: [PATCH v6 07/10] Display current tcp memory allocation in kmem cgroup
Posted by Glauber Costa on Fri, 25 Nov 2011 17:38:13 GMT

This patch introduces kmem.tcp.usage_in_bytes file, living in the
kmem_cgroup filesystem. It is a simple read-only file that displays the
amount of kernel memory currently consumed by the cgroup.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: David S. Miller <davem@davemloft.net>
CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
CC: Eric W. Biederman <ebiederm@xmission.com>

 Documentation/cgroups/memory.txt | 1 +
 net/ipv4/tcp_memcg.c | 21 ++++++++++++++++++++
 2 files changed, 22 insertions(+), 0 deletions(-)

diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
index c1db134..00f1a88 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_memcg.c b/net/ipv4/tcp_memcg.c
index b3721c3..a1ab613 100644
--- a/net/ipv4/tcp_memcg.c
+++ b/net/ipv4/tcp_memcg.c
@@ -16,6 +16,11 @@ 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)
@@ -158,6 +163,19 @@ 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);
@@ -167,6 +185,9 @@ 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 v6 08/10] Display current tcp failcnt in kmem cgroup
Posted by Glauber Costa on Fri, 25 Nov 2011 17:38:14 GMT
View Forum Message <> Reply to Message

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

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: David S. Miller <davem@davemloft.net>
CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
CC: Eric W. Biederman <ebiederm@xmission.com>
---
 net/ipv4/tcp_memcg.c | 31 ++++++++++++++++++++++++++++++++
 1 files changed, 31 insertions(+), 0 deletions(-)

diff --git a/net/ipv4/tcp_memcg.c b/net/ipv4/tcp_memcg.c

```
index a1ab613..da8d9c0 100644
--- a/net/ipv4/tcp_memcg.c
+++ b/net/ipv4/tcp_memcg.c
@@ -8,6 +8,7 @@
 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[] = {
  {
@@ -21,6 +22,12 @@ 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)
@@ -188,12 +195,36 @@ 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 v6 09/10] Display maximum tcp memory allocation in kmem cgroup
Posted by Glauber Costa on Fri, 25 Nov 2011 17:38:15 GMT
View Forum Message <> Reply to Message

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

All others, will see the maximum memory ever used by this cgroup.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: David S. Miller <davem@davemloft.net>
CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
CC: Eric W. Biederman <ebiederm@xmission.com>
---
 net/ipv4/tcp_memcg.c | 12 ++++++++++-
 1 files changed, 11 insertions(+), 1 deletions(-)

```
diff --git a/net/ipv4/tcp_memcg.c b/net/ipv4/tcp_memcg.c
index da8d9c0..e702f6a 100644
--- a/net/ipv4/tcp_memcg.c
+++ b/net/ipv4/tcp_memcg.c
@@ -28,6 +28,12 @@ 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)
@@ -196,7 +202,8 @@ 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();
@@ -217,6 +224,9 @@ static int tcp_cgroup_reset(struct cgroup *cont, unsigned int event)
  tcp = tcp_from_cgproto(cg_proto);

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

Subject: [PATCH v6 10/10] Disable task moving when using kernel memory accounting

Posted by Glauber Costa on Fri, 25 Nov 2011 17:38:16 GMT

View Forum Message <> Reply to Message

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

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

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

```
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 2df5d3c..ab7e57b 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -5451,10 +5451,19 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
 {
  int ret = 0;
  struct mem_cgroup *mem = mem_cgroup_from_cont(cgroup);
+ struct mem_cgroup *from = mem_cgroup_from_task(p);
+
+#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) && defined(CONFIG_INET)
+ if (from != mem && !mem_cgroup_is_root(from) &&
+     res_counter_read_u64(&from->tcp_mem.tcp_memory_allocated, RES_USAGE)) {
+  printk(KERN_WARNING "Can't move tasks between cgroups: "
+   "Kernel memory held. task: %s\n", p->comm);
+  return 1;
+ }
+#endif

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

   VM_BUG_ON(from == mem);

@@ -5622,6 +5631,18 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
     struct cgroup *cgroup,
     struct task_struct *p)
 {
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cgroup);
+ struct mem_cgroup *from = mem_cgroup_from_task(p);
+
+#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) && defined(CONFIG_INET)
+ if (from != mem && !mem_cgroup_is_root(from) &&
+     res_counter_read_u64(&from->tcp_mem.tcp_memory_allocated, RES_USAGE)) {
+  printk(KERN_WARNING "Can't move tasks between cgroups: "
+   "Kernel memory held. task: %s\n", p->comm);
+  return 1;
+ }
+#endif
+
  return 0;
 }
 static void mem_cgroup_cancel_attach(struct cgroup_subsys *ss,
--
1.7.6.4
```

Subject: Re: [PATCH v6 01/10] Basic kernel memory functionality for the Memory Controller
Posted by KAMEZAWA Hiroyuki on Mon, 28 Nov 2011 02:24:41 GMT
View Forum Message <> Reply to Message

On Fri, 25 Nov 2011 15:38:07 -0200
Glauber Costa <glommer@parallels.com> wrote:

> This patch lays down the foundation for the kernel memory component
> of the Memory Controller.
>
> As of today, I am only laying down the following files:
>
> * memory.independent_kmem_limit
> * memory.kmem.limit_in_bytes (currently ignored)
> * memory.kmem.usage_in_bytes (always zero)
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> Reviewed-by: Kirill A. Shutemov <kirill@shutemov.name>
> CC: Paul Menage <paul@paulmenage.org>
> CC: Greg Thelen <gthelen@google.com>
> ---
> Documentation/cgroups/memory.txt | 36 ++++++++++++-
> init/Kconfig | 14 +++++
> mm/memcontrol.c | 107 ++++++++++++++++++++++++++++++++++++++--
> 3 files changed, 150 insertions(+), 7 deletions(-)
>
> diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
> index 06eb6d9..bf00cd2 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 +262,31 @@ 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.
> +Kernel memory limits are not imposed for the root cgroup.
> +
> +Memory limits as specified by the standard Memory Controller may or may not
> +take kernel memory into consideration. This is achieved through the file
> +memory.independent_kmem_limit. A Value different than 0 will allow for kernel
> +memory to be controlled separately.
> +
> +When kernel memory limits are not independent, the limit values set in
> +memory.kmem files are ignored.
> +
> +Currently no soft limit is implemented for kernel memory. It is future work
> +to trigger slab reclaim when those limits are reached.
> +
> +CAUTION: As of this writing, the kmem extention may prevent tasks from moving
> +among cgroups. If a task has kmem accounting in a cgroup, the task cannot be
> +moved until the kmem resource is released. Also, until the resource is fully
> +released, the cgroup cannot be destroyed. So, please consider your use cases
> +and set kmem extention config option carefully.
> +

This seems that memcg 'has' kernel memory limiting feature for all kinds of kmem..
Could you add a list of "currently controled kmems" section ?

And update the list in later patch ?

Thanks,
-Kame

Subject: Re: [PATCH v6 02/10] foundations of per-cgroup memory pressure controlling.
Posted by KAMEZAWA Hiroyuki on Mon, 28 Nov 2011 02:55:41 GMT
View Forum Message <> Reply to Message

On Fri, 25 Nov 2011 15:38:08 -0200
Glauber Costa <glommer@parallels.com> wrote:

> This patch replaces all uses of struct sock fields' memory_pressure,
> memory_allocated, sockets_allocated, and sysctl_mem to acessor
> macros. Those macros can either receive a socket argument, or a mem_cgroup
> argument, depending on the context they live in.
>
> Since we're only doing a macro wrapping here, no performance impact at all is
> expected in the case where we don't have cgroups disabled.
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: David S. Miller <davem@davemloft.net>
> CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
> CC: Eric W. Biederman <ebiederm@xmission.com>
> CC: Eric Dumazet <eric.dumazet@gmail.com>

I have some comments on the style. Maybe a nitpick but many patches were
sent for fixing conding style in memcg recently.

+static inline int *sk_memory_pressure(const struct sock *sk)
+{
+ return sk->sk_prot->memory_pressure;
+}
+
+static inline long sk_prot_mem(const struct sock *sk, int index)
+{
+ long *prot = sk->sk_prot->sysctl_mem;
+ return prot[index];
+}
+

I don't think sk_prot_mem() is an easy to undestand name.
sk_prot_memory_limit() ?

> +static inline int
> +kcg_sockets_allocated_sum_positive(struct proto *prot, struct mem_cgroup *cg)

> +{
> + return percpu_counter_sum_positive(prot->sockets_allocated);
> +}
> +
> +static inline long
> +kcg_memory_allocated(struct proto *prot, struct mem_cgroup *cg)
> +{
> + return atomic_long_read(prot->memory_allocated);
> +}
>

I don't like 'kcg'. What it means ?
memory_cgrou_prot_socekts_allocated() ? and
memory_cgroup_prot_memory_allocated() ?

And the variable for memory cgroup should be 'memcg'.
http://www.spinics.net/lists/linux-mm/msg26781.html
So, please rename.

> #ifdef CONFIG_PROC_FS
> /* Called with local bh disabled */
> diff --git a/include/net/tcp.h b/include/net/tcp.h
> index e147f42..ccaa3b6 100644
<snip>

  seq_printf(seq, "RAW: inuse %d\n",
> diff --git a/net/ipv4/tcp.c b/net/ipv4/tcp.c
> index 34f5db1..89a2bfe 100644
> --- a/net/ipv4/tcp.c
> +++ b/net/ipv4/tcp.c
> @@ -319,9 +319,11 @@ EXPORT_SYMBOL(tcp_memory_pressure);
>
> void tcp_enter_memory_pressure(struct sock *sk)
> {
> - if (!tcp_memory_pressure) {
> + int *memory_pressure = sk_memory_pressure(sk);
> +

Don't you need !memory_pressure check here ?

Hmm, can this function be

+static inline int *sk_memory_pressure(const struct sock *sk)
+{
+ return sk->sk_prot->memory_pressure;
+}

as

```
static inline bool sk_under_prot_memory_pressure(const struct sock *sk)
{
 if (sk->sk_prot->memory_pressure &&
     *sk->sk_prot->memory_pressure)
  return true;

 return false;
}
```

and have sk_set/unset_prot_memory_pressure(), ?

```
> + if (!*memory_pressure) {
>    NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPMEMORYPRESSURES);
> -  tcp_memory_pressure = 1;
> +  *memory_pressure = 1;
>   }
>  }
>  EXPORT_SYMBOL(tcp_enter_memory_pressure);
> diff --git a/net/ipv4/tcp_input.c b/net/ipv4/tcp_input.c
> index 52b5c2d..3df862d 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_memory_pressure(sk)) {
```

Don't you need to check !*sk_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_memory_pressure(sk) &&
```

> +   sk_memory_allocated(sk) < sk_prot_mem(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_memory_pressure(sk))
>   tp->rcv_ssthresh = min(tp->rcv_ssthresh, 4U * tp->advmss);

Ditto.

>
>   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_memory_pressure(sk))
>   return 0;

again.

Thanks,
-Kame

Subject: Re: [PATCH v6 04/10] Account tcp memory as kernel memory
Posted by KAMEZAWA Hiroyuki on Mon, 28 Nov 2011 03:14:18 GMT
View Forum Message <> Reply to Message

Some nitpicks.

On Fri, 25 Nov 2011 15:38:10 -0200
Glauber Costa <glommer@parallels.com> wrote:

> Now that we account and control tcp memory buffers memory for pressure
> controlling purposes, display this information as part of the normal memcg
> files and other usages.
>

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

> static inline
> int mm_match_cgroup(const struct mm_struct *mm, const struct mem_cgroup *cgroup)
> {
> diff --git a/include/net/sock.h b/include/net/sock.h
> index d802761..da38de2 100644
> --- a/include/net/sock.h
> +++ b/include/net/sock.h
> @@ -65,6 +65,9 @@
> #include <net/dst.h>
> #include <net/checksum.h>
>
> +int sockets_populate(struct cgroup *cgrp, struct cgroup_subsys *ss);
> +void sockets_destroy(struct cgroup *cgrp, struct cgroup_subsys *ss);
> +
> /*

Hmm, what is this 'populate' function for ?
mem_cgroup_sockets_init() ?

>   * 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_memcg.h b/include/net/tcp_memcg.h
> new file mode 100644
> index 0000000..5f5e158
> --- /dev/null
> +++ b/include/net/tcp_memcg.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 5f29194..2df5d3c 100644

> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -49,6 +49,8 @@
> #include <linux/cpu.h>
> #include <linux/oom.h>
> #include "internal.h"
> +#include <net/sock.h>
> +#include <net/tcp_memcg.h>

ok, tcp_memcg.h ... some other men may like tcp_memcontrol.h..

>
> #include <asm/uaccess.h>
>
<snip>

> static int alloc_mem_cgroup_per_zone_info(struct mem_cgroup *mem, int node)
> @@ -4954,7 +4983,7 @@ static void mem_cgroup_put(struct mem_cgroup *mem)
> /*
>   * Returns the parent mem_cgroup in memcgroup hierarchy with hierarchy enabled.
>   */
> -static struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *mem)
> +struct mem_cgroup *parent_mem_cgroup(struct mem_cgroup *mem)
> {
>   if (!mem->res.parent)
>   return NULL;
> @@ -5037,6 +5066,7 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
>   res_counter_init(&mem->res, &parent->res);
>   res_counter_init(&mem->memsw, &parent->memsw);
>   res_counter_init(&mem->kmem, &parent->kmem);
> +

unnecessary blank line.

>   /*
>   * We increment refcnt of the parent to ensure that we can
>   * safely access it on res_counter_charge/uncharge.
> @@ -5053,6 +5083,7 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
>   mem->last_scanned_node = MAX_NUMNODES;
>   INIT_LIST_HEAD(&mem->oom_notify);
>
> +
ditto.

<snip>

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

Subject: Re: [PATCH v6 06/10] tcp buffer limitation: per-cgroup limit
Posted by KAMEZAWA Hiroyuki on Mon, 28 Nov 2011 03:24:52 GMT
View Forum Message <> Reply to Message

some comments.

On Fri, 25 Nov 2011 15:38:12 -0200
Glauber Costa <glommer@parallels.com> wrote:

> This patch uses the "tcp.limit_in_bytes" field of the kmem_cgroup to
> effectively control the amount of kernel memory pinned by a cgroup.
>
> This value is ignored in the root cgroup, and in all others,
> caps the value specified by the admin in the net namespaces'
> view of tcp_sysctl_mem.
>
> If namespaces are being used, the admin is allowed to set a
> value bigger than cgroup's maximum, the same way it is allowed
> to set pretty much unlimited values in a real box.
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: David S. Miller <davem@davemloft.net>
> CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
> CC: Eric W. Biederman <ebiederm@xmission.com>

<snip>

> EXPORT_SYMBOL(tcp_destroy_cgroup);
> +
> +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;
> + int i;
> + int ret;
> +
> + cg_proto = tcp_prot.proto_cgroup(memcg);
> + if (!cg_proto)
> + return -EINVAL;
> +
> + tcp = tcp_from_cgproto(cg_proto);
> +
> + ret = res_counter_set_limit(&tcp->tcp_memory_allocated, val);

Here, you changed the limit.

> + if (ret)
> + return ret;

> +
> + val >>= PAGE_SHIFT;

Here, you modifies 'val'

> +
> + for (i = 0; i < 3; i++)
> + tcp->tcp_prot_mem[i] = min_t(long, val,
> +   net->ipv4.sysctl_tcp_mem[i]);
> +
> + if (val == RESOURCE_MAX)
> + jump_label_dec(&memcg_socket_limit_enabled);

the 'val' never be RESOUECE_MAX.

> + else {
> + u64 old_lim;
> + old_lim = res_counter_read_u64(&tcp->tcp_memory_allocated,
> +   RES_LIMIT);

old_lim is not already overwritten ?

> + if (old_lim == RESOURCE_MAX)
> +   jump_label_inc(&memcg_socket_limit_enabled);
> + }
> + return 0;
> +}
> +

Thanks,
-Kame

Subject: Re: [PATCH v6 07/10] Display current tcp memory allocation in kmem cgroup
Posted by KAMEZAWA Hiroyuki on Mon, 28 Nov 2011 04:20:27 GMT
View Forum Message <> Reply to Message

On Fri, 25 Nov 2011 15:38:13 -0200
Glauber Costa <glommer@parallels.com> wrote:

> This patch introduces kmem.tcp.usage_in_bytes file, living in the
> kmem_cgroup filesystem. It is a simple read-only file that displays the
> amount of kernel memory currently consumed by the cgroup.
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: David S. Miller <davem@davemloft.net>

> CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
> CC: Eric W. Biederman <ebiederm@xmission.com>

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

Subject: Re: [PATCH v6 08/10] Display current tcp failcnt in kmem cgroup
Posted by KAMEZAWA Hiroyuki on Mon, 28 Nov 2011 04:21:28 GMT
View Forum Message <> Reply to Message

On Fri, 25 Nov 2011 15:38:14 -0200
Glauber Costa <glommer@parallels.com> wrote:

> This patch introduces kmem.tcp.failcnt file, living in the
> kmem_cgroup filesystem. Following the pattern in the other
> memcg resources, this files keeps a counter of how many times
> allocation failed due to limits being hit in this cgroup.
> The root cgroup will always show a failcnt of 0.
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: David S. Miller <davem@davemloft.net>
> CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
> CC: Eric W. Biederman <ebiederm@xmission.com>

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

Subject: Re: [PATCH v6 09/10] Display maximum tcp memory allocation in kmem cgroup
Posted by KAMEZAWA Hiroyuki on Mon, 28 Nov 2011 04:22:15 GMT
View Forum Message <> Reply to Message

On Fri, 25 Nov 2011 15:38:15 -0200
Glauber Costa <glommer@parallels.com> wrote:

> This patch introduces kmem.tcp.max_usage_in_bytes file, living in the
> kmem_cgroup filesystem. The root cgroup will display a value equal
> to RESOURCE_MAX. This is to avoid introducing any locking schemes in
> the network paths when cgroups are not being actively used.
>
> All others, will see the maximum memory ever used by this cgroup.
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: David S. Miller <davem@davemloft.net>
> CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
> CC: Eric W. Biederman <ebiederm@xmission.com>

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

Subject: Re: [PATCH v6 10/10] Disable task moving when using kernel memory accounting
Posted by KAMEZAWA Hiroyuki on Mon, 28 Nov 2011 04:32:03 GMT
View Forum Message <> Reply to Message

On Fri, 25 Nov 2011 15:38:16 -0200
Glauber Costa <glommer@parallels.com> wrote:

> Since this code is still experimental, we are leaving the exact
> details of how to move tasks between cgroups when kernel memory
> accounting is used as future work.
>
> For now, we simply disallow movement if there are any pending
> accounted memory.
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
> ---
> mm/memcontrol.c | 23 ++++++++++++++++++++++-
> 1 files changed, 22 insertions(+), 1 deletions(-)
>
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 2df5d3c..ab7e57b 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -5451,10 +5451,19 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
> {
>   int ret = 0;
>   struct mem_cgroup *mem = mem_cgroup_from_cont(cgroup);
> + struct mem_cgroup *from = mem_cgroup_from_task(p);
> +
> +#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) && defined(CONFIG_INET)
> + if (from != mem && !mem_cgroup_is_root(from) &&
> +   res_counter_read_u64(&from->tcp_mem.tcp_memory_allocated, RES_USAGE)) {
> + printk(KERN_WARNING "Can't move tasks between cgroups: "
> +   "Kernel memory held. task: %s\n", p->comm);
> + return 1;
> + }
> +#endif

Hmm, the kernel memory is not guaranteed as being held by the 'task' ?

How about
"Now, moving task between cgroup is disallowed while the source cgroup
 containes kmem reference." ?

Hmm.. we need to fix this task-move/rmdir issue before production use.

Thanks,
-Kame

Subject: Re: [PATCH v6 10/10] Disable task moving when using kernel memory accounting
Posted by Glauber Costa on Mon, 28 Nov 2011 11:00:24 GMT
View Forum Message <> Reply to Message

On 11/28/2011 02:32 AM, KAMEZAWA Hiroyuki wrote:
> On Fri, 25 Nov 2011 15:38:16 -0200
> Glauber Costa<glommer@parallels.com> wrote:
>
>> Since this code is still experimental, we are leaving the exact
>> details of how to move tasks between cgroups when kernel memory
>> accounting is used as future work.
>>
>> For now, we simply disallow movement if there are any pending
>> accounted memory.
>>
>> Signed-off-by: Glauber Costa<glommer@parallels.com>
>> CC: Hiroyouki Kamezawa<kamezawa.hiroyu@jp.fujitsu.com>
>> ---
>>   mm/memcontrol.c | 23 ++++++++++++++++++++++-
>>   1 files changed, 22 insertions(+), 1 deletions(-)
>>
>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
>> index 2df5d3c..ab7e57b 100644
>> --- a/mm/memcontrol.c
>> +++ b/mm/memcontrol.c
>> @@ -5451,10 +5451,19 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
>>   {
>>   int ret = 0;
>>   struct mem_cgroup *mem = mem_cgroup_from_cont(cgroup);
>> + struct mem_cgroup *from = mem_cgroup_from_task(p);
>> +
>> +#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM)&& defined(CONFIG_INET)
>> + if (from != mem&& !mem_cgroup_is_root(from)&&
>> +   res_counter_read_u64(&from->tcp_mem.tcp_memory_allocated, RES_USAGE)) {
>> + printk(KERN_WARNING "Can't move tasks between cgroups: "
>> +   "Kernel memory held. task: %s\n", p->comm);
>> + return 1;
>> + }
>> +#endif

>
> Hmm, the kernel memory is not guaranteed as being held by the 'task' ?
>
> How about
> "Now, moving task between cgroup is disallowed while the source cgroup
> containes kmem reference." ?
>
> Hmm.. we need to fix this task-move/rmdir issue before production use.
>
>
> Thanks,
> -Kame
>
Hi Kame,

Let me tell you the direction I am going wrt task movement: The only reasons I haven't included so far, is that I believe it needs more testing, and as you know, I am right now more interested in getting past the initial barriers for inclusion. I am committed to fix anything that needs to be fixed - stylish or non-stylish before we remove the experimental flag.

So what I intend to do, is to basically
* lock the task,
* scan through its file descriptors list,
* identify which of them are sockets,
* cast them to struct sock *,
* see if it has a cgrp associated
* see if cgrp == from

At this point we can decrement sockets allocated by 1 in from, and memory_allocated by sk_forward_alloc (increasing by equal quantities in the destination cgroup)

I belive it will work.

Subject: Re: [PATCH v6 01/10] Basic kernel memory functionality for the Memory Controller
Posted by Glauber Costa on Mon, 28 Nov 2011 11:03:09 GMT
View Forum Message <> Reply to Message

On 11/28/2011 12:24 AM, KAMEZAWA Hiroyuki wrote:
> On Fri, 25 Nov 2011 15:38:07 -0200
> Glauber Costa<glommer@parallels.com> wrote:
>
>> This patch lays down the foundation for the kernel memory component
>> of the Memory Controller.

>>
>> As of today, I am only laying down the following files:
>>
>> * memory.independent_kmem_limit
>> * memory.kmem.limit_in_bytes (currently ignored)
>> * memory.kmem.usage_in_bytes (always zero)
>>
>> Signed-off-by: Glauber Costa<glommer@parallels.com>
>> Reviewed-by: Kirill A. Shutemov<kirill@shutemov.name>
>> CC: Paul Menage<paul@paulmenage.org>
>> CC: Greg Thelen<gthelen@google.com>
>> ---
>> Documentation/cgroups/memory.txt | 36 ++++++++++++-
>> init/Kconfig | 14 +++++
>> mm/memcontrol.c | 107 +++++++++++++++++++++++++++++++++++++--
>> 3 files changed, 150 insertions(+), 7 deletions(-)
>>
>> diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
>> index 06eb6d9..bf00cd2 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 +262,31 @@ 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.
>> +Kernel memory limits are not imposed for the root cgroup.
>> +
>> +Memory limits as specified by the standard Memory Controller may or may not
>> +take kernel memory into consideration. This is achieved through the file
>> +memory.independent_kmem_limit. A Value different than 0 will allow for kernel
>> +memory to be controlled separately.
>> +
>> +When kernel memory limits are not independent, the limit values set in
>> +memory.kmem files are ignored.
>> +
>> +Currently no soft limit is implemented for kernel memory. It is future work
>> +to trigger slab reclaim when those limits are reached.
>> +
>> +CAUTION: As of this writing, the kmem extention may prevent tasks from moving
>> +among cgroups. If a task has kmem accounting in a cgroup, the task cannot be
>> +moved until the kmem resource is released. Also, until the resource is fully
>> +released, the cgroup cannot be destroyed. So, please consider your use cases
>> +and set kmem extention config option carefully.
>> +
>
> This seems that memcg 'has' kernel memory limiting feature for all kinds of kmem..
> Could you add a list of "currently controled kmems" section ?
> And update the list in later patch ?
>
> Thanks,
> -Kame
>
>
Hi Kame,

Thanks for your review.

Since none of your comments are blockers, I'd prefer to send follow up patches if you don't mind - assuming Dave won't have any restrictions himself that would prevent him from picking this series. If I have to resend it anyway, I'll be more than happy to address them all in my next submission

---

Subject: Re: [PATCH v6 01/10] Basic kernel memory functionality for the Memory Controller
Posted by KAMEZAWA Hiroyuki on Mon, 28 Nov 2011 23:55:18 GMT
View Forum Message <> Reply to Message

On Mon, 28 Nov 2011 09:03:09 -0200
Glauber Costa <glommer@parallels.com> wrote:

> On 11/28/2011 12:24 AM, KAMEZAWA Hiroyuki wrote:
> > On Fri, 25 Nov 2011 15:38:07 -0200
> > Glauber Costa<glommer@parallels.com> wrote:
> >
> >> This patch lays down the foundation for the kernel memory component
> >> of the Memory Controller.
> >>
> >> As of today, I am only laying down the following files:
> >>
> >>   * memory.independent_kmem_limit
> >>   * memory.kmem.limit_in_bytes (currently ignored)
> >>   * memory.kmem.usage_in_bytes (always zero)
> >>
> >> Signed-off-by: Glauber Costa<glommer@parallels.com>
> >> Reviewed-by: Kirill A. Shutemov<kirill@shutemov.name>
> >> CC: Paul Menage<paul@paulmenage.org>
> >> CC: Greg Thelen<gthelen@google.com>
> >> ---
> >>   Documentation/cgroups/memory.txt | 36 ++++++++++++-
> >>   init/Kconfig | 14 +++++
> >>   mm/memcontrol.c | 107 ++++++++++++++++++++++++++++++++++++--
> >>   3 files changed, 150 insertions(+), 7 deletions(-)
> >>
> >> diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
> >> index 06eb6d9..bf00cd2 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 +262,31 @@ 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.
> >> +Kernel memory limits are not imposed for the root cgroup.
> >> +
> >> +Memory limits as specified by the standard Memory Controller may or may not
> >> +take kernel memory into consideration. This is achieved through the file
> >> +memory.independent_kmem_limit. A Value different than 0 will allow for kernel
> >> +memory to be controlled separately.
> >> +
> >> +When kernel memory limits are not independent, the limit values set in
> >> +memory.kmem files are ignored.
> >> +

> >> +Currently no soft limit is implemented for kernel memory. It is future work
> >> +to trigger slab reclaim when those limits are reached.
> >> +
> >> +CAUTION: As of this writing, the kmem extention may prevent tasks from moving
> >> +among cgroups. If a task has kmem accounting in a cgroup, the task cannot be
> >> +moved until the kmem resource is released. Also, until the resource is fully
> >> +released, the cgroup cannot be destroyed. So, please consider your use cases
> >> +and set kmem extention config option carefully.
> >> +
> >
> > This seems that memcg 'has' kernel memory limiting feature for all kinds of kmem..
> > Could you add a list of "currently controled kmems" section ?
> > And update the list in later patch ?
> >
> > Thanks,
> > -Kame
> >
> >
> Hi Kame,
>
> Thanks for your review.
> Since none of your comments are blockers, I'd prefer to send follow up
> patches if you don't mind - assuming Dave won't have any restrictions
> himself that would prevent him from picking this series. If I have to
> resend it anyway, I'll be more than happy to address them all in my next
> submission
>

As you like. But please clarify my comment which pointed out bugs in patch 02/10
and 06/10 aren't correct.

Thanks,
-Kame

Subject: Re: [PATCH v6 02/10] foundations of per-cgroup memory pressure controlling.
Posted by Glauber Costa on Tue, 29 Nov 2011 09:19:12 GMT
View Forum Message <> Reply to Message

On 11/28/2011 12:55 AM, KAMEZAWA Hiroyuki wrote:
> On Fri, 25 Nov 2011 15:38:08 -0200
> Glauber Costa<glommer@parallels.com> wrote:
>
>> This patch replaces all uses of struct sock fields' memory_pressure,
>> memory_allocated, sockets_allocated, and sysctl_mem to acessor
>> macros. Those macros can either receive a socket argument, or a mem_cgroup
>> argument, depending on the context they live in.

>>
>> Since we're only doing a macro wrapping here, no performance impact at all is
>> expected in the case where we don't have cgroups disabled.
>>
>> Signed-off-by: Glauber Costa<glommer@parallels.com>
>> CC: David S. Miller<davem@davemloft.net>
>> CC: Hiroyouki Kamezawa<kamezawa.hiroyu@jp.fujitsu.com>
>> CC: Eric W. Biederman<ebiederm@xmission.com>
>> CC: Eric Dumazet<eric.dumazet@gmail.com>
>
> I have some comments on the style. Maybe a nitpick but many patches were
> sent for fixing conding style in memcg recently.
>
> +static inline int *sk_memory_pressure(const struct sock *sk)
> +{
> + return sk->sk_prot->memory_pressure;
> +}
> +
> +static inline long sk_prot_mem(const struct sock *sk, int index)
> +{
> + long *prot = sk->sk_prot->sysctl_mem;
> + return prot[index];
> +}
> +
>
> I don't think sk_prot_mem() is an easy to undestand name.
> sk_prot_memory_limit() ?
>
>> +static inline int
>> +kcg_sockets_allocated_sum_positive(struct proto *prot, struct mem_cgroup *cg)
>> +{
>> + return percpu_counter_sum_positive(prot->sockets_allocated);
>> +}
>> +
>> +static inline long
>> +kcg_memory_allocated(struct proto *prot, struct mem_cgroup *cg)
>> +{
>> + return atomic_long_read(prot->memory_allocated);
>> +}
>>
>
> I don't like 'kcg'. What it means ?
> memory_cgrou_prot_socekts_allocated() ? and
> memory_cgroup_prot_memory_allocated() ?
>
> And the variable for memory cgroup should be 'memcg'.
> http://www.spinics.net/lists/linux-mm/msg26781.html
> So, please rename.

>
>
>
>>   #ifdef CONFIG_PROC_FS
>>   /* Called with local bh disabled */
>> diff --git a/include/net/tcp.h b/include/net/tcp.h
>> index e147f42..ccaa3b6 100644
> <snip>
>
>   seq_printf(seq, "RAW: inuse %d\n",
>> diff --git a/net/ipv4/tcp.c b/net/ipv4/tcp.c
>> index 34f5db1..89a2bfe 100644
>> --- a/net/ipv4/tcp.c
>> +++ b/net/ipv4/tcp.c
>> @@ -319,9 +319,11 @@ EXPORT_SYMBOL(tcp_memory_pressure);
>>
>>   void tcp_enter_memory_pressure(struct sock *sk)
>>   {
>> - if (!tcp_memory_pressure) {
>> + int *memory_pressure = sk_memory_pressure(sk);
>> +
>
> Don't you need !memory_pressure check here ?

Not really. Note that the original tcp code doesn't have it as well.
The generic networking code deals with many protocols, not all of them
have memory pressure functionality implemented. Therefore, a check is
needed. If we're inside tcp boundaries, we can pretty much assume
memory_pressure is present.

> Hmm, can this function be
>
> +static inline int *sk_memory_pressure(const struct sock *sk)
> +{
> + return sk->sk_prot->memory_pressure;
> +}
>
> as
>
> static inline bool sk_under_prot_memory_pressure(const struct sock *sk)
> {
> if (sk->sk_prot->memory_pressure&&
> *sk->sk_prot->memory_pressure)
>   return true;
>
> return false;
> }
>

> and have sk_set/unset_prot_memory_pressure(), ?

Yes, it could. Would it be preferable for you?
>
>
>> + if (!*memory_pressure) {
>>   NET_INC_STATS(sock_net(sk), LINUX_MIB_TCPMEMORYPRESSURES);
>> - tcp_memory_pressure = 1;
>> + *memory_pressure = 1;
>>   }
>>   }
>>   EXPORT_SYMBOL(tcp_enter_memory_pressure);
>> diff --git a/net/ipv4/tcp_input.c b/net/ipv4/tcp_input.c
>> index 52b5c2d..3df862d 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_memory_pressure(sk)) {
>
> Don't you need to check !*sk_memory_pressure(sk) ?

good catch!
yes.

>
>
>
>>   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_memory_pressure(sk)&&
>> +   sk_memory_allocated(sk)< sk_prot_mem(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_memory_pressure(sk))
>>   tp->rcv_ssthresh = min(tp->rcv_ssthresh, 4U * tp->advmss);
>
> Ditto.
>
>
>>
>>   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_memory_pressure(sk))
>>   return 0;
>
> again.
>
> Thanks,
> -Kame
>
Ok, I will fix this and respin it.

Subject: Re: [PATCH v6 01/10] Basic kernel memory functionality for the Memory Controller
Posted by Michal Hocko on Wed, 14 Dec 2011 16:38:44 GMT
View Forum Message <> Reply to Message

Sorry for jumping in that late but I was busy recently...

On Fri 25-11-11 15:38:07, 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>
> Reviewed-by: Kirill A. Shutemov <kirill@shutemov.name>
> CC: Paul Menage <paul@paulmenage.org>
> CC: Greg Thelen <gthelen@google.com>
> ---
> Documentation/cgroups/memory.txt | 36 ++++++++++++-
> init/Kconfig | 14 +++++
> mm/memcontrol.c | 107 ++++++++++++++++++++++++++++++++++++--
> 3 files changed, 150 insertions(+), 7 deletions(-)
>
> diff --git a/Documentation/cgroups/memory.txt b/Documentation/cgroups/memory.txt
> index 06eb6d9..bf00cd2 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 to me what happens in enabled/disabled cases. Let's say
they are not independent. Do they form a single limit 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 +262,31 @@ 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.

Per kmem cache? Or what is the granularity?

> 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.
> +Kernel memory limits are not imposed for the root cgroup.
> +
> +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.

This suggests that the independent_kmem_limit is toggle to
enable/disable accounting. Wouldn't kmem_limit_enabled (0/1 or on/off)
be more obvious in that case?

Also a description what happens when the limit is reached (in both cases) would be helpful.

[...]
> @@ -343,9 +352,14 @@ 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 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)
> @@ -3838,10 +3852,17 @@ static inline u64 mem_cgroup_usage(struct mem_cgroup *mem, bool swap)
>   u64 val;
>
>   if (!mem_cgroup_is_root(mem)) {
> + val = 0;
> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> + if (!mem->kmem_independent_accounting)
> +   val = res_counter_read_u64(&mem->kmem, RES_USAGE);
> +#endif

OK so this suggests that independent accounting really means kmem+user_usage.

>   if (!swap)
> -   return res_counter_read_u64(&mem->res, RES_USAGE);
> +   val += res_counter_read_u64(&mem->res, RES_USAGE);
>   else
> -   return res_counter_read_u64(&mem->memsw, RES_USAGE);
> +   val += res_counter_read_u64(&mem->memsw, RES_USAGE);
> +

> + return val;
> }
>
> val = mem_cgroup_recursive_stat(mem, MEM_CGROUP_STAT_CACHE);
[...]
--
Michal Hocko
SUSE Labs
SUSE LINUX s.r.o.
Lihovarska 1060/12
190 00 Praha 9
Czech Republic

Subject: Re: [PATCH v6 01/10] Basic kernel memory functionality for the Memory Controller
Posted by Michal Hocko on Wed, 14 Dec 2011 16:45:46 GMT
View Forum Message <> Reply to Message

On Wed 14-12-11 17:38:44, Michal Hocko wrote:
> Sorry for jumping in that late but I was busy recently...

And obviously an old version of the patch in my mailbox. I will comment
on the v9 now. Sorry...
[...]
--
Michal Hocko
SUSE Labs
SUSE LINUX s.r.o.
Lihovarska 1060/12
190 00 Praha 9
Czech Republic
