Subject: [PATCH] memory.min_usage
Posted by yamamoto on Tue, 04 Dec 2007 04:09:34 GMT
View Forum Message <> Reply to Message

hi,

here's a patch to implement memory.min_usage,
which controls the minimum memory usage for a cgroup.

it works similarly to mlock;
global memory reclamation doesn't reclaim memory from
cgroups whose memory usage is below the value.
setting it too high is a dangerous operation.

it's against 2.6.24-rc3-mm2 + memory.swappiness patch i posted here yesterday.
but it's logically independent from the swappiness patch.

todo:
- restrict non-root user's operation ragardless of owner of cgroupfs files?
- make oom killer aware of this?

YAMAMOTO Takashi

Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

```
--- linux-2.6.24-rc3-mm2-swappiness/include/linux/memcontrol.h.BACKUP2 2007-12-03
13:52:37.000000000 +0900
+++ linux-2.6.24-rc3-mm2-swappiness/include/linux/memcontrol.h 2007-12-03
14:07:45.000000000 +0900
@@ -47,6 +47,7 @@ extern int mem_cgroup_cache_charge(struc
      gfp_t gfp_mask);
 int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem);
 extern int mem_cgroup_swappiness(struct mem_cgroup *mem);
+extern int mem_cgroup_canreclaim(struct page *page, struct mem_cgroup *mem);

 static inline struct mem_cgroup *mm_cgroup(const struct mm_struct *mm)
 {
--- linux-2.6.24-rc3-mm2-swappiness/mm/memcontrol.c.BACKUP2 2007-12-03
13:52:13.000000000 +0900
+++ linux-2.6.24-rc3-mm2-swappiness/mm/memcontrol.c 2007-12-04 11:42:07.000000000 +0900
@@ -134,6 +134,7 @@ struct mem_cgroup {
  unsigned long control_type; /* control RSS or RSS+Pagecache */
  int prev_priority; /* for recording reclaim priority */
  unsigned int swappiness; /* swappiness */
+ unsigned long long min_usage; /* XXX should be a part of res_counter? */
  /*
```

```
   * statistics.
   */
@@ -1096,6 +1097,22 @@ static u64 mem_cgroup_swappiness_read(st
  return mem->swappiness;
 }

+static int mem_cgroup_min_usage_write(struct cgroup *cont, struct cftype *cft,
+           u64 val)
+{
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
+
+ mem->min_usage = val;
+ return 0;
+}
+
+static u64 mem_cgroup_min_usage_read(struct cgroup *cont, struct cftype *cft)
+{
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
+
+ return mem->min_usage;
+}
+
 static struct cftype mem_cgroup_files[] = {
  {
   .name = "usage_in_bytes",
@@ -1132,6 +1149,11 @@ static struct cftype mem_cgroup_files[]
   .write_uint = mem_cgroup_swappiness_write,
   .read_uint = mem_cgroup_swappiness_read,
  },
+ {
+  .name = "min_usage_in_bytes",
+  .write_uint = mem_cgroup_min_usage_write,
+  .read_uint = mem_cgroup_min_usage_read,
+ },
 };

 /* XXX probably it's better to move try_to_free_mem_cgroup_pages to
@@ -1142,6 +1164,36 @@ int mem_cgroup_swappiness(struct mem_cgr
  return mem->swappiness;
 }

+int mem_cgroup_canreclaim(struct page *page, struct mem_cgroup *mem1)
+{
+ struct page_cgroup *pc;
+ int result = 1;
+
+ if (mem1 != NULL)
+  return 1;
```

```
+
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (pc) {
+  struct mem_cgroup *mem2 = pc->mem_cgroup;
+  unsigned long long min_usage;
+
+  BUG_ON(mem2 == NULL);
+  min_usage = mem2->min_usage;
+  if (min_usage != 0) {
+   unsigned long flags;
+
+   spin_lock_irqsave(&mem2->res.lock, flags);
+   if (mem2->res.usage <= min_usage)
+    result = 0;
+   spin_unlock_irqrestore(&mem2->res.lock, flags);
+  }
+ }
+ unlock_page_cgroup(page);
+
+ return result;
+}
+
 static int alloc_mem_cgroup_per_zone_info(struct mem_cgroup *mem, int node)
 {
  struct mem_cgroup_per_node *pn;
--- linux-2.6.24-rc3-mm2-swappiness/mm/vmscan.c.BACKUP2 2007-12-03 13:52:22.000000000
+0900
+++ linux-2.6.24-rc3-mm2-swappiness/mm/vmscan.c 2007-12-03 14:11:42.000000000 +0900
@@ -531,6 +531,11 @@ static unsigned long shrink_page_list(st
      referenced && page_mapping_inuse(page))
    goto activate_locked;

+#ifdef CONFIG_CGROUP_MEM_CONT
+  if (!mem_cgroup_canreclaim(page, sc->mem_cgroup))
+   goto activate_locked;
+#endif /* CONFIG_CGROUP_MEM_CONT */
+
 #ifdef CONFIG_SWAP
   /*
    * Anonymous process memory has backing store?
@@ -1122,6 +1127,12 @@ static void shrink_active_list(unsigned
    list_add(&page->lru, &l_active);
    continue;
   }
+#ifdef CONFIG_CGROUP_MEM_CONT
+  if (!mem_cgroup_canreclaim(page, sc->mem_cgroup)) {
+   list_add(&page->lru, &l_active);
```

```
+   continue;
+  }
+#endif /* CONFIG_CGROUP_MEM_CONT */
   list_add(&page->lru, &l_inactive);
  }
```

Subject: Re: [PATCH] memory.min_usage
Posted by KAMEZAWA Hiroyuki on Tue, 04 Dec 2007 05:58:31 GMT
View Forum Message <> Reply to Message

On Tue, 4 Dec 2007 13:09:34 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> hi,
>
> here's a patch to implement memory.min_usage,
> which controls the minimum memory usage for a cgroup.
>
Thanks.

> todo:
> - restrict non-root user's operation ragardless of owner of cgroupfs files?

I like 'only-for-root' user. or CAP_SYS_RESOURCE.

> - make oom killer aware of this?
>
For OOM-Killer, IMHO, no care is necessary (now).
This function can be considered as a kind of mlock(). A user can be aware of
memory usage.
But, we may need some logic/reason to request this function.

> + unsigned long long min_usage; /* XXX should be a part of res_counter? */
>   /*
Maybe res_counter is better. (Added CC: to Pavel Emelianov)

> +int mem_cgroup_canreclaim(struct page *page, struct mem_cgroup *mem1)
> +{
> + struct page_cgroup *pc;
> + int result = 1;
> +

```
> + if (mem1 != NULL)
> +  return 1;
> +
> + lock_page_cgroup(page);
> + pc = page_get_page_cgroup(page);
> + if (pc) {
> +  struct mem_cgroup *mem2 = pc->mem_cgroup;
> +  unsigned long long min_usage;
> +
> +  BUG_ON(mem2 == NULL);
> +  min_usage = mem2->min_usage;
> +  if (min_usage != 0) {
> +   unsigned long flags;
> +
> +   spin_lock_irqsave(&mem2->res.lock, flags);
> +   if (mem2->res.usage <= min_usage)
> +    result = 0;
> +   spin_unlock_irqrestore(&mem2->res.lock, flags);
> +  }
> + }
> + unlock_page_cgroup(page);
> +
> + return result;
> +}
> +
```

How about adding lock_and_get_page_cgroup(page)/put_page_cgroup() ?
==
```
struct page_cgroup *pc lock_and_get_page_cgroup(page)
{
 struct page_cgroup *pc;

 lock_page_cgroup(page);
 pc = page_get_page_cgroup(page);
 if (atomic_inc_not_zero(&pc->ref_cnt))
  pc = NULL;
 unlock_page_cgroup(page);
 return pc;
}

struct page_cgroup *pc put_page_cgroup(struct page_cgroup *pc)
{
 mem_cgroup_uncharge(pc);
}
```
==

BTW, how about add a status to res_counter ?
My (based on your) current patch uses watermark_state.

Maybe we can change it to be resource_state and show following status.

RES_STATE_HIT_LIMIT,
RES_STATE_ABOVE_HWATER,
RES_STATE_ABOVE_LWATER,
RES_STATE_STABLE, ?
RES_STATE_BELOW_MIN,

Useful ?

> static int alloc_mem_cgroup_per_zone_info(struct mem_cgroup *mem, int node)
> {
> struct mem_cgroup_per_node *pn;
> --- linux-2.6.24-rc3-mm2-swappiness/mm/vmscan.c.BACKUP2 2007-12-03 13:52:22.000000000 +0900
> +++ linux-2.6.24-rc3-mm2-swappiness/mm/vmscan.c 2007-12-03 14:11:42.000000000 +0900
> @@ -531,6 +531,11 @@ static unsigned long shrink_page_list(st
> referenced && page_mapping_inuse(page))
> goto activate_locked;
>
> +#ifdef CONFIG_CGROUP_MEM_CONT
> + if (!mem_cgroup_canreclaim(page, sc->mem_cgroup))
> + goto activate_locked;
> +#endif /* CONFIG_CGROUP_MEM_CONT */
> +

Maybe
==
if (scan_global_lru(sc) && !
mem_cgroup_canreclaim(page, sc->mem-cgroup))
goto activate_locked:
==

> #ifdef CONFIG_SWAP

> /*
> * Anonymous process memory has backing store?
> @@ -1122,6 +1127,12 @@ static void shrink_active_list(unsigned
> list_add(&page->lru, &l_active);
> continue;
> }
> +#ifdef CONFIG_CGROUP_MEM_CONT
> + if (!mem_cgroup_canreclaim(page, sc->mem_cgroup)) {
> + list_add(&page->lru, &l_active);

> +   continue;
> + }
here too.

Thanks,
-Kame

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] memory.min_usage
Posted by Paul Menage on Tue, 04 Dec 2007 06:03:57 GMT
View Forum Message <> Reply to Message

On Dec 3, 2007 8:09 PM, YAMAMOTO Takashi <yamamoto@valinux.co.jp> wrote:
> +static u64 mem_cgroup_min_usage_read(struct cgroup *cont, struct cftype *cft)
> +{
> +   struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
> +
> +   return mem->min_usage;
> +}
> +

This won't be atomic on a 32-bit arch, and I don't think you have any locking.

Paul

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] memory.min_usage
Posted by yamamoto on Tue, 04 Dec 2007 07:01:21 GMT
View Forum Message <> Reply to Message

> > +int mem_cgroup_canreclaim(struct page *page, struct mem_cgroup *mem1)
> > +{
> > + struct page_cgroup *pc;
> > + int result = 1;
> > +
> > + if (mem1 != NULL)
> > + return 1;
> > +

```
> > + lock_page_cgroup(page);
> > + pc = page_get_page_cgroup(page);
> > + if (pc) {
> > +  struct mem_cgroup *mem2 = pc->mem_cgroup;
> > +  unsigned long long min_usage;
> > +
> > +  BUG_ON(mem2 == NULL);
> > +  min_usage = mem2->min_usage;
> > +  if (min_usage != 0) {
> > +   unsigned long flags;
> > +
> > +   spin_lock_irqsave(&mem2->res.lock, flags);
> > +   if (mem2->res.usage <= min_usage)
> > +    result = 0;
> > +   spin_unlock_irqrestore(&mem2->res.lock, flags);
> > +  }
> > + }
> > + unlock_page_cgroup(page);
> > +
> > + return result;
> > +}
> > +
>
> How about adding lock_and_get_page_cgroup(page)/put_page_cgroup() ?
> ==
> struct page_cgroup *pc lock_and_get_page_cgroup(page)
> {
>  struct page_cgroup *pc;
>
>  lock_page_cgroup(page);
>  pc = page_get_page_cgroup(page);
>  if (atomic_inc_not_zero(&pc->ref_cnt))
>   pc = NULL;
>  unlock_page_cgroup(page);
>  return pc;
> }
>
> struct page_cgroup *pc put_page_cgroup(struct page_cgroup *pc)
> {
>  mem_cgroup_uncharge(pc);
> }
> ==
```

i'm not sure if it's a good idea given that reference counting (atomic_foo)
has its own costs.

> BTW, how about add a status to res_counter ?
> My (based on your) current patch uses watermark_state.

>
> Maybe we can change it to be resource_state and show following status.
>
> RES_STATE_HIT_LIMIT,
> RES_STATE_ABOVE_HWATER,
> RES_STATE_ABOVE_LWATER,
> RES_STATE_STABLE, ?
> RES_STATE_BELOW_MIN,
>
> Useful ?

how about making res_counter use seq_lock?

> > static int alloc_mem_cgroup_per_zone_info(struct mem_cgroup *mem, int node)
> > {
> > struct mem_cgroup_per_node *pn;
> > --- linux-2.6.24-rc3-mm2-swappiness/mm/vmscan.c.BACKUP2 2007-12-03
13:52:22.000000000 +0900
> > +++ linux-2.6.24-rc3-mm2-swappiness/mm/vmscan.c 2007-12-03 14:11:42.000000000 +0900
> > @@ -531,6 +531,11 @@ static unsigned long shrink_page_list(st
> > referenced && page_mapping_inuse(page))
> > goto activate_locked;
> >
> > +#ifdef CONFIG_CGROUP_MEM_CONT
> > + if (!mem_cgroup_canreclaim(page, sc->mem_cgroup))
> > + goto activate_locked;
> > +#endif /* CONFIG_CGROUP_MEM_CONT */
> > +
>
> Maybe
> ==
> if (scan_global_lru(sc) && !
> mem_cgroup_canreclaim(page, sc->mem-cgroup))
> goto activate_locked:
> ==

i don't think the decision belongs to callers.
(at least it wasn't my intention.)

YAMAMOTO Takashi

Subject: Re: [PATCH] memory.min_usage

Posted by yamamoto on Tue, 04 Dec 2007 07:02:57 GMT
View Forum Message <> Reply to Message

> On Dec 3, 2007 8:09 PM, YAMAMOTO Takashi <yamamoto@valinux.co.jp> wrote:
> > +static u64 mem_cgroup_min_usage_read(struct cgroup *cont, struct cftype *cft)
> > +{
> > +	struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
> > +
> > +	return mem->min_usage;
> > +}
> > +
>
> This won't be atomic on a 32-bit arch, and I don't think you have any locking.
>
> Paul

sure. probably it's better to make it a part of res_counter.

YAMAMOTO Takashi

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] memory.min_usage
Posted by KAMEZAWA Hiroyuki on Tue, 04 Dec 2007 07:27:53 GMT
View Forum Message <> Reply to Message

On Tue, 4 Dec 2007 16:01:21 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> > BTW, how about add a status to res_counter ?
> > My (based on your) current patch uses watermark_state.
> >
> > Maybe we can change it to be resource_state and show following status.
> >
> > RES_STATE_HIT_LIMIT,
> > RES_STATE_ABOVE_HWATER,
> > RES_STATE_ABOVE_LWATER,
> > RES_STATE_STABLE, ?
> > RES_STATE_BELOW_MIN,
> >
> > Useful ?
>
> how about making res_counter use seq_lock?
>
I thought of that when I wrote hi/low watermark patches.

But I didn't try....
Hmm, but maybe worth trying.

Pavel, can we change res_counter->lock to seq_lock ?
If we can, I'll write a patch.

Thanks,
-Kame

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] memory.min_usage
Posted by KAMEZAWA Hiroyuki on Tue, 04 Dec 2007 07:45:59 GMT
View Forum Message <> Reply to Message

On Tue, 4 Dec 2007 16:27:53 +0900
KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> I thought of that when I wrote hi/low watermark patches.
> But I didn't try....
> Hmm, but maybe worth trying.
>
But even with seqlock, we'll have to disable irq.
And maybe my current high/low patch needs smp_rmb().

I'll consider a bit more.

Thanks,
-Kame

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] memory.min_usage
Posted by yamamoto on Tue, 04 Dec 2007 07:58:54 GMT
View Forum Message <> Reply to Message

> But even with seqlock, we'll have to disable irq.

for writers, sure.
readers don't need to disable irq.

YAMAMOTO Takashi

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] memory.min_usage (seqlock for res_counter)
Posted by KAMEZAWA Hiroyuki on Tue, 04 Dec 2007 10:54:36 GMT
View Forum Message <> Reply to Message

This is seqlock version res_counter.
Maybe this this will reduce # of spin_lock.

Pavel-san, How about this ?

-Kame
==
resource counter's spinlock can be seqlock. (res_counter doesn't include
any pointers.)

And it will be good to avoid atomic ops if we can when we just checks
value.

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

```
 include/linux/res_counter.h |   24 +++++++----------------
 kernel/res_counter.c        |   14 +++++++-------
 2 files changed, 14 insertions(+), 24 deletions(-)

Index: linux-2.6.24-rc3-mm2/include/linux/res_counter.h
===================================================================
--- linux-2.6.24-rc3-mm2.orig/include/linux/res_counter.h
+++ linux-2.6.24-rc3-mm2/include/linux/res_counter.h
@@ -12,6 +12,7 @@
  */

 #include <linux/cgroup.h>
+#include <linux/seqlock.h>

 /*
  * The core object. the cgroup that wishes to account for some
@@ -36,7 +37,7 @@ struct res_counter {
   * the lock to protect all of the above.
```

```
   * the routines below consider this to be IRQ-safe
   */
- spinlock_t lock;
+ seqlock_t lock;
 };

 /*
@@ -101,27 +102,17 @@ int res_counter_charge(struct res_counte
 void res_counter_uncharge_locked(struct res_counter *counter, unsigned long val);
 void res_counter_uncharge(struct res_counter *counter, unsigned long val);

-static inline bool res_counter_limit_check_locked(struct res_counter *cnt)
-{
- if (cnt->usage < cnt->limit)
-  return true;
-
- return false;
-}
-
-/*
- * Helper function to detect if the cgroup is within it's limit or
- * not. It's currently called from cgroup_rss_prepare()
- */
 static inline bool res_counter_check_under_limit(struct res_counter *cnt)
 {
  bool ret;
- unsigned long flags;
+ unsigned long seq;

- spin_lock_irqsave(&cnt->lock, flags);
- ret = res_counter_limit_check_locked(cnt);
- spin_unlock_irqrestore(&cnt->lock, flags);
+ do {
+  seq = read_seqbegin(&cnt->lock);
+  ret = (cnt->usage < cnt->limit);
+ } while (read_seqretry(&cnt->lock, seq));
  return ret;
 }

+
 #endif
Index: linux-2.6.24-rc3-mm2/kernel/res_counter.c
===================================================================
--- linux-2.6.24-rc3-mm2.orig/kernel/res_counter.c
+++ linux-2.6.24-rc3-mm2/kernel/res_counter.c
@@ -15,7 +15,7 @@

 void res_counter_init(struct res_counter *counter)
```

```
 {
- spin_lock_init(&counter->lock);
+ seqlock_init(&counter->lock);
  counter->limit = (unsigned long long)LLONG_MAX;
 }

@@ -35,9 +35,9 @@ int res_counter_charge(struct res_counte
  int ret;
  unsigned long flags;

- spin_lock_irqsave(&counter->lock, flags);
+ write_seqlock_irqsave(&counter->lock, flags);
  ret = res_counter_charge_locked(counter, val);
- spin_unlock_irqrestore(&counter->lock, flags);
+ write_sequnlock_irqrestore(&counter->lock, flags);
  return ret;
 }

@@ -53,9 +53,9 @@ void res_counter_uncharge(struct res_cou
 {
  unsigned long flags;

- spin_lock_irqsave(&counter->lock, flags);
+ write_seqlock_irqsave(&counter->lock, flags);
  res_counter_uncharge_locked(counter, val);
- spin_unlock_irqrestore(&counter->lock, flags);
+ write_sequnlock_irqrestore(&counter->lock, flags);
 }


@@ -122,10 +122,10 @@ ssize_t res_counter_write(struct res_cou
    if (*end != '\0')
     goto out_free;
  }
- spin_lock_irqsave(&counter->lock, flags);
+ write_seqlock_irqsave(&counter->lock, flags);
  val = res_counter_member(counter, member);
  *val = tmp;
- spin_unlock_irqrestore(&counter->lock, flags);
+ write_sequnlock_irqrestore(&counter->lock, flags);
  ret = nbytes;
 out_free:
  kfree(buf);
```

Subject: Re: [PATCH] memory.min_usage (seqlock for res_counter)
Posted by Pavel Emelianov on Tue, 04 Dec 2007 11:10:42 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> This is seqlock version res_counter.
> Maybe this this will reduce # of spin_lock.
>
> Pavel-san, How about this ?

AFAIS the readlock is used only in the check_under_limit(),
but I think, that even if we read usage and limit values
in this case non-atomically, this won't result in any
dramatic sequence at all. No?

On the other hand, we make the charge/uncharge routines
work longer :(

> -Kame
> ==
> resource counter's spinlock can be seqlock. (res_counter doesn't include
> any pointers.)
>
> And it will be good to avoid atomic ops if we can when we just checks
> value.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
>
>
>
> include/linux/res_counter.h | 24 +++++++----------------
> kernel/res_counter.c | 14 +++++++-------
> 2 files changed, 14 insertions(+), 24 deletions(-)
>
> Index: linux-2.6.24-rc3-mm2/include/linux/res_counter.h
> ===
> --- linux-2.6.24-rc3-mm2.orig/include/linux/res_counter.h
> +++ linux-2.6.24-rc3-mm2/include/linux/res_counter.h
> @@ -12,6 +12,7 @@
>   */
>
> #include <linux/cgroup.h>
> +#include <linux/seqlock.h>
>
> /*
>   * The core object. the cgroup that wishes to account for some
> @@ -36,7 +37,7 @@ struct res_counter {
>   * the lock to protect all of the above.

>   * the routines below consider this to be IRQ-safe
>   */
> - spinlock_t lock;
> + seqlock_t lock;
> };
>
> /*
> @@ -101,27 +102,17 @@ int res_counter_charge(struct res_counte
> void res_counter_uncharge_locked(struct res_counter *counter, unsigned long val);
> void res_counter_uncharge(struct res_counter *counter, unsigned long val);
>
> -static inline bool res_counter_limit_check_locked(struct res_counter *cnt)
> -{
> - if (cnt->usage < cnt->limit)
> - return true;
> -
> - return false;
> -}
> -
> -/*
> - * Helper function to detect if the cgroup is within it's limit or
> - * not. It's currently called from cgroup_rss_prepare()
> - */
> static inline bool res_counter_check_under_limit(struct res_counter *cnt)
> {
>   bool ret;
> - unsigned long flags;
> + unsigned long seq;
>
> - spin_lock_irqsave(&cnt->lock, flags);
> - ret = res_counter_limit_check_locked(cnt);
> - spin_unlock_irqrestore(&cnt->lock, flags);
> + do {
> + seq = read_seqbegin(&cnt->lock);
> + ret = (cnt->usage < cnt->limit);
> + } while (read_seqretry(&cnt->lock, seq));
>   return ret;
> }
>
> +
> #endif
> Index: linux-2.6.24-rc3-mm2/kernel/res_counter.c
> ===
> --- linux-2.6.24-rc3-mm2.orig/kernel/res_counter.c
> +++ linux-2.6.24-rc3-mm2/kernel/res_counter.c
> @@ -15,7 +15,7 @@
>
> void res_counter_init(struct res_counter *counter)

```
>  {
> - spin_lock_init(&counter->lock);
> + seqlock_init(&counter->lock);
>    counter->limit = (unsigned long long)LLONG_MAX;
>  }
>
> @@ -35,9 +35,9 @@ int res_counter_charge(struct res_counte
>    int ret;
>    unsigned long flags;
>
> - spin_lock_irqsave(&counter->lock, flags);
> + write_seqlock_irqsave(&counter->lock, flags);
>    ret = res_counter_charge_locked(counter, val);
> - spin_unlock_irqrestore(&counter->lock, flags);
> + write_sequnlock_irqrestore(&counter->lock, flags);
>    return ret;
>  }
>
> @@ -53,9 +53,9 @@ void res_counter_uncharge(struct res_cou
>  {
>    unsigned long flags;
>
> - spin_lock_irqsave(&counter->lock, flags);
> + write_seqlock_irqsave(&counter->lock, flags);
>    res_counter_uncharge_locked(counter, val);
> - spin_unlock_irqrestore(&counter->lock, flags);
> + write_sequnlock_irqrestore(&counter->lock, flags);
>  }
>
>
> @@ -122,10 +122,10 @@ ssize_t res_counter_write(struct res_cou
>     if (*end != '\0')
>       goto out_free;
>    }
> - spin_lock_irqsave(&counter->lock, flags);
> + write_seqlock_irqsave(&counter->lock, flags);
>    val = res_counter_member(counter, member);
>    *val = tmp;
> - spin_unlock_irqrestore(&counter->lock, flags);
> + write_sequnlock_irqrestore(&counter->lock, flags);
>    ret = nbytes;
>  out_free:
>    kfree(buf);
>
>
```

_____

Containers mailing list

Subject: Re: [PATCH] memory.min_usage
Posted by Balbir Singh on Tue, 04 Dec 2007 13:30:25 GMT
View Forum Message <> Reply to Message

YAMAMOTO Takashi wrote:
> hi,
>
> here's a patch to implement memory.min_usage,
> which controls the minimum memory usage for a cgroup.
>
> it works similarly to mlock;
> global memory reclamation doesn't reclaim memory from
> cgroups whose memory usage is below the value.
> setting it too high is a dangerous operation.
>
> it's against 2.6.24-rc3-mm2 + memory.swappiness patch i posted here yesterday.
> but it's logically independent from the swappiness patch.
>

Hi,

I think it's a good approach, but we need to make the reclaim dependent
on priority. I think that if we stop reclaiming from a cgroup even
though there is high global pressure, it might be a bad idea.

> todo:
> - restrict non-root user's operation ragardless of owner of cgroupfs files?

Sorry, I don't understand this and I think using file permissions is a
good idea.

> - make oom killer aware of this?
>

Yes, we might also need a memory controller version of

1. overcommit
2. oom_adj* parameters

> YAMAMOTO Takashi
>
>
> Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

```
> ---
>
> --- linux-2.6.24-rc3-mm2-swappiness/include/linux/memcontrol.h.BACKUP2 2007-12-03
13:52:37.000000000 +0900
> +++ linux-2.6.24-rc3-mm2-swappiness/include/linux/memcontrol.h 2007-12-03
14:07:45.000000000 +0900
> @@ -47,6 +47,7 @@ extern int mem_cgroup_cache_charge(struc
>       gfp_t gfp_mask);
>  int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem);
>  extern int mem_cgroup_swappiness(struct mem_cgroup *mem);
> +extern int mem_cgroup_canreclaim(struct page *page, struct mem_cgroup *mem);
>
>  static inline struct mem_cgroup *mm_cgroup(const struct mm_struct *mm)
>  {
> --- linux-2.6.24-rc3-mm2-swappiness/mm/memcontrol.c.BACKUP2 2007-12-03
13:52:13.000000000 +0900
> +++ linux-2.6.24-rc3-mm2-swappiness/mm/memcontrol.c 2007-12-04 11:42:07.000000000
+0900
> @@ -134,6 +134,7 @@ struct mem_cgroup {
>   unsigned long control_type; /* control RSS or RSS+Pagecache */
>   int prev_priority; /* for recording reclaim priority */
>   unsigned int swappiness; /* swappiness */
> + unsigned long long min_usage; /* XXX should be a part of res_counter? */
>   /*
>    * statistics.
>    */
> @@ -1096,6 +1097,22 @@ static u64 mem_cgroup_swappiness_read(st
>   return mem->swappiness;
>  }
>
> +static int mem_cgroup_min_usage_write(struct cgroup *cont, struct cftype *cft,
> +          u64 val)
> +{
> + struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
> +
> + mem->min_usage = val;
> + return 0;
> +}
> +
> +static u64 mem_cgroup_min_usage_read(struct cgroup *cont, struct cftype *cft)
> +{
> + struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
> +
> + return mem->min_usage;
> +}
> +
>  static struct cftype mem_cgroup_files[] = {
>   {
```

```
>    .name = "usage_in_bytes",
> @@ -1132,6 +1149,11 @@ static struct cftype mem_cgroup_files[]
>    .write_uint = mem_cgroup_swappiness_write,
>    .read_uint = mem_cgroup_swappiness_read,
>   },
> + {
> +  .name = "min_usage_in_bytes",
> +  .write_uint = mem_cgroup_min_usage_write,
> +  .read_uint = mem_cgroup_min_usage_read,
> + },
>  };
>
>  /* XXX probably it's better to move try_to_free_mem_cgroup_pages to
> @@ -1142,6 +1164,36 @@ int mem_cgroup_swappiness(struct mem_cgr
>   return mem->swappiness;
>  }
>
> +int mem_cgroup_canreclaim(struct page *page, struct mem_cgroup *mem1)
> +{
> + struct page_cgroup *pc;
> + int result = 1;
> +
> + if (mem1 != NULL)
> +  return 1;
> +
> + lock_page_cgroup(page);
> + pc = page_get_page_cgroup(page);
> + if (pc) {
> +  struct mem_cgroup *mem2 = pc->mem_cgroup;
> +  unsigned long long min_usage;
> +
> +  BUG_ON(mem2 == NULL);
> +  min_usage = mem2->min_usage;
> +  if (min_usage != 0) {
> +   unsigned long flags;
> +
> +   spin_lock_irqsave(&mem2->res.lock, flags);
> +   if (mem2->res.usage <= min_usage)
> +    result = 0;
> +   spin_unlock_irqrestore(&mem2->res.lock, flags);
> +  }
> + }
> + unlock_page_cgroup(page);
> +
> + return result;
> +}
> +
>  static int alloc_mem_cgroup_per_zone_info(struct mem_cgroup *mem, int node)
```

> {
> struct mem_cgroup_per_node *pn;
> --- linux-2.6.24-rc3-mm2-swappiness/mm/vmscan.c.BACKUP2 2007-12-03 13:52:22.000000000 +0900
> +++ linux-2.6.24-rc3-mm2-swappiness/mm/vmscan.c 2007-12-03 14:11:42.000000000 +0900
> @@ -531,6 +531,11 @@ static unsigned long shrink_page_list(st
> referenced && page_mapping_inuse(page))
> goto activate_locked;
>
> +#ifdef CONFIG_CGROUP_MEM_CONT
> + if (!mem_cgroup_canreclaim(page, sc->mem_cgroup))
> + goto activate_locked;
> +#endif /* CONFIG_CGROUP_MEM_CONT */
> +
> #ifdef CONFIG_SWAP
> /*
> * Anonymous process memory has backing store?
> @@ -1122,6 +1127,12 @@ static void shrink_active_list(unsigned
> list_add(&page->lru, &l_active);
> continue;
> }
> +#ifdef CONFIG_CGROUP_MEM_CONT
> + if (!mem_cgroup_canreclaim(page, sc->mem_cgroup)) {
> + list_add(&page->lru, &l_active);
> + continue;
> + }
> +#endif /* CONFIG_CGROUP_MEM_CONT */
> list_add(&page->lru, &l_inactive);
> }
>
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] memory.min_usage (seqlock for res_counter)
Posted by KAMEZAWA Hiroyuki on Wed, 05 Dec 2007 00:34:55 GMT
View Forum Message <> Reply to Message

On Tue, 04 Dec 2007 14:10:42 +0300
Pavel Emelyanov <xemul@openvz.org> wrote:

> KAMEZAWA Hiroyuki wrote:
> > This is seqlock version res_counter.
> > Maybe this this will reduce # of spin_lock.
> >
> > Pavel-san, How about this ?
>
> AFAIS the readlock is used only in the check_under_limit(),
> but I think, that even if we read usage and limit values
> in this case non-atomically, this won't result in any
> dramatic sequence at all. No?
>
Reader can detect *any* changes in res_counter member which happens
while they access res_counter between seq_begin/seq_retry.
Memory barrier and "sequence" of seq_lock guarantees this.
So..there is no dramatical situation.
(it's used for accesing xtime.)

I'm sorry if I miss your point.

Thanks,
-Kame

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH] memory.min_usage (seqlock for res_counter)
Posted by Pavel Emelianov on Wed, 05 Dec 2007 09:12:22 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> On Tue, 04 Dec 2007 14:10:42 +0300
> Pavel Emelyanov <xemul@openvz.org> wrote:
>
>> KAMEZAWA Hiroyuki wrote:
>>> This is seqlock version res_counter.
>>> Maybe this this will reduce # of spin_lock.
>>>
>>> Pavel-san, How about this ?

>> AFAIS the readlock is used only in the check_under_limit(),
>> but I think, that even if we read usage and limit values
>> in this case non-atomically, this won't result in any
>> dramatic sequence at all. No?
>>
> Reader can detect *any* changes in res_counter member which happens
> while they access res_counter between seq_begin/seq_retry.
> Memory barrier and "sequence" of seq_lock guarantees this.
> So..there is no dramatical situation.
> (it's used for accesing xtime.)
>
> I'm sorry if I miss your point.

Sorry, let me explain it in other words.

I think, that protection in reader, that guarantees that it
will see the valid result, is not very important - even if
we compare usage and limit not atomically nothing serious
will happen (in this particular case)

> Thanks,
> -Kame
>
>

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] memory.min_usage (seqlock for res_counter)
Posted by KAMEZAWA Hiroyuki on Wed, 05 Dec 2007 09:29:27 GMT
View Forum Message <> Reply to Message

On Wed, 05 Dec 2007 12:12:22 +0300
Pavel Emelyanov <xemul@openvz.org> wrote:

> Sorry, let me explain it in other words.
>
> I think, that protection in reader, that guarantees that it
> will see the valid result, is not very important - even if
> we compare usage and limit not atomically nothing serious
> will happen (in this particular case)
>
Maybe there is no serious situation (now).
But programmers don't assume that the function may not return trustable result.
And I think it shouldn be trustable AMAP.

I'd like to use seq_lock or res_counter_state, here.

BTW, I'm wondering I should hold off my patches until 2.6.25-rc series if they
make things complex.

Thanks,
-Kame

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH] memory.min_usage (seqlock for res_counter)
Posted by Pavel Emelianov on Wed, 05 Dec 2007 09:32:26 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> On Wed, 05 Dec 2007 12:12:22 +0300
> Pavel Emelyanov <xemul@openvz.org> wrote:
>
>> Sorry, let me explain it in other words.
>>
>> I think, that protection in reader, that guarantees that it
>> will see the valid result, is not very important - even if
>> we compare usage and limit not atomically nothing serious
>> will happen (in this particular case)
>>
> Maybe there is no serious situation (now).
> But programmers don't assume that the function may not return trustable result.
> And I think it shouldn be trustable AMAP.

Well... OK. Among other possible ways to achieve this goal
seqlocks is the most preferable one from my POV.

Thanks :)

> I'd like to use seq_lock or res_counter_state, here.
>
> BTW, I'm wondering I should hold off my patches until 2.6.25-rc series if they
> make things complex.

Actually, Andrew wrote that he will pay little attention to
new functionality till 2.6.24 release, so I think that serious
patches should really be held off.

That's why I don't send the kmem controller yet :(

> Thanks,
> -Kame

Thanks,
Pavel

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers