Subject: [RFC] [PATCH] memory controller statistics
Posted by yamamoto on Fri, 07 Sep 2007 03:39:42 GMT
View Forum Message <> Reply to Message

hi,

i implemented some statistics for your memory controller.

it's tested with 2.6.23-rc2-mm2 + memory controller v7.
i think it can be applied to 2.6.23-rc4-mm1 as well.

YAMOMOTO Takshi

todo: something like nr_active/inactive in /proc/vmstat.

```
--- ./mm/memcontrol.c.BACKUP 2007-08-29 17:13:09.000000000 +0900
+++ ./mm/memcontrol.c 2007-09-06 16:26:13.000000000 +0900
@@ -24,6 +24,7 @@
 #include <linux/page-flags.h>
 #include <linux/bit_spinlock.h>
 #include <linux/rcupdate.h>
+#include <linux/seq_file.h>
 #include <linux/swap.h>
 #include <linux/spinlock.h>
 #include <linux/fs.h>
@@ -33,6 +34,54 @@
 struct container_subsys mem_container_subsys;
 static const int MEM_CONTAINER_RECLAIM_RETRIES = 5;

+enum mem_container_stat_index {
+ /*
+  * for MEM_CONTAINER_TYPE_ALL, usage == pagecache + rss
+  */
+ MEMCONT_STAT_PAGECACHE,
+ MEMCONT_STAT_RSS,
+
+ /*
+  * redundant; usage == charge - uncharge
+  */
+ MEMCONT_STAT_CHARGE,
+ MEMCONT_STAT_UNCHARGE,
+
+ /*
+  * mostly for debug
+  */
+ MEMCONT_STAT_ISOLATE,
+ MEMCONT_STAT_ISOLATE_FAIL,
+ MEMCONT_STAT_NSTATS,
```

```
+};
+
+static const char * const mem_container_stat_desc[] = {
+ [MEMCONT_STAT_PAGECACHE] = "page_cache",
+ [MEMCONT_STAT_RSS] = "rss",
+ [MEMCONT_STAT_CHARGE] = "charge",
+ [MEMCONT_STAT_UNCHARGE] = "uncharge",
+ [MEMCONT_STAT_ISOLATE] = "isolate",
+ [MEMCONT_STAT_ISOLATE_FAIL] = "isolate_fail",
+};
+
+struct mem_container_stat {
+ atomic_t count[MEMCONT_STAT_NSTATS];
+};
+
+static void mem_container_stat_inc(struct mem_container_stat * stat,
+   enum mem_container_stat_index idx)
+{
+
+ atomic_inc(&stat->count[idx]);
+}
+
+static void mem_container_stat_dec(struct mem_container_stat * stat,
+   enum mem_container_stat_index idx)
+{
+
+ atomic_dec(&stat->count[idx]);
+}
+
 /*
  * The memory controller data structure. The memory controller controls both
  * page cache and RSS per container. We would eventually like to provide
@@ -62,6 +111,7 @@ struct mem_container {
   */
  spinlock_t lru_lock;
  unsigned long control_type; /* control RSS or RSS+Pagecache */
+ struct mem_container_stat stat;
 };

 /*
@@ -72,6 +122,12 @@ struct mem_container {
 #define PAGE_CONTAINER_LOCK_BIT  0x0
 #define PAGE_CONTAINER_LOCK   (1 << PAGE_CONTAINER_LOCK_BIT)

+/* XXX hack; shouldn't be here.  it really belongs to struct page_container. */
+#define PAGE_CONTAINER_CACHE_BIT  0x1
+#define PAGE_CONTAINER_CACHE   (1 << PAGE_CONTAINER_CACHE_BIT)
+
```

```
+#define PAGE_CONTAINER_FLAGS (PAGE_CONTAINER_LOCK |
PAGE_CONTAINER_CACHE)
+
 /*
  * A page_container page is associated with every page descriptor. The
  * page_container helps us identify information about the container
@@ -134,9 +190,9 @@ static inline int page_container_locked(
     &page->page_container);
 }

-void page_assign_page_container(struct page *page, struct page_container *pc)
+static void page_assign_page_container_flags(struct page *page, int flags,
+    struct page_container *pc)
 {
- int locked;

  /*
   * While resetting the page_container we might not hold the
@@ -145,14 +201,20 @@ void page_assign_page_container(struct p
   */
  if (pc)
   VM_BUG_ON(!page_container_locked(page));
- locked = (page->page_container & PAGE_CONTAINER_LOCK);
- page->page_container = ((unsigned long)pc | locked);
+ flags |= (page->page_container & PAGE_CONTAINER_LOCK);
+ page->page_container = ((unsigned long)pc | flags);
+}
+
+void page_assign_page_container(struct page *page, struct page_container *pc)
+{
+
+ page_assign_page_container_flags(page, 0, pc);
 }

 struct page_container *page_get_page_container(struct page *page)
 {
  return (struct page_container *)
-  (page->page_container & ~PAGE_CONTAINER_LOCK);
+  (page->page_container & ~PAGE_CONTAINER_FLAGS);
 }

 void __always_inline lock_page_container(struct page *page)
@@ -203,6 +265,7 @@ unsigned long mem_container_isolate_page
  LIST_HEAD(pc_list);
  struct list_head *src;
  struct page_container *pc;
+ struct mem_container_stat *stat = &mem_cont->stat;
```

```
  if (active)
   src = &mem_cont->active_list;
@@ -244,6 +307,9 @@ unsigned long mem_container_isolate_page
   if (__isolate_lru_page(page, mode) == 0) {
    list_move(&page->lru, dst);
    nr_taken++;
+   mem_container_stat_inc(stat, MEMCONT_STAT_ISOLATE);
+  } else {
+   mem_container_stat_inc(stat, MEMCONT_STAT_ISOLATE_FAIL);
   }
  }

@@ -260,9 +326,11 @@ unsigned long mem_container_isolate_page
  * 0 if the charge was successful
  * < 0 if the container is over its limit
  */
-int mem_container_charge(struct page *page, struct mm_struct *mm)
+static int mem_container_charge_common(struct page *page, struct mm_struct *mm,
+    int is_cache)
 {
  struct mem_container *mem;
+ struct mem_container_stat *stat;
  struct page_container *pc, *race_pc;
  unsigned long flags;
  unsigned long nr_retries = MEM_CONTAINER_RECLAIM_RETRIES;
@@ -360,7 +428,16 @@ int mem_container_charge(struct page *pa
  atomic_set(&pc->ref_cnt, 1);
  pc->mem_container = mem;
  pc->page = page;
- page_assign_page_container(page, pc);
+ page_assign_page_container_flags(page,
+     is_cache ? PAGE_CONTAINER_CACHE : 0, pc);
+
+ stat = &mem->stat;
+ if (is_cache) {
+  mem_container_stat_inc(stat, MEMCONT_STAT_PAGECACHE);
+ } else {
+  mem_container_stat_inc(stat, MEMCONT_STAT_RSS);
+ }
+ mem_container_stat_inc(stat, MEMCONT_STAT_CHARGE);

  spin_lock_irqsave(&mem->lru_lock, flags);
  list_add(&pc->lru, &mem->active_list);
@@ -377,6 +454,12 @@ err:
  return -ENOMEM;
 }

+int mem_container_charge(struct page *page, struct mm_struct *mm)
```

```
+{
+
+ return mem_container_charge_common(page, mm, 0);
+}
+
 /*
  * See if the cached pages should be charged at all?
  */
@@ -388,7 +471,7 @@ int mem_container_cache_charge(struct pa

  mem = rcu_dereference(mm->mem_container);
  if (mem->control_type == MEM_CONTAINER_TYPE_ALL)
-  return mem_container_charge(page, mm);
+  return mem_container_charge_common(page, mm, 1);
  else
   return 0;
 }
@@ -411,15 +494,29 @@ void mem_container_uncharge(struct page_
   return;

  if (atomic_dec_and_test(&pc->ref_cnt)) {
+  struct mem_container_stat *stat;
+  int is_cache;
+
   page = pc->page;
   lock_page_container(page);
   mem = pc->mem_container;
   css_put(&mem->css);
+  /* XXX */
+  is_cache = (page->page_container & PAGE_CONTAINER_CACHE) != 0;
   page_assign_page_container(page, NULL);
   unlock_page_container(page);
   res_counter_uncharge(&mem->res, 1);

+  stat = &mem->stat;
+  if (is_cache) {
+   mem_container_stat_dec(stat, MEMCONT_STAT_PAGECACHE);
+  } else {
+   mem_container_stat_dec(stat, MEMCONT_STAT_RSS);
+  }
+  mem_container_stat_inc(stat, MEMCONT_STAT_UNCHARGE);
+
    spin_lock_irqsave(&mem->lru_lock, flags);
+  BUG_ON(list_empty(&pc->lru));
    list_del_init(&pc->lru);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
   kfree(pc);
@@ -496,6 +593,44 @@ static ssize_t mem_control_type_read(str
```

```
    ppos, buf, s - buf);
 }

+static void mem_container_stat_init(struct mem_container_stat *stat)
+{
+ int i;
+
+ for (i = 0; i < ARRAY_SIZE(stat->count); i++) {
+  atomic_set(&stat->count[i], 0);
+ }
+}
+
+static int mem_control_stat_show(struct seq_file *m, void *arg)
+{
+ struct container *cont = m->private;
+ struct mem_container *mem_cont = mem_container_from_cont(cont);
+ struct mem_container_stat *stat = &mem_cont->stat;
+ int i;
+
+ for (i = 0; i < ARRAY_SIZE(stat->count); i++) {
+  seq_printf(m, "%s %u\n", mem_container_stat_desc[i],
+       (unsigned int)atomic_read(&stat->count[i]));
+ }
+ return 0;
+}
+
+static const struct file_operations mem_control_stat_file_operations = {
+ .read = seq_read,
+ .llseek = seq_lseek,
+ .release = single_release,
+};
+
+static int mem_control_stat_open(struct inode *unused, struct file *file)
+{
+ /* XXX __d_cont */
+ struct container *cont = file->f_dentry->d_parent->d_fsdata;
+
+ file->f_op = &mem_control_stat_file_operations;
+ return single_open(file, mem_control_stat_show, cont);
+}
+
 static struct cftype mem_container_files[] = {
  {
   .name = "usage",
@@ -518,6 +653,10 @@ static struct cftype mem_container_files
   .write = mem_control_type_write,
   .read = mem_control_type_read,
  },
```

```
+ {
+  .name = "stat",
+  .open = mem_control_stat_open,
+ },
 };

 static struct mem_container init_mem_container;
@@ -541,6 +680,7 @@ mem_container_create(struct container_su
  INIT_LIST_HEAD(&mem->inactive_list);
  spin_lock_init(&mem->lru_lock);
  mem->control_type = MEM_CONTAINER_TYPE_ALL;
+ mem_container_stat_init(&mem->stat);
  return &mem->css;
 }
```

_____

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

---

Subject: Re: [RFC] [PATCH] memory controller statistics
Posted by KAMEZAWA Hiroyuki on Fri, 07 Sep 2007 05:02:34 GMT
View Forum Message <> Reply to Message

Hi,

On Fri, 7 Sep 2007 12:39:42 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

```
> +enum mem_container_stat_index {
> + /*
> +  * for MEM_CONTAINER_TYPE_ALL, usage == pagecache + rss
> +  */
> + MEMCONT_STAT_PAGECACHE,
> + MEMCONT_STAT_RSS,
> +
> + /*
> +  * redundant; usage == charge - uncharge
> +  */
> + MEMCONT_STAT_CHARGE,
> + MEMCONT_STAT_UNCHARGE,
> +
> + /*
> +  * mostly for debug
> +  */
> + MEMCONT_STAT_ISOLATE,
> + MEMCONT_STAT_ISOLATE_FAIL,
```

> + MEMCONT_STAT_NSTATS,
> +};
> +
please add comments on each statistics name. It's uneasy to catch the meaning of
ISOLATE/ISOLATE_FAIL without comments.

> +static const char * const mem_container_stat_desc[] = {
> + [MEMCONT_STAT_PAGECACHE] = "page_cache",
> + [MEMCONT_STAT_RSS] = "rss",
> + [MEMCONT_STAT_CHARGE] = "charge",
> + [MEMCONT_STAT_UNCHARGE] = "uncharge",
> + [MEMCONT_STAT_ISOLATE] = "isolate",
> + [MEMCONT_STAT_ISOLATE_FAIL] = "isolate_fail",
> +};
> +
> +struct mem_container_stat {
> + atomic_t count[MEMCONT_STAT_NSTATS];
> +};
> +
> +static void mem_container_stat_inc(struct mem_container_stat * stat,
> + enum mem_container_stat_index idx)
> +{
> +
> + atomic_inc(&stat->count[idx]);
> +}
> +
> +static void mem_container_stat_dec(struct mem_container_stat * stat,
> + enum mem_container_stat_index idx)
> +{
> +
> + atomic_dec(&stat->count[idx]);
> +}
> +

Can we do this accounting as mod_zone_page_state()(in mm/vmstat.c) ?
(use per-cpu data for accounting.)

> +/* XXX hack; shouldn't be here. it really belongs to struct page_container. */
> +#define PAGE_CONTAINER_CACHE_BIT 0x1
> +#define PAGE_CONTAINER_CACHE (1 << PAGE_CONTAINER_CACHE_BIT)
> +

Is this used for remebering whether a page is charged as page-cache or not ?

> + page_assign_page_container_flags(page,

> +   is_cache ? PAGE_CONTAINER_CACHE : 0, pc);
> +
> + stat = &mem->stat;
> + if (is_cache) {
> + mem_container_stat_inc(stat, MEMCONT_STAT_PAGECACHE);
> + } else {
> + mem_container_stat_inc(stat, MEMCONT_STAT_RSS);
> + }

nitpick,in linux style, one-sentence block shouldn't have braces {}.

==
if (is_cache)
 mem_cont...
else
 mem_cont...
==

> + mem_container_stat_inc(stat, MEMCONT_STAT_CHARGE);
>
>   spin_lock_irqsave(&mem->lru_lock, flags);
>   list_add(&pc->lru, &mem->active_list);
> @@ -377,6 +454,12 @@ err:
>   return -ENOMEM;
> }
>
> +int mem_container_charge(struct page *page, struct mm_struct *mm)
> +{
> +
> + return mem_container_charge_common(page, mm, 0);
> +}
> +
> /*
>   * See if the cached pages should be charged at all?
>   */
> @@ -388,7 +471,7 @@ int mem_container_cache_charge(struct pa
>
>   mem = rcu_dereference(mm->mem_container);
>   if (mem->control_type == MEM_CONTAINER_TYPE_ALL)
> - return mem_container_charge(page, mm);
> + return mem_container_charge_common(page, mm, 1);
>   else
>   return 0;
> }
> @@ -411,15 +494,29 @@ void mem_container_uncharge(struct page_
>   return;
>

>   if (atomic_dec_and_test(&pc->ref_cnt)) {
> + struct mem_container_stat *stat;
> + int is_cache;
> +
>   page = pc->page;
>   lock_page_container(page);
>   mem = pc->mem_container;
>   css_put(&mem->css);
> + /* XXX */
This kind of comment is bad.

> + is_cache = (page->page_container & PAGE_CONTAINER_CACHE) != 0;
>   page_assign_page_container(page, NULL);
>   unlock_page_container(page);
>   res_counter_uncharge(&mem->res, 1);
>
> + stat = &mem->stat;
> + if (is_cache) {
> +   mem_container_stat_dec(stat, MEMCONT_STAT_PAGECACHE);
> + } else {
> +   mem_container_stat_dec(stat, MEMCONT_STAT_RSS);
> + }
> + mem_container_stat_inc(stat, MEMCONT_STAT_UNCHARGE);
> +
>   spin_lock_irqsave(&mem->lru_lock, flags);
> + BUG_ON(list_empty(&pc->lru));

Why this BUG_ON() is added ?

Thanks
-Kame

Subject: Re: [RFC] [PATCH] memory controller statistics
Posted by yamamoto on Fri, 07 Sep 2007 05:38:29 GMT
View Forum Message <> Reply to Message

hi,

thanks for comments.

> Hi,
>

> On Fri, 7 Sep 2007 12:39:42 +0900 (JST)
> yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
>
> > +enum mem_container_stat_index {
> > + /*
> > + * for MEM_CONTAINER_TYPE_ALL, usage == pagecache + rss
> > + */
> > + MEMCONT_STAT_PAGECACHE,
> > + MEMCONT_STAT_RSS,
> > +
> > + /*
> > + * redundant; usage == charge - uncharge
> > + */
> > + MEMCONT_STAT_CHARGE,
> > + MEMCONT_STAT_UNCHARGE,
> > +
> > + /*
> > + * mostly for debug
> > + */
> > + MEMCONT_STAT_ISOLATE,
> > + MEMCONT_STAT_ISOLATE_FAIL,
> > + MEMCONT_STAT_NSTATS,
> > +};
> > +
> please add comments on each statistics name.

sure.

> It's uneasy to catch the meaning of
> ISOLATE/ISOLATE_FAIL without comments.

they aren't useful for users who don't read the relevant code.
probably they should be just removed.

> > +static const char * const mem_container_stat_desc[] = {
> > + [MEMCONT_STAT_PAGECACHE] = "page_cache",
> > + [MEMCONT_STAT_RSS] = "rss",
> > + [MEMCONT_STAT_CHARGE] = "charge",
> > + [MEMCONT_STAT_UNCHARGE] = "uncharge",
> > + [MEMCONT_STAT_ISOLATE] = "isolate",
> > + [MEMCONT_STAT_ISOLATE_FAIL] = "isolate_fail",
> > +};
> > +
> > +struct mem_container_stat {
> > + atomic_t count[MEMCONT_STAT_NSTATS];
> > +};
> > +
> > +static void mem_container_stat_inc(struct mem_container_stat * stat,

> > +   enum mem_container_stat_index idx)
> > +{
> > +
> > + atomic_inc(&stat->count[idx]);
> > +}
> > +
> > +static void mem_container_stat_dec(struct mem_container_stat * stat,
> > +   enum mem_container_stat_index idx)
> > +{
> > +
> > + atomic_dec(&stat->count[idx]);
> > +}
> > +
>
> Can we do this accounting as mod_zone_page_state()(in mm/vmstat.c) ?
> (use per-cpu data for accounting.)

we can do so later.

> > +/* XXX hack; shouldn't be here. it really belongs to struct page_container. */
> > +#define PAGE_CONTAINER_CACHE_BIT 0x1
> > +#define PAGE_CONTAINER_CACHE   (1 << PAGE_CONTAINER_CACHE_BIT)
> > +
>
> Is this used for remebering whether a page is charged as page-cache or not ?

yes.

> > + page_assign_page_container_flags(page,
> > +   is_cache ? PAGE_CONTAINER_CACHE : 0, pc);
> > +
> > + stat = &mem->stat;
> > + if (is_cache) {
> > + mem_container_stat_inc(stat, MEMCONT_STAT_PAGECACHE);
> > + } else {
> > + mem_container_stat_inc(stat, MEMCONT_STAT_RSS);
> > + }
>
> nitpick,in linux style, one-sentence block shouldn't have braces {}.
>
> ==
> if (is_cache)
> mem_cont...
> else
> mem_cont...
> ==

sure.

> > + mem_container_stat_inc(stat, MEMCONT_STAT_CHARGE);
> >
> > spin_lock_irqsave(&mem->lru_lock, flags);
> > list_add(&pc->lru, &mem->active_list);
> > @@ -377,6 +454,12 @@ err:
> > return -ENOMEM;
> > }
> >
> > +int mem_container_charge(struct page *page, struct mm_struct *mm)
> > +{
> > +
> > + return mem_container_charge_common(page, mm, 0);
> > +}
> > +
> > /*
> > * See if the cached pages should be charged at all?
> > */
> > @@ -388,7 +471,7 @@ int mem_container_cache_charge(struct pa
> >
> > mem = rcu_dereference(mm->mem_container);
> > if (mem->control_type == MEM_CONTAINER_TYPE_ALL)
> > - return mem_container_charge(page, mm);
> > + return mem_container_charge_common(page, mm, 1);
> > else
> > return 0;
> > }
> > @@ -411,15 +494,29 @@ void mem_container_uncharge(struct page_
> > return;
> >
> > if (atomic_dec_and_test(&pc->ref_cnt)) {
> > + struct mem_container_stat *stat;
> > + int is_cache;
> > +
> > page = pc->page;
> > lock_page_container(page);
> > mem = pc->mem_container;
> > css_put(&mem->css);
> > + /* XXX */
> This kind of comment is bad.

sure.

> > + is_cache = (page->page_container & PAGE_CONTAINER_CACHE) != 0;
> > page_assign_page_container(page, NULL);
> > unlock_page_container(page);
> > res_counter_uncharge(&mem->res, 1);
> >

> > + stat = &mem->stat;
> > + if (is_cache) {
> > +   mem_container_stat_dec(stat, MEMCONT_STAT_PAGECACHE);
> > + } else {
> > +   mem_container_stat_dec(stat, MEMCONT_STAT_RSS);
> > + }
> > + mem_container_stat_inc(stat, MEMCONT_STAT_UNCHARGE);
> > +
> >   spin_lock_irqsave(&mem->lru_lock, flags);
> > + BUG_ON(list_empty(&pc->lru));
>
> Why this BUG_ON() is added ?
>
> Thanks
> -Kame

to ensure that my understanding is correct.

YAMAMOTO Takashi

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

---

Subject: Re: [RFC] [PATCH] memory controller statistics
Posted by yamamoto on Thu, 04 Oct 2007 05:13:32 GMT
View Forum Message <> Reply to Message

> hi,
>
> i implemented some statistics for your memory controller.
>
> it's tested with 2.6.23-rc2-mm2 + memory controller v7.
> i think it can be applied to 2.6.23-rc4-mm1 as well.
>
> YAMOMOTO Takshi
>
> todo: something like nr_active/inactive in /proc/vmstat.

here's the version i'm working on currently. any comments?

changes from the previous version:

- adapt to 2.6.23-rc8-mm2, container -> cgroup rename.
- reflect some of comments on this list.
 - rename some macros as suggested by balbir
 - sprinkle some inlines as suggested by balbir.

 - remove "isolate" statistics
 - remove PAGE_CONTAINER_CACHE hack and
   add "flags" member to page_container instead.
 - make counters atomic_long_t.
 - add some comments.
 - coding style.
- implement nr_active/nr_inactive. they show numbers of pages on
  per-cgroup lru lists.

todo:
- consider to make counters per-cpu.
- more statistics.

YAMAMOTO Takashi

--- linux-2.6.23-rc8-mm2-stat/mm/memcontrol.c.BACKUP 2007-10-01 17:19:57.000000000 +0900
+++ linux-2.6.23-rc8-mm2-stat/mm/memcontrol.c 2007-10-04 12:42:05.000000000 +0900
@@ -25,6 +25,7 @@
 #include <linux/backing-dev.h>
 #include <linux/bit_spinlock.h>
 #include <linux/rcupdate.h>
+#include <linux/seq_file.h>
 #include <linux/swap.h>
 #include <linux/spinlock.h>
 #include <linux/fs.h>
@@ -34,6 +35,52 @@
 struct cgroup_subsys mem_cgroup_subsys;
 static const int MEM_CGROUP_RECLAIM_RETRIES = 5;

+enum mem_cgroup_stat_index {
+ /*
+ * for MEM_CONTAINER_TYPE_ALL, usage == pagecache + rss
+ */
+ MEM_CGROUP_STAT_PAGECACHE, /* # of pages charged as cache */
+ MEM_CGROUP_STAT_RSS, /* # of pages charged as rss */
+
+ /*
+ * redundant; usage == charge - uncharge
+ */
+ MEM_CGROUP_STAT_CHARGE, /* # of pages charged */
+ MEM_CGROUP_STAT_UNCHARGE, /* # of pages uncharged */
+
+ MEM_CGROUP_STAT_ACTIVE, /* # of pages on active_list */
+ MEM_CGROUP_STAT_INACTIVE, /* # of pages on inactive_list */
+
+ MEM_CGROUP_STAT_NSTATS,
+};

```
+
+static const char * const mem_cgroup_stat_desc[] = {
+ [MEM_CGROUP_STAT_PAGECACHE] = "page_cache",
+ [MEM_CGROUP_STAT_RSS] = "rss",
+ [MEM_CGROUP_STAT_CHARGE] = "charge",
+ [MEM_CGROUP_STAT_UNCHARGE] = "uncharge",
+ [MEM_CGROUP_STAT_ACTIVE] = "nr_active",
+ [MEM_CGROUP_STAT_INACTIVE] = "nr_inactive",
+};
+
+struct mem_cgroup_stat {
+ atomic_long_t count[MEM_CGROUP_STAT_NSTATS];
+};
+
+static inline void mem_cgroup_stat_inc(struct mem_cgroup_stat * stat,
+   enum mem_cgroup_stat_index idx)
+{
+
+ atomic_long_inc(&stat->count[idx]);
+}
+
+static inline void mem_cgroup_stat_dec(struct mem_cgroup_stat * stat,
+   enum mem_cgroup_stat_index idx)
+{
+
+ atomic_long_dec(&stat->count[idx]);
+}
+
 /*
  * The memory controller data structure. The memory controller controls both
  * page cache and RSS per cgroup. We would eventually like to provide
@@ -63,6 +110,11 @@ struct mem_cgroup {
   */
  spinlock_t lru_lock;
  unsigned long control_type; /* control RSS or RSS+Pagecache */
+
+ /*
+  * statistics
+  */
+ struct mem_cgroup_stat stat;
 };

 /*
@@ -83,6 +135,9 @@ struct page_cgroup {
  struct mem_cgroup *mem_cgroup;
  atomic_t ref_cnt;  /* Helpful when pages move b/w  */
     /* mapped and cached states     */
+ int flags;
```

```
+#define PCGF_PAGECACHE 1 /* charged as page cache */
+#define PCGF_ACTIVE 2 /* on active_list */
 };

 enum {
@@ -164,10 +219,24 @@ static void __always_inline unlock_page_

 static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
- if (active)
-  list_move(&pc->lru, &pc->mem_cgroup->active_list);
- else
-  list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
+ struct mem_cgroup *mem = pc->mem_cgroup;
+ struct mem_cgroup_stat *stat = &mem->stat;
+
+ if (active) {
+  list_move(&pc->lru, &mem->active_list);
+  if ((pc->flags & PCGF_ACTIVE) == 0) {
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_INACTIVE);
+   mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_ACTIVE);
+   pc->flags |= PCGF_ACTIVE;
+  }
+ } else {
+  list_move(&pc->lru, &mem->inactive_list);
+  if ((pc->flags & PCGF_ACTIVE) != 0) {
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_ACTIVE);
+   mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_INACTIVE);
+   pc->flags &= ~PCGF_ACTIVE;
+  }
+ }
 }

 /*
@@ -256,10 +325,11 @@ unsigned long mem_cgroup_isolate_pages(u
  * 0 if the charge was successful
  * < 0 if the cgroup is over its limit
  */
-int mem_cgroup_charge(struct page *page, struct mm_struct *mm,
-    gfp_t gfp_mask)
+int mem_cgroup_charge_common(struct page *page, struct mm_struct *mm,
+    gfp_t gfp_mask, int is_cache)
 {
  struct mem_cgroup *mem;
+ struct mem_cgroup_stat *stat;
  struct page_cgroup *pc, *race_pc;
  unsigned long flags;
  unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
```

```
@@ -365,8 +435,17 @@ noreclaim:
  atomic_set(&pc->ref_cnt, 1);
  pc->mem_cgroup = mem;
  pc->page = page;
+ pc->flags = (is_cache ? PCGF_PAGECACHE : 0) | PCGF_ACTIVE;
  page_assign_page_cgroup(page, pc);

+ stat = &mem->stat;
+ if (is_cache)
+  mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_PAGECACHE);
+ else
+  mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_RSS);
+ mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_CHARGE);
+ mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_ACTIVE);
+
  spin_lock_irqsave(&mem->lru_lock, flags);
  list_add(&pc->lru, &mem->active_list);
  spin_unlock_irqrestore(&mem->lru_lock, flags);
@@ -382,6 +461,14 @@ err:
  return -ENOMEM;
 }

+int mem_cgroup_charge(struct page *page, struct mm_struct *mm,
+    gfp_t gfp_mask)
+{
+
+ return mem_cgroup_charge_common(page, mm, gfp_mask, 0);
+}
+
+
 /*
  * See if the cached pages should be charged at all?
  */
@@ -394,7 +481,7 @@ int mem_cgroup_cache_charge(struct page

  mem = rcu_dereference(mm->mem_cgroup);
  if (mem->control_type == MEM_CGROUP_TYPE_ALL)
-  return mem_cgroup_charge(page, mm, gfp_mask);
+  return mem_cgroup_charge_common(page, mm, gfp_mask, 1);
  else
   return 0;
 }
@@ -417,7 +504,11 @@ void mem_cgroup_uncharge(struct page_cgr
   return;

  if (atomic_dec_and_test(&pc->ref_cnt)) {
+  struct mem_cgroup_stat *stat;
+  int is_cache;
```

```
+
   page = pc->page;
+  is_cache = (pc->flags & PCGF_PAGECACHE) != 0;
   lock_page_cgroup(page);
   mem = pc->mem_cgroup;
   css_put(&mem->css);
@@ -425,7 +516,19 @@ void mem_cgroup_uncharge(struct page_cgr
   unlock_page_cgroup(page);
   res_counter_uncharge(&mem->res, PAGE_SIZE);

+  stat = &mem->stat;
+  if (is_cache)
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_PAGECACHE);
+  else
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_RSS);
+  mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_UNCHARGE);
+  if ((pc->flags & PCGF_ACTIVE) != 0)
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_ACTIVE);
+  else
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_INACTIVE);
+
    spin_lock_irqsave(&mem->lru_lock, flags);
+  BUG_ON(list_empty(&pc->lru));
    list_del_init(&pc->lru);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
   kfree(pc);
@@ -517,6 +620,43 @@ static ssize_t mem_control_type_read(str
    ppos, buf, s - buf);
 }

+static void mem_cgroup_stat_init(struct mem_cgroup_stat *stat)
+{
+ int i;
+
+ for (i = 0; i < ARRAY_SIZE(stat->count); i++)
+  atomic_long_set(&stat->count[i], 0);
+}
+
+static int mem_control_stat_show(struct seq_file *m, void *arg)
+{
+ struct cgroup *cont = m->private;
+ struct mem_cgroup *mem_cont = mem_cgroup_from_cont(cont);
+ struct mem_cgroup_stat *stat = &mem_cont->stat;
+ int i;
+
+ for (i = 0; i < ARRAY_SIZE(stat->count); i++)
+  seq_printf(m, "%s %u\n", mem_cgroup_stat_desc[i],
+      (unsigned int)atomic_long_read(&stat->count[i]));
```

```
+
+ return 0;
+}
+
+static const struct file_operations mem_control_stat_file_operations = {
+ .read = seq_read,
+ .llseek = seq_lseek,
+ .release = single_release,
+};
+
+static int mem_control_stat_open(struct inode *unused, struct file *file)
+{
+ /* XXX __d_cont */
+ struct cgroup *cont = file->f_dentry->d_parent->d_fsdata;
+
+ file->f_op = &mem_control_stat_file_operations;
+ return single_open(file, mem_control_stat_show, cont);
+}
+
 static struct cftype mem_cgroup_files[] = {
  {
   .name = "usage_in_bytes",
@@ -539,6 +679,10 @@ static struct cftype mem_cgroup_files[]
   .write = mem_control_type_write,
   .read = mem_control_type_read,
  },
+ {
+  .name = "stat",
+  .open = mem_control_stat_open,
+ },
 };

 static struct mem_cgroup init_mem_cgroup;
@@ -562,6 +706,7 @@ mem_cgroup_create(struct cgroup_subsys *
  INIT_LIST_HEAD(&mem->inactive_list);
  spin_lock_init(&mem->lru_lock);
  mem->control_type = MEM_CGROUP_TYPE_ALL;
+ mem_cgroup_stat_init(&mem->stat);
  return &mem->css;
 }
```

_____

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

Subject: Re: [RFC] [PATCH] memory controller statistics
Posted by Balbir Singh on Fri, 05 Oct 2007 03:51:06 GMT
View Forum Message <> Reply to Message

YAMAMOTO Takashi wrote:
>> hi,
>>
>> i implemented some statistics for your memory controller.
>>
>> it's tested with 2.6.23-rc2-mm2 + memory controller v7.
>> i think it can be applied to 2.6.23-rc4-mm1 as well.
>>
>> YAMOMOTO Takshi
>>
>> todo: something like nr_active/inactive in /proc/vmstat.
>
> here's the version i'm working on currently. any comments?
>
> changes from the previous version:
>
> - adapt to 2.6.23-rc8-mm2, container -> cgroup rename.
> - reflect some of comments on this list.
> - rename some macros as suggested by balbir
> - sprinkle some inlines as suggested by balbir.
> - remove "isolate" statistics
> - remove PAGE_CONTAINER_CACHE hack and
>   add "flags" member to page_container instead.
> - make counters atomic_long_t.
> - add some comments.
> - coding style.
> - implement nr_active/nr_inactive. they show numbers of pages on
>   per-cgroup lru lists.
>
> todo:
> - consider to make counters per-cpu.
> - more statistics.
>
> YAMAMOTO Takashi

Hi, YAMAMOTO-San

Looks much better, I did a quick review, it looks nice to me.
I'll test the patches and get back.

Thank you for working on this,

--
Warm Regards,
Balbir Singh

Linux Technology Center
IBM, ISTL

_____

Subject: Re: [RFC] [PATCH] memory controller statistics
Posted by Balbir Singh on Sun, 07 Oct 2007 05:56:59 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> YAMAMOTO Takashi wrote:
>>> hi,
>>>
>>> i implemented some statistics for your memory controller.
>>>
>>> it's tested with 2.6.23-rc2-mm2 + memory controller v7.
>>> i think it can be applied to 2.6.23-rc4-mm1 as well.
>>>
>>> YAMOMOTO Takshi
>>>
>>> todo: something like nr_active/inactive in /proc/vmstat.
>> here's the version i'm working on currently. any comments?
>>
>> changes from the previous version:
>>
>> - adapt to 2.6.23-rc8-mm2, container -> cgroup rename.
>> - reflect some of comments on this list.
>> - rename some macros as suggested by balbir
>> - sprinkle some inlines as suggested by balbir.
>> - remove "isolate" statistics
>> - remove PAGE_CONTAINER_CACHE hack and
>>   add "flags" member to page_container instead.
>> - make counters atomic_long_t.
>> - add some comments.
>> - coding style.
>> - implement nr_active/nr_inactive. they show numbers of pages on
>>   per-cgroup lru lists.
>>
>> todo:
>> - consider to make counters per-cpu.
>> - more statistics.
>>
>> YAMAMOTO Takashi
>
> Hi, YAMAMOTO-San

>
> Looks much better, I did a quick review, it looks nice to me.
> I'll test the patches and get back.
>
> Thank you for working on this,
>

Forgot to mention one other detail, could we track the statistics
in bytes? We track usage and limit in bytes currently. I am open
to opinions on this, but the last time we discussed numbers,
tracking bytes was the preferred approach.

--
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: [RFC] [PATCH] memory controller statistics
Posted by yamamoto on Mon, 08 Oct 2007 22:55:28 GMT
View Forum Message <> Reply to Message

> Forgot to mention one other detail, could we track the statistics
> in bytes? We track usage and limit in bytes currently. I am open
> to opinions on this, but the last time we discussed numbers,
> tracking bytes was the preferred approach.

while /proc/vmstat and others uses number of pages?
IMO, numbers of pages make more sense for this kind of statistics.

however, if there's a consensus, i can follow. after all,
it's merely a matter of taste.

YAMAMOTO Takashi

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

---

Subject: Re: [RFC] [PATCH] memory controller statistics
Posted by Balbir Singh on Tue, 09 Oct 2007 03:11:39 GMT

YAMAMOTO Takashi wrote:
>> Forgot to mention one other detail, could we track the statistics
>> in bytes? We track usage and limit in bytes currently. I am open
>> to opinions on this, but the last time we discussed numbers,
>> tracking bytes was the preferred approach.
>
> while /proc/vmstat and others uses number of pages?
> IMO, numbers of pages make more sense for this kind of statistics.
>

Yes, I thought so too, but consensus was built around using
bytes for limit and usage for the memory controller. Keeping
all stats in the same units would make it less confusing for users.

> however, if there's a consensus, i can follow. after all,
> it's merely a matter of taste.
>

I agree

> YAMAMOTO Takashi

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

Subject: Re: [RFC] [PATCH] memory controller statistics
Posted by yamamoto on Wed, 10 Oct 2007 01:01:17 GMT

hi,

> > i implemented some statistics for your memory controller.

here's a new version.

changes from the previous:
- make counters per-cpu.

- value *= PAGE_SIZE

YAMAMOTO Takashi

```
--- linux-2.6.23-rc8-mm2-stat/mm/memcontrol.c.BACKUP 2007-10-01 17:19:57.000000000 +0900
+++ linux-2.6.23-rc8-mm2-stat/mm/memcontrol.c 2007-10-09 13:41:43.000000000 +0900
@@ -25,6 +25,7 @@
 #include <linux/backing-dev.h>
 #include <linux/bit_spinlock.h>
 #include <linux/rcupdate.h>
+#include <linux/seq_file.h>
 #include <linux/swap.h>
 #include <linux/spinlock.h>
 #include <linux/fs.h>
@@ -34,6 +35,63 @@
 struct cgroup_subsys mem_cgroup_subsys;
 static const int MEM_CGROUP_RECLAIM_RETRIES = 5;

+enum mem_cgroup_stat_index {
+ /*
+  * for MEM_CONTAINER_TYPE_ALL, usage == pagecache + rss
+  */
+ MEM_CGROUP_STAT_PAGECACHE, /* # of pages charged as cache */
+ MEM_CGROUP_STAT_RSS,  /* # of pages charged as rss */
+
+ /*
+  * redundant; usage == charge - uncharge
+  */
+ MEM_CGROUP_STAT_CHARGE,  /* # of pages charged */
+ MEM_CGROUP_STAT_UNCHARGE, /* # of pages uncharged */
+
+ MEM_CGROUP_STAT_ACTIVE,  /* # of pages on active_list */
+ MEM_CGROUP_STAT_INACTIVE, /* # of pages on inactive_list */
+
+ MEM_CGROUP_STAT_NSTATS,
+};
+
+static const struct mem_cgroup_stat_desc {
+ const char *msg;
+ u64 unit;
+} mem_cgroup_stat_desc[] = {
+ [MEM_CGROUP_STAT_PAGECACHE] = { "page_cache", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_RSS] = { "rss", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_CHARGE] = { "charge", PAGE_SIZE },
+ [MEM_CGROUP_STAT_UNCHARGE] = { "uncharge", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_ACTIVE] = { "active", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_INACTIVE] = { "inactive", PAGE_SIZE, },
```

```
+};
+
+struct mem_cgroup_stat_cpu {
+ u64 count[MEM_CGROUP_STAT_NSTATS];
+} ____cacheline_aligned_in_smp;
+
+struct mem_cgroup_stat {
+ struct mem_cgroup_stat_cpu cpustat[NR_CPUS];
+};
+
+static inline void mem_cgroup_stat_inc(struct mem_cgroup_stat * stat,
+   enum mem_cgroup_stat_index idx)
+{
+ unsigned int cpu = get_cpu();
+
+ stat->cpustat[cpu].count[idx]++;
+ put_cpu();
+}
+
+static inline void mem_cgroup_stat_dec(struct mem_cgroup_stat * stat,
+   enum mem_cgroup_stat_index idx)
+{
+ unsigned int cpu = get_cpu();
+
+ stat->cpustat[cpu].count[idx]--;
+ put_cpu();
+}
+
 /*
  * The memory controller data structure. The memory controller controls both
  * page cache and RSS per cgroup. We would eventually like to provide
@@ -63,6 +121,11 @@ struct mem_cgroup {
   */
  spinlock_t lru_lock;
  unsigned long control_type; /* control RSS or RSS+Pagecache */
+
+ /*
+  * statistics
+  */
+ struct mem_cgroup_stat stat;
 };

 /*
@@ -83,6 +146,9 @@ struct page_cgroup {
  struct mem_cgroup *mem_cgroup;
  atomic_t ref_cnt;  /* Helpful when pages move b/w  */
     /* mapped and cached states     */
+ int flags;
```

```
+#define PCGF_PAGECACHE 1 /* charged as page cache */
+#define PCGF_ACTIVE 2 /* on active_list */
 };

 enum {
@@ -164,10 +230,24 @@ static void __always_inline unlock_page_

 static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
- if (active)
-  list_move(&pc->lru, &pc->mem_cgroup->active_list);
- else
-  list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
+ struct mem_cgroup *mem = pc->mem_cgroup;
+ struct mem_cgroup_stat *stat = &mem->stat;
+
+ if (active) {
+  list_move(&pc->lru, &mem->active_list);
+  if ((pc->flags & PCGF_ACTIVE) == 0) {
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_INACTIVE);
+   mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_ACTIVE);
+   pc->flags |= PCGF_ACTIVE;
+  }
+ } else {
+  list_move(&pc->lru, &mem->inactive_list);
+  if ((pc->flags & PCGF_ACTIVE) != 0) {
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_ACTIVE);
+   mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_INACTIVE);
+   pc->flags &= ~PCGF_ACTIVE;
+  }
+ }
 }

 /*
@@ -256,10 +336,11 @@ unsigned long mem_cgroup_isolate_pages(u
  * 0 if the charge was successful
  * < 0 if the cgroup is over its limit
  */
-int mem_cgroup_charge(struct page *page, struct mm_struct *mm,
-    gfp_t gfp_mask)
+int mem_cgroup_charge_common(struct page *page, struct mm_struct *mm,
+    gfp_t gfp_mask, int is_cache)
 {
  struct mem_cgroup *mem;
+ struct mem_cgroup_stat *stat;
  struct page_cgroup *pc, *race_pc;
  unsigned long flags;
  unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
```

```
@@ -365,8 +446,17 @@ noreclaim:
  atomic_set(&pc->ref_cnt, 1);
  pc->mem_cgroup = mem;
  pc->page = page;
+ pc->flags = (is_cache ? PCGF_PAGECACHE : 0) | PCGF_ACTIVE;
  page_assign_page_cgroup(page, pc);

+ stat = &mem->stat;
+ if (is_cache)
+  mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_PAGECACHE);
+ else
+  mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_RSS);
+ mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_CHARGE);
+ mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_ACTIVE);
+
  spin_lock_irqsave(&mem->lru_lock, flags);
  list_add(&pc->lru, &mem->active_list);
  spin_unlock_irqrestore(&mem->lru_lock, flags);
@@ -382,6 +472,14 @@ err:
  return -ENOMEM;
 }

+int mem_cgroup_charge(struct page *page, struct mm_struct *mm,
+    gfp_t gfp_mask)
+{
+
+ return mem_cgroup_charge_common(page, mm, gfp_mask, 0);
+}
+
+
 /*
  * See if the cached pages should be charged at all?
  */
@@ -394,7 +492,7 @@ int mem_cgroup_cache_charge(struct page

  mem = rcu_dereference(mm->mem_cgroup);
  if (mem->control_type == MEM_CGROUP_TYPE_ALL)
-  return mem_cgroup_charge(page, mm, gfp_mask);
+  return mem_cgroup_charge_common(page, mm, gfp_mask, 1);
  else
   return 0;
 }
@@ -417,7 +515,11 @@ void mem_cgroup_uncharge(struct page_cgr
   return;

  if (atomic_dec_and_test(&pc->ref_cnt)) {
+  struct mem_cgroup_stat *stat;
+  int is_cache;
```

```
+
   page = pc->page;
+  is_cache = (pc->flags & PCGF_PAGECACHE) != 0;
   lock_page_cgroup(page);
   mem = pc->mem_cgroup;
   css_put(&mem->css);
@@ -425,7 +527,19 @@ void mem_cgroup_uncharge(struct page_cgr
   unlock_page_cgroup(page);
   res_counter_uncharge(&mem->res, PAGE_SIZE);

+  stat = &mem->stat;
+  if (is_cache)
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_PAGECACHE);
+  else
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_RSS);
+  mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_UNCHARGE);
+  if ((pc->flags & PCGF_ACTIVE) != 0)
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_ACTIVE);
+  else
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_INACTIVE);
+
    spin_lock_irqsave(&mem->lru_lock, flags);
+  BUG_ON(list_empty(&pc->lru));
    list_del_init(&pc->lru);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
   kfree(pc);
@@ -517,6 +631,49 @@ static ssize_t mem_control_type_read(str
    ppos, buf, s - buf);
 }

+static void mem_cgroup_stat_init(struct mem_cgroup_stat *stat)
+{
+
+ memset(stat, 0, sizeof(*stat));
+}
+
+static int mem_control_stat_show(struct seq_file *m, void *arg)
+{
+ struct cgroup *cont = m->private;
+ struct mem_cgroup *mem_cont = mem_cgroup_from_cont(cont);
+ struct mem_cgroup_stat *stat = &mem_cont->stat;
+ int i;
+
+ for (i = 0; i < ARRAY_SIZE(stat->cpustat[0].count); i++) {
+  unsigned int cpu;
+  u64 val;
+
+  val = 0;
```

```
+  for (cpu = 0; cpu < NR_CPUS; cpu++)
+   val += stat->cpustat[cpu].count[i];
+
+  seq_printf(m, "%s %llu\n", mem_cgroup_stat_desc[i].msg,
+      (unsigned long long)(val * mem_cgroup_stat_desc[i].unit));
+ }
+
+ return 0;
+}
+
+static const struct file_operations mem_control_stat_file_operations = {
+ .read = seq_read,
+ .llseek = seq_lseek,
+ .release = single_release,
+};
+
+static int mem_control_stat_open(struct inode *unused, struct file *file)
+{
+ /* XXX __d_cont */
+ struct cgroup *cont = file->f_dentry->d_parent->d_fsdata;
+
+ file->f_op = &mem_control_stat_file_operations;
+ return single_open(file, mem_control_stat_show, cont);
+}
+
 static struct cftype mem_cgroup_files[] = {
  {
   .name = "usage_in_bytes",
@@ -539,6 +696,10 @@ static struct cftype mem_cgroup_files[]
   .write = mem_control_type_write,
   .read = mem_control_type_read,
  },
+ {
+  .name = "stat",
+  .open = mem_control_stat_open,
+ },
 };

 static struct mem_cgroup init_mem_cgroup;
@@ -562,6 +723,7 @@ mem_cgroup_create(struct cgroup_subsys *
  INIT_LIST_HEAD(&mem->inactive_list);
  spin_lock_init(&mem->lru_lock);
  mem->control_type = MEM_CGROUP_TYPE_ALL;
+ mem_cgroup_stat_init(&mem->stat);
  return &mem->css;
 }
```

Subject: Re: [RFC] [PATCH] memory controller statistics
Posted by KAMEZAWA Hiroyuki on Wed, 10 Oct 2007 06:44:15 GMT
View Forum Message <> Reply to Message

On Wed, 10 Oct 2007 10:01:17 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> hi,
>
> > > i implemented some statistics for your memory controller.
>
> here's a new version.
>
> changes from the previous:
> - make counters per-cpu.
> - value *= PAGE_SIZE
>

YAMAMOTO-san, I like this work and the patch seems good.
But will HUNK with my work to some extent ;)

So, I'd like to merge this patch on my patch set (against -mm) if you don't mind.
If it's ok, please give your "Signed-off-by" line. Then, I'll merge this to mine.

Thanks,
-Kame

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

Subject: Re: [RFC] [PATCH] memory controller statistics
Posted by yamamoto on Wed, 10 Oct 2007 08:19:23 GMT
View Forum Message <> Reply to Message

> On Wed, 10 Oct 2007 10:01:17 +0900 (JST)
> yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
>
> > hi,

> >
> > > > i implemented some statistics for your memory controller.
> >
> > here's a new version.
> >
> > changes from the previous:
> > - make counters per-cpu.
> > - value *= PAGE_SIZE
> >
>
> YAMAMOTO-san, I like this work and the patch seems good.
> But will HUNK with my work to some extent ;)
>
> So, I'd like to merge this patch on my patch set (against -mm) if you don't mind.
> If it's ok, please give your "Signed-off-by" line. Then, I'll merge this to mine.
>
> Thanks,
> -Kame

sure. here's the latest version.

changes from the previous:
 - fix a race in uncharge. (check PCGF_ACTIVE with mem->lru_lock held)

YAMAMOTO Takashi

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

--- linux-2.6.23-rc8-mm2-stat/mm/memcontrol.c.BACKUP 2007-10-01 17:19:57.000000000 +0900
+++ linux-2.6.23-rc8-mm2-stat/mm/memcontrol.c 2007-10-10 12:40:48.000000000 +0900
@@ -25,6 +25,7 @@
 #include <linux/backing-dev.h>
 #include <linux/bit_spinlock.h>
 #include <linux/rcupdate.h>
+#include <linux/seq_file.h>
 #include <linux/swap.h>
 #include <linux/spinlock.h>
 #include <linux/fs.h>
@@ -34,6 +35,63 @@
 struct cgroup_subsys mem_cgroup_subsys;
 static const int MEM_CGROUP_RECLAIM_RETRIES = 5;

+enum mem_cgroup_stat_index {
+ /*
+ * for MEM_CONTAINER_TYPE_ALL, usage == pagecache + rss
+ */

```
+ MEM_CGROUP_STAT_PAGECACHE, /* # of pages charged as cache */
+ MEM_CGROUP_STAT_RSS,  /* # of pages charged as rss */
+
+ /*
+  * redundant; usage == charge - uncharge
+  */
+ MEM_CGROUP_STAT_CHARGE,  /* # of pages charged */
+ MEM_CGROUP_STAT_UNCHARGE, /* # of pages uncharged */
+
+ MEM_CGROUP_STAT_ACTIVE,  /* # of pages on active_list */
+ MEM_CGROUP_STAT_INACTIVE, /* # of pages on inactive_list */
+
+ MEM_CGROUP_STAT_NSTATS,
+};
+
+static const struct mem_cgroup_stat_desc {
+ const char *msg;
+ u64 unit;
+} mem_cgroup_stat_desc[] = {
+ [MEM_CGROUP_STAT_PAGECACHE] = { "page_cache", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_RSS] = { "rss", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_CHARGE] = { "charge", PAGE_SIZE },
+ [MEM_CGROUP_STAT_UNCHARGE] = { "uncharge", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_ACTIVE] = { "active", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_INACTIVE] = { "inactive", PAGE_SIZE, },
+};
+
+struct mem_cgroup_stat_cpu {
+ u64 count[MEM_CGROUP_STAT_NSTATS];
+} ____cacheline_aligned_in_smp;
+
+struct mem_cgroup_stat {
+ struct mem_cgroup_stat_cpu cpustat[NR_CPUS];
+};
+
+static inline void mem_cgroup_stat_inc(struct mem_cgroup_stat * stat,
+   enum mem_cgroup_stat_index idx)
+{
+ unsigned int cpu = get_cpu();
+
+ stat->cpustat[cpu].count[idx]++;
+ put_cpu();
+}
+
+static inline void mem_cgroup_stat_dec(struct mem_cgroup_stat * stat,
+   enum mem_cgroup_stat_index idx)
+{
+ unsigned int cpu = get_cpu();
```

```
+
+ stat->cpustat[cpu].count[idx]--;
+ put_cpu();
+}
+
 /*
  * The memory controller data structure. The memory controller controls both
  * page cache and RSS per cgroup. We would eventually like to provide
@@ -63,6 +121,11 @@ struct mem_cgroup {
   */
  spinlock_t lru_lock;
  unsigned long control_type; /* control RSS or RSS+Pagecache */
+
+ /*
+  * statistics
+  */
+ struct mem_cgroup_stat stat;
 };

 /*
@@ -83,6 +146,9 @@ struct page_cgroup {
  struct mem_cgroup *mem_cgroup;
  atomic_t ref_cnt;  /* Helpful when pages move b/w  */
      /* mapped and cached states     */
+ int flags;
+#define PCGF_PAGECACHE 1 /* charged as page cache */
+#define PCGF_ACTIVE 2 /* on active_list */
 };

 enum {
@@ -164,10 +230,24 @@ static void __always_inline unlock_page_

 static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
- if (active)
-  list_move(&pc->lru, &pc->mem_cgroup->active_list);
- else
-  list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
+ struct mem_cgroup *mem = pc->mem_cgroup;
+ struct mem_cgroup_stat *stat = &mem->stat;
+
+ if (active) {
+  list_move(&pc->lru, &mem->active_list);
+  if ((pc->flags & PCGF_ACTIVE) == 0) {
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_INACTIVE);
+   mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_ACTIVE);
+   pc->flags |= PCGF_ACTIVE;
+  }
```

```
+ } else {
+  list_move(&pc->lru, &mem->inactive_list);
+  if ((pc->flags & PCGF_ACTIVE) != 0) {
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_ACTIVE);
+   mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_INACTIVE);
+   pc->flags &= ~PCGF_ACTIVE;
+  }
+ }
 }

 /*
@@ -256,10 +336,11 @@ unsigned long mem_cgroup_isolate_pages(u
  * 0 if the charge was successful
  * < 0 if the cgroup is over its limit
  */
-int mem_cgroup_charge(struct page *page, struct mm_struct *mm,
-    gfp_t gfp_mask)
+int mem_cgroup_charge_common(struct page *page, struct mm_struct *mm,
+    gfp_t gfp_mask, int is_cache)
 {
  struct mem_cgroup *mem;
+ struct mem_cgroup_stat *stat;
  struct page_cgroup *pc, *race_pc;
  unsigned long flags;
  unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
@@ -365,8 +446,17 @@ noreclaim:
  atomic_set(&pc->ref_cnt, 1);
  pc->mem_cgroup = mem;
  pc->page = page;
+ pc->flags = (is_cache ? PCGF_PAGECACHE : 0) | PCGF_ACTIVE;
  page_assign_page_cgroup(page, pc);

+ stat = &mem->stat;
+ if (is_cache)
+  mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_PAGECACHE);
+ else
+  mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_RSS);
+ mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_CHARGE);
+ mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_ACTIVE);
+
  spin_lock_irqsave(&mem->lru_lock, flags);
  list_add(&pc->lru, &mem->active_list);
  spin_unlock_irqrestore(&mem->lru_lock, flags);
@@ -382,6 +472,14 @@ err:
  return -ENOMEM;
 }

+int mem_cgroup_charge(struct page *page, struct mm_struct *mm,
```

```
+    gfp_t gfp_mask)
+{
+
+ return mem_cgroup_charge_common(page, mm, gfp_mask, 0);
+}
+
+
 /*
  * See if the cached pages should be charged at all?
  */
@@ -394,7 +492,7 @@ int mem_cgroup_cache_charge(struct page

  mem = rcu_dereference(mm->mem_cgroup);
  if (mem->control_type == MEM_CGROUP_TYPE_ALL)
-  return mem_cgroup_charge(page, mm, gfp_mask);
+  return mem_cgroup_charge_common(page, mm, gfp_mask, 1);
  else
   return 0;
 }
@@ -417,17 +515,34 @@ void mem_cgroup_uncharge(struct page_cgr
   return;

  if (atomic_dec_and_test(&pc->ref_cnt)) {
+  struct mem_cgroup_stat *stat;
+  int oflags;
+
   page = pc->page;
   lock_page_cgroup(page);
   mem = pc->mem_cgroup;
-  css_put(&mem->css);
   page_assign_page_cgroup(page, NULL);
   unlock_page_cgroup(page);
   res_counter_uncharge(&mem->res, PAGE_SIZE);

    spin_lock_irqsave(&mem->lru_lock, flags);
+  oflags = pc->flags;
+  BUG_ON(list_empty(&pc->lru));
    list_del_init(&pc->lru);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
+
+  stat = &mem->stat;
+  if ((oflags & PCGF_ACTIVE) != 0)
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_ACTIVE);
+  else
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_INACTIVE);
+  if ((oflags & PCGF_PAGECACHE) != 0)
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_PAGECACHE);
+  else
```

```
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_RSS);
+  mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_UNCHARGE);
+
+  css_put(&mem->css);
   kfree(pc);
  }
 }
@@ -517,6 +632,49 @@ static ssize_t mem_control_type_read(str
    ppos, buf, s - buf);
 }

+static void mem_cgroup_stat_init(struct mem_cgroup_stat *stat)
+{
+
+ memset(stat, 0, sizeof(*stat));
+}
+
+static int mem_control_stat_show(struct seq_file *m, void *arg)
+{
+ struct cgroup *cont = m->private;
+ struct mem_cgroup *mem_cont = mem_cgroup_from_cont(cont);
+ struct mem_cgroup_stat *stat = &mem_cont->stat;
+ int i;
+
+ for (i = 0; i < ARRAY_SIZE(stat->cpustat[0].count); i++) {
+  unsigned int cpu;
+  u64 val;
+
+  val = 0;
+  for (cpu = 0; cpu < NR_CPUS; cpu++)
+   val += stat->cpustat[cpu].count[i];
+
+  seq_printf(m, "%s %llu\n", mem_cgroup_stat_desc[i].msg,
+      (unsigned long long)(val * mem_cgroup_stat_desc[i].unit));
+ }
+
+ return 0;
+}
+
+static const struct file_operations mem_control_stat_file_operations = {
+ .read = seq_read,
+ .llseek = seq_lseek,
+ .release = single_release,
+};
+
+static int mem_control_stat_open(struct inode *unused, struct file *file)
+{
+ /* XXX __d_cont */
```

```
+ struct cgroup *cont = file->f_dentry->d_parent->d_fsdata;
+
+ file->f_op = &mem_control_stat_file_operations;
+ return single_open(file, mem_control_stat_show, cont);
+}
+
 static struct cftype mem_cgroup_files[] = {
  {
   .name = "usage_in_bytes",
@@ -539,6 +697,10 @@ static struct cftype mem_cgroup_files[]
   .write = mem_control_type_write,
   .read = mem_control_type_read,
  },
+ {
+  .name = "stat",
+  .open = mem_control_stat_open,
+ },
 };

 static struct mem_cgroup init_mem_cgroup;
@@ -562,6 +724,7 @@ mem_cgroup_create(struct cgroup_subsys *
  INIT_LIST_HEAD(&mem->inactive_list);
  spin_lock_init(&mem->lru_lock);
  mem->control_type = MEM_CGROUP_TYPE_ALL;
+ mem_cgroup_stat_init(&mem->stat);
  return &mem->css;
 }
```

_____

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