Subject: [RFC][only for review] memory controller bacground reclaim [0/5]
Posted by KAMEZAWA Hiroyuki on Wed, 28 Nov 2007 08:49:23 GMT
View Forum Message <> Reply to Message

Hi, this set is for memory controller background reclaim.

Merged YAMAMOTO-san's version onto 2.6.23-rc3-mm1 + my NUMA patch.
And splitted to several sets.

Major changes from his one is
- use kthread instead of work_queue
- adjust high/low watermark when limit changes automatically
  and set default value. (a user can specify his own later.)

This version is just for review. (I'll rebase this to the next -mm.)

Patch description
[1/5] ... just a fix
[2/5] ... add set/get interface to res_counter
[3/5] ... add high/low watermark to res_counter
[4/5] ... add high/low watermark value to memory cgroup
[5/5] ... add background reclaim to memory cgroup

Any comments are welcome.

Thanks,
-Kame

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

Subject: [RFC][only for review] memory controller bacground reclaim [1/5] spinlock fix in res_counter modif
Posted by KAMEZAWA Hiroyuki on Wed, 28 Nov 2007 08:51:35 GMT
View Forum Message <> Reply to Message

spinlock is necessary when someone changes res_counter value.
splited out from YAMAMOTO's background page reclaim for memory cgroup set.

Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
From: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

kernel/res_counter.c | 5 +++--

1 file changed, 3 insertions(+), 2 deletions(-)

Index: linux-2.6.24-rc3-mm1/kernel/res_counter.c
===
--- linux-2.6.24-rc3-mm1.orig/kernel/res_counter.c 2007-11-27 14:07:44.000000000 +0900
+++ linux-2.6.24-rc3-mm1/kernel/res_counter.c 2007-11-27 14:09:40.000000000 +0900
@@ -98,7 +98,7 @@
 {
  int ret;
  char *buf, *end;
- unsigned long long tmp, *val;
+ unsigned long long flags, tmp, *val;

  buf = kmalloc(nbytes + 1, GFP_KERNEL);
  ret = -ENOMEM;
@@ -121,9 +121,10 @@
   if (*end != '\0')
    goto out_free;
  }
-
+ spin_lock_irqsave(&counter->lock, flags);
  val = res_counter_member(counter, member);
  *val = tmp;
+ spin_unlock_irqrestore(&counter->lock, flags);
  ret = nbytes;
 out_free:
  kfree(buf);

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

---

Subject: [RFC][only for review] memory controller bacground reclaim [2/5] set/get ops for res_counter
Posted by KAMEZAWA Hiroyuki on Wed, 28 Nov 2007 08:52:39 GMT
View Forum Message <> Reply to Message

At implmenting high/low watermark in res_counter, it will be better to
adjust high/low value when limit changes. (or don't allow user to specify
high/low value)

This patch adds *internal* interface to modify resource value.
(If there are only limit/usage/failcnt, these routines are not necessary but..)
And will be used later.

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

```
 include/linux/res_counter.h |    7 +++++++
 kernel/res_counter.c        |   19 +++++++++++++++++++
 2 files changed, 26 insertions(+)

Index: linux-2.6.24-rc3-mm1/include/linux/res_counter.h
===================================================================
--- linux-2.6.24-rc3-mm1.orig/include/linux/res_counter.h 2007-11-28 14:18:21.000000000 +0900
+++ linux-2.6.24-rc3-mm1/include/linux/res_counter.h 2007-11-28 14:18:33.000000000 +0900
@@ -59,6 +59,13 @@
   int (*write_strategy)(char *buf, unsigned long long *val));

 /*
+ * A routine for set/get limitation value from kernel internal code.
+ * res->lock should be held before call this.
+ */
+unsigned long long res_counter_get(struct res_counter *counter, int member);
+void res_counter_set(struct res_counter *conter, int member,
+   unsigned long long val);
+/*
  * the field descriptors. one for each member of res_counter
  */

Index: linux-2.6.24-rc3-mm1/kernel/res_counter.c
===================================================================
--- linux-2.6.24-rc3-mm1.orig/kernel/res_counter.c 2007-11-28 14:18:21.000000000 +0900
+++ linux-2.6.24-rc3-mm1/kernel/res_counter.c 2007-11-28 14:18:33.000000000 +0900
@@ -75,6 +75,25 @@
  return NULL;
 }

+unsigned long long res_counter_get(struct res_counter *res, int member)
+{
+ unsigned long long *val;
+
+ val = res_counter_member(res, member);
+
+ return *val;
+}
+
+void res_counter_set(struct res_counter *res, int member,
+   unsigned long long newval)
+{
+ unsigned long long *val;
+
+ val = res_counter_member(res, member);
+ *val = newval;
+ return;
```

```
+}
+
 ssize_t res_counter_read(struct res_counter *counter, int member,
   const char __user *userbuf, size_t nbytes, loff_t *pos,
   int (*read_strategy)(unsigned long long val, char *st_buf))
```

Subject: [RFC][only for review] memory controller bacground reclaim [3/5] high/low watermark support in res
Posted by KAMEZAWA Hiroyuki on Wed, 28 Nov 2007 08:54:08 GMT
View Forum Message <> Reply to Message

This patch adds high/low watermark parameter to res_counter.
splitted out from YAMAMOTO's background page reclaim for memory cgroup set.

Changes:
 * added param watermark_state this allows status check without lock.

Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
From: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

```
 include/linux/res_counter.h |   27 ++++++++++++++++++++++++
 kernel/res_counter.c        |   46 ++++++++++++++++++++++++++++++++++++++++++-
 2 files changed, 72 insertions(+), 1 deletion(-)

Index: linux-2.6.24-rc3-mm1/include/linux/res_counter.h
===================================================================
--- linux-2.6.24-rc3-mm1.orig/include/linux/res_counter.h 2007-11-28 14:33:19.000000000 +0900
+++ linux-2.6.24-rc3-mm1/include/linux/res_counter.h 2007-11-28 16:37:32.000000000 +0900
@@ -19,6 +19,12 @@
  * the helpers described beyond
  */

+enum watermark_state {
+ RES_WATERMARK_BELOW_LOW,
+ RES_WATERMARK_ABOVE_LOW,
+ RES_WATERMARK_ABOVE_HIGH,
+};
+
 struct res_counter {
  /*
   * the current resource consumption level
```

```
@@ -33,10 +39,19 @@
   */
  unsigned long long failcnt;
  /*
+  * Watermarks
+  * Should be changed automatically when the limit is changed and
+  * keep low < high < limit.
+  */
+ unsigned long long high_watermark;
+ unsigned long long low_watermark;
+ /*
   * the lock to protect all of the above.
   * the routines below consider this to be IRQ-safe
   */
  spinlock_t lock;
+ /* can be read without lock */
+ enum watermark_state watermark_state;
 };

 /*
@@ -73,6 +88,8 @@
  RES_USAGE,
  RES_LIMIT,
  RES_FAILCNT,
+ RES_HIGH_WATERMARK,
+ RES_LOW_WATERMARK,
 };

 /*
@@ -131,4 +148,17 @@
  return ret;
 }

+/*
+ * Helper function for implementing high/low watermark to resource controller.
+ */
+static inline bool res_counter_below_low_watermark(struct res_counter *cnt)
+{
+ return (cnt->watermark_state == RES_WATERMARK_BELOW_LOW);
+}
+
+static inline bool res_counter_above_high_watermark(struct res_counter *cnt)
+{
+ return (cnt->watermark_state == RES_WATERMARK_ABOVE_HIGH);
+}
+
 #endif
Index: linux-2.6.24-rc3-mm1/kernel/res_counter.c
```

```
===================================================================
--- linux-2.6.24-rc3-mm1.orig/kernel/res_counter.c 2007-11-28 14:33:19.000000000 +0900
+++ linux-2.6.24-rc3-mm1/kernel/res_counter.c 2007-11-28 16:33:20.000000000 +0900
@@ -17,6 +17,9 @@
 {
  spin_lock_init(&counter->lock);
  counter->limit = (unsigned long long)LLONG_MAX;
+ counter->low_watermark = (unsigned long long)LLONG_MAX;
+ counter->high_watermark = (unsigned long long)LLONG_MAX;
+ counter->watermark_state = RES_WATERMARK_BELOW_LOW;
 }

 int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
@@ -27,6 +30,15 @@
  }

  counter->usage += val;
+
+ if (counter->usage > counter->high_watermark) {
+  counter->watermark_state = RES_WATERMARK_ABOVE_HIGH;
+  return 0;
+ }
+
+ if (counter->usage > counter->low_watermark)
+  counter->watermark_state = RES_WATERMARK_ABOVE_LOW;
+
  return 0;
 }

@@ -47,6 +59,13 @@
   val = counter->usage;

  counter->usage -= val;
+
+ if (counter->usage < counter->low_watermark) {
+  counter->watermark_state = RES_WATERMARK_BELOW_LOW;
+  return;
+ }
+ if (counter->usage < counter->high_watermark)
+  counter->watermark_state = RES_WATERMARK_ABOVE_LOW;
 }

 void res_counter_uncharge(struct res_counter *counter, unsigned long val)
@@ -69,6 +88,10 @@
   return &counter->limit;
  case RES_FAILCNT:
   return &counter->failcnt;
+ case RES_HIGH_WATERMARK:
```

```
+  return &counter->high_watermark;
+ case RES_LOW_WATERMARK:
+  return &counter->low_watermark;
  };

  BUG();
@@ -117,7 +140,7 @@
 {
  int ret;
  char *buf, *end;
- unsigned long long flags, tmp, *val;
+ unsigned long long flags, tmp, *val, limit, low, high;

  buf = kmalloc(nbytes + 1, GFP_KERNEL);
  ret = -ENOMEM;
@@ -141,6 +164,26 @@
    goto out_free;
  }
  spin_lock_irqsave(&counter->lock, flags);
+ /*
+  * High/Low watermark should be changed automatically AMAP.
+  */
+ switch (member) {
+  case RES_HIGH_WATERMARK:
+   limit = res_counter_get(counter, RES_LIMIT);
+   if (tmp > limit)
+    goto out_free;
+   low = res_counter_get(counter, RES_LOW_WATERMARK);
+   if (tmp <= low)
+    goto out_free;
+   break;
+  case RES_LOW_WATERMARK:
+   high= res_counter_get(counter, RES_HIGH_WATERMARK);
+   if (tmp >= high)
+    goto out_free;
+   break;
+  default:
+   break;
+ }
  val = res_counter_member(counter, member);
  *val = tmp;
  spin_unlock_irqrestore(&counter->lock, flags);
@@ -150,3 +193,4 @@
  return ret;
 }
+
```

Subject: [RFC][only for review] memory controller bacground reclaim [4/5] high/low watermark for memory con
Posted by KAMEZAWA Hiroyuki on Wed, 28 Nov 2007 08:56:07 GMT
View Forum Message <> Reply to Message

High Low watermark for page reclaiming in memory cgroup(1)

High-Low value here is implemented for support background reclaim.

- If USAGE is bigger than high watermark, background reclaim starts.
- If USAGE is lower than low watermark, background reclaim stops.

Each value is represented in bytes.
(Not configurable now, but can be easily.)
The routine which reclaim is in the next patch. This patch just adds
high/low watermark value.

Changes from YAMAMOTO-san's version
- I like automatic adjustment of high/low watermark.
  So, I added a code to modify high/low watermark every time the limit
  changes. A user can custorimize it by hand later.
  (I think 93%/97% is not so bad value. need investigation.)

TODO:
- update documentation.

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

mm/memcontrol.c | 63 ++------
1 file changed, 57 insertions(+), 6 deletions(-)

Index: linux-2.6.24-rc3-mm1/mm/memcontrol.c
===
--- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-28 16:30:44.000000000 +0900
+++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-28 16:44:57.000000000 +0900
@@ -108,16 +108,12 @@
 struct mem_cgroup_per_node *nodeinfo[MAX_NUMNODES];
};

+
/*
 * The memory controller data structure. The memory controller controls both

```
  * page cache and RSS per cgroup. We would eventually like to provide
  * statistics based on the statistics developed by Rik Van Riel for clock-pro,
  * to help the administrator determine what knobs to tune.
- *
- * TODO: Add a water mark for the memory controller. Reclaim will begin when
- * we hit the water mark. May be even add a low water mark, such that
- * no reclaim occurs from a cgroup at it's low water mark, this is
- * a feature that will be implemented much later in the future.
  */
 struct mem_cgroup {
  struct cgroup_subsys_state css;
@@ -162,6 +158,15 @@
 #define PAGE_CGROUP_FLAG_CACHE (0x1) /* charged as cache */
 #define PAGE_CGROUP_FLAG_ACTIVE (0x2) /* page is active in this cgroup */

+/*
+ * Default value for high-low watermark for start/stop background reclaim.
+ * represented in percent here. can be customized by user later.
+ * This is applied always limit change.
+ */
+#define DEFAULT_WATERMARK_PERCENT_LOW (93ULL)
+#define DEFAULT_WATERMARK_PERCENT_HIGH (97ULL)
+
+
 static inline int page_cgroup_nid(struct page_cgroup *pc)
 {
  return page_to_nid(pc->page);
@@ -926,6 +931,29 @@
  return 0;
 }

+static void mem_cgroup_init_watermark(struct cgroup *cont)
+{
+ struct mem_cgroup *mem;
+ unsigned long long val, high, low;
+ unsigned long flags;
+
+ mem = mem_cgroup_from_cont(cont);
+ spin_lock_irqsave(&mem->res.lock, flags);
+ val = res_counter_get(&mem->res, RES_LIMIT);
+ if (val == (unsigned long long) LLONG_MAX) {
+  low = (unsigned long long) LLONG_MAX;
+  high = (unsigned long long) LLONG_MAX;
+ } else {
+  low = val * DEFAULT_WATERMARK_PERCENT_LOW / 100ULL;
+  high = val * DEFAULT_WATERMARK_PERCENT_HIGH / 100ULL;
+ }
+ res_counter_set(&mem->res, RES_LOW_WATERMARK, low);
```

```
+ res_counter_set(&mem->res, RES_HIGH_WATERMARK, high);
+ spin_unlock_irqrestore(&mem->res.lock, flags);
+ return;
+}
+
+
 static ssize_t mem_cgroup_read(struct cgroup *cont,
    struct cftype *cft, struct file *file,
    char __user *userbuf, size_t nbytes, loff_t *ppos)
@@ -944,6 +972,17 @@
     mem_cgroup_write_strategy);
 }

+static ssize_t mem_cgroup_write_limit(struct cgroup *cont, struct cftype *cft,
+    struct file *file, const char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+ ssize_t ret;
+ ret = mem_cgroup_write(cont, cft, file, userbuf, nbytes, ppos);
+ if (ret > 0)
+  mem_cgroup_init_watermark(cont);
+ return ret;
+}
+
 static ssize_t mem_control_type_write(struct cgroup *cont,
    struct cftype *cft, struct file *file,
    const char __user *userbuf,
@@ -1088,7 +1127,7 @@
  {
   .name = "limit_in_bytes",
   .private = RES_LIMIT,
-  .write = mem_cgroup_write,
+  .write = mem_cgroup_write_limit,
   .read = mem_cgroup_read,
  },
  {
@@ -1102,6 +1141,18 @@
   .read = mem_control_type_read,
  },
  {
+  .name = "low_watermark_in_bytes",
+  .private = RES_LOW_WATERMARK,
+  .write = mem_cgroup_write,
+  .read = mem_cgroup_read,
+ },
+ {
+  .name = "high_watermark_in_bytes",
+  .private = RES_HIGH_WATERMARK,
```

```
+  .write = mem_cgroup_write,
+  .read = mem_cgroup_read,
+ },
+ {
   .name = "force_empty",
   .write = mem_force_empty_write,
   .read = mem_force_empty_read,
```

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

---

Subject: [RFC][only for review] memory controller bacground reclaim [5/5]
Posted by KAMEZAWA Hiroyuki on Wed, 28 Nov 2007 08:57:13 GMT
View Forum Message <> Reply to Message

Create a daemon which does background page reclaim.

This daemon
  * starts when usage > high_watermark
  * stops when usage < low_watermark.

Because kthread_run() cannot be used when init_mem_cgroup is initialized(Sigh),
thread for init_mem_cgroup is invoked later by initcall.

Changes from YAMAMOTO-san's version
 * use kthread instead of workqueue.

Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
From: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

```
 mm/memcontrol.c |   79
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 79 insertions(+)

Index: linux-2.6.24-rc3-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-28 16:44:57.000000000 +0900
+++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-28 17:21:57.000000000 +0900
@@ -30,6 +30,8 @@
 #include <linux/spinlock.h>
 #include <linux/fs.h>
 #include <linux/seq_file.h>
+#include <linux/kthread.h>
```

```
+#include <linux/freezer.h>

 #include <asm/uaccess.h>

@@ -122,6 +124,13 @@
   */
  struct res_counter res;
  /*
+  * background reclaim
+  */
+ struct {
+  wait_queue_head_t waitq;
+  struct task_struct *thread;
+ } daemon;
+ /*
   * Per cgroup active and inactive list, similar to the
   * per zone LRU lists.
   */
@@ -401,6 +410,17 @@
  }
 }

+static void
+mem_cgroup_schedule_reclaim(struct mem_cgroup *mem)
+{
+ if (!unlikely(mem->daemon.thread))
+  return;
+ if (!waitqueue_active(&mem->daemon.waitq))
+  return;
+ wake_up_interruptible(&mem->daemon.waitq);
+}
+
+
 int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem)
 {
  int ret;
@@ -677,6 +697,8 @@
    mem_cgroup_out_of_memory(mem, GFP_KERNEL);
   goto free_pc;
  }
+ if (res_counter_above_high_watermark(&mem->res))
+  mem_cgroup_schedule_reclaim(mem);

  atomic_set(&pc->ref_cnt, 1);
  pc->mem_cgroup = mem;
@@ -832,6 +854,38 @@
 }
```

```
 /*
+ * Background page reclaim daeom for memory controller.
+ */
+
+static int mem_cgroup_reclaim_daemon(void *data)
+{
+ DEFINE_WAIT(wait);
+ struct mem_cgroup *mem = data;
+
+ css_get(&mem->css);
+ set_freezable();
+
+ while (!kthread_should_stop()) {
+  prepare_to_wait(&mem->daemon.waitq, &wait, TASK_INTERRUPTIBLE);
+
+  if (res_counter_below_low_watermark(&mem->res)) {
+   if (!kthread_should_stop()) {
+    schedule();
+    try_to_freeze();
+   }
+   finish_wait(&mem->daemon.waitq, &wait);
+   continue;
+  }
+  finish_wait(&mem->daemon.waitq, &wait);
+  try_to_free_mem_cgroup_pages(mem, GFP_HIGHUSER_MOVABLE);
+ }
+
+ css_put(&mem->css);
+
+ return 0;
+}
+
+/*
  * This routine traverse page_cgroup in given list and drop them all.
  * This routine ignores page_cgroup->ref_cnt.
  * *And* this routine doesn't reclaim page itself, just removes page_cgroup.
@@ -1045,6 +1100,7 @@
 {
  struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
  int ret;
+
  ret = mem_cgroup_force_empty(mem);
  if (!ret)
   ret = nbytes;
@@ -1188,6 +1244,16 @@

 static struct mem_cgroup init_mem_cgroup;
```

```
+static int __init mem_cgroup_reclaim_init(void)
+{
+ init_mem_cgroup.daemon.thread = kthread_run(mem_cgroup_reclaim_daemon,
+     &init_mem_cgroup, "memcontd");
+ if (IS_ERR(init_mem_cgroup.daemon.thread))
+  BUG();
+ return 0;
+}
+late_initcall(mem_cgroup_reclaim_init);
+
 static struct cgroup_subsys_state *
 mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
 {
@@ -1212,6 +1278,17 @@
   if (alloc_mem_cgroup_per_zone_info(mem, node))
    goto free_out;

+ /* Memory Reclaim Daemon per cgroup */
+ init_waitqueue_head(&mem->daemon.waitq);
+ if (mem != &init_mem_cgroup) {
+  /* Complicated...but we cannot call kthread create here..*/
+  /* init call will later assign kthread */
+  mem->daemon.thread = kthread_run(mem_cgroup_reclaim_daemon,
+      mem, "memcontd");
+  if (IS_ERR(mem->daemon.thread))
+   goto free_out;
+ }
+
  return &mem->css;
 free_out:
  for_each_node_state(node, N_POSSIBLE)
@@ -1226,6 +1303,7 @@
 {
  struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
  mem_cgroup_force_empty(mem);
+ kthread_stop(mem->daemon.thread);
 }

 static void mem_cgroup_destroy(struct cgroup_subsys *ss,
```

Subject: Re: [RFC][only for review] memory controller bacground reclaim [5/5]

KAMEZAWA Hiroyuki wrote:
> Create a daemon which does background page reclaim.
>
> This daemon
> * starts when usage > high_watermark
> * stops when usage < low_watermark.
>
> Because kthread_run() cannot be used when init_mem_cgroup is initialized(Sigh),
> thread for init_mem_cgroup is invoked later by initcall.
>
> Changes from YAMAMOTO-san's version
> * use kthread instead of workqueue.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> From: YAMAMOTO Takashi <yamamoto@valinux.co.jp>
>
>
>
> mm/memcontrol.c | 79
++
> 1 file changed, 79 insertions(+)
>
> Index: linux-2.6.24-rc3-mm1/mm/memcontrol.c
> ===
> --- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-28 16:44:57.000000000 +0900
> +++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-28 17:21:57.000000000 +0900
> @@ -30,6 +30,8 @@
> #include <linux/spinlock.h>
> #include <linux/fs.h>
> #include <linux/seq_file.h>
> +#include <linux/kthread.h>
> +#include <linux/freezer.h>
>
> #include <asm/uaccess.h>
>
> @@ -122,6 +124,13 @@
>   */
> struct res_counter res;
> /*
> + * background reclaim
> + */
> + struct {
> + wait_queue_head_t waitq;
> + struct task_struct *thread;
> + } daemon;

Does this HAS to be a struct?

```
> + /*
>    * Per cgroup active and inactive list, similar to the
>    * per zone LRU lists.
>    */
> @@ -401,6 +410,17 @@
>   }
>  }
>
> +static void
> +mem_cgroup_schedule_reclaim(struct mem_cgroup *mem)
> +{
> + if (!unlikely(mem->daemon.thread))
> +  return;
> + if (!waitqueue_active(&mem->daemon.waitq))
> +  return;
> + wake_up_interruptible(&mem->daemon.waitq);
> +}
> +
> +
>  int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem)
>  {
>   int ret;
> @@ -677,6 +697,8 @@
>     mem_cgroup_out_of_memory(mem, GFP_KERNEL);
>    goto free_pc;
>   }
> + if (res_counter_above_high_watermark(&mem->res))
> +  mem_cgroup_schedule_reclaim(mem);
>
>   atomic_set(&pc->ref_cnt, 1);
>   pc->mem_cgroup = mem;
> @@ -832,6 +854,38 @@
>  }
>
>  /*
> + * Background page reclaim daeom for memory controller.
> + */
> +
> +static int mem_cgroup_reclaim_daemon(void *data)
> +{
> + DEFINE_WAIT(wait);
> + struct mem_cgroup *mem = data;
> +
> + css_get(&mem->css);
```

Won't this prevent the css from being removed?

```
> + set_freezable();
> +
> + while (!kthread_should_stop()) {
> +  prepare_to_wait(&mem->daemon.waitq, &wait, TASK_INTERRUPTIBLE);
> +
> +  if (res_counter_below_low_watermark(&mem->res)) {
> +   if (!kthread_should_stop()) {
> +    schedule();
> +    try_to_freeze();
> +   }
> +   finish_wait(&mem->daemon.waitq, &wait);
> +   continue;
> +  }
> +  finish_wait(&mem->daemon.waitq, &wait);
> +  try_to_free_mem_cgroup_pages(mem, GFP_HIGHUSER_MOVABLE);
> + }
> +
> + css_put(&mem->css);
> +
> + return 0;
> +}
> +
> +/*
>   * This routine traverse page_cgroup in given list and drop them all.
>   * This routine ignores page_cgroup->ref_cnt.
>   * *And* this routine doesn't reclaim page itself, just removes page_cgroup.
> @@ -1045,6 +1100,7 @@
>  {
>   struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
>   int ret;
> +
```

This hunk is not needed :)

```
>   ret = mem_cgroup_force_empty(mem);
>   if (!ret)
>    ret = nbytes;
> @@ -1188,6 +1244,16 @@
>
>  static struct mem_cgroup init_mem_cgroup;
>
> +static int __init mem_cgroup_reclaim_init(void)
> +{
> + init_mem_cgroup.daemon.thread = kthread_run(mem_cgroup_reclaim_daemon,
> +     &init_mem_cgroup, "memcontd");
> + if (IS_ERR(init_mem_cgroup.daemon.thread))
> +  BUG();
```

> + return 0;
> +}
> +late_initcall(mem_cgroup_reclaim_init);
> +
> static struct cgroup_subsys_state *
> mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
> {
> @@ -1212,6 +1278,17 @@
>   if (alloc_mem_cgroup_per_zone_info(mem, node))
>   goto free_out;
>
> + /* Memory Reclaim Daemon per cgroup */
> + init_waitqueue_head(&mem->daemon.waitq);
> + if (mem != &init_mem_cgroup) {
> + /* Complicated...but we cannot call kthread create here..*/
> + /* init call will later assign kthread */
> + mem->daemon.thread = kthread_run(mem_cgroup_reclaim_daemon,
> +   mem, "memcontd");
> + if (IS_ERR(mem->daemon.thread))
> +   goto free_out;

goto free_mem_cgroup_per_zone_info()?

> + }
> +
>   return &mem->css;
> free_out:
>   for_each_node_state(node, N_POSSIBLE)
> @@ -1226,6 +1303,7 @@
> {
>   struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
>   mem_cgroup_force_empty(mem);
> + kthread_stop(mem->daemon.thread);
> }
>
> static void mem_cgroup_destroy(struct cgroup_subsys *ss,
>
>

Subject: Re: [RFC][only for review] memory controller bacground reclaim [1/5] spinlock fix in res_counter m

KAMEZAWA Hiroyuki wrote:
> spinlock is necessary when someone changes res_counter value.
> splited out from YAMAMOTO's background page reclaim for memory cgroup set.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> From: YAMAMOTO Takashi <yamamoto@valinux.co.jp>
>
>
> kernel/res_counter.c | 5 +++--
> 1 file changed, 3 insertions(+), 2 deletions(-)
>
> Index: linux-2.6.24-rc3-mm1/kernel/res_counter.c
> ===
> --- linux-2.6.24-rc3-mm1.orig/kernel/res_counter.c 2007-11-27 14:07:44.000000000 +0900
> +++ linux-2.6.24-rc3-mm1/kernel/res_counter.c 2007-11-27 14:09:40.000000000 +0900
> @@ -98,7 +98,7 @@
> {
>   int ret;
>   char *buf, *end;
> - unsigned long long tmp, *val;
> + unsigned long long flags, tmp, *val;

Flags for irqsave should be unsigned long. No?

>   buf = kmalloc(nbytes + 1, GFP_KERNEL);
>   ret = -ENOMEM;
> @@ -121,9 +121,10 @@
>   if (*end != '\0')
>   goto out_free;
>   }
> -
> + spin_lock_irqsave(&counter->lock, flags);
>   val = res_counter_member(counter, member);
>   *val = tmp;
> + spin_unlock_irqrestore(&counter->lock, flags);
>   ret = nbytes;
> out_free:
>   kfree(buf);
>
>

Subject: Re: [RFC][only for review] memory controller bacground reclaim [2/5] set/get ops for res_counter
Posted by Pavel Emelianov on Wed, 28 Nov 2007 11:09:26 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> At implmenting high/low watermark in res_counter, it will be better to
> adjust high/low value when limit changes. (or don't allow user to specify
> high/low value)
>
> This patch adds *internal* interface to modify resource value.
> (If there are only limit/usage/failcnt, these routines are not necessary but..)
> And will be used later.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
> include/linux/res_counter.h | 7 +++++++
> kernel/res_counter.c | 19 +++++++++++++++++++
> 2 files changed, 26 insertions(+)
>
> Index: linux-2.6.24-rc3-mm1/include/linux/res_counter.h
> ===
> --- linux-2.6.24-rc3-mm1.orig/include/linux/res_counter.h 2007-11-28 14:18:21.000000000 +0900
> +++ linux-2.6.24-rc3-mm1/include/linux/res_counter.h 2007-11-28 14:18:33.000000000 +0900
> @@ -59,6 +59,13 @@
>   int (*write_strategy)(char *buf, unsigned long long *val));
>
> /*
> + * A routine for set/get limitation value from kernel internal code.
> + * res->lock should be held before call this.
> + */
> +unsigned long long res_counter_get(struct res_counter *counter, int member);
> +void res_counter_set(struct res_counter *conter, int member,
> +   unsigned long long val);
> +/*
>   * the field descriptors. one for each member of res_counter
>   */
>
> Index: linux-2.6.24-rc3-mm1/kernel/res_counter.c
> ===
> --- linux-2.6.24-rc3-mm1.orig/kernel/res_counter.c 2007-11-28 14:18:21.000000000 +0900
> +++ linux-2.6.24-rc3-mm1/kernel/res_counter.c 2007-11-28 14:18:33.000000000 +0900
> @@ -75,6 +75,25 @@
> return NULL;
> }
>
> +unsigned long long res_counter_get(struct res_counter *res, int member)
> +{

> + unsigned long long *val;
> +
> + val = res_counter_member(res, member);
> +
> + return *val;
> +}
> +
> +void res_counter_set(struct res_counter *res, int member,
> +   unsigned long long newval)
> +{
> + unsigned long long *val;
> +
> + val = res_counter_member(res, member);
> + *val = newval;

You put locking here in the res_counter_write() (patch #1). Why
is it missed here?

> + return;
> +}
> +
> ssize_t res_counter_read(struct res_counter *counter, int member,
>   const char __user *userbuf, size_t nbytes, loff_t *pos,
>   int (*read_strategy)(unsigned long long val, char *st_buf))
>
>

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

---

Subject: Re: [RFC][only for review] memory controller bacground reclaim [3/5] high/low watermark support in
Posted by Pavel Emelianov on Wed, 28 Nov 2007 11:12:53 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> This patch adds high/low watermark parameter to res_counter.
> splitted out from YAMAMOTO's background page reclaim for memory cgroup set.
>
> Changes:
> * added param watermark_state this allows status check without lock.
>
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> From: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

>
> include/linux/res_counter.h | 27 ++++++++++++++++++++++++
> kernel/res_counter.c | 46 ++-
> 2 files changed, 72 insertions(+), 1 deletion(-)
>
> Index: linux-2.6.24-rc3-mm1/include/linux/res_counter.h
> ===
> --- linux-2.6.24-rc3-mm1.orig/include/linux/res_counter.h 2007-11-28 14:33:19.000000000
+0900
> +++ linux-2.6.24-rc3-mm1/include/linux/res_counter.h 2007-11-28 16:37:32.000000000 +0900
> @@ -19,6 +19,12 @@
>   * the helpers described beyond
>   */
>
> +enum watermark_state {
> + RES_WATERMARK_BELOW_LOW,
> + RES_WATERMARK_ABOVE_LOW,
> + RES_WATERMARK_ABOVE_HIGH,
> +};
> +
> struct res_counter {
>   /*
>   * the current resource consumption level
> @@ -33,10 +39,19 @@
>   */
>   unsigned long long failcnt;
>   /*
> + * Watermarks
> + * Should be changed automatically when the limit is changed and
> + * keep low < high < limit.
> + */
> + unsigned long long high_watermark;
> + unsigned long long low_watermark;
> + /*
>   * the lock to protect all of the above.
>   * the routines below consider this to be IRQ-safe
>   */
>   spinlock_t lock;
> + /* can be read without lock */
> + enum watermark_state watermark_state;
> };
>
> /*
> @@ -73,6 +88,8 @@
>   RES_USAGE,
>   RES_LIMIT,
>   RES_FAILCNT,
> + RES_HIGH_WATERMARK,

> + RES_LOW_WATERMARK,

I'd prefer some shorter names. Like RES_HWMARK and RES_LWMARK.

> };
>
> /*
> @@ -131,4 +148,17 @@
>   return ret;
> }
>
> +/*
> + * Helper function for implementing high/low watermark to resource controller.
> + */
> +static inline bool res_counter_below_low_watermark(struct res_counter *cnt)
> +{
> + return (cnt->watermark_state == RES_WATERMARK_BELOW_LOW);
> +}
> +
> +static inline bool res_counter_above_high_watermark(struct res_counter *cnt)
> +{
> + return (cnt->watermark_state == RES_WATERMARK_ABOVE_HIGH);
> +}
> +
> #endif
> Index: linux-2.6.24-rc3-mm1/kernel/res_counter.c
> ===
> --- linux-2.6.24-rc3-mm1.orig/kernel/res_counter.c 2007-11-28 14:33:19.000000000 +0900
> +++ linux-2.6.24-rc3-mm1/kernel/res_counter.c 2007-11-28 16:33:20.000000000 +0900
> @@ -17,6 +17,9 @@
> {
>   spin_lock_init(&counter->lock);
>   counter->limit = (unsigned long long)LLONG_MAX;
> + counter->low_watermark = (unsigned long long)LLONG_MAX;
> + counter->high_watermark = (unsigned long long)LLONG_MAX;
> + counter->watermark_state = RES_WATERMARK_BELOW_LOW;
> }
>
> int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
> @@ -27,6 +30,15 @@
>   }
>
>   counter->usage += val;
> +
> + if (counter->usage > counter->high_watermark) {
> + counter->watermark_state = RES_WATERMARK_ABOVE_HIGH;
> + return 0;
> + }

"else" would look much better here :)

> + if (counter->usage > counter->low_watermark)
> + counter->watermark_state = RES_WATERMARK_ABOVE_LOW;
> +
>   return 0;
> }
>
> @@ -47,6 +59,13 @@
>   val = counter->usage;
>
>   counter->usage -= val;
> +
> + if (counter->usage < counter->low_watermark) {
> + counter->watermark_state = RES_WATERMARK_BELOW_LOW;
> + return;
> + }

and here

> + if (counter->usage < counter->high_watermark)
> + counter->watermark_state = RES_WATERMARK_ABOVE_LOW;
> }
>
> void res_counter_uncharge(struct res_counter *counter, unsigned long val)
> @@ -69,6 +88,10 @@
>   return &counter->limit;
>   case RES_FAILCNT:
>   return &counter->failcnt;
> + case RES_HIGH_WATERMARK:
> + return &counter->high_watermark;
> + case RES_LOW_WATERMARK:
> + return &counter->low_watermark;
>   };
>
>   BUG();
> @@ -117,7 +140,7 @@
> {
>   int ret;
>   char *buf, *end;
> - unsigned long long flags, tmp, *val;
> + unsigned long long flags, tmp, *val, limit, low, high;
>
>   buf = kmalloc(nbytes + 1, GFP_KERNEL);
>   ret = -ENOMEM;
> @@ -141,6 +164,26 @@
>   goto out_free;

```
>   }
>   spin_lock_irqsave(&counter->lock, flags);
> + /*
> +  * High/Low watermark should be changed automatically AMAP.
> +  */
> + switch (member) {
> +  case RES_HIGH_WATERMARK:
> +   limit = res_counter_get(counter, RES_LIMIT);
> +   if (tmp > limit)
> +    goto out_free;
> +   low = res_counter_get(counter, RES_LOW_WATERMARK);
> +   if (tmp <= low)
> +    goto out_free;
> +   break;
> +  case RES_LOW_WATERMARK:
> +   high= res_counter_get(counter, RES_HIGH_WATERMARK);
> +   if (tmp >= high)
> +    goto out_free;
> +   break;
```

Why there's no checks for limit? Smth like

```
case RES_LIMIT:
 high = res_counter_get(counter, RES_HIGH_WATERMARK);
 if (tmp < high)
  goto out_free;
```

```
> +  default:
> +   break;
> + }
>   val = res_counter_member(counter, member);
>   *val = tmp;
>   spin_unlock_irqrestore(&counter->lock, flags);
> @@ -150,3 +193,4 @@
>  out:
>   return ret;
>  }
> +
>
>
```

Subject: Re: [RFC][only for review] memory controller bacground reclaim [4/5] high/low watermark for memory
Posted by Pavel Emelianov on Wed, 28 Nov 2007 11:20:33 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> High Low watermark for page reclaiming in memory cgroup(1)
>
> High-Low value here is implemented for support background reclaim.
>
> - If USAGE is bigger than high watermark, background reclaim starts.
> - If USAGE is lower than low watermark, background reclaim stops.
>
> Each value is represented in bytes.
> (Not configurable now, but can be easily.)
> The routine which reclaim is in the next patch. This patch just adds
> high/low watermark value.
>
> Changes from YAMAMOTO-san's version
> - I like automatic adjustment of high/low watermark.
> So, I added a code to modify high/low watermark every time the limit
> changes. A user can custorimize it by hand later.
> (I think 93%/97% is not so bad value. need investigation.)
>
> TODO:
> - update documentation.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
> mm/memcontrol.c | 63 ++------
> 1 file changed, 57 insertions(+), 6 deletions(-)
>
> Index: linux-2.6.24-rc3-mm1/mm/memcontrol.c
> ===
> --- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-28 16:30:44.000000000 +0900
> +++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-28 16:44:57.000000000 +0900
> @@ -108,16 +108,12 @@
> struct mem_cgroup_per_node *nodeinfo[MAX_NUMNODES];
> };
>
> +
> /*
> * The memory controller data structure. The memory controller controls both
> * page cache and RSS per cgroup. We would eventually like to provide
> * statistics based on the statistics developed by Rik Van Riel for clock-pro,
> * to help the administrator determine what knobs to tune.
> - *
> - * TODO: Add a water mark for the memory controller. Reclaim will begin when
> - * we hit the water mark. May be even add a low water mark, such that

> - * no reclaim occurs from a cgroup at it's low water mark, this is
> - * a feature that will be implemented much later in the future.
> */
> struct mem_cgroup {
> struct cgroup_subsys_state css;
> @@ -162,6 +158,15 @@
> #define PAGE_CGROUP_FLAG_CACHE (0x1) /* charged as cache */
> #define PAGE_CGROUP_FLAG_ACTIVE (0x2) /* page is active in this cgroup */
>
> +/*
> + * Default value for high-low watermark for start/stop background reclaim.
> + * represented in percent here. can be customized by user later.
> + * This is applied always limit change.
> + */
> +#define DEFAULT_WATERMARK_PERCENT_LOW (93ULL)
> +#define DEFAULT_WATERMARK_PERCENT_HIGH (97ULL)
> +
> +
> static inline int page_cgroup_nid(struct page_cgroup *pc)
> {
> return page_to_nid(pc->page);
> @@ -926,6 +931,29 @@
> return 0;
> }
>
> +static void mem_cgroup_init_watermark(struct cgroup *cont)
> +{
> + struct mem_cgroup *mem;
> + unsigned long long val, high, low;
> + unsigned long flags;
> +
> + mem = mem_cgroup_from_cont(cont);
> + spin_lock_irqsave(&mem->res.lock, flags);
> + val = res_counter_get(&mem->res, RES_LIMIT);
> + if (val == (unsigned long long) LLONG_MAX) {
> + low = (unsigned long long) LLONG_MAX;
> + high = (unsigned long long) LLONG_MAX;
> + } else {
> + low = val * DEFAULT_WATERMARK_PERCENT_LOW / 100ULL;
> + high = val * DEFAULT_WATERMARK_PERCENT_HIGH / 100ULL;
> + }
> + res_counter_set(&mem->res, RES_LOW_WATERMARK, low);
> + res_counter_set(&mem->res, RES_HIGH_WATERMARK, high);
> + spin_unlock_irqrestore(&mem->res.lock, flags);
> + return;
> +}
> +
> +

> static ssize_t mem_cgroup_read(struct cgroup *cont,
>   struct cftype *cft, struct file *file,
>   char __user *userbuf, size_t nbytes, loff_t *ppos)
> @@ -944,6 +972,17 @@
>   mem_cgroup_write_strategy);
> }
>
> +static ssize_t mem_cgroup_write_limit(struct cgroup *cont, struct cftype *cft,
> +   struct file *file, const char __user *userbuf,
> +   size_t nbytes, loff_t *ppos)
> +{
> + ssize_t ret;
> + ret = mem_cgroup_write(cont, cft, file, userbuf, nbytes, ppos);
> + if (ret > 0)
> + mem_cgroup_init_watermark(cont);

No, please, no! I'd be very disappointed if I tune high and low watermarks
carefully and then they are silently re-set after I tune the limit. Better
(see my comment to patch #3) return -EINVAL in case I try to set limit
below hwmark. Please :)

> + return ret;
> +}
> +
> static ssize_t mem_control_type_write(struct cgroup *cont,
>   struct cftype *cft, struct file *file,
>   const char __user *userbuf,
> @@ -1088,7 +1127,7 @@
>   {
>   .name = "limit_in_bytes",
>   .private = RES_LIMIT,
> - .write = mem_cgroup_write,
> + .write = mem_cgroup_write_limit,
>   .read = mem_cgroup_read,
>   },
>   {
> @@ -1102,6 +1141,18 @@
>   .read = mem_control_type_read,
>   },
>   {
> + .name = "low_watermark_in_bytes",
> + .private = RES_LOW_WATERMARK,
> + .write = mem_cgroup_write,
> + .read = mem_cgroup_read,
> + },
> + {
> + .name = "high_watermark_in_bytes",
> + .private = RES_HIGH_WATERMARK,

> + .write = mem_cgroup_write,
> + .read = mem_cgroup_read,
> + },
> + {
>   .name = "force_empty",
>   .write = mem_force_empty_write,
>   .read = mem_force_empty_read,
>
>

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

---

Subject: Re: [RFC][only for review] memory controller bacground reclaim [4/5] high/low watermark for memory
Posted by Pavel Emelianov on Wed, 28 Nov 2007 12:20:42 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> High Low watermark for page reclaiming in memory cgroup(1)
>
> High-Low value here is implemented for support background reclaim.
>
> - If USAGE is bigger than high watermark, background reclaim starts.
> - If USAGE is lower than low watermark, background reclaim stops.
>
> Each value is represented in bytes.
> (Not configurable now, but can be easily.)
> The routine which reclaim is in the next patch. This patch just adds
> high/low watermark value.
>
> Changes from YAMAMOTO-san's version
>   - I like automatic adjustment of high/low watermark.
>   So, I added a code to modify high/low watermark every time the limit
>   changes. A user can custorimize it by hand later.
>   (I think 93%/97% is not so bad value. need investigation.)
>
> TODO:
> - update documentation.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
> mm/memcontrol.c | 63 ++------
> 1 file changed, 57 insertions(+), 6 deletions(-)
>

> Index: linux-2.6.24-rc3-mm1/mm/memcontrol.c
> ===
> --- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-28 16:30:44.000000000 +0900
> +++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-28 16:44:57.000000000 +0900
> @@ -108,16 +108,12 @@
> struct mem_cgroup_per_node *nodeinfo[MAX_NUMNODES];
> };
>
> +
> /*
> * The memory controller data structure. The memory controller controls both
> * page cache and RSS per cgroup. We would eventually like to provide
> * statistics based on the statistics developed by Rik Van Riel for clock-pro,
> * to help the administrator determine what knobs to tune.
> - *
> - * TODO: Add a water mark for the memory controller. Reclaim will begin when
> - * we hit the water mark. May be even add a low water mark, such that
> - * no reclaim occurs from a cgroup at it's low water mark, this is
> - * a feature that will be implemented much later in the future.
> */
> struct mem_cgroup {
> struct cgroup_subsys_state css;
> @@ -162,6 +158,15 @@
> #define PAGE_CGROUP_FLAG_CACHE (0x1) /* charged as cache */
> #define PAGE_CGROUP_FLAG_ACTIVE (0x2) /* page is active in this cgroup */
>
> +/*
> + * Default value for high-low watermark for start/stop background reclaim.
> + * represented in percent here. can be customized by user later.
> + * This is applied always limit change.
> + */
> +#define DEFAULT_WATERMARK_PERCENT_LOW (93ULL)
> +#define DEFAULT_WATERMARK_PERCENT_HIGH (97ULL)
> +
> +
> static inline int page_cgroup_nid(struct page_cgroup *pc)
> {
> return page_to_nid(pc->page);
> @@ -926,6 +931,29 @@
> return 0;
> }
>
> +static void mem_cgroup_init_watermark(struct cgroup *cont)
> +{
> + struct mem_cgroup *mem;
> + unsigned long long val, high, low;
> + unsigned long flags;
> +

> + mem = mem_cgroup_from_cont(cont);
> + spin_lock_irqsave(&mem->res.lock, flags);
> + val = res_counter_get(&mem->res, RES_LIMIT);
> + if (val == (unsigned long long) LLONG_MAX) {
> + low = (unsigned long long) LLONG_MAX;
> + high = (unsigned long long) LLONG_MAX;
> + } else {
> + low = val * DEFAULT_WATERMARK_PERCENT_LOW / 100ULL;
> + high = val * DEFAULT_WATERMARK_PERCENT_HIGH / 100ULL;

BTW, I tried to compile such a code:

unsigned long long x, y;
y = ...;
x = y / 100ULL;

(similar to yours) and that's what I got:

kernel/built-in.o: In function `xxx':
: undefined reference to `__udivdi3'

It looks like i386 doesn't have any support for ULL divisions.
It doesn't have it in CPU, and I thought that it was some-how
emulated, but it is not...

Did I miss something?

> + }
> + res_counter_set(&mem->res, RES_LOW_WATERMARK, low);
> + res_counter_set(&mem->res, RES_HIGH_WATERMARK, high);
> + spin_unlock_irqrestore(&mem->res.lock, flags);
> + return;
> +}
> +
> +
> static ssize_t mem_cgroup_read(struct cgroup *cont,
>   struct cftype *cft, struct file *file,
>   char __user *userbuf, size_t nbytes, loff_t *ppos)
> @@ -944,6 +972,17 @@
>   mem_cgroup_write_strategy);
> }
>
> +static ssize_t mem_cgroup_write_limit(struct cgroup *cont, struct cftype *cft,
> +   struct file *file, const char __user *userbuf,
> +   size_t nbytes, loff_t *ppos)
> +{
> + ssize_t ret;
> + ret = mem_cgroup_write(cont, cft, file, userbuf, nbytes, ppos);

```
> + if (ret > 0)
> +  mem_cgroup_init_watermark(cont);
> + return ret;
> +}
> +
>  static ssize_t mem_control_type_write(struct cgroup *cont,
>      struct cftype *cft, struct file *file,
>      const char __user *userbuf,
> @@ -1088,7 +1127,7 @@
>   {
>    .name = "limit_in_bytes",
>    .private = RES_LIMIT,
> -  .write = mem_cgroup_write,
> +  .write = mem_cgroup_write_limit,
>    .read = mem_cgroup_read,
>   },
>   {
> @@ -1102,6 +1141,18 @@
>    .read = mem_control_type_read,
>   },
>   {
> +  .name = "low_watermark_in_bytes",
> +  .private = RES_LOW_WATERMARK,
> +  .write = mem_cgroup_write,
> +  .read = mem_cgroup_read,
> + },
> + {
> +  .name = "high_watermark_in_bytes",
> +  .private = RES_HIGH_WATERMARK,
> +  .write = mem_cgroup_write,
> +  .read = mem_cgroup_read,
> + },
> + {
>    .name = "force_empty",
>    .write = mem_force_empty_write,
>    .read = mem_force_empty_read,
>
>
```


Subject: Re: [RFC][only for review] memory controller bacground reclaim [1/5] spinlock fix in res_counter m

Posted by KAMEZAWA Hiroyuki on Thu, 29 Nov 2007 01:14:39 GMT
View Forum Message <> Reply to Message

On Wed, 28 Nov 2007 14:08:31 +0300
Pavel Emelyanov <xemul@openvz.org> wrote:

> KAMEZAWA Hiroyuki wrote:
> > spinlock is necessary when someone changes res_counter value.
> > splited out from YAMAMOTO's background page reclaim for memory cgroup set.
> >
> > Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> > From: YAMAMOTO Takashi <yamamoto@valinux.co.jp>
> >
> >
> > kernel/res_counter.c | 5 +++--
> > 1 file changed, 3 insertions(+), 2 deletions(-)
> >
> > Index: linux-2.6.24-rc3-mm1/kernel/res_counter.c
> > ===
> > --- linux-2.6.24-rc3-mm1.orig/kernel/res_counter.c 2007-11-27 14:07:44.000000000 +0900
> > +++ linux-2.6.24-rc3-mm1/kernel/res_counter.c 2007-11-27 14:09:40.000000000 +0900
> > @@ -98,7 +98,7 @@
> > {
> >   int ret;
> >   char *buf, *end;
> > - unsigned long long tmp, *val;
> > + unsigned long long flags, tmp, *val;
>
> Flags for irqsave should be unsigned long. No?
>
yes. It must be.

Thanks, I'll fix.
-Kame

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

Subject: Re: [RFC][only for review] memory controller bacground reclaim [2/5] set/get ops for res_counter
Posted by KAMEZAWA Hiroyuki on Thu, 29 Nov 2007 01:16:07 GMT
View Forum Message <> Reply to Message

On Wed, 28 Nov 2007 14:09:26 +0300
Pavel Emelyanov <xemul@openvz.org> wrote:

> > +void res_counter_set(struct res_counter *res, int member,
> > +   unsigned long long newval)
> > +{
> > + unsigned long long *val;
> > +
> > + val = res_counter_member(res, member);
> > + *val = newval;
>
> You put locking here in the res_counter_write() (patch #1). Why
> is it missed here?
>
Sorry, my mistake.
I'll drop this patch because automatic value adjustment patch will be dropped.

Thanks,
-Kame

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

---

Subject: Re: [RFC][only for review] memory controller bacground reclaim [3/5] high/low watermark support in
Posted by KAMEZAWA Hiroyuki on Thu, 29 Nov 2007 01:18:23 GMT
View Forum Message <> Reply to Message

On Wed, 28 Nov 2007 14:12:53 +0300
Pavel Emelyanov <xemul@openvz.org> wrote:
> > /*
> > @@ -73,6 +88,8 @@
> >   RES_USAGE,
> >   RES_LIMIT,
> >   RES_FAILCNT,
> > + RES_HIGH_WATERMARK,
> > + RES_LOW_WATERMARK,
>
> I'd prefer some shorter names. Like RES_HWMARK and RES_LWMARK.
>
Hmm, ok.

> >   counter->usage += val;
> > +
> > + if (counter->usage > counter->high_watermark) {
> > + counter->watermark_state = RES_WATERMARK_ABOVE_HIGH;

> > + return 0;
> > + }
>
> "else" would look much better here :)
I agree and will fix. thanks.

>
> > + if (counter->usage > counter->low_watermark)
> > + counter->watermark_state = RES_WATERMARK_ABOVE_LOW;
> > +
> >   return 0;
> > }
> >
> > @@ -47,6 +59,13 @@
> >   val = counter->usage;
> >
> >   counter->usage -= val;
> > +
> > + if (counter->usage < counter->low_watermark) {
> > + counter->watermark_state = RES_WATERMARK_BELOW_LOW;
> > + return;
> > + }
>
> and here
>
here, too.

> > + if (counter->usage < counter->high_watermark)
> > + counter->watermark_state = RES_WATERMARK_ABOVE_LOW;
> > }
> >
> > void res_counter_uncharge(struct res_counter *counter, unsigned long val)
> > @@ -69,6 +88,10 @@
> >   return &counter->limit;
> >   case RES_FAILCNT:
> >   return &counter->failcnt;
> > + case RES_HIGH_WATERMARK:
> > + return &counter->high_watermark;
> > + case RES_LOW_WATERMARK:
> > + return &counter->low_watermark;
> >   };
> >
> >   BUG();
> > @@ -117,7 +140,7 @@
> > {
> >   int ret;
> >   char *buf, *end;
> > - unsigned long long flags, tmp, *val;

> > + unsigned long long flags, tmp, *val, limit, low, high;
> >
> > buf = kmalloc(nbytes + 1, GFP_KERNEL);
> > ret = -ENOMEM;
> > @@ -141,6 +164,26 @@
> > goto out_free;
> > }
> > spin_lock_irqsave(&counter->lock, flags);
> > + /*
> > + * High/Low watermark should be changed automatically AMAP.
> > + */
> > + switch (member) {
> > + case RES_HIGH_WATERMARK:
> > + limit = res_counter_get(counter, RES_LIMIT);
> > + if (tmp > limit)
> > + goto out_free;
> > + low = res_counter_get(counter, RES_LOW_WATERMARK);
> > + if (tmp <= low)
> > + goto out_free;
> > + break;
> > + case RES_LOW_WATERMARK:
> > + high= res_counter_get(counter, RES_HIGH_WATERMARK);
> > + if (tmp >= high)
> > + goto out_free;
> > + break;
>
> Why there's no checks for limit? Smth like
>
> case RES_LIMIT:
> high = res_counter_get(counter, RES_HIGH_WATERMARK);
> if (tmp < high)
> goto out_free;
>
Ok, I'll drop automatic adjustment patch and add check for LIMIT.

Thanks,
-Kame

Subject: Re: [RFC][only for review] memory controller bacground reclaim [4/5] high/low watermark for memory
Posted by KAMEZAWA Hiroyuki on Thu, 29 Nov 2007 01:20:44 GMT

On Wed, 28 Nov 2007 15:20:42 +0300
Pavel Emelyanov <xemul@openvz.org> wrote:
> > + mem = mem_cgroup_from_cont(cont);
> > + spin_lock_irqsave(&mem->res.lock, flags);
> > + val = res_counter_get(&mem->res, RES_LIMIT);
> > + if (val == (unsigned long long) LLONG_MAX) {
> > + low = (unsigned long long) LLONG_MAX;
> > + high = (unsigned long long) LLONG_MAX;
> > + } else {
> > + low = val * DEFAULT_WATERMARK_PERCENT_LOW / 100ULL;
> > + high = val * DEFAULT_WATERMARK_PERCENT_HIGH / 100ULL;
>
> BTW, I tried to compile such a code:
>
> unsigned long long x, y;
> y = ...;
> x = y / 100ULL;
>
> (similar to yours) and that's what I got:
>
> kernel/built-in.o: In function `xxx':
> : undefined reference to `__udivdi3'
>
> It looks like i386 doesn't have any support for ULL divisions.
> It doesn't have it in CPU, and I thought that it was some-how
> emulated, but it is not...
>
> Did I miss something?
>
Ah, I didn't try i386...
But I'll drop this automatic watermark adjustment part.

Thanks,
-Kame

Subject: Re: [RFC][only for review] memory controller bacground reclaim [5/5]
Posted by KAMEZAWA Hiroyuki on Thu, 29 Nov 2007 01:26:09 GMT

On Wed, 28 Nov 2007 14:06:22 +0300

Pavel Emelyanov <xemul@openvz.org> wrote:
> > + struct {
> > + wait_queue_head_t waitq;
> > + struct task_struct *thread;
> > + } daemon;
>
> Does this HAS to be a struct?
>
No, but for shorter name of member.
(I'd like to add another waitq for avoiding contention in reclaiming,
 If I can.)

> > + * Background page reclaim daeom for memory controller.
> > + */
> > +
> > +static int mem_cgroup_reclaim_daemon(void *data)
> > +{
> > + DEFINE_WAIT(wait);
> > + struct mem_cgroup *mem = data;
> > +
> > + css_get(&mem->css);
>
> Won't this prevent the css from being removed?
>
This refcnt is dropped when this thread dies.
pre_destroy handler handles this at rmdir().

> >   struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
> >   int ret;
> > +
>
> This hunk is not needed :)
>
yes :)

> >   ret = mem_cgroup_force_empty(mem);
> >   if (!ret)
> >   ret = nbytes;
> > @@ -1188,6 +1244,16 @@
> >
> > static struct mem_cgroup init_mem_cgroup;
> >
> > +static int __init mem_cgroup_reclaim_init(void)
> > +{
> > + init_mem_cgroup.daemon.thread = kthread_run(mem_cgroup_reclaim_daemon,
> > +   &init_mem_cgroup, "memcontd");

> > + if (IS_ERR(init_mem_cgroup.daemon.thread))
> > + BUG();
> > + return 0;
> > +}
> > +late_initcall(mem_cgroup_reclaim_init);
> > +
> > static struct cgroup_subsys_state *
> > mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
> > {
> > @@ -1212,6 +1278,17 @@
> >   if (alloc_mem_cgroup_per_zone_info(mem, node))
> >   goto free_out;
> >
> > + /* Memory Reclaim Daemon per cgroup */
> > + init_waitqueue_head(&mem->daemon.waitq);
> > + if (mem != &init_mem_cgroup) {
> > + /* Complicated...but we cannot call kthread create here..*/
> > + /* init call will later assign kthread */
> > + mem->daemon.thread = kthread_run(mem_cgroup_reclaim_daemon,
> > +   mem, "memcontd");
> > + if (IS_ERR(mem->daemon.thread))
> > +   goto free_out;
>
> goto free_mem_cgroup_per_zone_info()?
>
Ah.. there may be some bugs. will fix soon.

Thank you for all your comments.

Regards,
-Kame

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

---

Subject: Re: [RFC][only for review] memory controller bacground reclaim [4/5] high/low watermark for memory
Posted by KAMEZAWA Hiroyuki on Thu, 29 Nov 2007 01:27:29 GMT
View Forum Message <> Reply to Message

On Wed, 28 Nov 2007 14:20:33 +0300
Pavel Emelyanov <xemul@openvz.org> wrote:

> > +static ssize_t mem_cgroup_write_limit(struct cgroup *cont, struct cftype *cft,
> > +   struct file *file, const char __user *userbuf,

> > +   size_t nbytes, loff_t *ppos)
> > +{
> > + ssize_t ret;
> > + ret = mem_cgroup_write(cont, cft, file, userbuf, nbytes, ppos);
> > + if (ret > 0)
> > + mem_cgroup_init_watermark(cont);
>
> No, please, no! I'd be very disappointed if I tune high and low watermarks
> carefully and then they are silently re-set after I tune the limit. Better
> (see my comment to patch #3) return -EINVAL in case I try to set limit
> below hwmark. Please :)
>
Ok, I'll drop this.

Thanks,
-Kame

Subject: Re: [RFC][only for review] memory controller bacground reclaim [3/5] high/low watermark support in
Posted by yamamoto on Thu, 29 Nov 2007 02:56:08 GMT
View Forum Message <> Reply to Message

> This patch adds high/low watermark parameter to res_counter.
> splitted out from YAMAMOTO's background page reclaim for memory cgroup set.

thanks.

> + * Watermarks
> + * Should be changed automatically when the limit is changed and
> + * keep low < high < limit.
> + */
> + unsigned long long high_watermark;
> + unsigned long long low_watermark;
> + /*

do you have any specific reason not to allow low == high == limit?
if you want to ensure low < high < limit, it's better to make the
default values below meet the condition as well.
to me, it seems weird to prevent users from making these values back to
the default.

YAMAMOTO Takashi

> @@ -17,6 +17,9 @@
> {
> spin_lock_init(&counter->lock);
> counter->limit = (unsigned long long)LLONG_MAX;
> + counter->low_watermark = (unsigned long long)LLONG_MAX;
> + counter->high_watermark = (unsigned long long)LLONG_MAX;
> + counter->watermark_state = RES_WATERMARK_BELOW_LOW;
> }
>
> int res_counter_charge_locked(struct res_counter *counter, unsigned long val)

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

---

Subject: Re: [RFC][only for review] memory controller bacground reclaim [3/5] high/low watermark support in
Posted by KAMEZAWA Hiroyuki on Thu, 29 Nov 2007 03:24:02 GMT
View Forum Message <> Reply to Message

On Thu, 29 Nov 2007 11:56:08 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> > This patch adds high/low watermark parameter to res_counter.
> > splitted out from YAMAMOTO's background page reclaim for memory cgroup set.
>
> thanks.
>
> > + * Watermarks
> > + * Should be changed automatically when the limit is changed and
> > + * keep low < high < limit.
> > + */
> > + unsigned long long high_watermark;
> > + unsigned long long low_watermark;
> > + /*
>
> do you have any specific reason not to allow low == high == limit?
No.

> if you want to ensure low < high < limit, it's better to make the
> default values below meet the condition as well.
Hmm, I see.

> to me, it seems weird to prevent users from making these values back to
> the default.
>

will fix.
LLONG_MAX-1 for high
LLONG_MAX-2 for low ...?

BTW, it should be "low + PAGE_SIZE < high + PAGE_SIZE < limit" ...?
Does anyone have good idea ?

Thanks,
-Kame

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

---

Subject: Re: [RFC][only for review] memory controller bacground reclaim [3/5] high/low watermark support in
Posted by yamamoto on Thu, 29 Nov 2007 03:36:59 GMT
View Forum Message <> Reply to Message

> > to me, it seems weird to prevent users from making these values back to
> > the default.
> >
> will fix.
> LLONG_MAX-1 for high
> LLONG_MAX-2 for low ...?

imo it's better to simply allow low == high == limit.

> BTW, it should be "low + PAGE_SIZE < high + PAGE_SIZE < limit" ...?

it shouldn't, unless you are going to throw away the res_counter abstraction.

YAMAMOTO Takashi

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

---

Subject: Re: [RFC][only for review] memory controller bacground reclaim [0/5] (Does anyone have an idea abo
Posted by KAMEZAWA Hiroyuki on Thu, 29 Nov 2007 11:53:24 GMT
View Forum Message <> Reply to Message

On Wed, 28 Nov 2007 17:49:23 +0900

KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:

> Hi, this set is for memory controller background reclaim.
>
> Merged YAMAMOTO-san's version onto 2.6.23-rc3-mm1 + my NUMA patch.
> And splitted to several sets.
>
> Major changes from his one is
> - use kthread instead of work_queue
> - adjust high/low watermark when limit changes automatically
>   and set default value. (a user can specify his own later.)
>
FYI rather than RFC.

I wrote attached patch and run kernbench on 8CPU/2Node NUMA/ia64.
It does make -j 32.

Memory limitation was 800M. Low/High watermark here was 750M/780M.

== These numbers are stable to some extent.==
2.6.24-rc3-mm2: (Limit: 800M)
Average Optimal -j 32 Load Run:
Elapsed Time 358.933----------(*)
User Time 1069.63
System Time 140.667
Percent CPU 337.333
Context Switches 220821
Sleeps 196912

2.6.24-rc3-mm2 + throttle (Limit:800M)
Average Optimal -j 32 Load Run:
Elapsed Time 266.697----------(*)
User Time 1105.39
System Time 124.423
Percent CPU 471.667
Context Switches 251797
Sleeps 231038

2.6.24-rc3-mm2 + throttle + High/Low watermark.
(low:750M High:780M Limit:800M)
Average Optimal -j 32 Load Run:
Elapsed Time 266.844----------(*)
User Time 1112.9
System Time 112.273
Percent CPU 473.667
Context Switches 251795
Sleeps 220339
==

Seems throttling reclaim has some good effect (for kernbench).
Does anyone have an idea for throttling reclaiming of memory controller ?

Thanks,
-Kame
==
Just and Experimental. add throttling at starting direct reclaim.

```
 mm/memcontrol.c |   22 ++++++++++++++++++----
 1 file changed, 18 insertions(+), 4 deletions(-)

Index: linux-2.6.24-rc3-mm2/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc3-mm2.orig/mm/memcontrol.c
+++ linux-2.6.24-rc3-mm2/mm/memcontrol.c
@@ -133,6 +133,7 @@ struct mem_cgroup {

  unsigned long control_type; /* control RSS or RSS+Pagecache */
  int prev_priority; /* for recording reclaim priority */
+ atomic_t reclaimers; /* # of processes which calls reclaim */
  /*
   * statistics.
   */
@@ -635,14 +636,27 @@ retry:
   */
  while (res_counter_charge(&mem->res, PAGE_SIZE)) {
   bool is_atomic = gfp_mask & GFP_ATOMIC;
+  int limit;
   /*
    * We cannot reclaim under GFP_ATOMIC, fail the charge
    */
   if (is_atomic)
    goto noreclaim;
-
-  if (try_to_free_mem_cgroup_pages(mem, gfp_mask))
-   continue;
+  /*
+   * Experimental:
+   * Obviously, we need some throttling here.
+   */
+  limit = num_online_cpus()/4 + 1;
+  if (atomic_add_unless(&mem->reclaimers, 1, limit)) {
+   if (try_to_free_mem_cgroup_pages(mem, gfp_mask)) {
+    atomic_dec(&mem->reclaimers);
+    continue;
+   }
```

```
+   atomic_dec(&mem->reclaimers);
+  } else {
+   yield();
+   nr_retries++;
+  }

   /*
    * try_to_free_mem_cgroup_pages() might not give us a full
@@ -1168,7 +1182,7 @@ mem_cgroup_create(struct cgroup_subsys *

  mem->control_type = MEM_CGROUP_TYPE_ALL;
  memset(&mem->info, 0, sizeof(mem->info));
-
+ atomic_set(&mem->reclaimers, 0);
  for_each_node_state(node, N_POSSIBLE)
   if (alloc_mem_cgroup_per_zone_info(mem, node))
    goto free_out;
```

Subject: Re: [RFC][only for review] memory controller bacground reclaim [0/5] (Does anyone have an idea abo
Posted by Balbir Singh on Thu, 29 Nov 2007 14:42:51 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> On Wed, 28 Nov 2007 17:49:23 +0900
> KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
>
>> Hi, this set is for memory controller background reclaim.
>>
>> Merged YAMAMOTO-san's version onto 2.6.23-rc3-mm1 + my NUMA patch.
>> And splitted to several sets.
>>
>> Major changes from his one is
>> - use kthread instead of work_queue
>> - adjust high/low watermark when limit changes automatically
>>   and set default value. (a user can specify his own later.)
>>
> FYI rather than RFC.

>
> I wrote attached patch and run kernbench on 8CPU/2Node NUMA/ia64.
> It does make -j 32.
>
> Memory limitation was 800M. Low/High watermark here was 750M/780M.
>
> == These numbers are stable to some extent.==
> 2.6.24-rc3-mm2: (Limit: 800M)
> Average Optimal -j 32 Load Run:
> Elapsed Time 358.933----------(*)
> User Time 1069.63
> System Time 140.667
> Percent CPU 337.333
> Context Switches 220821
> Sleeps 196912
>
> 2.6.24-rc3-mm2 + throttle (Limit:800M)
> Average Optimal -j 32 Load Run:
> Elapsed Time 266.697----------(*)
> User Time 1105.39
> System Time 124.423
> Percent CPU 471.667
> Context Switches 251797
> Sleeps 231038
>
> 2.6.24-rc3-mm2 + throttle + High/Low watermark.
> (low:750M High:780M Limit:800M)
> Average Optimal -j 32 Load Run:
> Elapsed Time 266.844----------(*)
> User Time 1112.9
> System Time 112.273
> Percent CPU 473.667
> Context Switches 251795
> Sleeps 220339
> ==
>

Looks good to me, was there any impact on memory.failcnt?

> Seems throttling reclaim has some good effect (for kernbench).
> Does anyone have an idea for throttling reclaiming of memory controller ?
>

In the past I've run workloads of apache+geronimo+open trade, I've run
linear sequential memory access tests, kernbench, lmbench, database
benchmarks (DOTS, pgbench, etc). I think Lee Schermerhorn has a very
interesting setup (that I need to learn to replicate).

> Thanks,
> -Kame
--
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][only for review] memory controller bacground reclaim [4/5] high/low watermark for memory
Posted by Oren Laadan on Thu, 29 Nov 2007 19:55:56 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> On Wed, 28 Nov 2007 15:20:42 +0300
> Pavel Emelyanov <xemul@openvz.org> wrote:
>>> + mem = mem_cgroup_from_cont(cont);
>>> + spin_lock_irqsave(&mem->res.lock, flags);
>>> + val = res_counter_get(&mem->res, RES_LIMIT);
>>> + if (val == (unsigned long long) LLONG_MAX) {
>>> + low = (unsigned long long) LLONG_MAX;
>>> + high = (unsigned long long) LLONG_MAX;
>>> + } else {
>>> + low = val * DEFAULT_WATERMARK_PERCENT_LOW / 100ULL;
>>> + high = val * DEFAULT_WATERMARK_PERCENT_HIGH / 100ULL;
>> BTW, I tried to compile such a code:
>>
>> unsigned long long x, y;
>> y = ...;
>> x = y / 100ULL;
>>
>> (similar to yours) and that's what I got:
>>
>> kernel/built-in.o: In function `xxx':
>> : undefined reference to `__udivdi3'
>>
>> It looks like i386 doesn't have any support for ULL divisions.
>> It doesn't have it in CPU, and I thought that it was some-how
>> emulated, but it is not...
>>
>> Did I miss something?
>>
> Ah, I didn't try i386...

> But I'll drop this automatic watermark adjustment part.

FYI, if you do need it, you can do long long division on i386 using
the macro "do_div()", defined in "include/asm-i386/div64.h".

Oren.

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

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

---

Subject: Re: [RFC][only for review] memory controller bacground reclaim [4/5] high/low watermark for memory
Posted by KAMEZAWA Hiroyuki on Fri, 30 Nov 2007 00:26:42 GMT
View Forum Message <> Reply to Message

On Thu, 29 Nov 2007 14:55:56 -0500
Oren Laadan <orenl@cs.columbia.edu> wrote:

> >> It looks like i386 doesn't have any support for ULL divisions.
> >> It doesn't have it in CPU, and I thought that it was some-how
> >> emulated, but it is not...
> >>
> >> Did I miss something?
> >>
> > Ah, I didn't try i386...
> > But I'll drop this automatic watermark adjustment part.
>
> FYI, if you do need it, you can do long long division on i386 using
> the macro "do_div()", defined in "include/asm-i386/div64.h".
>
Thank you.
I'll check it.

-Kame

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [RFC][only for review] memory controller bacground reclaim [0/5] (Does anyone have an idea abo
Posted by KAMEZAWA Hiroyuki on Fri, 30 Nov 2007 00:28:57 GMT
View Forum Message <> Reply to Message

On Thu, 29 Nov 2007 20:12:51 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> > == These numbers are stable to some extent.==
> > 2.6.24-rc3-mm2: (Limit: 800M)
> > Average Optimal -j 32 Load Run:
> > Elapsed Time 358.933----------(*)
> > User Time 1069.63
> > System Time 140.667
> > Percent CPU 337.333
> > Context Switches 220821
> > Sleeps 196912
> >
> > 2.6.24-rc3-mm2 + throttle (Limit:800M)
> > Average Optimal -j 32 Load Run:
> > Elapsed Time 266.697----------(*)
> > User Time 1105.39
> > System Time 124.423
> > Percent CPU 471.667
> > Context Switches 251797
> > Sleeps 231038
> >
> > 2.6.24-rc3-mm2 + throttle + High/Low watermark.
> > (low:750M High:780M Limit:800M)
> > Average Optimal -j 32 Load Run:
> > Elapsed Time 266.844----------(*)
> > User Time 1112.9
> > System Time 112.273
> > Percent CPU 473.667
> > Context Switches 251795
> > Sleeps 220339
> > ==
> >
>
> Looks good to me, was there any impact on memory.failcnt?
>
This version o patch doesn't care it. (I'll fix.)
I just wanted to ask someone has (another) throttling patch or idea.

> > Seems throttling reclaim has some good effect (for kernbench).
> > Does anyone have an idea for throttling reclaiming of memory controller ?

> >
>
> In the past I've run workloads of apache+geronimo+open trade, I've run
> linear sequential memory access tests, kernbench, lmbench, database
> benchmarks (DOTS, pgbench, etc). I think Lee Schermerhorn has a very
> interesting setup (that I need to learn to replicate).
>
Ok, thanks.
I will reflesh and post new one.

Regards
-Kame

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

---

Subject: Re: [RFC][only for review] memory controller bacground reclaim [4/5] high/low watermark for memory
Posted by Paul Menage on Sat, 01 Dec 2007 07:09:20 GMT
View Forum Message <> Reply to Message

On Nov 28, 2007 12:56 AM, KAMEZAWA Hiroyuki
<kamezawa.hiroyu@jp.fujitsu.com> wrote:
> {
> + .name = "low_watermark_in_bytes",
> + .private = RES_LOW_WATERMARK,
> + .write = mem_cgroup_write,
> + .read = mem_cgroup_read,
> + },
> + {
> + .name = "high_watermark_in_bytes",
> + .private = RES_HIGH_WATERMARK,
> + .write = mem_cgroup_write,
> + .read = mem_cgroup_read,
> + },

>From a style point of view, I dislike having the "in_bytes" suffix
tacked on to all the memory controller filenames.

If people really want this to be self-documenting, how about we allow
cgroup control files to specify metadata, which would be presented to
the user via an auto-generated "api" file.

As an example, the addition above might then look something like:

```
{
    .name = "low_watermark",
    .units = "bytes",
    .description = "usage below which background reclaim stops",
    .write = mem_cgroup_write,
    .read = mem_cgroup_read,
}
```

which would correspond to a line in the "mem.api" auto-generated control file as

low_watermark: usage below which background reclaim stops (bytes)

Paul

Subject: Re: [RFC][only for review] memory controller bacground reclaim [5/5]
Posted by Paul Menage on Sat, 01 Dec 2007 07:16:26 GMT
View Forum Message <> Reply to Message

On Nov 28, 2007 12:57 AM, KAMEZAWA Hiroyuki
<kamezawa.hiroyu@jp.fujitsu.com> wrote:

```
> @@ -1226,6 +1303,7 @@
>  {
>         struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
>         mem_cgroup_force_empty(mem);
> +       kthread_stop(mem->daemon.thread);
>  }
>
```

This means that if the user tries to delete the cgroup (hence calling
the predestroy handlers) but fails (e.g. because some other subsystem
is still busy) then the cgroup's reclaim daemon will have exited but
the cgroup will still exist. I think you should wait until the actual
destroy call, and synchronize with the exiting reclaim daemon.

Paul

Subject: Re: [RFC][only for review] memory controller bacground reclaim [4/5] high/low watermark for memory
Posted by Balbir Singh on Sat, 01 Dec 2007 10:45:36 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On Nov 28, 2007 12:56 AM, KAMEZAWA Hiroyuki
> <kamezawa.hiroyu@jp.fujitsu.com> wrote:
>> {
>> + .name = "low_watermark_in_bytes",
>> + .private = RES_LOW_WATERMARK,
>> + .write = mem_cgroup_write,
>> + .read = mem_cgroup_read,
>> + },
>> + {
>> + .name = "high_watermark_in_bytes",
>> + .private = RES_HIGH_WATERMARK,
>> + .write = mem_cgroup_write,
>> + .read = mem_cgroup_read,
>> + },
>
> From a style point of view, I dislike having the "in_bytes" suffix
> tacked on to all the memory controller filenames.
>

The idea came from Andrew, when he suggested it.

Please see /proc/sys/vm. We have files like min_free_kbyes, I think it's
a good idea to tell the user what units are used.

> If people really want this to be self-documenting, how about we allow
> cgroup control files to specify metadata, which would be presented to
> the user via an auto-generated "api" file.
>
> As an example, the addition above might then look something like:
>
> {
> .name = "low_watermark",
> .units = "bytes",
> .description = "usage below which background reclaim stops",
> .write = mem_cgroup_write,
> .read = mem_cgroup_read,
> }
>
> which would correspond to a line in the "mem.api" auto-generated control file as
>

The user is expected to cat "memory.api" in order to figure out how to
use the file?

> low_watermark: usage below which background reclaim stops (bytes)
>
> Paul

--
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][only for review] memory controller bacground reclaim [4/5] high/low watermark for memory
Posted by Paul Menage on Sat, 01 Dec 2007 16:55:12 GMT
View Forum Message <> Reply to Message

On Dec 1, 2007 2:45 AM, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> >
> > which would correspond to a line in the "mem.api" auto-generated control file as
> >
>
> The user is expected to cat "memory.api" in order to figure out how to
> use the file?
>

Yes, the very first time they try to use a particular cgroup, if
they've not bothered to read any documentation about it. There has to
be some documentation *somewhere*, and putting some of the basic bits
at a point where they're immediately accessible could be an advantage.

Paul

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

---

Subject: Re: [RFC][only for review] memory controller bacground reclaim [4/5] high/low watermark for memory
Posted by Balbir Singh on Sat, 01 Dec 2007 17:16:19 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On Dec 1, 2007 2:45 AM, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>>> which would correspond to a line in the "mem.api" auto-generated control file as
>>>
>> The user is expected to cat "memory.api" in order to figure out how to
>> use the file?
>>
>
> Yes, the very first time they try to use a particular cgroup, if
> they've not bothered to read any documentation about it. There has to
> be some documentation *somewhere*, and putting some of the basic bits
> at a point where they're immediately accessible could be an advantage.
>
> Paul

I worry about the kernel size bloat due to adding documentation in the
kernel. I don't think the documentation belongs to the kernel image, but
using file names like /proc/sys/vm/min_free_kbytes is a much better idea.

--
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