Subject: [RFC][PATCH] memory cgroup enhancements updated [0/10] intro
Posted by KAMEZAWA Hiroyuki on Fri, 19 Oct 2007 09:24:36 GMT
View Forum Message <> Reply to Message

These patches are for memory cgroup on my queue.
Just dumping before week-end.
I'd like to post these against the next -mm.

This is RFC again.
Comments for previous version is reflected AMAP, thanks.

Some of patches has no change. Several pathces are new.

[1/10] fix try_to_free_mem_cgroup_pages() for NUMA (NEW)
[2/10] force_empty for drop all page_cgroup in empty cgroup. (REFLESHED)
[3/10] remember page is charged as page cache (changed #define placement)
[4/10] remember page is on active list (changed #define placement)
[5/10] per cpu memory cgroup accounting
[6/10] show per cpu memory cgroup accounting
[7/10] account failcnt for RSS and CACHE (NEW)
[8/10] add pre_destroy handler for cgroup (NEW)
[9/10] do force_empty at rmdir() implicitly (NEW)
[10/10] NUMA-aware accounting. (NEW)

Any comments are welcome.

Thanks,
-Kame

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

Subject: [RFC][PATCH] memory cgroup enhancements updated [1/10] try_to_free_mem_cgroup_pages bugfix
Posted by KAMEZAWA Hiroyuki on Fri, 19 Oct 2007 09:29:52 GMT
View Forum Message <> Reply to Message

Because NODE_DATA(node)->node_zonelists[] is guaranteed to contain
all necessary zones, it is not necessary to use for_each_online_node.

And this for_each_online_node() makes reclaim routine start always
from node 0. This is bad.

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

```
 mm/vmscan.c |    8 +++-----
 1 file changed, 3 insertions(+), 5 deletions(-)

Index: devel-2.6.23-mm1/mm/vmscan.c
===================================================================
--- devel-2.6.23-mm1.orig/mm/vmscan.c
+++ devel-2.6.23-mm1/mm/vmscan.c
@@ -1375,15 +1375,13 @@ unsigned long try_to_free_mem_cgroup_pag
   .mem_cgroup = mem_cont,
   .isolate_pages = mem_cgroup_isolate_pages,
  };
- int node;
+ int node = numa_node_id();
  struct zone **zones;
  int target_zone = gfp_zone(GFP_HIGHUSER_MOVABLE);

- for_each_online_node(node) {
-  zones = NODE_DATA(node)->node_zonelists[target_zone].zones;
-  if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
+ zones = NODE_DATA(node)->node_zonelists[target_zone].zones;
+ if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
    return 1;
- }
  return 0;
 }
 #endif
```

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

---

Subject: [RFC][PATCH] memory cgroup enhancements updated [2/10] force empty interface

Posted by KAMEZAWA Hiroyuki on Fri, 19 Oct 2007 09:30:31 GMT
View Forum Message <> Reply to Message

This patch adds an interface "memory.force_empty".
Any write to this file will drop all charges in this cgroup if
there is no task under.

%echo 1 > /...../memory.force_empty

will drop all charges of memory cgroup if cgroup's tasks is empty.

This is useful to invoke rmdir() against memory cgroup successfully.

Tested and worked well on x86_64/fake-NUMA system.

Changelog v4 -> v5:
 - added comments to mem_cgroup_force_empty()
 - made mem_force_empty_read return -EINVAL
 - cleanup mem_cgroup_force_empty_list()
 - removed SWAP_CLUSTER_MAX

Changelog v3 -> v4:
 - adjusted to 2.6.23-mm1
 - fixed typo
 - changes buf[2]="0" to static const

Changelog v2 -> v3:
 - changed the name from force_reclaim to force_empty.

Changelog v1 -> v2:
 - added a new interface force_reclaim.
 - changes spin_lock to spin_lock_irqsave().

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

 mm/memcontrol.c | 110 ++----
 1 file changed, 103 insertions(+), 7 deletions(-)

Index: devel-2.6.23-mm1/mm/memcontrol.c
===
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -480,6 +480,7 @@ void mem_cgroup_uncharge(struct page_cgr
   page = pc->page;
   /*
    * get page->cgroup and clear it under lock.
+   * force_empty can drop page->cgroup without checking refcnt.
    */
   if (clear_page_cgroup(page, pc) == pc) {
    mem = pc->mem_cgroup;
@@ -489,13 +490,6 @@ void mem_cgroup_uncharge(struct page_cgr
    list_del_init(&pc->lru);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
    kfree(pc);
- } else {
-   /*

```
-    * Note:This will be removed when force-empty patch is
-    * applied. just show warning here.
-    */
-   printk(KERN_ERR "Race in mem_cgroup_uncharge() ?");
-   dump_stack();
   }
  }
 }
@@ -543,6 +537,76 @@ retry:
  return;
 }

+/*
+ * 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.
+ */
+#define FORCE_UNCHARGE_BATCH (128)
+static void
+mem_cgroup_force_empty_list(struct mem_cgroup *mem, struct list_head *list)
+{
+ struct page_cgroup *pc;
+ struct page *page;
+ int count;
+ unsigned long flags;
+
+retry:
+ count = FORCE_UNCHARGE_BATCH;
+ spin_lock_irqsave(&mem->lru_lock, flags);
+
+ while (--count && !list_empty(list)) {
+  pc = list_entry(list->prev, struct page_cgroup, lru);
+  page = pc->page;
+  /* Avoid race with charge */
+  atomic_set(&pc->ref_cnt, 0);
+  if (clear_page_cgroup(page, pc) == pc) {
+   css_put(&mem->css);
+   res_counter_uncharge(&mem->res, PAGE_SIZE);
+   list_del_init(&pc->lru);
+   kfree(pc);
+  } else  /* being uncharged ? ...do relax */
+   break;
+ }
+ spin_unlock_irqrestore(&mem->lru_lock, flags);
+ if (!list_empty(list)) {
+  cond_resched();
+  goto retry;
+ }
```

```
+ return;
+}
+
+/*
+ * make mem_cgroup's charge to be 0 if there is no task.
+ * This enables deleting this mem_cgroup.
+ */
+
+int mem_cgroup_force_empty(struct mem_cgroup *mem)
+{
+ int ret = -EBUSY;
+ css_get(&mem->css);
+ /*
+  * page reclaim code (kswapd etc..) will move pages between
+`  * active_list <-> inactive_list while we don't take a lock.
+  * So, we have to do loop here until all lists are empty.
+  */
+ while (!(list_empty(&mem->active_list) &&
+   list_empty(&mem->inactive_list))) {
+  if (atomic_read(&mem->css.cgroup->count) > 0)
+   goto out;
+  /* drop all page_cgroup in active_list */
+  mem_cgroup_force_empty_list(mem, &mem->active_list);
+  /* drop all page_cgroup in inactive_list */
+  mem_cgroup_force_empty_list(mem, &mem->inactive_list);
+ }
+ ret = 0;
+out:
+ css_put(&mem->css);
+ return ret;
+}
+
+
+
 int mem_cgroup_write_strategy(char *buf, unsigned long long *tmp)
 {
  *tmp = memparse(buf, &buf);
@@ -628,6 +692,33 @@ static ssize_t mem_control_type_read(str
    ppos, buf, s - buf);
 }

+
+static ssize_t mem_force_empty_write(struct cgroup *cont,
+    struct cftype *cft, struct file *file,
+    const char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
```

```
+ int ret;
+ ret = mem_cgroup_force_empty(mem);
+ if (!ret)
+  ret = nbytes;
+ return ret;
+}
+
+/*
+ * Note: This should be removed if cgroup supports write-only file.
+ */
+
+static ssize_t mem_force_empty_read(struct cgroup *cont,
+    struct cftype *cft,
+    struct file *file, char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+ return -EINVAL;
+}
+
+
 static struct cftype mem_cgroup_files[] = {
  {
   .name = "usage_in_bytes",
@@ -650,6 +741,11 @@ static struct cftype mem_cgroup_files[]
   .write = mem_control_type_write,
   .read = mem_control_type_read,
  },
+ {
+  .name = "force_empty",
+  .write = mem_force_empty_write,
+  .read = mem_force_empty_read,
+ },
 };

 static struct mem_cgroup init_mem_cgroup;
```

Subject: [RFC][PATCH] memory cgroup enhancements updated [3/10] remember pagecache
Posted by KAMEZAWA Hiroyuki on Fri, 19 Oct 2007 09:31:06 GMT
View Forum Message <> Reply to Message

Add PCGF_PAGECACHE flag to page_cgroup to remember "this page is

```
charged as page-cache."
This is very useful for implementing precise accounting in memory cgroup.

Changelog v1 -> v2
  - moved #define to out-side of struct definition

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

 mm/memcontrol.c |   18 ++++++++++++++---
 1 file changed, 15 insertions(+), 3 deletions(-)

Index: devel-2.6.23-mm1/mm/memcontrol.c
===================================================================
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -83,7 +83,9 @@ 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  (0x1) /* charged as page-cache */

 enum {
  MEM_CGROUP_TYPE_UNSPEC = 0,
@@ -315,8 +317,8 @@ 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)
+static int mem_cgroup_charge_common(struct page *page, struct mm_struct *mm,
+    gfp_t gfp_mask, int is_cache)
 {
  struct mem_cgroup *mem;
  struct page_cgroup *pc;
@@ -418,6 +420,10 @@ noreclaim:
  atomic_set(&pc->ref_cnt, 1);
  pc->mem_cgroup = mem;
  pc->page = page;
+ if (is_cache)
+  pc->flags = PCGF_PAGECACHE;
+ else
+  pc->flags = 0;
  if (page_cgroup_assign_new_page_cgroup(page, pc)) {
   /*
    * an another charge is added to this page already.
```

```
@@ -442,6 +448,12 @@ 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?
  */
@@ -454,7 +466,7 @@ 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;
 }
```

Subject: [RFC][PATCH] memory cgroup enhancements updated [4/10] record page cgroup on active list

Posted by KAMEZAWA Hiroyuki on Fri, 19 Oct 2007 09:31:48 GMT

View Forum Message <> Reply to Message

Remember page_cgroup is on active_list or not in page_cgroup->flags.

Against 2.6.23-mm1.

Changelog v1->v2
 - moved #define to out-side of struct definition

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

```
 mm/memcontrol.c |   12 ++++++++----
 1 file changed, 8 insertions(+), 4 deletions(-)

Index: devel-2.6.23-mm1/mm/memcontrol.c
```

```
===================================================================
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -86,6 +86,7 @@ struct page_cgroup {
  int  flags;
 };
 #define PCGF_PAGECACHE  (0x1) /* charged as page-cache */
+#define PCGF_ACTIVE  (0x2) /* page_cgroup is on active list */

 enum {
  MEM_CGROUP_TYPE_UNSPEC = 0,
@@ -208,10 +209,13 @@ clear_page_cgroup(struct page *page, str

 static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
- if (active)
+ if (active) {
+  pc->flags |= PCGF_ACTIVE;
   list_move(&pc->lru, &pc->mem_cgroup->active_list);
- else
+ } else {
+  pc->flags &= ~PCGF_ACTIVE;
   list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
+ }
 }

 int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem)
@@ -421,9 +425,9 @@ noreclaim:
  pc->mem_cgroup = mem;
  pc->page = page;
  if (is_cache)
-  pc->flags = PCGF_PAGECACHE;
+  pc->flags = PCGF_PAGECACHE | PCGF_ACTIVE;
  else
-  pc->flags = 0;
+  pc->flags = PCGF_ACTIVE;
  if (page_cgroup_assign_new_page_cgroup(page, pc)) {
   /*
    * an another charge is added to this page already.
```

_____

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

---

Subject: [RFC][PATCH] memory cgroup enhancements updated [5/10] per cpu

Add statistics account infrastructure for memory controller.

Changelog v1 -> v2
 - Removed Charge/Uncharge counter
 - reflected comments.
   - changes __move_lists() args.
   - changes __mem_cgroup_stat_add() name, comment and added VM_BUGON

Changes from original:
 - divided into 2 patch (account and show info)
 - changed from u64 to s64
 - added mem_cgroup_stat_add() and batched statistics modification logic.
 - removed stat init code because mem_cgroup is allocated by kzalloc().

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

 mm/memcontrol.c | 120 ++----
 1 file changed, 113 insertions(+), 7 deletions(-)

Index: devel-2.6.23-mm1/mm/memcontrol.c
===
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -35,6 +35,64 @@ struct cgroup_subsys mem_cgroup_subsys;
 static const int MEM_CGROUP_RECLAIM_RETRIES = 5;

 /*
+ * Statistics for memory cgroup.
+ */
+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 */
+
+ /*
+ * usage = charge - uncharge.
+ */
+ MEM_CGROUP_STAT_ACTIVE,   /* # of pages in active list */
+ MEM_CGROUP_STAT_INACTIVE, /* # of pages on inactive list */
+
+ MEM_CGROUP_STAT_NSTATS,

```
+};
+
+struct mem_cgroup_stat_cpu {
+ s64 count[MEM_CGROUP_STAT_NSTATS];
+} ____cacheline_aligned_in_smp;
+
+struct mem_cgroup_stat {
+ struct mem_cgroup_stat_cpu cpustat[NR_CPUS];
+};
+
+/*
+ * For batching....mem_cgroup_charge_statistics()(see below).
+ * MUST be called under preempt_disable().
+ */
+static inline void __mem_cgroup_stat_add(struct mem_cgroup_stat *stat,
+                enum mem_cgroup_stat_index idx, int val)
+{
+ int cpu = smp_processor_id();
+#ifdef CONFIG_PREEMPT
+ VM_BUG_ON(preempt_count() == 0);
+#endif
+ stat->cpustat[cpu].count[idx] += val;
+}
+
+static inline void mem_cgroup_stat_inc(struct mem_cgroup_stat *stat,
+  enum mem_cgroup_stat_index idx)
+{
+ preempt_disable();
+ __mem_cgroup_stat_add(stat, idx, 1);
+ preempt_enable();
+}
+
+static inline void mem_cgroup_stat_dec(struct mem_cgroup_stat *stat,
+  enum mem_cgroup_stat_index idx)
+{
+ preempt_disable();
+ __mem_cgroup_stat_add(stat, idx, -1);
+ preempt_enable();
+}
+
+
+/*
  * 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,
@@ -63,6 +121,10 @@ struct mem_cgroup {
   */
  spinlock_t lru_lock;
```

```
  unsigned long control_type; /* control RSS or RSS+Pagecache */
+ /*
+  * statistics.
+  */
+ struct mem_cgroup_stat stat;
 };

 /*
@@ -96,6 +158,33 @@ enum {
  MEM_CGROUP_TYPE_MAX,
 };

+/*
+ * Batched statistics modification.
+ * We have to modify several values at charge/uncharge..
+ */
+static inline void
+mem_cgroup_charge_statistics(struct mem_cgroup *mem, int flags, int charge)
+{
+ int val = (charge)? 1 : -1;
+ struct mem_cgroup_stat *stat = &mem->stat;
+ preempt_disable();
+
+ if (flags & PCGF_PAGECACHE)
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_PAGECACHE, val);
+ else
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_RSS, val);
+
+ if (flags & PCGF_ACTIVE)
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE, val);
+ else
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE, val);
+
+ preempt_enable();
+}
+
+
+
+
 static struct mem_cgroup init_mem_cgroup;

 static inline
@@ -209,12 +298,27 @@ clear_page_cgroup(struct page *page, str

 static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
+ int moved = 0;
+ struct mem_cgroup *mem = pc->mem_cgroup;
```

```
+
+ if (active && (pc->flags & PCGF_ACTIVE) == 0)
+  moved = 1; /* Move from inactive to active */
+ else if (!active && (pc->flags & PCGF_ACTIVE))
+  moved = -1; /* Move from active to inactive */
+
+ if (moved) {
+  struct mem_cgroup_stat *stat = &mem->stat;
+  preempt_disable();
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE, moved);
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE, -moved);
+  preempt_enable();
+ }
  if (active) {
   pc->flags |= PCGF_ACTIVE;
-  list_move(&pc->lru, &pc->mem_cgroup->active_list);
+  list_move(&pc->lru, &mem->active_list);
  } else {
   pc->flags &= ~PCGF_ACTIVE;
-  list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
+  list_move(&pc->lru, &mem->inactive_list);
  }
 }

@@ -233,15 +337,12 @@ int task_in_mem_cgroup(struct task_struc
  */
 void mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
- struct mem_cgroup *mem;
  if (!pc)
   return;

- mem = pc->mem_cgroup;
-
- spin_lock(&mem->lru_lock);
+ spin_lock(&pc->mem_cgroup->lru_lock);
  __mem_cgroup_move_lists(pc, active);
- spin_unlock(&mem->lru_lock);
+ spin_unlock(&pc->mem_cgroup->lru_lock);
 }

 unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
@@ -440,6 +541,9 @@ noreclaim:
   goto retry;
  }

+ /* Update statistics vector */
+ mem_cgroup_charge_statistics(mem, pc->flags, true);
```

```
+
  spin_lock_irqsave(&mem->lru_lock, flags);
  list_add(&pc->lru, &mem->active_list);
  spin_unlock_irqrestore(&mem->lru_lock, flags);
@@ -505,6 +609,7 @@ void mem_cgroup_uncharge(struct page_cgr
    spin_lock_irqsave(&mem->lru_lock, flags);
    list_del_init(&pc->lru);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
+   mem_cgroup_charge_statistics(mem, pc->flags, false);
    kfree(pc);
   }
  }
@@ -580,6 +685,7 @@ retry:
    css_put(&mem->css);
    res_counter_uncharge(&mem->res, PAGE_SIZE);
    list_del_init(&pc->lru);
+   mem_cgroup_charge_statistics(mem, pc->flags, false);
    kfree(pc);
   } else  /* being uncharged ? ...do relax */
    break;
```

Subject: [RFC][PATCH] memory cgroup enhancements updated [6/10] memory.stat interface
Posted by KAMEZAWA Hiroyuki on Fri, 19 Oct 2007 09:33:08 GMT
View Forum Message <> Reply to Message

Show accounted information of memory cgroup by memory.stat file

Changelog v1->v2
 - dropped Charge/Uncharge entry.

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

 mm/memcontrol.c | 52 ++
 1 file changed, 52 insertions(+)

Index: devel-2.6.23-mm1/mm/memcontrol.c
===
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -28,6 +28,7 @@

```
 #include <linux/swap.h>
 #include <linux/spinlock.h>
 #include <linux/fs.h>
+#include <linux/seq_file.h>

 #include <asm/uaccess.h>

@@ -841,6 +842,53 @@ static ssize_t mem_force_empty_read(stru
 }


+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_ACTIVE] = { "active", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_INACTIVE] = { "inactive", PAGE_SIZE, },
+};
+
+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;
+  s64 val;
+
+  val = 0;
+  for (cpu = 0; cpu < NR_CPUS; cpu++)
+   val += stat->cpustat[cpu].count[i];
+  val *= mem_cgroup_stat_desc[i].unit;
+  seq_printf(m, "%s %lld\n", mem_cgroup_stat_desc[i].msg, val);
+ }
+ 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",
@@ -868,6 +916,10 @@ static struct cftype mem_cgroup_files[]
   .write = mem_force_empty_write,
   .read = mem_force_empty_read,
  },
+ {
+  .name = "stat",
+  .open = mem_control_stat_open,
+ },
 };

 static struct mem_cgroup init_mem_cgroup;
```

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

---

Subject: [RFC][PATCH] memory cgroup enhancements updated [7/10] RSS/CACHE failcnt

Posted by KAMEZAWA Hiroyuki on Fri, 19 Oct 2007 09:34:52 GMT
View Forum Message <> Reply to Message

cgroup's resource has "failure" counter. But I think memory cgroup
has 2 types of failure
 - failure of cache
 - failure of RSS

This patch adds above 2 information to stat file.
Above information is shown in "byte". But I wonder showing just counter
is better o not...rather than PAGE_SIZE.

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

```
 mm/memcontrol.c |   13 ++++++++++++
 1 file changed, 13 insertions(+)

Index: devel-2.6.23-mm1/mm/memcontrol.c
===================================================================
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -50,6 +50,11 @@ enum mem_cgroup_stat_index {
   */
  MEM_CGROUP_STAT_ACTIVE,    /* # of pages in active list */
  MEM_CGROUP_STAT_INACTIVE,  /* # of pages on inactive list */
+ /*
+  * precise failcnt
+  */
+ MEM_CGROUP_STAT_FAIL_RSS,  /* # of failure in RSS charging */
+ MEM_CGROUP_STAT_FAIL_CACHE,/* # of failure in CACHE charging */

  MEM_CGROUP_STAT_NSTATS,
 };
@@ -486,6 +491,12 @@ retry:
   */
  while (res_counter_charge(&mem->res, PAGE_SIZE)) {
   bool is_atomic = gfp_mask & GFP_ATOMIC;
+  if (is_cache)
+   mem_cgroup_stat_inc(&mem->stat,
+    MEM_CGROUP_STAT_FAIL_CACHE);
+  else
+   mem_cgroup_stat_inc(&mem->stat,
+    MEM_CGROUP_STAT_FAIL_RSS);
   /*
    * We cannot reclaim under GFP_ATOMIC, fail the charge
    */
@@ -850,6 +861,8 @@ static const struct mem_cgroup_stat_desc
  [MEM_CGROUP_STAT_RSS] = { "rss", PAGE_SIZE, },
  [MEM_CGROUP_STAT_ACTIVE] = { "active", PAGE_SIZE, },
  [MEM_CGROUP_STAT_INACTIVE] = { "inactive", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_FAIL_RSS] = { "rss_failure", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_FAIL_CACHE] = { "cache_failure", PAGE_SIZE, },
 };

 static int mem_control_stat_show(struct seq_file *m, void *arg)
```

Subject: [RFC][PATCH] memory cgroup enhancements updated [8/10] add pre_destroy handler
Posted by KAMEZAWA Hiroyuki on Fri, 19 Oct 2007 09:35:44 GMT
View Forum Message <> Reply to Message

My main purpose of this patch is for memory controller..

This patch adds a handler "pre_destroy" to cgroup_subsys.
It is called before cgroup_rmdir() checks all subsys's refcnt.

I think this is useful for subsyses which have some extra refs
even if there are no tasks in cgroup.

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

```
 include/linux/cgroup.h |    1 +
 kernel/cgroup.c        |    7 +++++++
 2 files changed, 8 insertions(+)

Index: devel-2.6.23-mm1/include/linux/cgroup.h
===================================================================
--- devel-2.6.23-mm1.orig/include/linux/cgroup.h
+++ devel-2.6.23-mm1/include/linux/cgroup.h
@@ -233,6 +233,7 @@ int cgroup_is_descendant(const struct cg
 struct cgroup_subsys {
  struct cgroup_subsys_state *(*create)(struct cgroup_subsys *ss,
        struct cgroup *cont);
+ void (*pre_destroy)(struct cgroup_subsys *ss, struct cgroup *cont);
  void (*destroy)(struct cgroup_subsys *ss, struct cgroup *cont);
  int (*can_attach)(struct cgroup_subsys *ss,
      struct cgroup *cont, struct task_struct *tsk);
Index: devel-2.6.23-mm1/kernel/cgroup.c
===================================================================
--- devel-2.6.23-mm1.orig/kernel/cgroup.c
+++ devel-2.6.23-mm1/kernel/cgroup.c
@@ -2158,6 +2158,13 @@ static int cgroup_rmdir(struct inode *un
  parent = cont->parent;
  root = cont->root;
  sb = root->sb;
+ /*
+  * Notify subsyses that rmdir() request comes.
+  */
+ for_each_subsys(root, ss) {
+  if ((cont->subsys[ss->subsys_id]) && ss->pre_destroy)
+   ss->pre_destroy(ss, cont);
+ }

  if (cgroup_has_css_refs(cont)) {
   mutex_unlock(&cgroup_mutex);
```

Subject: [RFC][PATCH] memory cgroup enhancements updated [9/10] implicit force empty at rmdir
Posted by KAMEZAWA Hiroyuki on Fri, 19 Oct 2007 09:36:27 GMT
View Forum Message <> Reply to Message

This patch adds pre_destroy handler for mem_cgroup and try to make
mem_cgroup empty at rmdir().

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

 mm/memcontrol.c | 8 ++++++++
 1 file changed, 8 insertions(+)

Index: devel-2.6.23-mm1/mm/memcontrol.c
===
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -959,6 +959,13 @@ mem_cgroup_create(struct cgroup_subsys *
  return &mem->css;
 }

+static void mem_cgroup_pre_destroy(struct cgroup_subsys *ss,
+   struct cgroup *cont)
+{
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
+ mem_cgroup_force_empty(mem);
+}
+
 static void mem_cgroup_destroy(struct cgroup_subsys *ss,
     struct cgroup *cont)
 {
@@ -1010,6 +1017,7 @@ struct cgroup_subsys mem_cgroup_subsys =
  .name = "memory",
  .subsys_id = mem_cgroup_subsys_id,
  .create = mem_cgroup_create,
+ .pre_destroy = mem_cgroup_pre_destroy,
  .destroy = mem_cgroup_destroy,
  .populate = mem_cgroup_populate,
  .attach = mem_cgroup_move_task,

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

Subject: [RFC][PATCH] memory cgroup enhancements updated [10/10] NUMA aware account
Posted by KAMEZAWA Hiroyuki on Fri, 19 Oct 2007 09:37:30 GMT
View Forum Message <> Reply to Message

This patch is a trial for making stat for memory_cgroup NUMA-aware.

 * dividing per-cpu stat to handle nid information.
 * add nid information to page_cgroup
 * Because init routine has to call kmalloc, init_early is set to be 0.

This is just a trial at this stage. Any comments are welcome.
Works well on my fake NUMA system.
I think we can add "numastat" based on this.

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

```
 mm/memcontrol.c |   97 +++++++++++++++++++++++++++++++++++++++++++++++++---------------
 1 file changed, 71 insertions(+), 26 deletions(-)

Index: devel-2.6.23-mm1/mm/memcontrol.c
===================================================================
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -29,6 +29,7 @@
 #include <linux/spinlock.h>
 #include <linux/fs.h>
 #include <linux/seq_file.h>
+#include <linux/vmalloc.h>

 #include <asm/uaccess.h>

@@ -59,12 +60,18 @@ enum mem_cgroup_stat_index {
  MEM_CGROUP_STAT_NSTATS,
 };

+#ifndef CONFIG_NUMA
 struct mem_cgroup_stat_cpu {
- s64 count[MEM_CGROUP_STAT_NSTATS];
+ s64 count[1][MEM_CGROUP_STAT_NSTATS];
 } ____cacheline_aligned_in_smp;
+#else
```

```
+struct mem_cgroup_stat_cpu {
+ s64 count[MAX_NUMNODES][MEM_CGROUP_STAT_NSTATS];
+} ____cacheline_aligned_in_smp;
+#endif

 struct mem_cgroup_stat {
- struct mem_cgroup_stat_cpu cpustat[NR_CPUS];
+ struct mem_cgroup_stat_cpu *cpustat[NR_CPUS];
 };

 /*
@@ -72,28 +79,28 @@ struct mem_cgroup_stat {
  * MUST be called under preempt_disable().
  */
 static inline void __mem_cgroup_stat_add(struct mem_cgroup_stat *stat,
-                enum mem_cgroup_stat_index idx, int val)
+  enum mem_cgroup_stat_index idx, int nid, int val)
 {
  int cpu = smp_processor_id();
 #ifdef CONFIG_PREEMPT
  VM_BUG_ON(preempt_count() == 0);
 #endif
- stat->cpustat[cpu].count[idx] += val;
+ stat->cpustat[cpu]->count[nid][idx] += val;
 }

 static inline void mem_cgroup_stat_inc(struct mem_cgroup_stat *stat,
-  enum mem_cgroup_stat_index idx)
+  enum mem_cgroup_stat_index idx, int nid)
 {
  preempt_disable();
- __mem_cgroup_stat_add(stat, idx, 1);
+ __mem_cgroup_stat_add(stat, idx, nid, 1);
  preempt_enable();
 }

 static inline void mem_cgroup_stat_dec(struct mem_cgroup_stat *stat,
-  enum mem_cgroup_stat_index idx)
+  enum mem_cgroup_stat_index idx, int nid)
 {
  preempt_disable();
- __mem_cgroup_stat_add(stat, idx, -1);
+ __mem_cgroup_stat_add(stat, idx, nid, -1);
  preempt_enable();
 }

@@ -149,6 +156,7 @@ struct page_cgroup {
  struct list_head lru;  /* per cgroup LRU list */
```

```
  struct page *page;
  struct mem_cgroup *mem_cgroup;
+ int nid;
  atomic_t ref_cnt;  /* Helpful when pages move b/w  */
     /* mapped and cached states     */
  int  flags;
@@ -169,21 +177,23 @@ enum {
  * We have to modify several values at charge/uncharge..
  */
 static inline void
-mem_cgroup_charge_statistics(struct mem_cgroup *mem, int flags, int charge)
+mem_cgroup_charge_statistics(struct mem_cgroup *mem, int nid,
+    int flags, int charge)
 {
  int val = (charge)? 1 : -1;
  struct mem_cgroup_stat *stat = &mem->stat;
  preempt_disable();

  if (flags & PCGF_PAGECACHE)
-  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_PAGECACHE, val);
+  __mem_cgroup_stat_add(stat,
+    MEM_CGROUP_STAT_PAGECACHE, nid, val);
  else
-  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_RSS, val);
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_RSS, nid, val);

  if (flags & PCGF_ACTIVE)
-  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE, val);
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE, nid, val);
  else
-  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE, val);
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE, nid, val);

  preempt_enable();
 }
@@ -315,8 +325,10 @@ static void __mem_cgroup_move_lists(stru
  if (moved) {
   struct mem_cgroup_stat *stat = &mem->stat;
   preempt_disable();
-  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE, moved);
-  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE, -moved);
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE,
+       pc->nid, moved);
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE,
+     pc->nid, -moved);
   preempt_enable();
  }
  if (active) {
```

```
@@ -493,10 +505,10 @@ retry:
   bool is_atomic = gfp_mask & GFP_ATOMIC;
   if (is_cache)
    mem_cgroup_stat_inc(&mem->stat,
-    MEM_CGROUP_STAT_FAIL_CACHE);
+    MEM_CGROUP_STAT_FAIL_CACHE, page_to_nid(page));
   else
    mem_cgroup_stat_inc(&mem->stat,
-    MEM_CGROUP_STAT_FAIL_RSS);
+    MEM_CGROUP_STAT_FAIL_RSS, page_to_nid(page));
   /*
    * We cannot reclaim under GFP_ATOMIC, fail the charge
    */
@@ -537,6 +549,7 @@ noreclaim:
  atomic_set(&pc->ref_cnt, 1);
  pc->mem_cgroup = mem;
  pc->page = page;
+ pc->nid = page_to_nid(page);
  if (is_cache)
   pc->flags = PCGF_PAGECACHE | PCGF_ACTIVE;
  else
@@ -554,7 +567,7 @@ noreclaim:
  }

  /* Update statistics vector */
- mem_cgroup_charge_statistics(mem, pc->flags, true);
+ mem_cgroup_charge_statistics(mem, pc->nid, pc->flags, true);

  spin_lock_irqsave(&mem->lru_lock, flags);
  list_add(&pc->lru, &mem->active_list);
@@ -621,7 +634,8 @@ void mem_cgroup_uncharge(struct page_cgr
    spin_lock_irqsave(&mem->lru_lock, flags);
    list_del_init(&pc->lru);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
-   mem_cgroup_charge_statistics(mem, pc->flags, false);
+   mem_cgroup_charge_statistics(mem, pc->nid, pc->flags,
+       false);
    kfree(pc);
   }
  }
@@ -657,6 +671,7 @@ void mem_cgroup_end_migration(struct pag
 void mem_cgroup_page_migration(struct page *page, struct page *newpage)
 {
  struct page_cgroup *pc;
+ struct mem_cgroup *mem;
 retry:
  pc = page_get_page_cgroup(page);
  if (!pc)
```

```
@@ -664,6 +679,11 @@ retry:
  if (clear_page_cgroup(page, pc) != pc)
   goto retry;
  pc->page = newpage;
+ pc->nid = page_to_nid(newpage);
+ mem = pc->mem_cgroup;
+ mem_cgroup_charge_statistics(mem, page_to_nid(page), pc->flags, false);
+ mem_cgroup_charge_statistics(mem,
+    page_to_nid(newpage), pc->flags, true);
  lock_page_cgroup(newpage);
  page_assign_page_cgroup(newpage, pc);
  unlock_page_cgroup(newpage);
@@ -697,7 +717,8 @@ retry:
    css_put(&mem->css);
    res_counter_uncharge(&mem->res, PAGE_SIZE);
    list_del_init(&pc->lru);
-   mem_cgroup_charge_statistics(mem, pc->flags, false);
+   mem_cgroup_charge_statistics(mem, pc->flags, pc->nid,
+     false);
    kfree(pc);
   } else  /* being uncharged ? ...do relax */
    break;
@@ -872,13 +893,16 @@ static int mem_control_stat_show(struct
  struct mem_cgroup_stat *stat = &mem_cont->stat;
  int i;

- for (i = 0; i < ARRAY_SIZE(stat->cpustat[0].count); i++) {
+ for (i = 0; i < MEM_CGROUP_STAT_NSTATS; i++) {
   unsigned int cpu;
+  int node;
   s64 val;

   val = 0;
-  for (cpu = 0; cpu < NR_CPUS; cpu++)
-   val += stat->cpustat[cpu].count[i];
+  for_each_possible_cpu(cpu)
+   for_each_node_state(node, N_POSSIBLE)
+    val += stat->cpustat[cpu]->count[node][i];
+
   val *= mem_cgroup_stat_desc[i].unit;
   seq_printf(m, "%s %lld\n", mem_cgroup_stat_desc[i].msg, val);
  }
@@ -941,12 +965,14 @@ static struct cgroup_subsys_state *
 mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
 {
  struct mem_cgroup *mem;
+ int cpu;
```

```
  if (unlikely((cont->parent) == NULL)) {
   mem = &init_mem_cgroup;
   init_mm.mem_cgroup = mem;
- } else
+ } else {
   mem = kzalloc(sizeof(struct mem_cgroup), GFP_KERNEL);
+ }

  if (mem == NULL)
   return NULL;
@@ -956,6 +982,17 @@ 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;
+
+ for_each_possible_cpu(cpu) {
+  int nid = cpu_to_node(cpu);
+  struct mem_cgroup_stat_cpu *mcsc;
+  if (sizeof(*mcsc) < PAGE_SIZE)
+   mcsc = kmalloc_node(sizeof(*mcsc), GFP_KERNEL, nid);
+  else
+   mcsc = vmalloc_node(sizeof(*mcsc), nid);
+  memset(mcsc, 0, sizeof(*mcsc));
+  mem->stat.cpustat[cpu] = mcsc;
+ }
  return &mem->css;
 }

@@ -969,7 +1006,15 @@ static void mem_cgroup_pre_destroy(struc
 static void mem_cgroup_destroy(struct cgroup_subsys *ss,
     struct cgroup *cont)
 {
- kfree(mem_cgroup_from_cont(cont));
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
+ int cpu;
+ for_each_possible_cpu(cpu) {
+  if (sizeof(struct mem_cgroup_stat_cpu) < PAGE_SIZE)
+   kfree(mem->stat.cpustat[cpu]);
+  else
+   vfree(mem->stat.cpustat[cpu]);
+ }
+ kfree(mem);
 }

 static int mem_cgroup_populate(struct cgroup_subsys *ss,
@@ -1021,5 +1066,5 @@ struct cgroup_subsys mem_cgroup_subsys =
  .destroy = mem_cgroup_destroy,
  .populate = mem_cgroup_populate,
```

```
  .attach = mem_cgroup_move_task,
- .early_init = 1,
+ .early_init = 0,
 };
```

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

---

Subject: Re: [RFC][PATCH] memory cgroup enhancements updated [7/10] RSS/CACHE failcnt
Posted by Paul Menage on Fri, 19 Oct 2007 16:54:23 GMT
View Forum Message <> Reply to Message

On 10/19/07, KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> cgroup's resource has "failure" counter. But I think memory cgroup
> has 2 types of failure
> - failure of cache
> - failure of RSS

Why do you think these are significantly different? Also, what kind of failure do we get when a task tries to fault on a mapping for a file-backed VMA? That's both RSS (since it's due to be mapped into the process' memory) and pagecache (assuming a cache-backed filesystem).

Paul

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

---

Subject: Re: [RFC][PATCH] memory cgroup enhancements updated [7/10] RSS/CACHE failcnt
Posted by KAMEZAWA Hiroyuki on Fri, 19 Oct 2007 22:33:38 GMT
View Forum Message <> Reply to Message

On Fri, 19 Oct 2007 09:54:23 -0700
"Paul Menage" <menage@google.com> wrote:

> On 10/19/07, KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> > cgroup's resource has "failure" counter. But I think memory cgroup
> > has 2 types of failure
> > - failure of cache
> > - failure of RSS

>
> Why do you think these are significantly different? Also, what kind of
> failure do we get when a task tries to fault on a mapping for a
> file-backed VMA? That's both RSS (since it's due to be mapped into the
> process' memory) and pagecache (assuming a cache-backed filesystem).
>
One is for checking how memory cgroup works.
When memory reaches limit and amount of CACHE doesn't change but
CACHE.failure increases, maybe CACHE is rotating.

One is for debug, I'd like to check swapiness <-> RSS.failure:CACHE.failure
releationship. It's ok to turn these params to be DEBUG option.

File-backed VMA gets RSS.failure.

Thanks,
-Kame

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

---

Subject: Re: [RFC][PATCH] memory cgroup enhancements updated [7/10] RSS/CACHE failcnt
Posted by KAMEZAWA Hiroyuki on Fri, 19 Oct 2007 22:40:26 GMT
View Forum Message <> Reply to Message

On Sat, 20 Oct 2007 07:33:38 +0900
KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> One is for debug, I'd like to check swapiness <-> RSS.failure:CACHE.failure
> releationship. It's ok to turn these params to be DEBUG option.
>
AH...but it's maybe better to check # of page fault in cgroup directly.
I'll consider again.

Thanks,
-Kame

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

---

Subject: Re: [RFC][PATCH] memory cgroup enhancements updated [1/10] try_to_free_mem_cgroup_pages bugfix

KAMEZAWA Hiroyuki wrote:
> Because NODE_DATA(node)->node_zonelists[] is guaranteed to contain
> all necessary zones, it is not necessary to use for_each_online_node.
>
> And this for_each_online_node() makes reclaim routine start always
> from node 0. This is bad.
>

But with this change, we'll start reclaim on the node that the task
hit the limit on - right?

> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
>
>
> mm/vmscan.c | 8 +++-----
> 1 file changed, 3 insertions(+), 5 deletions(-)
>
> Index: devel-2.6.23-mm1/mm/vmscan.c
> ===
> --- devel-2.6.23-mm1.orig/mm/vmscan.c
> +++ devel-2.6.23-mm1/mm/vmscan.c
> @@ -1375,15 +1375,13 @@ unsigned long try_to_free_mem_cgroup_pag
> .mem_cgroup = mem_cont,
> .isolate_pages = mem_cgroup_isolate_pages,
> };
> - int node;
> + int node = numa_node_id();
> struct zone **zones;
> int target_zone = gfp_zone(GFP_HIGHUSER_MOVABLE);
>
> - for_each_online_node(node) {
> - zones = NODE_DATA(node)->node_zonelists[target_zone].zones;
> - if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
> + zones = NODE_DATA(node)->node_zonelists[target_zone].zones;
> + if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
> return 1;
> - }
> return 0;
> }
> #endif
>

--
Warm Regards,

Balbir Singh
Linux Technology Center
IBM, ISTL

_____

Subject: Re: [RFC][PATCH] memory cgroup enhancements updated [1/10] try_to_free_mem_cgroup_pages bugfix
Posted by KAMEZAWA Hiroyuki on Tue, 23 Oct 2007 04:19:14 GMT
View Forum Message <> Reply to Message

On Tue, 23 Oct 2007 09:30:53 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> KAMEZAWA Hiroyuki wrote:
> > Because NODE_DATA(node)->node_zonelists[] is guaranteed to contain
> > all necessary zones, it is not necessary to use for_each_online_node.
> >
> > And this for_each_online_node() makes reclaim routine start always
> > from node 0. This is bad.
> >
>
> But with this change, we'll start reclaim on the node that the task
> hit the limit on - right?
>
yes.
I wonder it can be changed if we have precise information and some logic.
But no concrete idea of better logic now.

-Kame

_____

Subject: Re: [RFC][PATCH] memory cgroup enhancements updated [2/10] force empty interface
Posted by Balbir Singh on Wed, 24 Oct 2007 13:52:00 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> This patch adds an interface "memory.force_empty".

> Any write to this file will drop all charges in this cgroup if
> there is no task under.
>
> %echo 1 > /...../memory.force_empty
>
> will drop all charges of memory cgroup if cgroup's tasks is empty.
>
> This is useful to invoke rmdir() against memory cgroup successfully.
>
> Tested and worked well on x86_64/fake-NUMA system.
>
> Changelog v4 -> v5:
>   - added comments to mem_cgroup_force_empty()
>   - made mem_force_empty_read return -EINVAL
>   - cleanup mem_cgroup_force_empty_list()
>   - removed SWAP_CLUSTER_MAX
>
> Changelog v3 -> v4:
>   - adjusted to 2.6.23-mm1
>   - fixed typo
>   - changes buf[2]="0" to static const
>
> Changelog v2 -> v3:
>   - changed the name from force_reclaim to force_empty.
>
> Changelog v1 -> v2:
>   - added a new interface force_reclaim.
>   - changes spin_lock to spin_lock_irqsave().
>
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
>
> mm/memcontrol.c | 110
++----
> 1 file changed, 103 insertions(+), 7 deletions(-)
>
> Index: devel-2.6.23-mm1/mm/memcontrol.c
> ===
> --- devel-2.6.23-mm1.orig/mm/memcontrol.c
> +++ devel-2.6.23-mm1/mm/memcontrol.c
> @@ -480,6 +480,7 @@ void mem_cgroup_uncharge(struct page_cgr
>   page = pc->page;
>   /*
>   * get page->cgroup and clear it under lock.
> +   * force_empty can drop page->cgroup without checking refcnt.
>   */
>   if (clear_page_cgroup(page, pc) == pc) {

```
>     mem = pc->mem_cgroup;
> @@ -489,13 +490,6 @@ void mem_cgroup_uncharge(struct page_cgr
>     list_del_init(&pc->lru);
>     spin_unlock_irqrestore(&mem->lru_lock, flags);
>     kfree(pc);
> -  } else {
> -   /*
> -    * Note:This will be removed when force-empty patch is
> -    * applied. just show warning here.
> -    */
> -   printk(KERN_ERR "Race in mem_cgroup_uncharge() ?");
> -   dump_stack();
>    }
>   }
>  }
> @@ -543,6 +537,76 @@ retry:
>   return;
>  }
>
> +/*
> + * 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.
> + */
> +#define FORCE_UNCHARGE_BATCH (128)
> +static void
> +mem_cgroup_force_empty_list(struct mem_cgroup *mem, struct list_head *list)
> +{
> + struct page_cgroup *pc;
> + struct page *page;
> + int count;
> + unsigned long flags;
> +
> +retry:
> + count = FORCE_UNCHARGE_BATCH;
> + spin_lock_irqsave(&mem->lru_lock, flags);
> +
> + while (--count && !list_empty(list)) {
> +  pc = list_entry(list->prev, struct page_cgroup, lru);
> +  page = pc->page;
> +  /* Avoid race with charge */
> +  atomic_set(&pc->ref_cnt, 0);
> +  if (clear_page_cgroup(page, pc) == pc) {
> +   css_put(&mem->css);
> +   res_counter_uncharge(&mem->res, PAGE_SIZE);
> +   list_del_init(&pc->lru);
> +   kfree(pc);
> +  } else  /* being uncharged ? ...do relax */
```

> +   break;
> + }
> + spin_unlock_irqrestore(&mem->lru_lock, flags);
> + if (!list_empty(list)) {
> + cond_resched();
> + goto retry;
> + }
> + return;
> +}
> +

We could potentially share some of this code with the background reclaim
code being worked upon by YAMAMOTO-San.

> +/*
> + * make mem_cgroup's charge to be 0 if there is no task.
> + * This enables deleting this mem_cgroup.
> + */
> +
> +int mem_cgroup_force_empty(struct mem_cgroup *mem)
> +{
> + int ret = -EBUSY;
> + css_get(&mem->css);
> + /*
> + * page reclaim code (kswapd etc..) will move pages between
> +` * active_list <-> inactive_list while we don't take a lock.
> + * So, we have to do loop here until all lists are empty.
> + */
> + while (!(list_empty(&mem->active_list) &&
> +   list_empty(&mem->inactive_list))) {
> + if (atomic_read(&mem->css.cgroup->count) > 0)
> +   goto out;
> + /* drop all page_cgroup in active_list */
> + mem_cgroup_force_empty_list(mem, &mem->active_list);
> + /* drop all page_cgroup in inactive_list */
> + mem_cgroup_force_empty_list(mem, &mem->inactive_list);
> + }
> + ret = 0;
> +out:
> + css_put(&mem->css);
> + return ret;
> +}
> +
> +
> +
> int mem_cgroup_write_strategy(char *buf, unsigned long long *tmp)
> {
>   *tmp = memparse(buf, &buf);

```
> @@ -628,6 +692,33 @@ static ssize_t mem_control_type_read(str
>     ppos, buf, s - buf);
>  }
>
> +
> +static ssize_t mem_force_empty_write(struct cgroup *cont,
> +    struct cftype *cft, struct file *file,
> +    const char __user *userbuf,
> +    size_t nbytes, loff_t *ppos)
> +{
> + struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
> + int ret;
> + ret = mem_cgroup_force_empty(mem);
> + if (!ret)
> +  ret = nbytes;
> + return ret;
> +}
> +
> +/*
> + * Note: This should be removed if cgroup supports write-only file.
> + */
> +
> +static ssize_t mem_force_empty_read(struct cgroup *cont,
> +    struct cftype *cft,
> +    struct file *file, char __user *userbuf,
> +    size_t nbytes, loff_t *ppos)
> +{
> + return -EINVAL;
> +}
> +
> +
>  static struct cftype mem_cgroup_files[] = {
>   {
>    .name = "usage_in_bytes",
> @@ -650,6 +741,11 @@ static struct cftype mem_cgroup_files[]
>    .write = mem_control_type_write,
>    .read = mem_control_type_read,
>   },
> + {
> +  .name = "force_empty",
> +  .write = mem_force_empty_write,
> +  .read = mem_force_empty_read,
> + },
>  };
>
>  static struct mem_cgroup init_mem_cgroup;
>
```

--
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 cgroup enhancements updated [3/10] remember pagecache
Posted by Balbir Singh on Wed, 24 Oct 2007 13:56:34 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> Add PCGF_PAGECACHE flag to page_cgroup to remember "this page is
> charged as page-cache."
> This is very useful for implementing precise accounting in memory cgroup.
>
> Changelog v1 -> v2
> - moved #define to out-side of struct definition
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>
>
> mm/memcontrol.c | 18 ++++++++++++++---
> 1 file changed, 15 insertions(+), 3 deletions(-)
>
> Index: devel-2.6.23-mm1/mm/memcontrol.c
> ===
> --- devel-2.6.23-mm1.orig/mm/memcontrol.c
> +++ devel-2.6.23-mm1/mm/memcontrol.c
> @@ -83,7 +83,9 @@ 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 (0x1) /* charged as page-cache */
>
> enum {
> MEM_CGROUP_TYPE_UNSPEC = 0,
> @@ -315,8 +317,8 @@ 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)
> +static int mem_cgroup_charge_common(struct page *page, struct mm_struct *mm,
> +   gfp_t gfp_mask, int is_cache)
> {
>   struct mem_cgroup *mem;
>   struct page_cgroup *pc;
> @@ -418,6 +420,10 @@ noreclaim:
>   atomic_set(&pc->ref_cnt, 1);
>   pc->mem_cgroup = mem;
>   pc->page = page;
> + if (is_cache)
> + pc->flags = PCGF_PAGECACHE;

I prefer PAGE_CGROUP_CACHE since cache can be page cache/swap cache.

> + else
> + pc->flags = 0;
>   if (page_cgroup_assign_new_page_cgroup(page, pc)) {
>   /*
>   * an another charge is added to this page already.
> @@ -442,6 +448,12 @@ 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);

Could we define

```
enum {
 MEM_CGROUP_CHARGE_TYPE_CACHE  = 0,
 MEM_CGROUP_CHARGE_TYPE_MAPPED = 1,
};
```

and use the enums here and below.

> +}
> +
> /*
>   * See if the cached pages should be charged at all?
>   */
> @@ -454,7 +466,7 @@ 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;
> }
>

--
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 cgroup enhancements updated [8/10] add pre_destroy handler
Posted by Balbir Singh on Wed, 24 Oct 2007 14:49:34 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> My main purpose of this patch is for memory controller..
>
> This patch adds a handler "pre_destroy" to cgroup_subsys.
> It is called before cgroup_rmdir() checks all subsys's refcnt.
>
> I think this is useful for subsyses which have some extra refs
> even if there are no tasks in cgroup.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
> include/linux/cgroup.h | 1 +
> kernel/cgroup.c | 7 +++++++
> 2 files changed, 8 insertions(+)
>
> Index: devel-2.6.23-mm1/include/linux/cgroup.h
> ===
> --- devel-2.6.23-mm1.orig/include/linux/cgroup.h
> +++ devel-2.6.23-mm1/include/linux/cgroup.h
> @@ -233,6 +233,7 @@ int cgroup_is_descendant(const struct cg
> struct cgroup_subsys {
>   struct cgroup_subsys_state *(*create)(struct cgroup_subsys *ss,
>   struct cgroup *cont);

> + void (*pre_destroy)(struct cgroup_subsys *ss, struct cgroup *cont);
>   void (*destroy)(struct cgroup_subsys *ss, struct cgroup *cont);
>   int (*can_attach)(struct cgroup_subsys *ss,
>   struct cgroup *cont, struct task_struct *tsk);
> Index: devel-2.6.23-mm1/kernel/cgroup.c
> ===
> --- devel-2.6.23-mm1.orig/kernel/cgroup.c
> +++ devel-2.6.23-mm1/kernel/cgroup.c
> @@ -2158,6 +2158,13 @@ static int cgroup_rmdir(struct inode *un
>   parent = cont->parent;
>   root = cont->root;
>   sb = root->sb;
> + /*
> + * Notify subsyses that rmdir() request comes.
> + */
> + for_each_subsys(root, ss) {
> + if ((cont->subsys[ss->subsys_id]) && ss->pre_destroy)
> +   ss->pre_destroy(ss, cont);
> + }
>

Is pre_destroy really required? Can't we do what we do here in destroy?

>   if (cgroup_has_css_refs(cont)) {
>   mutex_unlock(&cgroup_mutex);
>

--
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 cgroup enhancements updated [10/10] NUMA aware account
Posted by Balbir Singh on Wed, 24 Oct 2007 14:59:08 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> This patch is a trial for making stat for memory_cgroup NUMA-aware.
>
> * dividing per-cpu stat to handle nid information.

> * add nid information to page_cgroup
> * Because init routine has to call kmalloc, init_early is set to be 0.
>
> This is just a trial at this stage. Any comments are welcome.
> Works well on my fake NUMA system.
> I think we can add "numastat" based on this.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
>
> mm/memcontrol.c | 97 ++----------
> 1 file changed, 71 insertions(+), 26 deletions(-)
>
> Index: devel-2.6.23-mm1/mm/memcontrol.c
> ===
> --- devel-2.6.23-mm1.orig/mm/memcontrol.c
> +++ devel-2.6.23-mm1/mm/memcontrol.c
> @@ -29,6 +29,7 @@
> #include <linux/spinlock.h>
> #include <linux/fs.h>
> #include <linux/seq_file.h>
> +#include <linux/vmalloc.h>
>
> #include <asm/uaccess.h>
>
> @@ -59,12 +60,18 @@ enum mem_cgroup_stat_index {
>   MEM_CGROUP_STAT_NSTATS,
> };
>
> +#ifndef CONFIG_NUMA
> struct mem_cgroup_stat_cpu {
> - s64 count[MEM_CGROUP_STAT_NSTATS];
> + s64 count[1][MEM_CGROUP_STAT_NSTATS];
> } ____cacheline_aligned_in_smp;
> +#else
> +struct mem_cgroup_stat_cpu {
> + s64 count[MAX_NUMNODES][MEM_CGROUP_STAT_NSTATS];
> +} ____cacheline_aligned_in_smp;
> +#endif
>
> struct mem_cgroup_stat {
> - struct mem_cgroup_stat_cpu cpustat[NR_CPUS];
> + struct mem_cgroup_stat_cpu *cpustat[NR_CPUS];
> };
>
> /*
> @@ -72,28 +79,28 @@ struct mem_cgroup_stat {
>   * MUST be called under preempt_disable().

```
>   */
>  static inline void __mem_cgroup_stat_add(struct mem_cgroup_stat *stat,
> -                enum mem_cgroup_stat_index idx, int val)
> +  enum mem_cgroup_stat_index idx, int nid, int val)
>  {
>   int cpu = smp_processor_id();
>  #ifdef CONFIG_PREEMPT
>   VM_BUG_ON(preempt_count() == 0);
>  #endif
> - stat->cpustat[cpu].count[idx] += val;
> + stat->cpustat[cpu]->count[nid][idx] += val;
>  }
>
>  static inline void mem_cgroup_stat_inc(struct mem_cgroup_stat *stat,
> -  enum mem_cgroup_stat_index idx)
> +  enum mem_cgroup_stat_index idx, int nid)
>  {
>   preempt_disable();
> - __mem_cgroup_stat_add(stat, idx, 1);
> + __mem_cgroup_stat_add(stat, idx, nid, 1);
>   preempt_enable();
>  }
>
>  static inline void mem_cgroup_stat_dec(struct mem_cgroup_stat *stat,
> -  enum mem_cgroup_stat_index idx)
> +  enum mem_cgroup_stat_index idx, int nid)
>  {
>   preempt_disable();
> - __mem_cgroup_stat_add(stat, idx, -1);
> + __mem_cgroup_stat_add(stat, idx, nid, -1);
>   preempt_enable();
>  }
>
> @@ -149,6 +156,7 @@ struct page_cgroup {
>   struct list_head lru;  /* per cgroup LRU list */
>   struct page *page;
>   struct mem_cgroup *mem_cgroup;
> + int nid;
>   atomic_t ref_cnt;  /* Helpful when pages move b/w  */
>       /* mapped and cached states     */
>   int  flags;
> @@ -169,21 +177,23 @@ enum {
>   * We have to modify several values at charge/uncharge..
>   */
>  static inline void
> -mem_cgroup_charge_statistics(struct mem_cgroup *mem, int flags, int charge)
> +mem_cgroup_charge_statistics(struct mem_cgroup *mem, int nid,
> +    int flags, int charge)
```

```
>  {
>   int val = (charge)? 1 : -1;
>   struct mem_cgroup_stat *stat = &mem->stat;
>   preempt_disable();
>
>   if (flags & PCGF_PAGECACHE)
> -  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_PAGECACHE, val);
> +  __mem_cgroup_stat_add(stat,
> +    MEM_CGROUP_STAT_PAGECACHE, nid, val);
>   else
> -  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_RSS, val);
> +  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_RSS, nid, val);
>
>   if (flags & PCGF_ACTIVE)
> -  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE, val);
> +  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE, nid, val);
>   else
> -  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE, val);
> +  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE, nid, val);
>
>   preempt_enable();
>  }
> @@ -315,8 +325,10 @@ static void __mem_cgroup_move_lists(stru
>   if (moved) {
>    struct mem_cgroup_stat *stat = &mem->stat;
>    preempt_disable();
> -  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE, moved);
> -  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE, -moved);
> +  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE,
> +       pc->nid, moved);
> +  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE,
> +     pc->nid, -moved);
>    preempt_enable();
>   }
>   if (active) {
> @@ -493,10 +505,10 @@ retry:
>    bool is_atomic = gfp_mask & GFP_ATOMIC;
>    if (is_cache)
>     mem_cgroup_stat_inc(&mem->stat,
> -    MEM_CGROUP_STAT_FAIL_CACHE);
> +    MEM_CGROUP_STAT_FAIL_CACHE, page_to_nid(page));
>    else
>     mem_cgroup_stat_inc(&mem->stat,
> -    MEM_CGROUP_STAT_FAIL_RSS);
> +    MEM_CGROUP_STAT_FAIL_RSS, page_to_nid(page));
>    /*
>     * We cannot reclaim under GFP_ATOMIC, fail the charge
>     */
```

```
> @@ -537,6 +549,7 @@ noreclaim:
>   atomic_set(&pc->ref_cnt, 1);
>   pc->mem_cgroup = mem;
>   pc->page = page;
> + pc->nid = page_to_nid(page);
>   if (is_cache)
>    pc->flags = PCGF_PAGECACHE | PCGF_ACTIVE;
>   else
> @@ -554,7 +567,7 @@ noreclaim:
>   }
>
>   /* Update statistics vector */
> - mem_cgroup_charge_statistics(mem, pc->flags, true);
> + mem_cgroup_charge_statistics(mem, pc->nid, pc->flags, true);
>
>   spin_lock_irqsave(&mem->lru_lock, flags);
>   list_add(&pc->lru, &mem->active_list);
> @@ -621,7 +634,8 @@ void mem_cgroup_uncharge(struct page_cgr
>     spin_lock_irqsave(&mem->lru_lock, flags);
>     list_del_init(&pc->lru);
>     spin_unlock_irqrestore(&mem->lru_lock, flags);
> -   mem_cgroup_charge_statistics(mem, pc->flags, false);
> +   mem_cgroup_charge_statistics(mem, pc->nid, pc->flags,
> +       false);
>     kfree(pc);
>    }
>   }
> @@ -657,6 +671,7 @@ void mem_cgroup_end_migration(struct pag
>  void mem_cgroup_page_migration(struct page *page, struct page *newpage)
>  {
>   struct page_cgroup *pc;
> + struct mem_cgroup *mem;
>  retry:
>   pc = page_get_page_cgroup(page);
>   if (!pc)
> @@ -664,6 +679,11 @@ retry:
>   if (clear_page_cgroup(page, pc) != pc)
>    goto retry;
>   pc->page = newpage;
> + pc->nid = page_to_nid(newpage);
> + mem = pc->mem_cgroup;
> + mem_cgroup_charge_statistics(mem, page_to_nid(page), pc->flags, false);
> + mem_cgroup_charge_statistics(mem,
> +     page_to_nid(newpage), pc->flags, true);
>   lock_page_cgroup(newpage);
>   page_assign_page_cgroup(newpage, pc);
>   unlock_page_cgroup(newpage);
> @@ -697,7 +717,8 @@ retry:
```

```
>     css_put(&mem->css);
>     res_counter_uncharge(&mem->res, PAGE_SIZE);
>     list_del_init(&pc->lru);
> -   mem_cgroup_charge_statistics(mem, pc->flags, false);
> +    mem_cgroup_charge_statistics(mem, pc->flags, pc->nid,
> +      false);
>     kfree(pc);
>    } else  /* being uncharged ? ...do relax */
>     break;
> @@ -872,13 +893,16 @@ static int mem_control_stat_show(struct
>   struct mem_cgroup_stat *stat = &mem_cont->stat;
>   int i;
>
> - for (i = 0; i < ARRAY_SIZE(stat->cpustat[0].count); i++) {
> + for (i = 0; i < MEM_CGROUP_STAT_NSTATS; i++) {
>    unsigned int cpu;
> +  int node;
>    s64 val;
>
>    val = 0;
> -  for (cpu = 0; cpu < NR_CPUS; cpu++)
> -   val += stat->cpustat[cpu].count[i];
> +  for_each_possible_cpu(cpu)
> +   for_each_node_state(node, N_POSSIBLE)
> +    val += stat->cpustat[cpu]->count[node][i];
> +
>    val *= mem_cgroup_stat_desc[i].unit;
>    seq_printf(m, "%s %lld\n", mem_cgroup_stat_desc[i].msg, val);
>   }
> @@ -941,12 +965,14 @@ static struct cgroup_subsys_state *
>  mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
>  {
>   struct mem_cgroup *mem;
> + int cpu;
>
>   if (unlikely((cont->parent) == NULL)) {
>    mem = &init_mem_cgroup;
>    init_mm.mem_cgroup = mem;
> - } else
> + } else {
>    mem = kzalloc(sizeof(struct mem_cgroup), GFP_KERNEL);
> + }
>
>   if (mem == NULL)
>    return NULL;
> @@ -956,6 +982,17 @@ 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;
> +
> + for_each_possible_cpu(cpu) {
> + int nid = cpu_to_node(cpu);
> + struct mem_cgroup_stat_cpu *mcsc;
> + if (sizeof(*mcsc) < PAGE_SIZE)
> +   mcsc = kmalloc_node(sizeof(*mcsc), GFP_KERNEL, nid);
> + else
> +   mcsc = vmalloc_node(sizeof(*mcsc), nid);

Do we need to use the vmalloc() pool? I think we might be better off
using a dedicated slab for ourselves

> + memset(mcsc, 0, sizeof(*mcsc));
> + mem->stat.cpustat[cpu] = mcsc;
> + }
>   return &mem->css;
> }
>
> @@ -969,7 +1006,15 @@ static void mem_cgroup_pre_destroy(struc
> static void mem_cgroup_destroy(struct cgroup_subsys *ss,
>   struct cgroup *cont)
> {
> - kfree(mem_cgroup_from_cont(cont));
> + struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
> + int cpu;
> + for_each_possible_cpu(cpu) {
> + if (sizeof(struct mem_cgroup_stat_cpu) < PAGE_SIZE)
> +   kfree(mem->stat.cpustat[cpu]);
> + else
> +   vfree(mem->stat.cpustat[cpu]);
> + }
> + kfree(mem);
> }
>
> static int mem_cgroup_populate(struct cgroup_subsys *ss,
> @@ -1021,5 +1066,5 @@ struct cgroup_subsys mem_cgroup_subsys =
>   .destroy = mem_cgroup_destroy,
>   .populate = mem_cgroup_populate,
>   .attach = mem_cgroup_move_task,
> - .early_init = 1,
> + .early_init = 0,

I don't understand why this change is required here?

> };
>

--
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 cgroup enhancements updated [8/10] add pre_destroy handler
Posted by KAMEZAWA Hiroyuki on Wed, 24 Oct 2007 15:46:14 GMT
View Forum Message <> Reply to Message

On Wed, 24 Oct 2007 20:19:34 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> > + /*
> > + * Notify subsyses that rmdir() request comes.
> > + */
> > + for_each_subsys(root, ss) {
> > + if ((cont->subsys[ss->subsys_id]) && ss->pre_destroy)
> > +   ss->pre_destroy(ss, cont);
> > + }
> >
>
> Is pre_destroy really required? Can't we do what we do here in destroy?
>

I think keeping "we can destroy cgroup only when refcnt=0" semantics is better.

Thanks,
-Kame

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

---

Subject: Re: [RFC][PATCH] memory cgroup enhancements updated [3/10] remember pagecache
Posted by KAMEZAWA Hiroyuki on Wed, 24 Oct 2007 15:47:24 GMT
View Forum Message <> Reply to Message

On Wed, 24 Oct 2007 19:26:34 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> Could we define
>
> enum {
> MEM_CGROUP_CHARGE_TYPE_CACHE = 0,
> MEM_CGROUP_CHARGE_TYPE_MAPPED = 1,
> };
>
> and use the enums here and below.
>
Okay, I'll use this approach.

Thanks,
-Kame

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

---

Subject: Re: [RFC][PATCH] memory cgroup enhancements updated [10/10] NUMA aware account
Posted by KAMEZAWA Hiroyuki on Wed, 24 Oct 2007 15:53:04 GMT
View Forum Message <> Reply to Message

On Wed, 24 Oct 2007 20:29:08 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> > + for_each_possible_cpu(cpu) {
> > + int nid = cpu_to_node(cpu);
> > + struct mem_cgroup_stat_cpu *mcsc;
> > + if (sizeof(*mcsc) < PAGE_SIZE)
> > +   mcsc = kmalloc_node(sizeof(*mcsc), GFP_KERNEL, nid);
> > + else
> > +   mcsc = vmalloc_node(sizeof(*mcsc), nid);
>
> Do we need to use the vmalloc() pool? I think we might be better off
> using a dedicated slab for ourselves
>
I admit this part is complicated. But ia64's MAX_NUMNODES=1024 and stat
can be increased. we need vmalloc. I'll rewrite this part to be
better looking.

> > + memset(mcsc, 0, sizeof(*mcsc));
> > + mem->stat.cpustat[cpu] = mcsc;
> > + }

> >   return &mem->css;
> > }
> >
> > @@ -969,7 +1006,15 @@ static void mem_cgroup_pre_destroy(struc
> > static void mem_cgroup_destroy(struct cgroup_subsys *ss,
> >   struct cgroup *cont)
> > {
> > - kfree(mem_cgroup_from_cont(cont));
> > + struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
> > + int cpu;
> > + for_each_possible_cpu(cpu) {
> > + if (sizeof(struct mem_cgroup_stat_cpu) < PAGE_SIZE)
> > +   kfree(mem->stat.cpustat[cpu]);
> > + else
> > +   vfree(mem->stat.cpustat[cpu]);
> > + }
> > + kfree(mem);
> > }
> >
> > static int mem_cgroup_populate(struct cgroup_subsys *ss,
> > @@ -1021,5 +1066,5 @@ struct cgroup_subsys mem_cgroup_subsys =
> >   .destroy = mem_cgroup_destroy,
> >   .populate = mem_cgroup_populate,
> >   .attach = mem_cgroup_move_task,
> > - .early_init = 1,
> > + .early_init = 0,
>
> I don't understand why this change is required here?
>
If early_init = 1, we cannot call kmalloc/vmalloc at initializing init_mem_cgroup.
It's too early.

Thanks,
-Kame

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