Subject: [PATCH][RFC] dirty balancing for cgroups
Posted by yamamoto on Wed, 09 Jul 2008 06:00:34 GMT
View Forum Message <> Reply to Message

hi,

the following patch is a simple implementation of
dirty balancing for cgroups. any comments?

it depends on the following fix:
 http://lkml.org/lkml/2008/7/8/428

YAMAMOTO Takashi

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

```
diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
index 23c02e2..f5453cc 100644
--- a/include/linux/cgroup_subsys.h
+++ b/include/linux/cgroup_subsys.h
@@ -52,3 +52,9 @@ SUBSYS(memrlimit_cgroup)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR
+SUBSYS(memdirtylimit_cgroup)
+#endif
+
+/* */
diff --git a/include/linux/memdirtylimitcgroup.h b/include/linux/memdirtylimitcgroup.h
new file mode 100644
index 0000000..667d312
--- /dev/null
+++ b/include/linux/memdirtylimitcgroup.h
@@ -0,0 +1,47 @@
+
+/*
+ * memdirtylimitcgroup.h COPYRIGHT FUJITSU LIMITED 2008
+ *
+ * Author: yamamoto@valinux.co.jp
+ */
+
+struct task_struct;
+
+#if defined(CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR)
```

```
+
+void memdirtylimitcgroup_dirty_inc(struct task_struct *);
+void memdirtylimitcgroup_dirty_limit(struct task_struct *, long *);
+void memdirtylimitcgroup_change_shift(int);
+void memdirtylimitcgroup_init(int);
+
+#else /* defined(CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR) */
+
+static inline void
+memdirtylimitcgroup_dirty_inc(struct task_struct *t)
+{
+
+ /* nothing */
+}
+
+static inline void
+memdirtylimitcgroup_dirty_limit(struct task_struct *t, long *dirtyp)
+{
+
+ /* nothing */
+}
+
+static inline void
+memdirtylimitcgroup_change_shift(int shift)
+{
+
+ /* nothing */
+}
+
+static inline void
+memdirtylimitcgroup_init(int shift)
+{
+
+ /* nothing */
+}
+
+#endif /* defined(CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR) */
diff --git a/init/Kconfig b/init/Kconfig
index 162d462..985bac8 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -418,6 +418,12 @@ config CGROUP_MEMRLIMIT_CTLR
    memory RSS and Page Cache control. Virtual address space control
    is provided by this controller.

+config CGROUP_MEMDIRTYLIMIT_CTLR
+ bool "Memory Dirty Limit Controller for Control Groups"
+ depends on CGROUPS && RESOURCE_COUNTERS
```

```
+ help
+   XXX TBD
+
 config SYSFS_DEPRECATED
  bool

diff --git a/mm/Makefile b/mm/Makefile
index f54232d..8603d19 100644
--- a/mm/Makefile
+++ b/mm/Makefile
@@ -35,4 +35,5 @@ obj-$(CONFIG_SMP) += allocpercpu.o
 obj-$(CONFIG_QUICKLIST) += quicklist.o
 obj-$(CONFIG_CGROUP_MEM_RES_CTLR) += memcontrol.o
 obj-$(CONFIG_CGROUP_MEMRLIMIT_CTLR) += memrlimitcgroup.o
+obj-$(CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR) += memdirtylimitcgroup.o
 obj-$(CONFIG_MMU_NOTIFIER) += mmu_notifier.o
diff --git a/mm/memdirtylimitcgroup.c b/mm/memdirtylimitcgroup.c
new file mode 100644
index 0000000..b70b33d
--- /dev/null
+++ b/mm/memdirtylimitcgroup.c
@@ -0,0 +1,179 @@
+
+/*
+ * memdirtylimitcgroup.c COPYRIGHT FUJITSU LIMITED 2008
+ *
+ * Author: yamamoto@valinux.co.jp
+ */
+
+#include <linux/err.h>
+#include <linux/cgroup.h>
+#include <linux/rcupdate.h>
+#include <linux/slab.h>
+#include <linux/memdirtylimitcgroup.h>
+
+#include <asm/div64.h>
+
+static struct prop_descriptor vm_cgroup_dirties;
+
+struct memdirtylimit_cgroup {
+ struct cgroup_subsys_state dlcg_css;
+ spinlock_t dlcg_lock;
+ struct prop_local_single dlcg_dirties;
+};
+
+static struct cgroup_subsys_state *
+task_to_css(struct task_struct *task)
+{
```

```
+
+ return task_subsys_state(task, memdirtylimit_cgroup_subsys_id);
+}
+
+static struct memdirtylimit_cgroup *
+css_to_dlcg(struct cgroup_subsys_state *css)
+{
+
+ return container_of(css, struct memdirtylimit_cgroup, dlcg_css);
+}
+
+static struct cgroup_subsys_state *
+cg_to_css(struct cgroup *cg)
+{
+
+ return cgroup_subsys_state(cg, memdirtylimit_cgroup_subsys_id);
+}
+
+static struct memdirtylimit_cgroup *
+cg_to_dlcg(struct cgroup *cg)
+{
+
+ return css_to_dlcg(cg_to_css(cg));
+}
+
+/* ---------------------------------------- */
+
+static void
+getfraction(struct memdirtylimit_cgroup *dlcg, long *numeratorp,
+    long *denominatorp)
+{
+
+ spin_lock(&dlcg->dlcg_lock);
+ prop_fraction_single(&vm_cgroup_dirties, &dlcg->dlcg_dirties,
+    numeratorp, denominatorp);
+ spin_unlock(&dlcg->dlcg_lock);
+}
+
+/* ---------------------------------------- */
+
+void
+memdirtylimitcgroup_dirty_inc(struct task_struct *t)
+{
+ struct memdirtylimit_cgroup *dlcg;
+
+ rcu_read_lock();
+ dlcg = css_to_dlcg(task_to_css(t));
+ spin_lock(&dlcg->dlcg_lock);
```

```
+ prop_inc_single(&vm_cgroup_dirties, &dlcg->dlcg_dirties);
+ spin_unlock(&dlcg->dlcg_lock);
+ rcu_read_unlock();
+}
+
+void
+memdirtylimitcgroup_dirty_limit(struct task_struct *t, long *dirtyp)
+{
+ struct memdirtylimit_cgroup *dlcg;
+ unsigned long dirty = *dirtyp;
+ uint64_t tmp;
+ long numerator;
+ long denominator;
+
+ BUG_ON(*dirtyp < 0);
+
+ rcu_read_lock();
+ dlcg = css_to_dlcg(task_to_css(t));
+ getfraction(dlcg, &numerator, &denominator);
+ rcu_read_unlock();
+
+ tmp = (uint64_t)(dirty >> 1) * numerator;
+ do_div(tmp, denominator);
+ *dirtyp = dirty - (unsigned long)tmp;
+}
+
+void
+memdirtylimitcgroup_change_shift(int shift)
+{
+
+ prop_change_shift(&vm_cgroup_dirties, shift);
+}
+
+void
+memdirtylimitcgroup_init(int shift)
+{
+
+ prop_descriptor_init(&vm_cgroup_dirties, shift);
+}
+
+/* ---------------------------------------- */
+
+static u64
+memdirtylimit_cgroup_read_fraction(struct cgroup *cg, struct cftype *cft)
+{
+ struct memdirtylimit_cgroup *dlcg;
+ uint64_t result;
+ long numerator;
```

```
+ long denominator;
+
+ dlcg = cg_to_dlcg(cg);
+ getfraction(dlcg, &numerator, &denominator);
+ result = (uint64_t)100 * numerator;
+ do_div(result, denominator);
+ return result;
+}
+
+static const struct cftype files[] = {
+ {
+  .name = "fraction",
+  .read_u64 = memdirtylimit_cgroup_read_fraction,
+ },
+};
+
+static int
+memdirtylimit_cgroup_populate(struct cgroup_subsys *ss, struct cgroup *cg)
+{
+
+ return cgroup_add_files(cg, ss, files, ARRAY_SIZE(files));
+}
+
+static struct cgroup_subsys_state *
+memdirtylimit_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cg)
+{
+ struct memdirtylimit_cgroup *dlcg;
+ int error;
+
+ dlcg = kzalloc(sizeof(*dlcg), GFP_KERNEL);
+ if (dlcg == NULL)
+  return ERR_PTR(-ENOMEM);
+ error = prop_local_init_single(&dlcg->dlcg_dirties);
+ if (error != 0) {
+  kfree(dlcg);
+  return ERR_PTR(error);
+ }
+ spin_lock_init(&dlcg->dlcg_lock);
+ return &dlcg->dlcg_css;
+}
+
+static void
+memdirtylimit_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cg)
+{
+ struct memdirtylimit_cgroup *dlcg = cg_to_dlcg(cg);
+
+ prop_local_destroy_single(&dlcg->dlcg_dirties);
+ kfree(dlcg);
```

```
+}
+
+struct cgroup_subsys memdirtylimit_cgroup_subsys = {
+ .name = "memdirtylimit",
+ .subsys_id = memdirtylimit_cgroup_subsys_id,
+ .create = memdirtylimit_cgroup_create,
+ .destroy = memdirtylimit_cgroup_destroy,
+ .populate = memdirtylimit_cgroup_populate,
+};
diff --git a/mm/page-writeback.c b/mm/page-writeback.c
index e6fa69e..f971532 100644
--- a/mm/page-writeback.c
+++ b/mm/page-writeback.c
@@ -34,6 +34,7 @@
 #include <linux/syscalls.h>
 #include <linux/buffer_head.h>
 #include <linux/pagevec.h>
+#include <linux/memdirtylimitcgroup.h>

 /*
  * The maximum number of pages to writeout in a single bdflush/kupdate
@@ -152,6 +153,7 @@ int dirty_ratio_handler(struct ctl_table *table, int write,
   int shift = calc_period_shift();
   prop_change_shift(&vm_completions, shift);
   prop_change_shift(&vm_dirties, shift);
+  memdirtylimitcgroup_change_shift(shift);
  }
  return ret;
 }
@@ -393,6 +395,8 @@ get_dirty_limits(long *pbackground, long *pdirty, long *pbdi_dirty,
  if (bdi) {
   u64 bdi_dirty;
   long numerator, denominator;
+  long task_dirty;
+  long cgroup_dirty;

   /*
    * Calculate this BDI's share of the dirty ratio.
@@ -408,7 +412,11 @@ get_dirty_limits(long *pbackground, long *pdirty, long *pbdi_dirty,

   *pbdi_dirty = bdi_dirty;
   clip_bdi_dirty_limit(bdi, dirty, pbdi_dirty);
-  task_dirty_limit(current, pbdi_dirty);
+  task_dirty = *pbdi_dirty;
+  task_dirty_limit(current, &task_dirty);
+  cgroup_dirty = *pbdi_dirty;
+  memdirtylimitcgroup_dirty_limit(current, &cgroup_dirty);
+  *pbdi_dirty = min(task_dirty, cgroup_dirty);
```

```
 }
}

@@ -842,6 +850,7 @@ void __init page_writeback_init(void)
  shift = calc_period_shift();
  prop_descriptor_init(&vm_completions, shift);
  prop_descriptor_init(&vm_dirties, shift);
+ memdirtylimitcgroup_init(shift);
 }

 /**
@@ -1105,6 +1114,7 @@ int __set_page_dirty_nobuffers(struct page *page)
  }

  task_dirty_inc(current);
+ memdirtylimitcgroup_dirty_inc(current);

  return 1;
 }
```

Subject: Re: [PATCH][RFC] dirty balancing for cgroups
Posted by KAMEZAWA Hiroyuki on Thu, 10 Jul 2008 23:50:47 GMT
View Forum Message <> Reply to Message

On Wed, 9 Jul 2008 15:00:34 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> hi,
>
> the following patch is a simple implementation of
> dirty balancing for cgroups. any comments?
>
> it depends on the following fix:
> http://lkml.org/lkml/2008/7/8/428
>

A few comments ;)

- This looks simple but, could you merge this into memory resource controller ?
  (if conflict, I'll queue on my stack.)
- Do you have some number ? or How we can test this works well ?
- please CC to linux-mm.

Thanks,
-Kame

> YAMAMOTO Takashi
>
>
> Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>
> ---
>
> diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
> index 23c02e2..f5453cc 100644
> --- a/include/linux/cgroup_subsys.h
> +++ b/include/linux/cgroup_subsys.h
> @@ -52,3 +52,9 @@ SUBSYS(memrlimit_cgroup)
> #endif
>
> /* */
> +
> +#ifdef CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR
> +SUBSYS(memdirtylimit_cgroup)
> +#endif
> +
> +/* */
> diff --git a/include/linux/memdirtylimitcgroup.h b/include/linux/memdirtylimitcgroup.h
> new file mode 100644
> index 0000000..667d312
> --- /dev/null
> +++ b/include/linux/memdirtylimitcgroup.h
> @@ -0,0 +1,47 @@
> +
> +/*
> + * memdirtylimitcgroup.h COPYRIGHT FUJITSU LIMITED 2008
> + *
> + * Author: yamamoto@valinux.co.jp
> + */
> +
> +struct task_struct;
> +
> +#if defined(CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR)
> +
> +void memdirtylimitcgroup_dirty_inc(struct task_struct *);
> +void memdirtylimitcgroup_dirty_limit(struct task_struct *, long *);
> +void memdirtylimitcgroup_change_shift(int);
> +void memdirtylimitcgroup_init(int);
> +
> +#else /* defined(CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR) */
> +
> +static inline void

> +memdirtylimitcgroup_dirty_inc(struct task_struct *t)
> +{
> +
> + /* nothing */
> +}
> +
> +static inline void
> +memdirtylimitcgroup_dirty_limit(struct task_struct *t, long *dirtyp)
> +{
> +
> + /* nothing */
> +}
> +
> +static inline void
> +memdirtylimitcgroup_change_shift(int shift)
> +{
> +
> + /* nothing */
> +}
> +
> +static inline void
> +memdirtylimitcgroup_init(int shift)
> +{
> +
> + /* nothing */
> +}
> +
> +#endif /* defined(CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR) */
> diff --git a/init/Kconfig b/init/Kconfig
> index 162d462..985bac8 100644
> --- a/init/Kconfig
> +++ b/init/Kconfig
> @@ -418,6 +418,12 @@ config CGROUP_MEMRLIMIT_CTLR
> memory RSS and Page Cache control. Virtual address space control
> is provided by this controller.
>
> +config CGROUP_MEMDIRTYLIMIT_CTLR
> + bool "Memory Dirty Limit Controller for Control Groups"
> + depends on CGROUPS && RESOURCE_COUNTERS
> + help
> + XXX TBD
> +
> config SYSFS_DEPRECATED
> bool
>
> diff --git a/mm/Makefile b/mm/Makefile
> index f54232d..8603d19 100644
> --- a/mm/Makefile

> +++ b/mm/Makefile
> @@ -35,4 +35,5 @@ obj-$(CONFIG_SMP) += allocpercpu.o
> obj-$(CONFIG_QUICKLIST) += quicklist.o
> obj-$(CONFIG_CGROUP_MEM_RES_CTLR) += memcontrol.o
> obj-$(CONFIG_CGROUP_MEMRLIMIT_CTLR) += memrlimitcgroup.o
> +obj-$(CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR) += memdirtylimitcgroup.o
> obj-$(CONFIG_MMU_NOTIFIER) += mmu_notifier.o
> diff --git a/mm/memdirtylimitcgroup.c b/mm/memdirtylimitcgroup.c
> new file mode 100644
> index 0000000..b70b33d
> --- /dev/null
> +++ b/mm/memdirtylimitcgroup.c
> @@ -0,0 +1,179 @@
> +
> +/*
> + * memdirtylimitcgroup.c COPYRIGHT FUJITSU LIMITED 2008
> + *
> + * Author: yamamoto@valinux.co.jp
> + */
> +
> +#include <linux/err.h>
> +#include <linux/cgroup.h>
> +#include <linux/rcupdate.h>
> +#include <linux/slab.h>
> +#include <linux/memdirtylimitcgroup.h>
> +
> +#include <asm/div64.h>
> +
> +static struct prop_descriptor vm_cgroup_dirties;
> +
> +struct memdirtylimit_cgroup {
> + struct cgroup_subsys_state dlcg_css;
> + spinlock_t dlcg_lock;
> + struct prop_local_single dlcg_dirties;
> +};
> +
> +static struct cgroup_subsys_state *
> +task_to_css(struct task_struct *task)
> +{
> +
> + return task_subsys_state(task, memdirtylimit_cgroup_subsys_id);
> +}
> +
> +static struct memdirtylimit_cgroup *
> +css_to_dlcg(struct cgroup_subsys_state *css)
> +{
> +
> + return container_of(css, struct memdirtylimit_cgroup, dlcg_css);

```
> +}
> +
> +static struct cgroup_subsys_state *
> +cg_to_css(struct cgroup *cg)
> +{
> +
> + return cgroup_subsys_state(cg, memdirtylimit_cgroup_subsys_id);
> +}
> +
> +static struct memdirtylimit_cgroup *
> +cg_to_dlcg(struct cgroup *cg)
> +{
> +
> + return css_to_dlcg(cg_to_css(cg));
> +}
> +
> +/* ---------------------------------------- */
> +
> +static void
> +getfraction(struct memdirtylimit_cgroup *dlcg, long *numeratorp,
> +    long *denominatorp)
> +{
> +
> + spin_lock(&dlcg->dlcg_lock);
> + prop_fraction_single(&vm_cgroup_dirties, &dlcg->dlcg_dirties,
> +    numeratorp, denominatorp);
> + spin_unlock(&dlcg->dlcg_lock);
> +}
> +
> +/* ---------------------------------------- */
> +
> +void
> +memdirtylimitcgroup_dirty_inc(struct task_struct *t)
> +{
> + struct memdirtylimit_cgroup *dlcg;
> +
> + rcu_read_lock();
> + dlcg = css_to_dlcg(task_to_css(t));
> + spin_lock(&dlcg->dlcg_lock);
> + prop_inc_single(&vm_cgroup_dirties, &dlcg->dlcg_dirties);
> + spin_unlock(&dlcg->dlcg_lock);
> + rcu_read_unlock();
> +}
> +
> +void
> +memdirtylimitcgroup_dirty_limit(struct task_struct *t, long *dirtyp)
> +{
> + struct memdirtylimit_cgroup *dlcg;
```

```
> + unsigned long dirty = *dirtyp;
> + uint64_t tmp;
> + long numerator;
> + long denominator;
> +
> + BUG_ON(*dirtyp < 0);
> +
> + rcu_read_lock();
> + dlcg = css_to_dlcg(task_to_css(t));
> + getfraction(dlcg, &numerator, &denominator);
> + rcu_read_unlock();
> +
> + tmp = (uint64_t)(dirty >> 1) * numerator;
> + do_div(tmp, denominator);
> + *dirtyp = dirty - (unsigned long)tmp;
> +}
> +
> +void
> +memdirtylimitcgroup_change_shift(int shift)
> +{
> +
> + prop_change_shift(&vm_cgroup_dirties, shift);
> +}
> +
> +void
> +memdirtylimitcgroup_init(int shift)
> +{
> +
> + prop_descriptor_init(&vm_cgroup_dirties, shift);
> +}
> +
> +/* ---------------------------------------- */
> +
> +static u64
> +memdirtylimit_cgroup_read_fraction(struct cgroup *cg, struct cftype *cft)
> +{
> + struct memdirtylimit_cgroup *dlcg;
> + uint64_t result;
> + long numerator;
> + long denominator;
> +
> + dlcg = cg_to_dlcg(cg);
> + getfraction(dlcg, &numerator, &denominator);
> + result = (uint64_t)100 * numerator;
> + do_div(result, denominator);
> + return result;
> +}
> +
```

```
> +static const struct cftype files[] = {
> + {
> +  .name = "fraction",
> +  .read_u64 = memdirtylimit_cgroup_read_fraction,
> + },
> +};
> +
> +static int
> +memdirtylimit_cgroup_populate(struct cgroup_subsys *ss, struct cgroup *cg)
> +{
> +
> + return cgroup_add_files(cg, ss, files, ARRAY_SIZE(files));
> +}
> +
> +static struct cgroup_subsys_state *
> +memdirtylimit_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cg)
> +{
> + struct memdirtylimit_cgroup *dlcg;
> + int error;
> +
> + dlcg = kzalloc(sizeof(*dlcg), GFP_KERNEL);
> + if (dlcg == NULL)
> +  return ERR_PTR(-ENOMEM);
> + error = prop_local_init_single(&dlcg->dlcg_dirties);
> + if (error != 0) {
> +  kfree(dlcg);
> +  return ERR_PTR(error);
> + }
> + spin_lock_init(&dlcg->dlcg_lock);
> + return &dlcg->dlcg_css;
> +}
> +
> +static void
> +memdirtylimit_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cg)
> +{
> + struct memdirtylimit_cgroup *dlcg = cg_to_dlcg(cg);
> +
> + prop_local_destroy_single(&dlcg->dlcg_dirties);
> + kfree(dlcg);
> +}
> +
> +struct cgroup_subsys memdirtylimit_cgroup_subsys = {
> + .name = "memdirtylimit",
> + .subsys_id = memdirtylimit_cgroup_subsys_id,
> + .create = memdirtylimit_cgroup_create,
> + .destroy = memdirtylimit_cgroup_destroy,
> + .populate = memdirtylimit_cgroup_populate,
> +};
```

> diff --git a/mm/page-writeback.c b/mm/page-writeback.c
> index e6fa69e..f971532 100644
> --- a/mm/page-writeback.c
> +++ b/mm/page-writeback.c
> @@ -34,6 +34,7 @@
> #include <linux/syscalls.h>
> #include <linux/buffer_head.h>
> #include <linux/pagevec.h>
> +#include <linux/memdirtylimitcgroup.h>
>
> /*
>   * The maximum number of pages to writeout in a single bdflush/kupdate
> @@ -152,6 +153,7 @@ int dirty_ratio_handler(struct ctl_table *table, int write,
>   int shift = calc_period_shift();
>   prop_change_shift(&vm_completions, shift);
>   prop_change_shift(&vm_dirties, shift);
> + memdirtylimitcgroup_change_shift(shift);
>   }
>   return ret;
> }
> @@ -393,6 +395,8 @@ get_dirty_limits(long *pbackground, long *pdirty, long *pbdi_dirty,
>   if (bdi) {
>   u64 bdi_dirty;
>   long numerator, denominator;
> + long task_dirty;
> + long cgroup_dirty;
>
>   /*
>   * Calculate this BDI's share of the dirty ratio.
> @@ -408,7 +412,11 @@ get_dirty_limits(long *pbackground, long *pdirty, long *pbdi_dirty,
>
>   *pbdi_dirty = bdi_dirty;
>   clip_bdi_dirty_limit(bdi, dirty, pbdi_dirty);
> - task_dirty_limit(current, pbdi_dirty);
> + task_dirty = *pbdi_dirty;
> + task_dirty_limit(current, &task_dirty);
> + cgroup_dirty = *pbdi_dirty;
> + memdirtylimitcgroup_dirty_limit(current, &cgroup_dirty);
> + *pbdi_dirty = min(task_dirty, cgroup_dirty);
>   }
> }
>
> @@ -842,6 +850,7 @@ void __init page_writeback_init(void)
>   shift = calc_period_shift();
>   prop_descriptor_init(&vm_completions, shift);
>   prop_descriptor_init(&vm_dirties, shift);
> + memdirtylimitcgroup_init(shift);
> }

>
> /**
> @@ -1105,6 +1114,7 @@ int __set_page_dirty_nobuffers(struct page *page)
>   }
>
>   task_dirty_inc(current);
> + memdirtylimitcgroup_dirty_inc(current);
>
>   return 1;
> }
>

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

---

Subject: Re: [PATCH][RFC] dirty balancing for cgroups
Posted by yamamoto on Fri, 11 Jul 2008 04:06:57 GMT
View Forum Message <> Reply to Message

hi,

> On Wed, 9 Jul 2008 15:00:34 +0900 (JST)
> yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
>
> > hi,
> >
> > the following patch is a simple implementation of
> > dirty balancing for cgroups. any comments?
> >
> > it depends on the following fix:
> > http://lkml.org/lkml/2008/7/8/428
> >
>
> A few comments ;)

thanks for comments.

> - This looks simple but, could you merge this into memory resource controller ?

why?

>   (if conflict, I'll queue on my stack.)
> - Do you have some number ? or How we can test this works well ?

i have no numbers yet.

it should work as well as the one for tasks. (ie. i don't know. :)

> - please CC to linux-mm.

ok, i will.

YAMAMOTO Takashi

>
> Thanks,
> -Kame
>
> > YAMAMOTO Takashi
> > >
> > >
> > > Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>
> > > ---
> > >
> > > diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
> > > index 23c02e2..f5453cc 100644
> > > --- a/include/linux/cgroup_subsys.h
> > > +++ b/include/linux/cgroup_subsys.h
> > > @@ -52,3 +52,9 @@ SUBSYS(memrlimit_cgroup)
> > > #endif
> > >
> > > /* */
> > > +
> > > +#ifdef CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR
> > > +SUBSYS(memdirtylimit_cgroup)
> > > +#endif
> > > +
> > > +/* */
> > > diff --git a/include/linux/memdirtylimitcgroup.h b/include/linux/memdirtylimitcgroup.h
> > > new file mode 100644
> > > index 0000000..667d312
> > > --- /dev/null
> > > +++ b/include/linux/memdirtylimitcgroup.h
> > > @@ -0,0 +1,47 @@
> > > +
> > > +/*
> > > + * memdirtylimitcgroup.h COPYRIGHT FUJITSU LIMITED 2008
> > > + *
> > > + * Author: yamamoto@valinux.co.jp
> > > + */
> > > +
> > > +struct task_struct;
> > > +
> > > +#if defined(CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR)

> > +
> > +void memdirtylimitcgroup_dirty_inc(struct task_struct *);
> > +void memdirtylimitcgroup_dirty_limit(struct task_struct *, long *);
> > +void memdirtylimitcgroup_change_shift(int);
> > +void memdirtylimitcgroup_init(int);
> > +
> > +#else /* defined(CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR) */
> > +
> > +static inline void
> > +memdirtylimitcgroup_dirty_inc(struct task_struct *t)
> > +{
> > +
> > + /* nothing */
> > +}
> > +
> > +static inline void
> > +memdirtylimitcgroup_dirty_limit(struct task_struct *t, long *dirtyp)
> > +{
> > +
> > + /* nothing */
> > +}
> > +
> > +static inline void
> > +memdirtylimitcgroup_change_shift(int shift)
> > +{
> > +
> > + /* nothing */
> > +}
> > +
> > +static inline void
> > +memdirtylimitcgroup_init(int shift)
> > +{
> > +
> > + /* nothing */
> > +}
> > +
> > +#endif /* defined(CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR) */
> > diff --git a/init/Kconfig b/init/Kconfig
> > index 162d462..985bac8 100644
> > --- a/init/Kconfig
> > +++ b/init/Kconfig
> > @@ -418,6 +418,12 @@ config CGROUP_MEMRLIMIT_CTLR
> > memory RSS and Page Cache control. Virtual address space control
> > is provided by this controller.
> >
> > +config CGROUP_MEMDIRTYLIMIT_CTLR
> > + bool "Memory Dirty Limit Controller for Control Groups"
> > + depends on CGROUPS && RESOURCE_COUNTERS

> > + help
> > +   XXX TBD
> > +
> > config SYSFS_DEPRECATED
> >   bool
> >
> > diff --git a/mm/Makefile b/mm/Makefile
> > index f54232d..8603d19 100644
> > --- a/mm/Makefile
> > +++ b/mm/Makefile
> > @@ -35,4 +35,5 @@ obj-$(CONFIG_SMP) += allocpercpu.o
> > obj-$(CONFIG_QUICKLIST) += quicklist.o
> > obj-$(CONFIG_CGROUP_MEM_RES_CTLR) += memcontrol.o
> > obj-$(CONFIG_CGROUP_MEMRLIMIT_CTLR) += memrlimitcgroup.o
> > +obj-$(CONFIG_CGROUP_MEMDIRTYLIMIT_CTLR) += memdirtylimitcgroup.o
> > obj-$(CONFIG_MMU_NOTIFIER) += mmu_notifier.o
> > diff --git a/mm/memdirtylimitcgroup.c b/mm/memdirtylimitcgroup.c
> > new file mode 100644
> > index 0000000..b70b33d
> > --- /dev/null
> > +++ b/mm/memdirtylimitcgroup.c
> > @@ -0,0 +1,179 @@
> > +
> > +/*
> > + * memdirtylimitcgroup.c COPYRIGHT FUJITSU LIMITED 2008
> > + *
> > + * Author: yamamoto@valinux.co.jp
> > + */
> > +
> > +#include <linux/err.h>
> > +#include <linux/cgroup.h>
> > +#include <linux/rcupdate.h>
> > +#include <linux/slab.h>
> > +#include <linux/memdirtylimitcgroup.h>
> > +
> > +#include <asm/div64.h>
> > +
> > +static struct prop_descriptor vm_cgroup_dirties;
> > +
> > +struct memdirtylimit_cgroup {
> > + struct cgroup_subsys_state dlcg_css;
> > + spinlock_t dlcg_lock;
> > + struct prop_local_single dlcg_dirties;
> > +};
> > +
> > +static struct cgroup_subsys_state *
> > +task_to_css(struct task_struct *task)
> > +{

```
> > +
> > + return task_subsys_state(task, memdirtylimit_cgroup_subsys_id);
> > +}
> > +
> > +static struct memdirtylimit_cgroup *
> > +css_to_dlcg(struct cgroup_subsys_state *css)
> > +{
> > +
> > + return container_of(css, struct memdirtylimit_cgroup, dlcg_css);
> > +}
> > +
> > +static struct cgroup_subsys_state *
> > +cg_to_css(struct cgroup *cg)
> > +{
> > +
> > + return cgroup_subsys_state(cg, memdirtylimit_cgroup_subsys_id);
> > +}
> > +
> > +static struct memdirtylimit_cgroup *
> > +cg_to_dlcg(struct cgroup *cg)
> > +{
> > +
> > + return css_to_dlcg(cg_to_css(cg));
> > +}
> > +
> > +/* --------------------------------------- */
> > +
> > +static void
> > +getfraction(struct memdirtylimit_cgroup *dlcg, long *numeratorp,
> > +    long *denominatorp)
> > +{
> > +
> > + spin_lock(&dlcg->dlcg_lock);
> > + prop_fraction_single(&vm_cgroup_dirties, &dlcg->dlcg_dirties,
> > +    numeratorp, denominatorp);
> > + spin_unlock(&dlcg->dlcg_lock);
> > +}
> > +
> > +/* --------------------------------------- */
> > +
> > +void
> > +memdirtylimitcgroup_dirty_inc(struct task_struct *t)
> > +{
> > + struct memdirtylimit_cgroup *dlcg;
> > +
> > + rcu_read_lock();
> > + dlcg = css_to_dlcg(task_to_css(t));
> > + spin_lock(&dlcg->dlcg_lock);
```

> > + prop_inc_single(&vm_cgroup_dirties, &dlcg->dlcg_dirties);
> > + spin_unlock(&dlcg->dlcg_lock);
> > + rcu_read_unlock();
> > +}
> > +
> > +void
> > +memdirtylimitcgroup_dirty_limit(struct task_struct *t, long *dirtyp)
> > +{
> > + struct memdirtylimit_cgroup *dlcg;
> > + unsigned long dirty = *dirtyp;
> > + uint64_t tmp;
> > + long numerator;
> > + long denominator;
> > +
> > + BUG_ON(*dirtyp < 0);
> > +
> > + rcu_read_lock();
> > + dlcg = css_to_dlcg(task_to_css(t));
> > + getfraction(dlcg, &numerator, &denominator);
> > + rcu_read_unlock();
> > +
> > + tmp = (uint64_t)(dirty >> 1) * numerator;
> > + do_div(tmp, denominator);
> > + *dirtyp = dirty - (unsigned long)tmp;
> > +}
> > +
> > +void
> > +memdirtylimitcgroup_change_shift(int shift)
> > +{
> > +
> > + prop_change_shift(&vm_cgroup_dirties, shift);
> > +}
> > +
> > +void
> > +memdirtylimitcgroup_init(int shift)
> > +{
> > +
> > + prop_descriptor_init(&vm_cgroup_dirties, shift);
> > +}
> > +
> > +/* -- */
> > +
> > +static u64
> > +memdirtylimit_cgroup_read_fraction(struct cgroup *cg, struct cftype *cft)
> > +{
> > + struct memdirtylimit_cgroup *dlcg;
> > + uint64_t result;
> > + long numerator;

```
> > + long denominator;
> > +
> > + dlcg = cg_to_dlcg(cg);
> > + getfraction(dlcg, &numerator, &denominator);
> > + result = (uint64_t)100 * numerator;
> > + do_div(result, denominator);
> > + return result;
> > +}
> > +
> > +static const struct cftype files[] = {
> > + {
> > +  .name = "fraction",
> > +  .read_u64 = memdirtylimit_cgroup_read_fraction,
> > + },
> > +};
> > +
> > +static int
> > +memdirtylimit_cgroup_populate(struct cgroup_subsys *ss, struct cgroup *cg)
> > +{
> > +
> > + return cgroup_add_files(cg, ss, files, ARRAY_SIZE(files));
> > +}
> > +
> > +static struct cgroup_subsys_state *
> > +memdirtylimit_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cg)
> > +{
> > + struct memdirtylimit_cgroup *dlcg;
> > + int error;
> > +
> > + dlcg = kzalloc(sizeof(*dlcg), GFP_KERNEL);
> > + if (dlcg == NULL)
> > +  return ERR_PTR(-ENOMEM);
> > + error = prop_local_init_single(&dlcg->dlcg_dirties);
> > + if (error != 0) {
> > +  kfree(dlcg);
> > +  return ERR_PTR(error);
> > + }
> > + spin_lock_init(&dlcg->dlcg_lock);
> > + return &dlcg->dlcg_css;
> > +}
> > +
> > +static void
> > +memdirtylimit_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cg)
> > +{
> > + struct memdirtylimit_cgroup *dlcg = cg_to_dlcg(cg);
> > +
> > + prop_local_destroy_single(&dlcg->dlcg_dirties);
> > + kfree(dlcg);
```

> > +}
> > +
> > +struct cgroup_subsys memdirtylimit_cgroup_subsys = {
> > + .name = "memdirtylimit",
> > + .subsys_id = memdirtylimit_cgroup_subsys_id,
> > + .create = memdirtylimit_cgroup_create,
> > + .destroy = memdirtylimit_cgroup_destroy,
> > + .populate = memdirtylimit_cgroup_populate,
> > +};
> > diff --git a/mm/page-writeback.c b/mm/page-writeback.c
> > index e6fa69e..f971532 100644
> > --- a/mm/page-writeback.c
> > +++ b/mm/page-writeback.c
> > @@ -34,6 +34,7 @@
> > #include <linux/syscalls.h>
> > #include <linux/buffer_head.h>
> > #include <linux/pagevec.h>
> > +#include <linux/memdirtylimitcgroup.h>
> >
> > /*
> > * The maximum number of pages to writeout in a single bdflush/kupdate
> > @@ -152,6 +153,7 @@ int dirty_ratio_handler(struct ctl_table *table, int write,
> > int shift = calc_period_shift();
> > prop_change_shift(&vm_completions, shift);
> > prop_change_shift(&vm_dirties, shift);
> > + memdirtylimitcgroup_change_shift(shift);
> > }
> > return ret;
> > }
> > @@ -393,6 +395,8 @@ get_dirty_limits(long *pbackground, long *pdirty, long *pbdi_dirty,
> > if (bdi) {
> > u64 bdi_dirty;
> > long numerator, denominator;
> > + long task_dirty;
> > + long cgroup_dirty;
> >
> > /*
> > * Calculate this BDI's share of the dirty ratio.
> > @@ -408,7 +412,11 @@ get_dirty_limits(long *pbackground, long *pdirty, long *pbdi_dirty,
> >
> > *pbdi_dirty = bdi_dirty;
> > clip_bdi_dirty_limit(bdi, dirty, pbdi_dirty);
> > - task_dirty_limit(current, pbdi_dirty);
> > + task_dirty = *pbdi_dirty;
> > + task_dirty_limit(current, &task_dirty);
> > + cgroup_dirty = *pbdi_dirty;
> > + memdirtylimitcgroup_dirty_limit(current, &cgroup_dirty);
> > + *pbdi_dirty = min(task_dirty, cgroup_dirty);

> > }
> > }
> >
> > @@ -842,6 +850,7 @@ void __init page_writeback_init(void)
> > shift = calc_period_shift();
> > prop_descriptor_init(&vm_completions, shift);
> > prop_descriptor_init(&vm_dirties, shift);
> > + memdirtylimitcgroup_init(shift);
> > }
> >
> > /**
> > @@ -1105,6 +1114,7 @@ int __set_page_dirty_nobuffers(struct page *page)
> > }
> >
> > task_dirty_inc(current);
> > + memdirtylimitcgroup_dirty_inc(current);
> >
> > return 1;
> > }
> >
>
> _____
> 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: [PATCH][RFC] dirty balancing for cgroups
Posted by KAMEZAWA Hiroyuki on Fri, 11 Jul 2008 05:15:11 GMT
View Forum Message <> Reply to Message

On Fri, 11 Jul 2008 13:06:57 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> hi,
>
> > On Wed, 9 Jul 2008 15:00:34 +0900 (JST)
> > yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
> >
> > > hi,
> > >
> > > the following patch is a simple implementation of
> > > dirty balancing for cgroups. any comments?
> > >

> > > it depends on the following fix:
> > > http://lkml.org/lkml/2008/7/8/428
> > >
> >
> > A few comments ;)
>
> thanks for comments.
>
> > - This looks simple but, could you merge this into memory resource controller ?
>
> why?
>
3 points.
 1. Is this useful if used alone ?
 2. memcg requires this kind of feature, basically.
 3. I wonder I need more work to make this work well under memcg.

If chasing page->cgroup and memcg make this patch much more complex,
I think this style of implimentation is a choice.

 About 3.
    Does this works well if I changes get_dirty_limit()'s
    determine_dirtyable_memory() calculation under memcg ?
    But to do this seems not valid if dirty_ratio cgroup and memcg cgroup
    containes different set of tasks.

Thanks,
-Kame

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

---

Subject: Re: [PATCH][RFC] dirty balancing for cgroups
Posted by yamamoto on Fri, 11 Jul 2008 05:59:26 GMT
View Forum Message <> Reply to Message

> > > - This looks simple but, could you merge this into memory resource controller ?
> >
> > why?
> >
> 3 points.
> 1. Is this useful if used alone ?

it can be. why not?

> 2. memcg requires this kind of feature, basically.
>
> 3. I wonder I need more work to make this work well under memcg.

i'm not sure if i understand these points. can you explain a bit?

my patch penalizes heavy-writer cgroups as task_dirty_limit does
for heavy-writer tasks. i don't think that it's necessary to be
tied to the memory subsystem because i merely want to group writers.

otoh, if you want to limit the number (or percentage or whatever) of
dirty pages in a memory cgroup, it can't be done independently from
the memory subsystem, of course. it's another story, tho.

YAMAMOTO Takashi

>
> If chasing page->cgroup and memcg make this patch much more complex,
> I think this style of implimentation is a choice.
>
> About 3.
> Does this works well if I changes get_dirty_limit()'s
> determine_dirtyable_memory() calculation under memcg ?
> But to do this seems not valid if dirty_ratio cgroup and memcg cgroup
> containes different set of tasks.
>
> Thanks,
> -Kame

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

---

Subject: Re: [PATCH][RFC] dirty balancing for cgroups
Posted by KAMEZAWA Hiroyuki on Fri, 11 Jul 2008 07:09:52 GMT
View Forum Message <> Reply to Message

On Fri, 11 Jul 2008 14:59:26 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> > > > - This looks simple but, could you merge this into memory resource controller ?
> > >
> > > why?
> > >
> > 3 points.
> > 1. Is this useful if used alone ?
>

> it can be. why not?
>
> > 2. memcg requires this kind of feature, basically.
> >
> > 3. I wonder I need more work to make this work well under memcg.
>
> i'm not sure if i understand these points. can you explain a bit?
>
In my understanding, dirty_ratio is for helping memory (reclaim) subsystem.

See comments in fs/page-writeback.c:: determin_dirtyable_memory()
==
/*
 * Work out the current dirty-memory clamping and background writeout
 * thresholds.
 *
 * The main aim here is to lower them aggressively if there is a lot of mapped
 * memory around. To avoid stressing page reclaim with lots of unreclaimable
 * pages. It is better to clamp down on writers than to start swapping, and
 * performing lots of scanning.
 *
 * We only allow 1/2 of the currently-unmapped memory to be dirtied.
 *
 * We don't permit the clamping level to fall below 5% - that is getting rather
 * excessive.
 *
 * We make sure that the background writeout level is below the adjusted
 * clamping level.
==

"To avoid stressing page reclaim with lots of unreclaimable pages"

Then, I think memcg should support this for helping relcaim under memcg.

> my patch penalizes heavy-writer cgroups as task_dirty_limit does
> for heavy-writer tasks. i don't think that it's necessary to be
> tied to the memory subsystem because i merely want to group writers.
>
Hmm, maybe what I need is different from this ;)
Does not seem to be a help for memory reclaim under memcg.

Thanks,
-Kame

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

Subject: Re: [PATCH][RFC] dirty balancing for cgroups
Posted by yamamoto on Fri, 11 Jul 2008 08:34:46 GMT
View Forum Message <> Reply to Message

hi,

> > my patch penalizes heavy-writer cgroups as task_dirty_limit does
> > for heavy-writer tasks. i don't think that it's necessary to be
> > tied to the memory subsystem because i merely want to group writers.
> >
> Hmm, maybe what I need is different from this ;)
> Does not seem to be a help for memory reclaim under memcg.

to implement what you need, i think that we need to keep track of
the numbers of dirty-pages in each memory cgroups as a first step.
do you agree?

YAMAMOTO Takashi

Subject: Re: [PATCH][RFC] dirty balancing for cgroups
Posted by KAMEZAWA Hiroyuki on Fri, 11 Jul 2008 08:48:25 GMT
View Forum Message <> Reply to Message

On Fri, 11 Jul 2008 17:34:46 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> hi,
>
> > > my patch penalizes heavy-writer cgroups as task_dirty_limit does
> > > for heavy-writer tasks. i don't think that it's necessary to be
> > > tied to the memory subsystem because i merely want to group writers.
> > >
> > Hmm, maybe what I need is different from this ;)
> > Does not seem to be a help for memory reclaim under memcg.
>
> to implement what you need, i think that we need to keep track of
> the numbers of dirty-pages in each memory cgroups as a first step.
> do you agree?
>

yes, I think so, now.

may be not difficult but will add extra overhead ;(Sigh..

Thanks,
-Kame

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

---

Subject: Re: [PATCH][RFC] dirty balancing for cgroups
Posted by Peter Zijlstra on Mon, 14 Jul 2008 13:37:17 GMT
View Forum Message <> Reply to Message

On Wed, 2008-07-09 at 15:00 +0900, YAMAMOTO Takashi wrote:
> hi,
>
> the following patch is a simple implementation of
> dirty balancing for cgroups. any comments?
>
> it depends on the following fix:
> http://lkml.org/lkml/2008/7/8/428
>
> YAMAMOTO Takashi
>
>
> Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>
> ---

Yamamoto-san,

> @@ -408,7 +412,11 @@ get_dirty_limits(long *pbackground, long *pdirty, long *pbdi_dirty,
>
>   *pbdi_dirty = bdi_dirty;
>   clip_bdi_dirty_limit(bdi, dirty, pbdi_dirty);
> - task_dirty_limit(current, pbdi_dirty);
> + task_dirty = *pbdi_dirty;
> + task_dirty_limit(current, &task_dirty);
> + cgroup_dirty = *pbdi_dirty;
> + memdirtylimitcgroup_dirty_limit(current, &cgroup_dirty);
> + *pbdi_dirty = min(task_dirty, cgroup_dirty);
>   }
> }

I think this is wrong - is basically breaks task dirty throttling within
groups. You'd need a multiplicative operation, something like:

  bdi_dirty = dirty * p(bdi) * p(cgroup) * (1 - p(task))

However then we still have problems... see the next email further down
the thread.

Subject: Re: [PATCH][RFC] dirty balancing for cgroups
Posted by Peter Zijlstra on Mon, 14 Jul 2008 13:49:04 GMT
View Forum Message <> Reply to Message

On Fri, 2008-07-11 at 16:13 +0900, KAMEZAWA Hiroyuki wrote:
> On Fri, 11 Jul 2008 14:59:26 +0900 (JST)
> yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
>
> > > > > > - This looks simple but, could you merge this into memory resource controller ?
> > > > >
> > > > > why?
> > > > >
> > > > 3 points.
> > > > 1. Is this useful if used alone ?
> > >
> > it can be. why not?
> > >
> > > 2. memcg requires this kind of feature, basically.
> > >
> > > 3. I wonder I need more work to make this work well under memcg.
> > >
> > i'm not sure if i understand these points. can you explain a bit?
> > >
> In my understanding, dirty_ratio is for helping memory (reclaim) subsystem.
>
> See comments in fs/page-writeback.c:: determin_dirtyable_memory()
> ==
> /*
> * Work out the current dirty-memory clamping and background writeout
> * thresholds.
> *
> * The main aim here is to lower them aggressively if there is a lot of mapped
> * memory around. To avoid stressing page reclaim with lots of unreclaimable

> * pages. It is better to clamp down on writers than to start swapping, and
> * performing lots of scanning.
> *
> * We only allow 1/2 of the currently-unmapped memory to be dirtied.
> *
> * We don't permit the clamping level to fall below 5% - that is getting rather
> * excessive.
> *
> * We make sure that the background writeout level is below the adjusted
> * clamping level.
> ==
>
> "To avoid stressing page reclaim with lots of unreclaimable pages"
>
> Then, I think memcg should support this for helping relcaim under memcg.

That comment is unclear at best.

The dirty page limit avoids deadlocks under certain situations, the per BDI dirty limit avoids even mode deadlocks by providing isolation between BDIs.

The fundamental deadlock solved by the dirty page limit is the typical reclaim deadlock - needing memory to free memory. It does this by ensuring only some part of the total memory used for the page-cache can be dirty, thus we always have clean pages around that can be reclaimed so we can launder the dirty pages.

This on its own generates a new deadlock for stacked devices, imagine device A on top of B. When A generates loads of dirty pages it will eventually hit the dirty limit and we'd start to launder them. However in order to launder A's dirty pages we'd need to dirty pages for B, but we can't since we're at the global limit.

This problem is solved by introducing a per BDI dirty limit, by assigning each BDI an individual dirty limit (whoes sum is the total dirty limit) we avoid that deadlock. Take the previous example; A would start laundering its pages when it hits its own limit, B's operation isn't hampered by that.

[even when B's limit is 0 we're able to make progress, since we'll only
  wait for B's dirty page count to decrease - effectively reducing to
  sync writes.]

Of course this raises the question how to assign the various dirty limits - any fixed distribution is hard to maintain and suboptimial for most workloads.

We solve this by assigning each BDI a fraction proportional to its current launder speed. That is to say, if A launders pages twice as fast as B does, then A will get 2/3-rd of the total dirty page limit, versus 1/3-rd for B.

Then there is the task dirty stuff - this is basically a 'fix' for the problem where a slow writer gets starved by a fast reader. Imagine two tasks competing for bandwidth, 1 the fast writer and 2 the slow writer.

1 will dirty loads of pages but all things being equal 2 will have to wait for 1's dirty pages.

So what we do is lower the dirty limit for fast writers - so these get to wait sooner and slow writers have a little room to make progress before they too have to wait.

To properly solve this we'd need to track p_{bdi,task}. However that's intracktable. Therefore we approximate that with p_bdi * p_task. This approximation looses detail.

Imagine two tasks: 1 and 2, and two BDIs A and B (independent this time). If 1 is a (fast) writer to A and 2 is a (slow) writer to B, we need not throttle 2 sooner as there is no actual competition.

The full proportional tensor p_{bdi,task} can express this, but the simple approximation p_bdi * p_task can not.

The approximation will reduce 1's bandwidth a little even though there is no actual competition.

Now the problem this patch tries to address...

As you can see you'd need p_{bdi,cgroup,task} for it to work, and the obvious approximation p_bdi * p_cgroup * p_task will get even more coarse.

You could possibly attempt to do p_{bdi,cgroup} * p_task since the bdi and cgroup set are pretty static, but still that would be painful.

So, could you please give some more justification for this work, I'm not seeing the value in complicating all this just yet.

Thanks for reading this far,

Peter

_____

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

---

Subject: Re: Re: [PATCH][RFC] dirty balancing for cgroups
Posted by KAMEZAWA Hiroyuki on Mon, 14 Jul 2008 14:38:24 GMT
View Forum Message <> Reply to Message

----- Original Message -----
>> See comments in fs/page-writeback.c:: determin_dirtyable_memory()
>> ==
>> /*
>> * Work out the current dirty-memory clamping and background writeout
>> * thresholds.
>> *
>> * The main aim here is to lower them aggressively if there is a lot of map
ped
>> * memory around. To avoid stressing page reclaim with lots of unreclaimab
le
>> * pages. It is better to clamp down on writers than to start swapping, an
d
>> * performing lots of scanning.
>> *
>> * We only allow 1/2 of the currently-unmapped memory to be dirtied.
>> *
>> * We don't permit the clamping level to fall below 5% - that is getting ra
ther
>> * excessive.
>> *
>> * We make sure that the background writeout level is below the adjusted
>> * clamping level.
>> ==
>>
>> "To avoid stressing page reclaim with lots of unreclaimable pages"
>>
>> Then, I think memcg should support this for helping relcaim under memcg.
>
>That comment is unclear at best.
>
>The dirty page limit avoids deadlocks under certain situations, the per
>BDI dirty limit avoids even mode deadlocks by providing isolation
>between BDIs.
>
>

>The fundamental deadlock solved by the dirty page limit is the typical
>reclaim deadlock - needing memory to free memory. It does this by
>ensuring only some part of the total memory used for the page-cache can
>be dirty, thus we always have clean pages around that can be reclaimed
>so we can launder the dirty pages.
>
>This on its own generates a new deadlock for stacked devices, imagine
>device A on top of B. When A generates loads of dirty pages it will
>eventually hit the dirty limit and we'd start to launder them. However
>in order to launder A's dirty pages we'd need to dirty pages for B, but
>we can't since we're at the global limit.
>
>This problem is solved by introducing a per BDI dirty limit, by
>assigning each BDI an individual dirty limit (whoes sum is the total
>dirty limit) we avoid that deadlock. Take the previous example; A would
>start laundering its pages when it hits its own limit, B's operation
>isn't hampered by that.
>
>[even when B's limit is 0 we're able to make progress, since we'll only
> wait for B's dirty page count to decrease - effectively reducing to
> sync writes.]
>
>Of course this raises the question how to assign the various dirty
>limits - any fixed distribution is hard to maintain and suboptimial for
>most workloads.
>
>We solve this by assigning each BDI a fraction proportional to its
>current launder speed. That is to say, if A launders pages twice as fast
>as B does, then A will get 2/3-rd of the total dirty page limit, versus
>1/3-rd for B.
>
>
>Then there is the task dirty stuff - this is basically a 'fix' for the
>problem where a slow writer gets starved by a fast reader. Imagine two
>tasks competing for bandwidth, 1 the fast writer and 2 the slow writer.
>
>1 will dirty loads of pages but all things being equal 2 will have to
>wait for 1's dirty pages.
>
>So what we do is lower the dirty limit for fast writers - so these get
>to wait sooner and slow writers have a little room to make progress
>before they too have to wait.
>
>To properly solve this we'd need to track p_{bdi,task}. However that's
>intracktable. Therefore we approximate that with p_bdi * p_task. This
>approximation looses detail.
>
>Imagine two tasks: 1 and 2, and two BDIs A and B (independent this

>time). If 1 is a (fast) writer to A and 2 is a (slow) writer to B, we
>need not throttle 2 sooner as there is no actual competition.
>
>The full proportional tensor p_{bdi,task} can express this, but the
>simple approximation p_bdi * p_task can not.
>
>The approximation will reduce 1's bandwidth a little even though there
>is no actual competition.
>
>
>Now the problem this patch tries to address...
>
>As you can see you'd need p_{bdi,cgroup,task} for it to work, and the
>obvious approximation p_bdi * p_cgroup * p_task will get even more
>coarse.
>
>You could possibly attempt to do p_{bdi,cgroup} * p_task since the bdi
>and cgroup set are pretty static, but still that would be painful.
>
>So, could you please give some more justification for this work, I'm not
>seeing the value in complicating all this just yet.
>
>
>Thanks for reading this far,
>
Thank you for explanation. Maybe I misunderstood something.

What I thought was to keep some amount of clean pages under memcg
for smooth reclaiming. But after reading your explanation, I'm now
considering again...

about DEADLOCK:

memory cgroup's reclaim path is working as following now.
==
mem_cgroup_charge() or others.
  -> try_to_free_mem_cgroup_pages()
     -> shrink_zone()...
        -> subsystem may call alloc_pages()
           -> ....(*)

Unlike global LRU, memory_cgroup does not affect (*) because add_tp_page_cache
() is called here and maybe no deadlock.

about Memory Reclaim's health:

It doesn't seem good to allow users to make all pages under memcg
to be dirty without penalty(throttling).

Dirty/Written-backed pages are not to be reclaimed by first lru scan because
it has to be written out (and rolated to the tail of LRU.) and clean pages,
which are easy to be dropped, are dropped soon.

Under following situation
 - large file copy under memcg.
 - big coredump under memcg.

I think too much (clean) pages are swapped out by memcg's LRU because
of tons of dirty pages. (but I don't have enough data now. just my concern.)

What can I do against this ? Could you give me a hint ?

Thanks,
-Kame

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

---

Subject: Re: [PATCH][RFC] dirty balancing for cgroups
Posted by yamamoto on Thu, 17 Jul 2008 01:43:35 GMT
View Forum Message <> Reply to Message

hi,

> Now the problem this patch tries to address...
>
> As you can see you'd need p_{bdi,cgroup,task} for it to work, and the
> obvious approximation p_bdi * p_cgroup * p_task will get even more
> coarse.
>
> You could possibly attempt to do p_{bdi,cgroup} * p_task since the bdi
> and cgroup set are pretty static, but still that would be painful.

i chose min(p_bdi * p_cgroup, p_bdi * p_task) because i couldn't imagine
a case where p_bdi * p_cgroup * p_task is better.

> So, could you please give some more justification for this work, I'm not
> seeing the value in complicating all this just yet.

a simple example for which my patch can make some sense is:

 while :;do dd if=/dev/zero of=file conv=notrunc bs=4096 count=1;done

YAMAMOTO Takashi

_____

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