

---

Subject: [PATCH -mm 1/3] cgroup: use a hash table for css\_set finding

Posted by [Li Zefan](#) on Wed, 02 Apr 2008 02:16:45 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

When we attach a process to a different cgroup, the css\_set linked-list will be run through to find a suitable existing css\_set to use. This patch implements a hash table for better performance.

The following benchmarks has been tested:

For N in 1, 5, 10, 50, 100, 500, 1000, create N cgroups with one sleeping task in each, and then move an additional task through each cgroup in turn.

Here is a test result:

N Loop orig - Time(s) hash - Time(s)

```
-----
1 10000 1.201231728 1.196311177
5 2000 1.065743872 1.040566424
10 1000 0.991054735 0.986876440
50 200 0.976554203 0.969608733
100 100 0.998504680 0.969218270
500 20 1.157347764 0.962602963
1000 10 1.619521852 1.085140172
```

Signed-off-by: Li Zefan <lizf@cn.fujitsu.com>

---

```
include/linux/cgroup.h | 7 +++++-
kernel/cgroup.c        | 58 ++++++++++++++++++++++++++++++++++++++-----
2 files changed, 52 insertions(+), 13 deletions(-)
```

diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h

index 785a01c..c15c5e0 100644

--- a/include/linux/cgroup.h

+++ b/include/linux/cgroup.h

```
@@ -156,6 +156,12 @@ struct css_set {
    struct list_head list;
```

```
    /*
+ * List running through all cgroup groups in the same hash
+ * slot. Protected by css_set_lock
+ */
+ struct hlist_node hlist;
+
+ /*
+ * List running through all tasks using this cgroup
```

```

    * group. Protected by css_set_lock
    */
@@ -174,7 +180,6 @@ struct css_set {
    * during subsystem registration (at boot time).
    */
    struct cgroup_subsys_state *subsys[CGROUP_SUBSYS_COUNT];
-
};

/*
diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index e8e8ec4..78e5bde 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -44,6 +44,7 @@
#include <linux/kmod.h>
#include <linux/delayacct.h>
#include <linux/cgroupstats.h>
+#include <linux/hash.h>

#include <asm/atomic.h>

@@ -193,6 +194,26 @@ static struct cg_cgroup_link init_css_set_link;
static DEFINE_RWLOCK(css_set_lock);
static int css_set_count;

+/* hash table for cgroup groups. This improves the performance to
+ * find an existing css_set */
+#define CSS_SET_HASH_BITS 7
+#define CSS_SET_TABLE_SIZE (1 << CSS_SET_HASH_BITS)
+static struct hlist_head css_set_table[CSS_SET_TABLE_SIZE];
+
+static struct hlist_head *css_set_hash(struct cgroup_subsys_state *css[])
+{
+    int i;
+    int index;
+    unsigned long tmp = 0UL;
+
+    for (i = 0; i < CGROUP_SUBSYS_COUNT; i++)
+        tmp += (unsigned long)css[i];
+
+    index = hash_long(tmp, CSS_SET_HASH_BITS);
+
+    return &css_set_table[index];
+}
+
+/* We don't maintain the lists running through each css_set to its
+ * task until after the first call to cgroup_iter_start(). This

```

```

* reduces the fork()/exit() overhead for people who have cgroups
@@ -219,6 +240,7 @@ static int use_task_css_set_links;
static void unlink_css_set(struct css_set *cg)
{
    write_lock(&css_set_lock);
+ hlist_del(&cg->hlist);
    list_del(&cg->list);
    css_set_count--;
    while (!list_empty(&cg->cg_links)) {
@@ -284,9 +306,7 @@ static inline void put_css_set_taskexit(struct css_set *cg)
/*
 * find_existing_css_set() is a helper for
 * find_css_set(), and checks to see whether an existing
- * css_set is suitable. This currently walks a linked-list for
- * simplicity; a later patch will use a hash table for better
- * performance
+ * css_set is suitable.
 *
 * oldcg: the cgroup group that we're using before the cgroup
 * transition
@@ -303,7 +323,9 @@ static struct css_set *find_existing_css_set(
{
    int i;
    struct cgroupfs_root *root = cgrp->root;
- struct list_head *l = &init_css_set.list;
+ struct hlist_head *hhead;
+ struct hlist_node *node;
+ struct css_set *cg;

    /* Built the set of subsystem state objects that we want to
     * see in the new css_set */
@@ -320,18 +342,13 @@ static struct css_set *find_existing_css_set(
    }
}

- /* Look through existing cgroup groups to find one to reuse */
- do {
-     struct css_set *cg =
-         list_entry(l, struct css_set, list);
-     -
+ hhead = css_set_hash(template);
+ hlist_for_each_entry(cg, node, hhead, hlist) {
+     if (!memcmp(template, cg->subsys, sizeof(cg->subsys))) {
+         /* All subsystems matched */
+         return cg;
+     }
- /* Try the next cgroup group */
- l = l->next;

```

```

- } while (l != &init_css_set.list);
+ }

/* No existing cgroup group matched */
return NULL;
@@ -393,6 +410,8 @@ static struct css_set *find_css_set(
    struct list_head tmp_cg_links;
    struct cg_cgroup_link *link;

+ struct hlist_head *hhead;
+
/* First see if we already have a cgroup group that matches
 * the desired set */
write_lock(&css_set_lock);
@@ -417,6 +436,7 @@ static struct css_set *find_css_set(
    kref_init(&res->ref);
    INIT_LIST_HEAD(&res->cg_links);
    INIT_LIST_HEAD(&res->tasks);
+ INIT_HLIST_NODE(&res->hlist);

/* Copy the set of subsystem state objects generated in
 * find_existing_css_set() */
@@ -459,6 +479,11 @@ static struct css_set *find_css_set(
/* Link this cgroup group into the list */
list_add(&res->list, &init_css_set.list);
css_set_count++;
+
+ /* Add this cgroup group to the hash table */
+ hhead = css_set_hash(res->subsys);
+ hlist_add_head(&res->hlist, hhead);
+
write_unlock(&css_set_lock);

return res;
@@ -2499,6 +2524,7 @@ int __init cgroup_init_early(void)
    INIT_LIST_HEAD(&init_css_set.list);
    INIT_LIST_HEAD(&init_css_set.cg_links);
    INIT_LIST_HEAD(&init_css_set.tasks);
+ INIT_HLIST_NODE(&init_css_set.hlist);
css_set_count = 1;
init_cgroup_root(&rootnode);
list_add(&rootnode.root_list, &roots);
@@ -2511,6 +2537,9 @@ int __init cgroup_init_early(void)
    list_add(&init_css_set.link.cg_link_list,
        &init_css_set.cg_links);

+ for (i = 0; i < CSS_SET_TABLE_SIZE; i++)
+ INIT_HLIST_HEAD(&css_set_table[i]);

```

```

+
for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
    struct cgroup_subsys *ss = subsys[i];

@@ -2540,6 +2569,7 @@ int __init cgroup_init(void)
{
    int err;
    int i;
+ struct hlist_head *hhead;

    err = bdi_init(&cgroup_backing_dev_info);
    if (err)
@@ -2551,6 +2581,10 @@ int __init cgroup_init(void)
    cgroup_init_subsys(ss);
}

+ /* Add init_css_set to the hash table */
+ hhead = css_set_hash(init_css_set.subsys);
+ hlist_add_head(&init_css_set.hlist, hhead);
+
    err = register_filesystem(&cgroup_fs_type);
    if (err < 0)
        goto out;
--
1.5.4.rc3

```

---

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

---



---

Subject: Re: [PATCH -mm 1/3] cgroup: use a hash table for css\_set finding  
Posted by [Paul Menage](#) on Wed, 02 Apr 2008 10:23:59 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Tue, Apr 1, 2008 at 7:16 PM, Li Zefan <lizf@cn.fujitsu.com> wrote:

- > When we attach a process to a different cgroup, the css\_set linked-list
- > will be run through to find a suitable existing css\_set to use. This
- > patch implements a hash table for better performance.
- >
- > The following benchmarks has been tested:
- >
- > For N in 1, 5, 10, 50, 100, 500, 1000, create N cgroups with one sleeping
- > task in each, and then move an additional task through each cgroup in
- > turn.
- >
- > Here is a test result:
- >

```

> N      Loop  orig - Time(s) hash - Time(s)
> -----
> 1      10000  1.201231728    1.196311177
> 5       2000  1.065743872    1.040566424
> 10     1000   0.991054735    0.986876440
> 50      200   0.976554203    0.969608733
> 100     100   0.998504680    0.969218270
> 500      20   1.157347764    0.962602963
> 1000     10   1.619521852    1.085140172
>
>
> Signed-off-by: Li Zefan <lizf@cn.fujitsu.com>

```

Reviewed-by: Paul Menage <menage@google.com>

This looks good, thanks.

```

> ---
> include/linux/cgroup.h | 7 +++++-
> kernel/cgroup.c        | 58 ++++++++++++++++++++++++++++++++++++++-----
> 2 files changed, 52 insertions(+), 13 deletions(-)
>
> diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
> index 785a01c..c15c5e0 100644
> --- a/include/linux/cgroup.h
> +++ b/include/linux/cgroup.h
> @@ -156,6 +156,12 @@ struct css_set {
>     struct list_head list;
>
>     /*
> +    * List running through all cgroup groups in the same hash
> +    * slot. Protected by css_set_lock
> +    */
>     struct hlist_node hlist;
> +
> +    /*
> +    * List running through all tasks using this cgroup
> +    * group. Protected by css_set_lock
> +    */
> @@ -174,7 +180,6 @@ struct css_set {
>     * during subsystem registration (at boot time).
>     */
>     struct cgroup_subsys_state *subsys[CGROUP_SUBSYS_COUNT];
> -
> };
>
> /*
> diff --git a/kernel/cgroup.c b/kernel/cgroup.c

```

```

> index e8e8ec4..78e5bde 100644
> --- a/kernel/cgroup.c
> +++ b/kernel/cgroup.c
> @@ -44,6 +44,7 @@
> #include <linux/kmod.h>
> #include <linux/delayacct.h>
> #include <linux/cgroupstats.h>
> +#include <linux/hash.h>
>
> #include <asm/atomic.h>
>
> @@ -193,6 +194,26 @@ static struct cg_cgroup_link init_css_set_link;
> static DEFINE_RWLOCK(css_set_lock);
> static int css_set_count;
>
> +/* hash table for cgroup groups. This improves the performance to
> + * find an existing css_set */
> +#define CSS_SET_HASH_BITS    7
> +#define CSS_SET_TABLE_SIZE   (1 << CSS_SET_HASH_BITS)
> +static struct hlist_head css_set_table[CSS_SET_TABLE_SIZE];
> +
> +static struct hlist_head *css_set_hash(struct cgroup_subsys_state *css[])
> +{
> +    int i;
> +    int index;
> +    unsigned long tmp = 0UL;
> +
> +    for (i = 0; i < CGROUP_SUBSYS_COUNT; i++)
> +        tmp += (unsigned long)css[i];
> +
> +    index = hash_long(tmp, CSS_SET_HASH_BITS);
> +
> +    return &css_set_table[index];
> +}
> +
> /* We don't maintain the lists running through each css_set to its
> * task until after the first call to cgroup_iter_start(). This
> * reduces the fork()/exit() overhead for people who have cgroups
> @@ -219,6 +240,7 @@ static int use_task_css_set_links;
> static void unlink_css_set(struct css_set *cg)
> {
>     write_lock(&css_set_lock);
> +    hlist_del(&cg->hlist);
>     list_del(&cg->list);
>     css_set_count--;
>     while (!list_empty(&cg->cg_links)) {
> @@ -284,9 +306,7 @@ static inline void put_css_set_taskexit(struct css_set *cg)
> /*

```

```

> * find_existing_css_set() is a helper for
> * find_css_set(), and checks to see whether an existing
> - * css_set is suitable. This currently walks a linked-list for
> - * simplicity; a later patch will use a hash table for better
> - * performance
> + * css_set is suitable.
> *
> * oldcg: the cgroup group that we're using before the cgroup
> * transition
> @@ -303,7 +323,9 @@ static struct css_set *find_existing_css_set(
> {
>     int i;
>     struct cgroupfs_root *root = cgrp->root;
> -     struct list_head *l = &init_css_set.list;
> +     struct hlist_head *hhead;
> +     struct hlist_node *node;
> +     struct css_set *cg;
>
>     /* Built the set of subsystem state objects that we want to
>      * see in the new css_set */
> @@ -320,18 +342,13 @@ static struct css_set *find_existing_css_set(
>     }
> }
>
> -     /* Look through existing cgroup groups to find one to reuse */
> -     do {
> -         struct css_set *cg =
> -             list_entry(l, struct css_set, list);
> -
> +         hhead = css_set_hash(template);
> +         hlist_for_each_entry(cg, node, hhead, hlist) {
>             if (!memcmp(template, cg->subsys, sizeof(cg->subsys))) {
>                 /* All subsystems matched */
>                 return cg;
>             }
> -             /* Try the next cgroup group */
> -             l = l->next;
> -         } while (l != &init_css_set.list);
> +     }
>
>     /* No existing cgroup group matched */
>     return NULL;
> @@ -393,6 +410,8 @@ static struct css_set *find_css_set(
>     struct list_head tmp_cg_links;
>     struct cg_cgroup_link *link;
>
> +     struct hlist_head *hhead;
> +

```



```

> /* First see if we already have a cgroup group that matches
> * the desired set */
> write_lock(&css_set_lock);
> @@ -417,6 +436,7 @@ static struct css_set *find_css_set(
> kref_init(&res->ref);
> INIT_LIST_HEAD(&res->cg_links);
> INIT_LIST_HEAD(&res->tasks);
> + INIT_HLIST_NODE(&res->hlist);
>
> /* Copy the set of subsystem state objects generated in
> * find_existing_css_set() */
> @@ -459,6 +479,11 @@ static struct css_set *find_css_set(
> /* Link this cgroup group into the list */
> list_add(&res->list, &init_css_set.list);
> css_set_count++;
> +
> + /* Add this cgroup group to the hash table */
> + hhead = css_set_hash(res->subsys);
> + hlist_add_head(&res->hlist, hhead);
> +
> write_unlock(&css_set_lock);
>
> return res;
> @@ -2499,6 +2524,7 @@ int __init cgroup_init_early(void)
> INIT_LIST_HEAD(&init_css_set.list);
> INIT_LIST_HEAD(&init_css_set.cg_links);
> INIT_LIST_HEAD(&init_css_set.tasks);
> + INIT_HLIST_NODE(&init_css_set.hlist);
> css_set_count = 1;
> init_cgroup_root(&rootnode);
> list_add(&rootnode.root_list, &roots);
> @@ -2511,6 +2537,9 @@ int __init cgroup_init_early(void)
> list_add(&init_css_set.link.cg_link_list,
> &init_css_set.cg_links);
>
> + for (i = 0; i < CSS_SET_TABLE_SIZE; i++)
> + INIT_HLIST_HEAD(&css_set_table[i]);
> +
> for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
> struct cgroup_subsys *ss = subsys[i];
>
> @@ -2540,6 +2569,7 @@ int __init cgroup_init(void)
> {
> int err;
> int i;
> + struct hlist_head *hhead;
>
> err = bdi_init(&cgroup_backing_dev_info);

```

```

>     if (err)
> @@ -2551,6 +2581,10 @@ int __init cgroup_init(void)
>         cgroup_init_subsys(ss);
>     }
>
> +    /* Add init_css_set to the hash table */
> +    hhead = css_set_hash(init_css_set.subsys);
> +    hlist_add_head(&init_css_set.hlist, hhead);
> +
>     err = register_filesystem(&cgroup_fs_type);
>     if (err < 0)
>         goto out;
> --
> 1.5.4.rc3
>

```

---

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

---



---

Subject: Re: [PATCH -mm 1/3] cgroup: use a hash table for css\_set finding  
Posted by [KAMEZAWA Hiroyuki](#) on Thu, 03 Apr 2008 07:24:59 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Thu, 03 Apr 2008 13:52:43 +0800  
Li Zefan <lizf@cn.fujitsu.com> wrote:  
> +/\* hash table for cgroup groups. This improves the performance to  
> + \* find an existing css\_set \*/  
> +#define CSS\_SET\_HASH\_BITS 7  
> +#define CSS\_SET\_TABLE\_SIZE (1 << CSS\_SET\_HASH\_BITS)  
> +static struct hlist\_head css\_set\_table[CSS\_SET\_TABLE\_SIZE];

How above number is selected ?

```

> +static struct hlist_head *css_set_hash(struct cgroup_subsys_state *css[])
> +{
> + int i;
> + int index;
> + unsigned long tmp = 0UL;
> +
> + for (i = 0; i < CGROUP_SUBSYS_COUNT; i++)
> +     tmp += (unsigned long)css[i];
> +

```

maybe css[i]'s lower 2-3 bits will be ignored. because they are always 0.

And I don't like "+" for hash. how about

```
==  
for (i = 0; i < CGROUP_SUBSYS_COUNT; i++)  
    unsigned long x;  
    x = (unsigned long)css[i];  
    tmp = (x >> 16) ^ (x & 0xffff)
```

```
==  
or some func, which uses full bits.
```

```
> + index = hash_long(tmp, CSS_SET_HASH_BITS);  
> +  
> + return &css_set_table[index];  
> +}  
> +  
> /* We don't maintain the lists running through each css_set to its  
> * task until after the first call to cgroup_iter_start(). This  
> * reduces the fork()/exit() overhead for people who have cgroups  
> @@ -219,6 +240,7 @@ static int use_task_css_set_links;  
> static void unlink_css_set(struct css_set *cg)  
> {  
>     write_lock(&css_set_lock);  
> + hlist_del(&cg->hlist);  
>     list_del(&cg->list);  
>     css_set_count--;
```

This css\_set\_lock is worth to be rwlock ?  
how about per hashline spinlock ? (but per-hashline seems overkill..)

Thanks,  
-Kame

---

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

---

---

Subject: Re: [PATCH -mm 1/3] cgroup: use a hash table for css\_set finding  
Posted by [Li Zefan](#) on Thu, 03 Apr 2008 07:51:05 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

KAMEZAWA Hiroyuki wrote:  
> On Thu, 03 Apr 2008 13:52:43 +0800  
> Li Zefan <lizf@cn.fujitsu.com> wrote:  
>> +/\* hash table for cgroup groups. This improves the performance to  
>> + \* find an existing css\_set \*/  
>> +#define CSS\_SET\_HASH\_BITS 7  
>> +#define CSS\_SET\_TABLE\_SIZE (1 << CSS\_SET\_HASH\_BITS)

```
>> +static struct hlist_head css_set_table[CSS_SET_TABLE_SIZE];
>
> How above number is selected ?
>
```

I suppose 100 will be suitable, so i would like to choose from 6 or 7 bits.

```
>> +static struct hlist_head *css_set_hash(struct cgroup_subsys_state *css[])
>> +{
>> + int i;
>> + int index;
>> + unsigned long tmp = 0UL;
>> +
>> + for (i = 0; i < CGROUP_SUBSYS_COUNT; i++)
>> + tmp += (unsigned long)css[i];
>> +
>
> maybe css[i]'s lower 2-3 bits will be ignored. because they are always 0.
>
> And I don't like "+" for hash. how about
> ==
> for (i = 0; i < CGROUP_SUBSYS_COUNT; i++)
> unsigned long x;
> x = (unsigned long)css[i];
> tmp = (x >> 16) ^ (x & 0xffff)
> ==
> or some func, which uses full bits.
>
```

I'm using hash\_long(), which has been proved to be a good one. And I've tested css\_set\_hash(), I run the css\_set benchmark with N == 1000, the for loop in find\_existing\_css\_set() will break out within 10 iterations for most cases, which is the expected result.

```
>
>> + index = hash_long(tmp, CSS_SET_HASH_BITS);
>> +
>> + return &css_set_table[index];
>> +}
>> +
>> /* We don't maintain the lists running through each css_set to its
>> * task until after the first call to cgroup_iter_start(). This
>> * reduces the fork()/exit() overhead for people who have cgroups
>> @@ -219,6 +240,7 @@ static int use_task_css_set_links;
>> static void unlink_css_set(struct css_set *cg)
>> {
>> write_lock(&css_set_lock);
>> + hlist_del(&cg->hlist);
```

```
>> list_del(&cg->list);
>> css_set_count--;
>
> This css_set_lock is worth to be rwlock ?
> how about per hashline spinlock ? (but per-hashline seems overkill..)
>
```

I think it's an overkill. And the css\_set\_lock protects not only the hash table.

Thanks for looking into this. :)

Regards,  
Li Zefan

---

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

---

---

Subject: Re: [PATCH -mm 1/3] cgroup: use a hash table for css\_set finding  
Posted by [Li Zefan](#) on Thu, 03 Apr 2008 10:13:13 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

```
>>> +static struct hlist_head *css_set_hash(struct cgroup_subsys_state *css[])
>>> +{
>>> + int i;
>>> + int index;
>>> + unsigned long tmp = 0UL;
>>> +
>>> + for (i = 0; i < CGROUP_SUBSYS_COUNT; i++)
>>> + tmp += (unsigned long)css[i];
>>> +
>> maybe css[i]'s lower 2-3 bits will be ignored. because they are always 0.
>>
>> And I don't like "+" for hash. how about
>> ==
>> for (i = 0; i < CGROUP_SUBSYS_COUNT; i++)
>> unsigned long x;
>> x = (unsigned long)css[i];
>> tmp = (x >> 16) ^ (x & 0xffff)
>> ==
>> or some func, which uses full bits.
>>
>
> I'm using hash_long(), which has been proved to be a good one. And I've tested
> css_set_hash(), I run the css_set benchmark with N == 1000, the for loop in
```

> find\_existing\_css\_set() will break out within 10 iterations for most cases,  
> which is the expected result.  
>

I just revised the css\_set\_hash() a bit, and It turned out we can improve the scalability.

Thanks, KAMEZAWA-san.

---

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

---

---

Subject: Re: [PATCH -mm 1/3] cgroup: use a hash table for css\_set finding  
Posted by [Li Zefan](#) on Thu, 03 Apr 2008 10:17:08 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

When we attach a process to a different cgroup, the css\_set linked-list will be run through to find a suitable existing css\_set to use. This patch implements a hash table for better performance.

The following benchmarks has been tested:

For N in 1, 5, 10, 50, 100, 500, 1000, create N cgroups with one sleeping task in each, and then move an additional task through each cgroup in turn.

Here is a test result:

N Loop orig - Time(s) hash - Time(s)  
-----  
1 10000 1.201231728 1.196311177  
5 2000 1.065743872 1.040566424  
10 1000 0.991054735 0.986876440  
50 200 0.976554203 0.969608733  
100 100 0.998504680 0.969218270  
500 20 1.157347764 0.962602963  
1000 10 1.619521852 1.085140172

Signed-off-by: Li Zefan <lizf@cn.fujitsu.com>  
Reviewed-by: Paul Menage <menage@google.com>

---

include/linux/cgroup.h | 7 +++++-  
kernel/cgroup.c | 59 ++++++-----  
2 files changed, 53 insertions(+), 13 deletions(-)

```

diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
index 2d1d151..f585b7c 100644
--- a/include/linux/cgroup.h
+++ b/include/linux/cgroup.h
@@ -156,6 +156,12 @@ struct css_set {
    struct list_head list;

    /*
+ * List running through all cgroup groups in the same hash
+ * slot. Protected by css_set_lock
+ */
+ struct hlist_node hlist;
+
+ /*
 * List running through all tasks using this cgroup
 * group. Protected by css_set_lock
 */
@@ -174,7 +180,6 @@ struct css_set {
    * during subsystem registration (at boot time).
    */
    struct cgroup_subsys_state *subsys[CGROUP_SUBSYS_COUNT];
-
};

/*
diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index 85f31ad..23e2573 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -44,6 +44,7 @@
#include <linux/kmod.h>
#include <linux/delayacct.h>
#include <linux/cgroupstats.h>
+#include <linux/hash.h>

#include <asm/atomic.h>

@@ -193,6 +194,27 @@ static struct cg_cgroup_link init_css_set_link;
static DEFINE_RWLOCK(css_set_lock);
static int css_set_count;

+/* hash table for cgroup groups. This improves the performance to
+ * find an existing css_set */
+#define CSS_SET_HASH_BITS 7
+#define CSS_SET_TABLE_SIZE (1 << CSS_SET_HASH_BITS)
+static struct hlist_head css_set_table[CSS_SET_TABLE_SIZE];
+
+static struct hlist_head *css_set_hash(struct cgroup_subsys_state *css[])

```

```

+{
+ int i;
+ int index;
+ unsigned long tmp = 0UL;
+
+ for (i = 0; i < CGROUP_SUBSYS_COUNT; i++)
+ tmp += (unsigned long)css[i];
+ tmp = (tmp >> 16) ^ tmp;
+
+ index = hash_long(tmp, CSS_SET_HASH_BITS);
+
+ return &css_set_table[index];
+}
+
/* We don't maintain the lists running through each css_set to its
 * task until after the first call to cgroup_iter_start(). This
 * reduces the fork()/exit() overhead for people who have cgroups
@@ -219,6 +241,7 @@ static int use_task_css_set_links;
static void unlink_css_set(struct css_set *cg)
{
    write_lock(&css_set_lock);
+ hlist_del(&cg->hlist);
    list_del(&cg->list);
    css_set_count--;
    while (!list_empty(&cg->cg_links)) {
@@ -284,9 +307,7 @@ static inline void put_css_set_taskexit(struct css_set *cg)
/*
 * find_existing_css_set() is a helper for
 * find_css_set(), and checks to see whether an existing
- * css_set is suitable. This currently walks a linked-list for
- * simplicity; a later patch will use a hash table for better
- * performance
+ * css_set is suitable.
 *
 * oldcg: the cgroup group that we're using before the cgroup
 * transition
@@ -303,7 +324,9 @@ static struct css_set *find_existing_css_set(
{
    int i;
    struct cgroupfs_root *root = cgrp->root;
- struct list_head *l = &init_css_set.list;
+ struct hlist_head *hhead;
+ struct hlist_node *node;
+ struct css_set *cg;

    /* Built the set of subsystem state objects that we want to
     * see in the new css_set */
@@ -320,18 +343,13 @@ static struct css_set *find_existing_css_set(

```



```

    }
}

- /* Look through existing cgroup groups to find one to reuse */
- do {
-     struct css_set *cg =
-     list_entry(l, struct css_set, list);
-
+ hhead = css_set_hash(template);
+ hlist_for_each_entry(cg, node, hhead, hlist) {
+     if (!memcmp(template, cg->subsys, sizeof(cg->subsys))) {
+         /* All subsystems matched */
+         return cg;
+     }
- /* Try the next cgroup group */
- l = l->next;
- } while (l != &init_css_set.list);
+ }

    /* No existing cgroup group matched */
    return NULL;
@@ -393,6 +411,8 @@ static struct css_set *find_css_set(
    struct list_head tmp_cg_links;
    struct cg_cgroup_link *link;

+ struct hlist_head *hhead;
+
    /* First see if we already have a cgroup group that matches
     * the desired set */
    write_lock(&css_set_lock);
@@ -417,6 +437,7 @@ static struct css_set *find_css_set(
    kref_init(&res->ref);
    INIT_LIST_HEAD(&res->cg_links);
    INIT_LIST_HEAD(&res->tasks);
+ INIT_HLIST_NODE(&res->hlist);

    /* Copy the set of subsystem state objects generated in
     * find_existing_css_set() */
@@ -459,6 +480,11 @@ static struct css_set *find_css_set(
    /* Link this cgroup group into the list */
    list_add(&res->list, &init_css_set.list);
    css_set_count++;
+
+ /* Add this cgroup group to the hash table */
+ hhead = css_set_hash(res->subsys);
+ hlist_add_head(&res->hlist, hhead);
+
    write_unlock(&css_set_lock);

```

```

return res;
@@ -2503,6 +2529,7 @@ int __init cgroup_init_early(void)
    INIT_LIST_HEAD(&init_css_set.list);
    INIT_LIST_HEAD(&init_css_set.cg_links);
    INIT_LIST_HEAD(&init_css_set.tasks);
+ INIT_HLIST_NODE(&init_css_set.hlist);
    css_set_count = 1;
    init_cgroup_root(&rootnode);
    list_add(&rootnode.root_list, &roots);
@@ -2515,6 +2542,9 @@ int __init cgroup_init_early(void)
    list_add(&init_css_set_link.cg_link_list,
        &init_css_set.cg_links);

+ for (i = 0; i < CSS_SET_TABLE_SIZE; i++)
+ INIT_HLIST_HEAD(&css_set_table[i]);
+
    for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
        struct cgroup_subsys *ss = subsys[i];

@@ -2544,6 +2574,7 @@ int __init cgroup_init(void)
    {
        int err;
        int i;
+ struct hlist_head *hhead;

        err = bdi_init(&cgroup_backing_dev_info);
        if (err)
@@ -2555,6 +2586,10 @@ int __init cgroup_init(void)
        cgroup_init_subsys(ss);
    }

+ /* Add init_css_set to the hash table */
+ hhead = css_set_hash(init_css_set.subsys);
+ hlist_add_head(&init_css_set.hlist, hhead);
+
    err = register_filesystem(&cgroup_fs_type);
    if (err < 0)
        goto out;
--
1.5.4.rc3

```

---

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

---