

---

Subject: Re: Re: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list

Posted by [dev](#) on Wed, 01 Nov 2006 10:48:06 GMT

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

---

Neil Brown wrote:

> On Monday October 30, [dev@sw.ru](mailto:dev@sw.ru) wrote:

>

>>David,

>>

>>

>>>>The proposed fix prevents this issue by using per-sb dentry LRU list. It  
>>>>provides very quickly search for the unused dentries for given super block thus  
>>>>forcing shrinking always making good progress.

>>>

>>>

>>>We've been down this path before:

>>>

>>> <http://marc.theaimsgroup.com/?l=linux-kernel&m=114861109717260&w=2>

>>>

>>>A lot of comments on per-sb unused dentry lists were made in  
>>>the threads associated with the above. other solutions were  
>>>found to the problem that the above patch addressed, but I don't  
>>>think any of them have made it to mainline yet. You might want  
>>>to have a bit of a read of these threads first...

>>

>>The major difference between our patch and the one discussed in the link  
>>it that we keep both global and per-sb dentry LRU lists.  
>>Thus, when needed normal LRU is used and prune logic is unchanged,  
>>while umount/remount use per-sb list and do its job faster.

>

>

> Yes, we have been down this path before - several times I think.

> Below is the patch that I like (not tested recently - just rediffed  
> and reviewed).

>

> NeilBrown

>

> Subject: Reduce contention in dentry\_unused when unmounting.

>

> When we unmount a filesystem we need to release all dentries.

> We currently

> - move a collection of dentries to the end of the dentry\_unused list

> - call prune\_dcache to prune that number of dentries.

>

> If lots of other dentries are added to the end of the list while  
> the prune\_dcache proceeds (e.g. another filesystem is unmounted),  
> this can involve a lot of wasted time wandering through the  
> list looking for dentries that we had previously found.

```

>
> This patch allows the dentry_unused list to temporarily be multiple
> lists.
> When unmounting, dentries that are found to require pruning are moved
> to a temporary list, but accounted as though they were on dentry_unused.
> Then this list is passed to prune_dcache for freeing. Any entries
> that are not pruned for whatever reason are added to the end of
> dentry_unused.
>
> Also change shrink_dcache_sb to simply call shrink_dcache_parent.
> This avoids a long walk of the LRU.
>
> Signed-off-by: Neil Brown <neilb@suse.de>
>
> ### Diffstat output
> ./fs/dcache.c | 104 ++++++-----
> 1 file changed, 30 insertions(+), 74 deletions(-)
>
> diff .prev/fs/dcache.c ./fs/dcache.c
> --- .prev/fs/dcache.c 2006-10-31 15:22:10.000000000 +1100
> +++ ./fs/dcache.c 2006-10-31 15:37:22.000000000 +1100
> @@ -384,8 +384,8 @@ static void prune_one_dentry(struct dent
> /**
>  * prune_dcache - shrink the dcache
>  * @count: number of entries to try and free
>  * @sb: if given, ignore dentries for other superblocks
>  *       which are being unmounted.
>  * @list: If given, remove from this list instead of
>  *        from dentry_unused.
>  *
>  * Shrink the dcache. This is done when we need
>  * more memory, or simply when we need to unmount
> @@ -394,11 +394,21 @@ static void prune_one_dentry(struct dent
>  *
>  * This function may fail to free any resources if
>  * all the dentries are in use.
> + *
> + * Any dentries that were not removed due to the @count
> + * limit will be splice on to the end of dentry_unused,
> + * so they should already be founded in dentry_stat.nr_unused.
>  */
>
> -static void prune_dcache(int count, struct super_block *sb)
> +static void prune_dcache(int count, struct list_head *list)
> {
> + int have_list = list != NULL;
> +
> + spin_lock(&dcache_lock);

```

```

> + if (!have_list)
> + /* use the dentry_unused list */
> + list = &dentry_unused;
> +
> for (; count ; count--) {
> struct dentry *dentry;
> struct list_head *tmp;
> @@ -406,23 +416,11 @@ static void prune_dcache(int count, stru
>
> cond_resched_lock(&dcache_lock);
>
> - tmp = dentry_unused.prev;
> - if (sb) {
> - /* Try to find a dentry for this sb, but don't try
> -  * too hard, if they aren't near the tail they will
> -  * be moved down again soon
> -  */
> - int skip = count;
> - while (skip && tmp != &dentry_unused &&
> - list_entry(tmp, struct dentry, d_lru)->d_sb != sb) {
> - skip--;
> - tmp = tmp->prev;
> - }
> - }
> - if (tmp == &dentry_unused)
> + tmp = list->prev;
> + if (tmp == list)
> break;
> list_del_init(tmp);
> - prefetch(dentry_unused.prev);
> + prefetch(list->prev);
> dentry_stat.nr_unused--;
> dentry = list_entry(tmp, struct dentry, d_lru);
>
> @@ -455,7 +453,7 @@ static void prune_dcache(int count, stru
>  * If this dentry is for "my" filesystem, then I can prune it
>  * without taking the s_umount lock (I already hold it).
>  */
> - if (sb && dentry->d_sb == sb) {
> + if (have_list) {
> prune_one_dentry(dentry);
> continue;
> }
> @@ -485,68 +483,25 @@ static void prune_dcache(int count, stru
> list_add(&dentry->d_lru, &dentry_unused);
> dentry_stat.nr_unused++;
> }
> + /* split any remaining entries back onto dentry_unused */

```

```

> + if (have_list)
> + list_splice(list, dentry_unused.prev);
> spin_unlock(&dcache_lock);
> }
>
> -/*
> - * Shrink the dcache for the specified super block.
> - * This allows us to unmount a device without disturbing
> - * the dcache for the other devices.
> - *
> - * This implementation makes just two traversals of the
> - * unused list. On the first pass we move the selected
> - * dentries to the most recent end, and on the second
> - * pass we free them. The second pass must restart after
> - * each dput(), but since the target dentries are all at
> - * the end, it's really just a single traversal.
> - */
> -
> /**
>  * shrink_dcache_sb - shrink dcache for a superblock
>  * @sb: superblock
>  *
>  * Shrink the dcache for the specified super block. This
>  * is used to free the dcache before unmounting a file
>  * system
>  * is used to reduce the dcache presence of a file system
>  * before re-mounting, and when invalidating the device
>  * holding a file system.
>  */
>
> void shrink_dcache_sb(struct super_block * sb)
> {
> - struct list_head *tmp, *next;
> - struct dentry *dentry;
> -
> - /*
> - * Pass one ... move the dentries for the specified
> - * superblock to the most recent end of the unused list.
> - */
> - spin_lock(&dcache_lock);
> - list_for_each_safe(tmp, next, &dentry_unused) {
> - dentry = list_entry(tmp, struct dentry, d_lru);
> - if (dentry->d_sb != sb)
> - continue;
> - list_move(tmp, &dentry_unused);
> - }
> -
> - /*

```

```

> - * Pass two ... free the dentries for this superblock.
> - */
> -repeat:
> - list_for_each_safe(tmp, next, &dentry_unused) {
> - dentry = list_entry(tmp, struct dentry, d_lru);
> - if (dentry->d_sb != sb)
> - continue;
> - dentry_stat.nr_unused--;
> - list_del_init(tmp);
> - spin_lock(&dentry->d_lock);
> - if (atomic_read(&dentry->d_count)) {
> - spin_unlock(&dentry->d_lock);
> - continue;
> - }
> - prune_one_dentry(dentry);
> - cond_resched_lock(&dcache_lock);
> - goto repeat;
> - }
> - spin_unlock(&dcache_lock);
> + shrink_dcache_parent(sb->s_root);
<<<< AFAICS, doing so you introduced a leak of anonymous dentries.

```

d\_alloc\_anon() calls d\_alloc() with parent == NULL, i.e. dentries have no parent and are not linked to the sb->s\_root...

BTW, looking at it, I found that s\_anon field on super block is not used any more.

we can add BUG\_ON(!hlist\_empty(&sb->s\_anon)) in generic\_shutdown\_super to avoid such issues like this.

maybe we can fix it adding something like:

```

while (!hlist_empty(&sb->s_anon))
    prune_dcache(MAX_INT, &sb->s_anon);

```

```

> }
>
> /*
> @@ -739,7 +694,7 @@ positive:
>
> /*
> * Search the dentry child list for the specified parent,
> - * and move any unused dentries to the end of the unused
> + * and move any unused dentries to the end of a new unused
> * list for prune_dcache(). We descend to the next level
> * whenever the d_subdirs list is non-empty and continue
> * searching.
> @@ -751,7 +706,7 @@ positive:
> * drop the lock and return early due to latency
> * constraints.
> */

```

```

> -static int select_parent(struct dentry * parent)
> +static int select_parent(struct dentry * parent, struct list_head *new)
> {
>     struct dentry *this_parent = parent;
>     struct list_head *next;
> @@ -775,7 +730,7 @@ resume:
>     * of the unused list for prune_dcache
>     */
>     if (!atomic_read(&dentry->d_count)) {
> - list_add_tail(&dentry->d_lru, &dentry_unused);
> + list_add_tail(&dentry->d_lru, new);
>     dentry_stat.nr_unused++;
>     found++;
> }
> @@ -819,9 +774,10 @@ out:
> void shrink_dcache_parent(struct dentry * parent)
> {
>     int found;
> + LIST_HEAD(list);
>
> - while ((found = select_parent(parent)) != 0)
> - prune_dcache(found, parent->d_sb);
> + while ((found = select_parent(parent, &list)) != 0)
> + prune_dcache(found, &list);
> }
>
> /*
>

```

---