Subject: [PATCH 2.6.19-rc3] VFS: per-sb dentry lru list
Posted by vaverin on Fri, 27 Oct 2006 14:05:50 GMT
View Forum Message <> Reply to Message

From: Vasily Averin <vvs@sw.ru>

Virtuozzo/OpenVZ linux kernel team has discovered that umount/remount can last for hours looping in shrink_dcache_sb() without much successes. Since during shrinking s_umount semaphore is taken lots of other unrelated operations like sync can stop working until shrink finished.

It happens due to very long unused dentries list (>1 million of dentries), so that it takes shrink_dcache_sb() longer then 1/HZ seconds to get to the required dentry and after freeing this single dentry it reschedules and restarts.

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.

It was well tested on 2.6.9-based Virtuozzo/OpenVZ kernels, but the port to the latest mainstream kernel is not tested.

Signed-off-by: Kirill Korotaev <dev@openvz.org>
Signed-off-by: Vasily Averin <vvs@sw.ru>

```
--- linux-2.6.19-rc3/fs/dcache.c.shrsb 2006-10-27 10:45:11.000000000 +0400
+++ linux-2.6.19-rc3/fs/dcache.c 2006-10-27 15:49:54.000000000 +0400
@@ -173,6 +173,7 @@ repeat:
   if (list_empty(&dentry->d_lru)) {
    dentry->d_flags |= DCACHE_REFERENCED;
    list_add(&dentry->d_lru, &dentry_unused);
+  list_add(&dentry->d_sb_lru, &dentry->d_sb->s_dentry_unused);
    dentry_stat.nr_unused++;
   }
  spin_unlock(&dentry->d_lock);
@@ -190,6 +191,7 @@ kill_it: {
   */
    if (!list_empty(&dentry->d_lru)) {
     list_del(&dentry->d_lru);
+   list_del(&dentry->d_sb_lru);
     dentry_stat.nr_unused--;
    }
    list_del(&dentry->d_u.d_child);
@@ -270,6 +272,7 @@ static inline struct dentry * __dget_loc
  if (!list_empty(&dentry->d_lru)) {
   dentry_stat.nr_unused--;
   list_del_init(&dentry->d_lru);
+  list_del_init(&dentry->d_sb_lru);
```

```
 }
 return dentry;
}
@@ -398,6 +401,10 @@ static void prune_one_dentry(struct dent

 static void prune_dcache(int count, struct super_block *sb)
 {
+ struct list_head *lru_head;
+
+ lru_head = sb ? &sb->s_dentry_unused : &dentry_unused;
+
  spin_lock(&dcache_lock);
  for (; count ; count--) {
   struct dentry *dentry;
@@ -406,25 +413,16 @@ 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 = lru_head->prev;
+  if (tmp == lru_head)
    break;
-  list_del_init(tmp);
-  prefetch(dentry_unused.prev);
+
+  prefetch(lru_head->prev);
   dentry_stat.nr_unused--;
-  dentry = list_entry(tmp, struct dentry, d_lru);
+  dentry = sb ? list_entry(tmp, struct dentry, d_sb_lru) :
+    list_entry(tmp, struct dentry, d_lru);
+  list_del_init(&dentry->d_lru);
+  list_del_init(&dentry->d_sb_lru);

   spin_lock(&dentry->d_lock);
   /*
```

```
@@ -440,6 +438,8 @@ static void prune_dcache(int count, stru
  if (dentry->d_flags & DCACHE_REFERENCED) {
   dentry->d_flags &= ~DCACHE_REFERENCED;
    list_add(&dentry->d_lru, &dentry_unused);
+   list_add(&dentry->d_sb_lru,
+     &dentry->d_sb->s_dentry_unused);
    dentry_stat.nr_unused++;
    spin_unlock(&dentry->d_lock);
   continue;
@@ -455,7 +455,8 @@ 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 (sb) {
+   BUG_ON(dentry->d_sb != sb);
   prune_one_dentry(dentry);
   continue;
  }
@@ -480,6 +481,8 @@ static void prune_dcache(int count, stru
  spin_unlock(&dentry->d_lock);
  /* Inserting dentry to tail of the list leads to cycle */
   list_add(&dentry->d_lru, &dentry_unused);
+  list_add(&dentry->d_sb_lru,
+    &dentry->d_sb->s_dentry_unused);
  dentry_stat.nr_unused++;
 }
 spin_unlock(&dcache_lock);
@@ -509,31 +512,15 @@ static void prune_dcache(int count, stru

 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);
- }
+ while (!list_empty(&sb->s_dentry_unused)) {
+  struct dentry *dentry;
```

```
- /*
-  * 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 = list_entry((&sb->s_dentry_unused)->next,
+     struct dentry, d_sb_lru);
   dentry_stat.nr_unused--;
-  list_del_init(tmp);
+  list_del_init(&dentry->d_lru);
+  list_del_init(&dentry->d_sb_lru);
   spin_lock(&dentry->d_lock);
   if (atomic_read(&dentry->d_count)) {
    spin_unlock(&dentry->d_lock);
@@ -541,7 +528,6 @@ repeat:
   }
   prune_one_dentry(dentry);
   cond_resched_lock(&dcache_lock);
-  goto repeat;
  }
  spin_unlock(&dcache_lock);
 }
@@ -563,6 +549,7 @@ static void shrink_dcache_for_umount_sub
  if (!list_empty(&dentry->d_lru)) {
   dentry_stat.nr_unused--;
   list_del_init(&dentry->d_lru);
+  list_del_init(&dentry->d_sb_lru);
  }
  __d_drop(dentry);
  spin_unlock(&dcache_lock);
@@ -580,6 +567,7 @@ static void shrink_dcache_for_umount_sub
     if (!list_empty(&loop->d_lru)) {
      dentry_stat.nr_unused--;
      list_del_init(&loop->d_lru);
+     list_del_init(&loop->d_sb_lru);
     }

     __d_drop(loop);
@@ -766,6 +754,7 @@ resume:
   if (!list_empty(&dentry->d_lru)) {
    dentry_stat.nr_unused--;
    list_del_init(&dentry->d_lru);
+   list_del_init(&dentry->d_sb_lru);
   }
```

```
   /*
    * move only zero ref count dentries to the end
@@ -773,6 +762,8 @@ resume:
    */
   if (!atomic_read(&dentry->d_count)) {
    list_add_tail(&dentry->d_lru, &dentry_unused);
+   list_add_tail(&dentry->d_sb_lru,
+     &dentry->d_sb->s_dentry_unused);
    dentry_stat.nr_unused++;
    found++;
   }
@@ -892,6 +883,7 @@ struct dentry *d_alloc(struct dentry * p
 #endif
  INIT_HLIST_NODE(&dentry->d_hash);
  INIT_LIST_HEAD(&dentry->d_lru);
+ INIT_LIST_HEAD(&dentry->d_sb_lru);
  INIT_LIST_HEAD(&dentry->d_subdirs);
  INIT_LIST_HEAD(&dentry->d_alias);

--- linux-2.6.19-rc3/fs/super.c.shrsb 2006-10-24 10:29:13.000000000 +0400
+++ linux-2.6.19-rc3/fs/super.c 2006-10-27 15:36:33.000000000 +0400
@@ -69,6 +69,7 @@ static struct super_block *alloc_super(s
   INIT_LIST_HEAD(&s->s_io);
   INIT_LIST_HEAD(&s->s_files);
   INIT_LIST_HEAD(&s->s_instances);
+  INIT_LIST_HEAD(&s->s_dentry_unused);
   INIT_HLIST_HEAD(&s->s_anon);
   INIT_LIST_HEAD(&s->s_inodes);
   init_rwsem(&s->s_umount);
--- linux-2.6.19-rc3/include/linux/dcache.h.shrsb 2006-10-24 10:29:14.000000000
+0400
+++ linux-2.6.19-rc3/include/linux/dcache.h 2006-10-27 15:36:33.000000000 +0400
@@ -94,6 +94,7 @@ struct dentry {
  struct qstr d_name;

  struct list_head d_lru;  /* LRU list */
+ struct list_head d_sb_lru; /* per-sb LRU list */
  /*
   * d_child and d_rcu can share memory
   */
--- linux-2.6.19-rc3/include/linux/fs.h.shrsb 2006-10-24 10:29:14.000000000 +0400
+++ linux-2.6.19-rc3/include/linux/fs.h 2006-10-27 15:36:33.000000000 +0400
@@ -939,6 +939,7 @@ struct super_block {
  struct list_head s_dirty; /* dirty inodes */
  struct list_head s_io;  /* parked for writeback */
  struct hlist_head s_anon;  /* anonymous dentries for (nfs) exporting */
+ struct list_head s_dentry_unused;
  struct list_head s_files;
```

struct block_device *s_bdev;