Subject: [PATCH 7/7] example shrinker for memcg-aware dcache
Posted by Glauber Costa on Tue, 21 Feb 2012 11:34:39 GMT
View Forum Message <> Reply to Message

This is an example of a memcg-aware shrinker for the dcache.
In the way it works today, it tries to not interfere much with
the existing shrinker. We can probably make a list of sb's per-memcg -
having the root memcg to hold them all, but this adds a lot
of complications along the way.

This version is simpler, and serves to pave the way to future
work.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Kirill A. Shutemov <kirill@shutemov.name>
CC: Greg Thelen <gthelen@google.com>
CC: Johannes Weiner <jweiner@redhat.com>
CC: Michal Hocko <mhocko@suse.cz>
CC: Hiroyouki Kamezawa <kamezawa.hiroyu@jp.fujitsu.com>
CC: Paul Turner <pjt@google.com>
CC: Frederic Weisbecker <fweisbec@gmail.com>
CC: Dave Chinner <david@fromorbit.com>
---
 fs/dcache.c | 98 ++
 include/linux/dcache.h | 1 +
 2 files changed, 99 insertions(+), 0 deletions(-)

diff --git a/fs/dcache.c b/fs/dcache.c
index a452c19..98a0490 100644
--- a/fs/dcache.c
+++ b/fs/dcache.c
@@ -234,6 +234,7 @@ static void dentry_lru_add(struct dentry *dentry)
  if (list_empty(&dentry->d_lru)) {
   spin_lock(&dcache_lru_lock);
   list_add(&dentry->d_lru, &dentry->d_sb->s_dentry_lru);
+ list_add(&dentry->d_memcg_lru, &dentry->d_cache->lru);
   dentry->d_cache->nr_objects++;
   dentry->d_sb->s_nr_dentry_unused++;
   dentry_stat.nr_unused++;
@@ -243,9 +244,12 @@ static void dentry_lru_add(struct dentry *dentry)

 static void __dentry_lru_del(struct dentry *dentry)
 {
+ WARN_ON(list_empty(&dentry->d_memcg_lru));
  list_del_init(&dentry->d_lru);
+ list_del_init(&dentry->d_memcg_lru);
  dentry->d_flags &= ~DCACHE_SHRINK_LIST;
  dentry->d_sb->s_nr_dentry_unused--;

```
+ dentry->d_cache->nr_objects--;
  dentry_stat.nr_unused--;
 }

@@ -283,10 +287,13 @@ static void dentry_lru_move_list(struct dentry *dentry, struct list_head
*list)
  spin_lock(&dcache_lru_lock);
  if (list_empty(&dentry->d_lru)) {
   list_add_tail(&dentry->d_lru, list);
+  list_add_tail(&dentry->d_memcg_lru, &dentry->d_cache->lru);
   dentry->d_sb->s_nr_dentry_unused++;
+  dentry->d_cache->nr_objects++;
   dentry_stat.nr_unused++;
  } else {
   list_move_tail(&dentry->d_lru, list);
+  list_move_tail(&dentry->d_memcg_lru, &dentry->d_cache->lru);
  }
  spin_unlock(&dcache_lru_lock);
 }
@@ -771,6 +778,96 @@ static void shrink_dentry_list(struct list_head *list)
  rcu_read_unlock();
 }

+static void dcache_shrink_dentry_list(struct list_head *list)
+{
+ struct dentry *dentry;
+
+ rcu_read_lock();
+ for (;;) {
+  dentry = list_entry_rcu(list->prev, struct dentry, d_memcg_lru);
+  if (&dentry->d_memcg_lru == list)
+   break; /* empty */
+
+  spin_lock(&dentry->d_lock);
+
+  WARN_ON(dentry->d_cache == &dentry_cache);
+
+  if (dentry != list_entry(list->prev, struct dentry, d_memcg_lru)) {
+   spin_unlock(&dentry->d_lock);
+   continue;
+  }
+
+  /*
+   * We found an inuse dentry which was not removed from
+   * the LRU because of laziness during lookup.  Do not free
+   * it - just keep it off the LRU list.
+   */
+  if (dentry->d_count) {
```

```
+   dentry_lru_del(dentry);
+   spin_unlock(&dentry->d_lock);
+   continue;
+  }
+
+  rcu_read_unlock();
+
+  try_prune_one_dentry(dentry);
+
+  rcu_read_lock();
+ }
+ rcu_read_unlock();
+}
+
+static int dcache_shrink_memcg(struct shrinker *shrink, struct shrink_control *sc)
+{
+ struct memcg_kmem_cache *kcache;
+ int count = sc->nr_to_scan;
+ struct dentry *dentry;
+ struct mem_cgroup *memcg;
+ LIST_HEAD(referenced);
+ LIST_HEAD(tmp);
+
+ memcg = sc->memcg;
+ kcache = memcg_cache_get(memcg, CACHE_DENTRY);
+ if (!count) {
+  return kcache->nr_objects;
+ }
+relock:
+ spin_lock(&dcache_lru_lock);
+ while (!list_empty(&kcache->lru)) {
+
+  dentry = list_entry(kcache->lru.prev, struct dentry, d_memcg_lru);
+
+  BUG_ON(dentry->d_cache != kcache);
+
+  if (!spin_trylock(&dentry->d_lock)) {
+   spin_unlock(&dcache_lru_lock);
+   cpu_relax();
+   goto relock;
+  }
+
+  if (dentry->d_flags & DCACHE_REFERENCED) {
+   dentry->d_flags &= ~DCACHE_REFERENCED;
+   list_move(&dentry->d_memcg_lru, &referenced);
+   spin_unlock(&dentry->d_lock);
+  } else {
+   list_move_tail(&dentry->d_memcg_lru, &tmp);
```

```
+   spin_unlock(&dentry->d_lock);
+   if (!--count)
+    break;
+  }
+
+  cond_resched_lock(&dcache_lru_lock);
+ }
+ if (!list_empty(&referenced))
+  list_splice(&referenced, &kcache->lru);
+
+ spin_unlock(&dcache_lru_lock);
+
+ dcache_shrink_dentry_list(&tmp);
+
+ return sc->nr_to_scan - count;
+}
+
 /**
  * prune_dcache_sb - shrink the dcache
  * @sb: superblock
@@ -1244,6 +1341,7 @@ struct dentry *__d_alloc(struct super_block *sb, const struct qstr
*name)
  dentry->d_cache = cache;
  INIT_HLIST_BL_NODE(&dentry->d_hash);
  INIT_LIST_HEAD(&dentry->d_lru);
+ INIT_LIST_HEAD(&dentry->d_memcg_lru);
  INIT_LIST_HEAD(&dentry->d_subdirs);
  INIT_LIST_HEAD(&dentry->d_alias);
  INIT_LIST_HEAD(&dentry->d_u.d_child);
diff --git a/include/linux/dcache.h b/include/linux/dcache.h
index 4d94b657..4af7be3 100644
--- a/include/linux/dcache.h
+++ b/include/linux/dcache.h
@@ -135,6 +135,7 @@ struct dentry {
  void *d_fsdata;   /* fs-specific data */

  struct list_head d_lru;  /* LRU list */
+ struct list_head d_memcg_lru; /* per-memcg LRU list */
  /*
   * d_child and d_rcu can share memory
   */
--
1.7.7.6
```