

[CFT] kmem_cache_alloc_node

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2003-11/4602.html>

From: Manfred Spraul (manfred_at_colorfullife.com)

Date: 11/29/03

Date: Sat, 29 Nov 2003 17:21:40 +0100
To: linux-kernel@vger.kernel.org

Hi,

I've written a prototype `kmem_cache_alloc_node` function: I'm not yet convinced that it's really necessary to guarantee that `kmalloc` and `kmem_cache_alloc` are strictly node-local (adds noticable costs to the hot paths, and many objects will be touched by multiple nodes anyway), but at least the `cpu` bound data structures should be allocated from the right node.

The attached patch adds a simple `kmem_cache_alloc_node` function and moves the `cpu` local structures within slab onto the right node.

One problem is the bootstrap: there is an `alloc_pages_node(,cpu_to_node())` during `CPU_UP_PREPARE`: Does that work, or is that too early?

Please test it, I don't have access to suitable hardware.

The patch also includes fixes for two accounting bugs in error paths, I'll send them seperately to Andrew.

--
Manfred

```
// $Header$
// Kernel Version:
// VERSION = 2
// PATCHLEVEL = 6
// SUBLEVEL = 0
// EXTRAVERSION = -test11
---- 2.6/include/linux/slab.h 2003-10-25 20:44:55.000000000 +0200
+++ build-2.6/include/linux/slab.h 2003-11-29 15:42:59.000000000 +0100
@@ -62,6 +62,7 @@
extern int kmem_cache_destroy(kmem_cache_t *);
extern int kmem_cache_shrink(kmem_cache_t *);
extern void *kmem_cache_alloc(kmem_cache_t *, int);
+extern void *kmem_cache_alloc_node(kmem_cache_t *, int);
extern void kmem_cache_free(kmem_cache_t *, void *);
extern unsigned int kmem_cache_size(kmem_cache_t *);
```

```

---- 2.6/mm/slab.c 2003-11-29 09:46:35.000000000 +0100
+++ build-2.6/mm/slab.c 2003-11-29 17:11:46.000000000 +0100
@@ -578,12 +578,26 @@
    }
}

-/*
- * Note: if someone calls kmem_cache_alloc() on the new
- * cpu before the cpuup callback had a chance to allocate
- * the head arrays, it will oops.
- * Is CPU_ONLINE early enough?
- */
+static struct array_cache *alloc_arraycache(int cpu, int entries, int batchcount)
+{
+ int memsize = sizeof(void*)*entries+sizeof(struct array_cache);
+ struct array_cache *nc = NULL;
+
+ if (cpu != -1) {
+ nc = kmem_cache_alloc_node(kmem_find_general_cachep(memsize, GFP_KERNEL),
+ cpu_to_node(cpu));
+ }
+ if (!nc)
+ nc = kmalloc(memsize, GFP_KERNEL);
+ if (nc) {
+ nc->avail = 0;
+ nc->limit = entries;
+ nc->batchcount = batchcount;
+ nc->touched = 0;
+ }
+ return nc;
+}
+
static int __devinit cpuup_callback(struct notifier_block *nfb,
unsigned long action,
void *hcpu)
@@ -595,25 +609,18 @@
    case CPU_UP_PREPARE:
        down(&cache_chain_sem);
        list_for_each(p, &cache_chain) {
- int memsize;
+ kmem_cache_t* cachep = list_entry(p, kmem_cache_t, next);
        struct array_cache *nc;

- kmem_cache_t* cachep = list_entry(p, kmem_cache_t, next);
- memsize = sizeof(void*)*cachep->limit+sizeof(struct array_cache);
- nc = kmalloc(memsize, GFP_KERNEL);
+ nc = alloc_arraycache(cpu, cachep->limit, cachep->batchcount);
        if (!nc)
            goto bad;

- nc->avail = 0;

```

Linux-Kernel: [CFT] kmem_cache_alloc_node

```

- nc->limit = cachep->limit;
- nc->batchcount = cachep->batchcount;
- nc->touched = 0;

    spin_lock_irq(&cachep->spinlock);
    cachep->array[cpu] = nc;
    cachep->free_limit = (1+num_online_cpus())*cachep->batchcount
        + cachep->num;
    spin_unlock_irq(&cachep->spinlock);
-
    }
    up(&cache_chain_sem);
    break;
@@ -800,24 +807,27 @@
 * did not request dmaable memory, we might get it, but that
 * would be relatively rare and ignorable.
 */
-static inline void *kmem_getpages(kmem_cache_t *cachep, unsigned long flags)
+static void *kmem_getpages(kmem_cache_t *cachep, int flags, int nodeid)
{
- void *addr;
+ struct page *page;

    flags |= cachep->gfpflags;
- if (cachep->flags & SLAB_RECLAIM_ACCOUNT)
- atomic_add(1 << cachep->gfporder, &slab_reclaim_pages);
- addr = (void*)__get_free_pages(flags, cachep->gfporder);
- if (addr) {
+ page = alloc_pages_node(nodeid, flags, cachep->gfporder);
+ if (page) {
    int i = (1 << cachep->gfporder);
- struct page *page = virt_to_page(addr);
+ void *ptr;

+ add_page_state(nr_slab, i);
+ if (cachep->flags & SLAB_RECLAIM_ACCOUNT)
+ atomic_add(i, &slab_reclaim_pages);
+ ptr = page_address(page);
    while (i-->0) {
        SetPageSlab(page);
        page++;
    }
+ return ptr;
    }
- return addr;
+ return NULL;
}

/*
@@ -1539,6 +1549,21 @@
}

```

```

}

+static void set_slab_attr(kmem_cache_t *cachep, struct slab *slabp, void *objp)
+{
+ int i;
+ struct page *page;
+
+ /* Nasty!!!!!! I hope this is OK. */
+ i = 1 << cachep->gfporder;
+ page = virt_to_page(objp);
+ do {
+ SET_PAGE_CACHE(page, cachep);
+ SET_PAGE_SLAB(page, slabp);
+ page++;
+ } while (--i);
+}
+
+/*
+ * Grow (by 1) the number of slabs within a cache. This is called by
+ * kmem_cache_alloc() when there are no active objs left in a cache.
+ @@ -1546,10 +1571,9 @@
+ static int cache_grow (kmem_cache_t * cachep, int flags)
+ {
+     struct slab *slabp;
- struct page *page;
+     void *objp;
+     size_t offset;
- unsigned int i, local_flags;
+ int local_flags;
+     unsigned long ctor_flags;

+     /* Be lazy and only check for valid flags here,
+ @@ -1593,25 +1617,15 @@
+     */
+     kmem_flagcheck(cachep, flags);

-
+     /* Get mem for the objs. */
- if (!(objp = kmem_getpages(cachep, flags)))
+ if (!(objp = kmem_getpages(cachep, flags, numa_node_id())))
+     goto failed;

+     /* Get slab management. */
+     if (!(slabp = alloc_slabmgmt(cachep, objp, offset, local_flags)))
+     goto opps1;

- /* Nasty!!!!!! I hope this is OK. */
- i = 1 << cachep->gfporder;
- page = virt_to_page(objp);
- do {
- SET_PAGE_CACHE(page, cachep);

```

```

- SET_PAGE_SLAB(page, slabp);
- inc_page_state(nr_slab);
- page++;
- } while (--i);
-
+ set_slab_attr(cachep, slabp, objp);
  cache_init_objs(cachep, slabp, ctor_flags);

  if (local_flags & __GFP_WAIT)
@@ -2074,6 +2088,82 @@
EXPORT_SYMBOL(kmem_cache_alloc);

/**
+ * kmem_cache_alloc_node - Allocate an object on the specified node
+ * @cachep: The cache to allocate from.
+ * @flags: See kmalloc().
+ * @nodeid: node number of the target node.
+ *
+ * Identical to kmem_cache_alloc, except that this function is slow
+ * and can sleep. And it will allocate memory on the given node, which
+ * can improve the performance for cpu bound structures.
+ */
+void *kmem_cache_alloc_node(kmem_cache_t *cachep, int nodeid)
+{
+ size_t offset;
+ void *objp;
+ struct slab *slabp;
+ kmem_bufctl_t next;
+
+ /* The main algorithms are not node aware, thus we have to cheat:
+ * We bypass all caches and allocate a new slab.
+ * The following code is a streamlined copy of cache_grow().
+ */
+
+ /* Get colour for the slab, and update the next value. */
+ spin_lock_irq(&cachep->spinlock);
+ offset = cachep->colour_next;
+ cachep->colour_next++;
+ if (cachep->colour_next >= cachep->colour)
+ cachep->colour_next = 0;
+ offset *= cachep->colour_off;
+ spin_unlock_irq(&cachep->spinlock);
+
+ /* Get mem for the objs. */
+ if (!(objp = kmem_getpages(cachep, GFP_KERNEL, nodeid)))
+ goto failed;
+
+ /* Get slab management. */
+ if (!(slabp = alloc_slabmgmt(cachep, objp, offset, GFP_KERNEL)))
+ goto opps1;
+

```


Linux-Kernel: [CFT] kmem_cache_alloc_node

```

    memset(&new.new,0,sizeof(new.new));
    for (i = 0; i < NR_CPUS; i++) {
- struct array_cache *ccnew;
-
- ccnew = kmalloc(sizeof(void*)*limit+
- sizeof(struct array_cache), GFP_KERNEL);
- if (!ccnew) {
+ new.new[i] = alloc_arraycache(i, limit, batchcount);
+ if (!new.new[i]) {
        for (i--; i >= 0; i--) kfree(new.new[i]);
        return -ENOMEM;
    }
- ccnew->avail = 0;
- ccnew->limit = limit;
- ccnew->batchcount = batchcount;
- ccnew->touched = 0;
- new.new[i] = ccnew;
    }
    new.cachep = cachep;

@@ -2310,14 +2391,9 @@
        spin_unlock_irq(&cachep->spinlock);
        kfree(ccold);
    }
- new_shared = kmalloc(sizeof(void*)*batchcount*shared+
- sizeof(struct array_cache), GFP_KERNEL);
+ new_shared = alloc_arraycache(-1, batchcount*shared, 0xbaadf00d);
    if (new_shared) {
        struct array_cache *old;
- new_shared->avail = 0;
- new_shared->limit = batchcount*shared;
- new_shared->batchcount = 0xbaadf00d;
- new_shared->touched = 0;

        spin_lock_irq(&cachep->spinlock);
        old = cachep->lists.shared;
---- 2.6/include/linux/page-flags.h 2003-10-25 20:44:06.000000000 +0200
+++ build-2.6/include/linux/page-flags.h 2003-11-29 14:45:57.000000000 +0100
@@ -133,6 +133,7 @@

#define inc_page_state(member) mod_page_state(member, 1UL)
#define dec_page_state(member) mod_page_state(member, 0UL - 1)
+#define add_page_state(member,delta) mod_page_state(member, (delta))
#define sub_page_state(member,delta) mod_page_state(member, 0UL - (delta))

```

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
 the body of a message to majordomo@vger.kernel.org
 More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>