

Re: cache limit

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2003-08/5747.html>

From: Takao Indoh (indou.takao_at_soft.fujitsu.com)

Date: 08/21/03

Date: Thu, 21 Aug 2003 09:49:45 +0900

To: linux-kernel@vger.kernel.org

Hi.

On Tue, 19 Aug 2003 00:39:49 -0400, "Anthony R." wrote:

>I would like to tune my kernel not to use as much memory for cache
>as it currently does. I have 2GB RAM, but when I am running one program
>that accesses a lot of files on my disk (like rsync), that program uses
>most of the cache, and other programs wind up swapping out. I'd prefer to
>have just rsync run slower because less of its data is cached, rather
>than have
>all my other programs run more slowly. rsync is not allocating memory,
>but the kernel is caching it at the expense of other programs.
>
>With 2GB on a system, I should never page out, but I consistently do and I
>need to tune the kernel to avoid that. Cache usage is around 1.4 GB!
>I never had this problem with earlier kernels. I've read a lot of comments
>where so-called experts poo-poo this problem, but it is real and
>repeatable and I am
>ready to take matters into my own hands to fix it. I am told the cache
>is replaced when
>another program needs more memory, so it shouldn't swap, but that is not
>the
>behaviour I am seeing.
>
>Can anyone help point me in the right direction?
>Do any kernel developers care about this?

I also have the same problem about pagecache. Pagecache become bigger until most of memory is used, and it influences the performance of other programs. It is a serious problem in the OLTP system. When the transaction increases, transaction processing stalls, because most of memory is used as pagecache and it takes many time to reclaim memory from pagecache. It causes the whole system's stalling. To avoid that, the limitation of pagecache is necessary.

Actually, in the system I constructed(RedHat AdvancedServer2.1, kernel 2.4.9based), the problem occurred due to pagecache. The system's maximum

Linux-Kernel: Re: cache limit

response time had to be less than 4 seconds, but owing to the pagecache, response time get uneven, and maximum time became 10 seconds. This trouble was solved by controlling pagecache using /proc/sys/vm/pagecache.

I made a patch to add new paramter /proc/sys/vm/pgcache-max. It controls maximum number of pages used as pagecache. An attached file is a mere test patch, so it may contain a bug or ugly code. Please let me know if there is an advice, comment, better implementation, and so on.

Thanks.

Takao Indoh
E-Mail : indou.takao@soft.fujitsu.com

```
diff -Nur linux-2.5.64/include/linux/gfp.h linux-2.5.64-new/include/linux/gfp.h
--- linux-2.5.64/include/linux/gfp.h Wed Mar 5 12:29:03 2003
+++ linux-2.5.64-new/include/linux/gfp.h Tue Apr 8 11:12:33 2003
@@ -18,6 +18,7 @@
#define __GFP_FS 0x80 /* Can call down to low-level FS? */
#define __GFP_COLD 0x100 /* Cache-cold page required */
#define __GFP_NOWARN 0x200 /* Suppress page allocation failure warning */
+#define __GFP_PGCACHE 0x400 /* Page-cache required */

#define GFP_ATOMIC (__GFP_HIGH)
#define GFP_NOIO (__GFP_WAIT)
diff -Nur linux-2.5.64/include/linux/mm.h linux-2.5.64-new/include/linux/mm.h
--- linux-2.5.64/include/linux/mm.h Wed Mar 5 12:28:56 2003
+++ linux-2.5.64-new/include/linux/mm.h Tue Apr 8 11:12:33 2003
@@ -22,6 +22,7 @@
extern unsigned long num_physpages;
extern void * high_memory;
extern int page_cluster;
+extern unsigned long max_pgcache;

#include <asm/page.h>
#include <asm/pgtable.h>
diff -Nur linux-2.5.64/include/linux/page-flags.h linux-2.5.64-new/include/linux/page-flags.h
--- linux-2.5.64/include/linux/page-flags.h Wed Mar 5 12:29:31 2003
+++ linux-2.5.64-new/include/linux/page-flags.h Tue Apr 8 11:12:33 2003
@@ -74,6 +74,7 @@
#define PG_mappedtodisk 17 /* Has blocks allocated on-disk */
#define PG_reclaim 18 /* To be reclaimed asap */
#define PG_compound 19 /* Part of a compound page */
+#define PG_pgcache 20 /* Page is used as pagecache */

/*
 * Global page accounting. One instance per CPU. Only unsigned longs are
@@ -255,6 +256,10 @@
#define PageCompound(page) test_bit(PG_compound, &(page)->flags)
```

Re: cache limit

Linux-Kernel: Re: cache limit

```
#define SetPageCompound(page) set_bit(PG_compound, &(page)->flags)
#define ClearPageCompound(page) clear_bit(PG_compound, &(page)->flags)
+
+##define PagePgcache(page) test_bit(PG_pgcache, &(page)->flags)
+##define SetPagePgcache(page) set_bit(PG_pgcache, &(page)->flags)
+##define ClearPagePgcache(page) clear_bit(PG_pgcache, &(page)->flags)

/*
 * The PageSwapCache predicate doesn't use a PG_flag at this time,
diff -Nur linux-2.5.64/include/linux/pagemap.h linux-2.5.64-new/include/linux/pagemap.h
--- linux-2.5.64/include/linux/pagemap.h Wed Mar 5 12:28:53 2003
+++ linux-2.5.64-new/include/linux/pagemap.h Tue Apr 8 11:12:33 2003
@@ -29,12 +29,12 @@

static inline struct page *page_cache_alloc(struct address_space *x)
{
- return alloc_pages(x->gfp_mask, 0);
+ return alloc_pages(x->gfp_mask|__GFP_PGCACHE, 0);
}

static inline struct page *page_cache_alloc_cold(struct address_space *x)
{
- return alloc_pages(x->gfp_mask|__GFP_COLD, 0);
+ return alloc_pages(x->gfp_mask|__GFP_COLD|__GFP_PGCACHE, 0);
}

typedef int filler_t(void *, struct page *);
@@ -80,6 +80,7 @@
    list_add(&page->list, &mapping->clean_pages);
    page->mapping = mapping;
    page->index = index;
+ SetPagePgcache(page);

    mapping->npages++;
    inc_page_state(nr_pagecache);
diff -Nur linux-2.5.64/include/linux/sysctl.h linux-2.5.64-new/include/linux/sysctl.h
--- linux-2.5.64/include/linux/sysctl.h Wed Mar 5 12:29:21 2003
+++ linux-2.5.64-new/include/linux/sysctl.h Tue Apr 8 11:12:33 2003
@@ -155,6 +155,7 @@
    VM_HUGETLB_PAGES=18, /* int: Number of available Huge Pages */
    VM_SWAPPINESS=19, /* Tendency to steal mapped memory */
    VM_LOWER_ZONE_PROTECTION=20, /* Amount of protection of lower zones */
+ VM_MAXPGCACHE=21, /* maximum number of page used as pagecache */
};

diff -Nur linux-2.5.64/kernel/sysctl.c linux-2.5.64-new/kernel/sysctl.c
--- linux-2.5.64/kernel/sysctl.c Wed Mar 5 12:28:58 2003
+++ linux-2.5.64-new/kernel/sysctl.c Tue Apr 8 11:12:33 2003
@@ -319,6 +319,8 @@
    &sysctl_lower_zone_protection, sizeof(sysctl_lower_zone_protection),
```

Linux-Kernel: Re: cache limit

```
    0644, NULL, &proc_dointvec_minmax, &sysctl_intvec, NULL, &zero,
    NULL, },
+ {VM_MAXPGCACHE, "pgcache-max", &max_pgcache, sizeof(unsigned long),
+ 0644, NULL, &proc_dointvec_minmax, &sysctl_intvec, NULL, &zero, NULL},
    {0}
};
```

```
diff -Nur linux-2.5.64/mm/filemap.c linux-2.5.64-new/mm/filemap.c
```

```
--- linux-2.5.64/mm/filemap.c Wed Mar 5 12:29:15 2003
```

```
+++ linux-2.5.64-new/mm/filemap.c Tue Apr 8 11:12:33 2003
```

```
@@ -86,6 +86,7 @@
```

```
    radix_tree_delete(&mapping->page_tree, page->index);
    list_del(&page->list);
    page->mapping = NULL;
+ ClearPagePgcache(page);
```

```
    mapping->nrpages--;
    dec_page_state(nr_pagecache);
```

```
@@ -437,7 +438,7 @@
```

```
    page = find_lock_page(mapping, index);
    if (!page) {
        if (!cached_page) {
- cached_page = alloc_page(gfp_mask);
+ cached_page = alloc_page(gfp_mask|__GFP_PGCACHE);
            if (!cached_page)
                return NULL;
```

```
    }
```

```
@@ -507,7 +508,7 @@
```

```
    return NULL;
}
    gfp_mask = mapping->gfp_mask & ~__GFP_FS;
- page = alloc_pages(gfp_mask, 0);
+ page = alloc_pages(gfp_mask|__GFP_PGCACHE, 0);
    if (page && add_to_page_cache_lru(page, mapping, index, gfp_mask)) {
        page_cache_release(page);
        page = NULL;
```

```
diff -Nur linux-2.5.64/mm/page_alloc.c linux-2.5.64-new/mm/page_alloc.c
```

```
--- linux-2.5.64/mm/page_alloc.c Wed Mar 5 12:28:58 2003
```

```
+++ linux-2.5.64-new/mm/page_alloc.c Tue Apr 8 17:21:03 2003
```

```
@@ -39,6 +39,7 @@
```

```
int nr_swap_pages;
int numnodes = 1;
int sysctl_lower_zone_protection = 0;
+unsigned long max_pgcache = ULONG_MAX;
```

```
/*
```

```
 * Used by page_zone() to look up the address of the struct zone whose
```

```
@@ -52,6 +53,9 @@
```

```
static int zone_balance_min[MAX_NR_ZONES] __initdata = { 20, 20, 20, };
static int zone_balance_max[MAX_NR_ZONES] __initdata = { 255, 255, 255, };
```

Linux-Kernel: Re: cache limit

```
+extern int shrink_pgcache(struct zonelist *zonelist, unsigned int gfp_mask,
+ unsigned int max_nrpage, struct page_state *ps);
+
+/*
+ * Temporary debugging check for pages not lying within a given zone.
+ */
@@ -548,6 +552,19 @@
+   classzone = zones[0];
+   if (classzone == NULL) /* no zones in the zonelist */
+       return NULL;
+
+
+ if (gfp_mask & __GFP_PGCACHE) {
+ struct page_state ps;
+ int nr_page;
+
+ min = 1UL << order;
+ get_page_state(&ps);
+ if (ps.nr_pagecache + min >= max_pgcache) {
+ /* try to shrink pagecache */
+ nr_page = ps.nr_pagecache + min - max_pgcache;
+ shrink_pgcache(zonelist, gfp_mask, nr_page, &ps);
+ }
+ }

+   /* Go through the zonelist once, looking for a zone with enough free */
+   min = 1UL << order;
diff -Nur linux-2.5.64/mm/swap_state.c linux-2.5.64-new/mm/swap_state.c
--- linux-2.5.64/mm/swap_state.c Wed Mar 5 12:29:17 2003
+++ linux-2.5.64-new/mm/swap_state.c Tue Apr 8 11:12:33 2003
@@ -360,7 +360,7 @@
+   * Get a new page to read into from swap.
+   */
+   if (!new_page) {
- new_page = alloc_page(GFP_HIGHUSER);
+ new_page = alloc_page(GFP_HIGHUSER|__GFP_PGCACHE);
+   if (!new_page)
+       break; /* Out of memory */
+   }
diff -Nur linux-2.5.64/mm/vmscan.c linux-2.5.64-new/mm/vmscan.c
--- linux-2.5.64/mm/vmscan.c Wed Mar 5 12:28:59 2003
+++ linux-2.5.64-new/mm/vmscan.c Tue Apr 8 11:12:33 2003
@@ -493,6 +493,13 @@
+       list_add(&page->lru, &zone->inactive_list);
+       continue;
+   }
+ if (gfp_mask & __GFP_PGCACHE) {
+ if (!PagePgcache(page)) {
+ SetPageLRU(page);
+ list_add(&page->lru, &zone->inactive_list);
+ continue;
+ }
+ }
```

Linux-Kernel: Re: cache limit

```
+ }
        list_add(&page->lru, &page_list);
        page_cache_get(page);
        nr_taken++;
@@ -737,6 +744,40 @@
    }
    return shrink_cache(nr_pages, zone, gfp_mask,
                       max_scan, nr_mapped);
+}
+
+/*
+ * Try to reclaim `nr_pages' from pagecache of this zone.
+ * Returns the number of reclaimed pages.
+ */
+int shrink_pgcache(struct zonelist *zonelist, unsigned int gfp_mask,
+signed int nr_pages, struct page_state *ps)
+{
+ struct zone **zones;
+ struct zone *first_classzone;
+ struct zone *zone;
+ unsigned int ret = 0, reclaim;
+ unsigned long rest_nr_page;
+ int dummy, i;
+
+ zones = zonelist->zones;
+ for (i = 0; zones[i] != NULL; i++) {
+ zone = zones[i];
+ first_classzone = zone->zone_pgdat->node_zones;
+ for (; zone >= first_classzone; zone--) {
+ if (zone->all_unreclaimable) /* all pages pinned */
+ continue;
+
+ rest_nr_page = nr_pages - ret;
+ reclaim = max(((zone->nr_inactive)>>2)+1, rest_nr_page);
+ ret += shrink_zone(zone, zone->nr_inactive,
+ gfp_mask|__GFP_PGCACHE,
+ reclaim, &dummy, ps, DEF_PRIORITY);
+ if (ret >= nr_pages)
+ return ret;
+ }
+ }
+ return ret;
+ }
+
+/*
```

-
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/>