

[PATCH 11/14] page-replace-move-refill.patch

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-12/msg08501.html>

- *From:* Peter Zijlstra <a.p.zijlstra@xxxxxxxxxx>
 - *Date:* Fri, 30 Dec 2005 23:42:04 +0100
-

From: Peter Zijlstra <a.p.zijlstra@xxxxxxxxxx>

Move refill_inactive_page() to the new mm/page_replace.c file. And couple its invocation to the: page_replace_candidate() function. Keep the scan-rate equal to nr_active/nr_inactive. Also make sure we only scan in full multiples of swap_cluster_max.

Kudos to Wu Fengguang for showing me the way to decouple the active and inactive scans.

Signed-off-by: Peter Zijlstra <a.p.zijlstra@xxxxxxxxxx>

```
include/linux/mm_page_replace.h | 2
include/linux/mmzone.h | 1
mm/page_alloc.c | 4
mm/page_replace.c | 180 +++++
mm/vmscan.c | 196 +-
-----
5 files changed, 189 insertions(+), 194 deletions(-)
```

Index: linux-2.6-git/mm/page_replace.c

```
-----
--- linux-2.6-git.orig/mm/page_replace.c
+++ linux-2.6-git/mm/page_replace.c
@@ -1,6 +1,27 @@
#include <linux/mm_page_replace.h>
#include <linux/swap.h>
#include <linux/pagevec.h>
+#include <linux/page-flags.h>
+#include <linux/init.h>
+#include <linux/rmap.h>
+#include <linux/buffer_head.h> /* for try_to_release_page(),
+ buffer_heads_over_limit */
+
+ /*
+ * From 0 .. 100. Higher means more swappy.
+ */
+int vm_swappiness = 60;
+static long total_memory;
```

[PATCH 11/14] page-replace-move-refill.patch

```
+
+static void refill_inactive_zone(struct zone *, int);
+
+static int __init page_replace_init(void)
+{
+ total_memory = nr_free_pagecache_pages();
+ return 0;
+}
+
+module_init(page_replace_init)

static inline void
add_page_to_inactive_list(struct zone *zone, struct page *page)
@@ -34,8 +55,8 @@ void __page_replace_insert(struct zone *
*
* returns how many pages were moved onto *@dst.
*/
-int isolate_lru_pages(int nr_to_scan, struct list_head *src,
- struct list_head *dst, int *scanned)
+static int isolate_lru_pages(int nr_to_scan, struct list_head *src,
+ struct list_head *dst, int *scanned)
{
int nr_taken = 0;
struct page *page;
@@ -70,6 +91,7 @@ void page_replace_candidates(struct zone
{
int nr_taken;
int nr_scan;
+ unsigned long long nr_scan_active;

spin_lock_irq(&zone->lru_lock);
nr_taken = isolate_lru_pages(nr_to_scan, &zone->inactive_list,
@@ -82,6 +104,18 @@ void page_replace_candidates(struct zone
mod_page_state_zone(zone, pgscan_kswapd, nr_scan);
else
mod_page_state_zone(zone, pgscan_direct, nr_scan);
+
+ /*
+ * Add one to `nr_to_scan' just to make sure that the kernel will
+ * slowly sift through the active list.
+ */
+ nr_scan_active = (nr_scan + 1ULL) * zone->nr_active * 1024ULL;
+ do_div(nr_scan_active, zone->nr_inactive + nr_taken + 1UL);
+ zone->nr_scan_active += nr_scan_active;
+ while (zone->nr_scan_active >= SWAP_CLUSTER_MAX * 1024UL) {
+ zone->nr_scan_active -= SWAP_CLUSTER_MAX * 1024UL;
+ refill_inactive_zone(zone, SWAP_CLUSTER_MAX);
+ }
+ }
}

/*
```

[PATCH 11/14] page-replace-move-refill.patch

```
@@ -112,3 +146,145 @@ void page_replace_reinsert(struct zone *
pagevec_release(&pvec);
}

+/*
+ * This moves pages from the active list to the inactive list.
+ *
+ * We move them the other way if the page is referenced by one or more
+ * processes, from rmap.
+ *
+ * If the pages are mostly unmapped, the processing is fast and it is
+ * appropriate to hold zone->lru_lock across the whole operation. But if
+ * the pages are mapped, the processing is slow (page_referenced()) so we
+ * should drop zone->lru_lock around each page. It's impossible to balance
+ * this, so instead we remove the pages from the LRU while processing them.
+ * It is safe to rely on PG_active against the non-LRU pages in here because
+ * nobody will play with that bit on a non-LRU page.
+ *
+ * The downside is that we have to touch page->_count against each page.
+ * But we had to alter page->flags anyway.
+ */
+static void refill_inactive_zone(struct zone *zone, int nr_pages)
+{
+ int pgmoved;
+ int pgdeactivate = 0;
+ int pgscanned;
+ LIST_HEAD(l_hold); /* The pages which were snipped off */
+ LIST_HEAD(l_inactive); /* Pages to go onto the inactive_list */
+ LIST_HEAD(l_active); /* Pages to go onto the active_list */
+ struct page *page;
+ struct pagevec pvec;
+ int reclaim_mapped = 0;
+ long mapped_ratio;
+ long distress;
+ long swap_tendency;
+
+ lru_add_drain();
+ spin_lock_irq(&zone->lru_lock);
+ pgmoved = isolate_lru_pages(nr_pages, &zone->active_list,
+ &l_hold, &pgscanned);
+ zone->pages_scanned += pgscanned;
+ zone->nr_active -= pgmoved;
+ spin_unlock_irq(&zone->lru_lock);
+
+ /*
+ * `distress' is a measure of how much trouble we're having reclaiming
+ * pages. 0 -> no problems. 100 -> great trouble.
+ */
+ distress = 100 >> zone->prev_priority;
+
+ /*
```

[PATCH 11/14] page-replace-move-refill.patch

```
+ * The point of this algorithm is to decide when to start reclaiming
+ * mapped memory instead of just pagecache. Work out how much memory
+ * is mapped.
+ */
+ mapped_ratio = (read_page_state(nr_mapped) * 100) / total_memory;
+
+ /*
+ * Now decide how much we really want to unmap some pages. The mapped
+ * ratio is downgraded – just because there's a lot of mapped memory
+ * doesn't necessarily mean that page reclaim isn't succeeding.
+ *
+ * The distress ratio is important – we don't want to start going oom.
+ *
+ * A 100% value of vm_swappiness overrides this algorithm altogether.
+ */
+ swap_tendency = mapped_ratio / 2 + distress + vm_swappiness;
+
+ /*
+ * Now use this metric to decide whether to start moving mapped memory
+ * onto the inactive list.
+ */
+ if (swap_tendency >= 100)
+ reclaim_mapped = 1;
+
+ while (!list_empty(&l_hold)) {
+ cond_resched();
+ page = lru_to_page(&l_hold);
+ list_del(&page->lru);
+ if (page_mapped(page)) {
+ if (!reclaim_mapped ||
+ (total_swap_pages == 0 && PageAnon(page)) ||
+ page_referenced(page, 0)) {
+ list_add(&page->lru, &l_active);
+ continue;
+ }
+ }
+ list_add(&page->lru, &l_inactive);
+ }
+
+ pagevec_init(&pvec, 1);
+ pgmoved = 0;
+ spin_lock_irq(&zone->lru_lock);
+ while (!list_empty(&l_inactive)) {
+ page = lru_to_page(&l_inactive);
+ prefetchw_prev_lru_page(page, &l_inactive, flags);
+ if (TestSetPageLRU(page))
+ BUG();
+ if (!TestClearPageActive(page))
+ BUG();
+ list_move(&page->lru, &zone->inactive_list);
+ pgmoved++;
+ }
```

[PATCH 11/14] page-replace-move-refill.patch

```
+ if (!pagevec_add(&pvec, page)) {
+ zone->nr_inactive += pgmoved;
+ spin_unlock_irq(&zone->lru_lock);
+ pgdeactivate += pgmoved;
+ pgmoved = 0;
+ if (buffer_heads_over_limit)
+ pagevec_strip(&pvec);
+ __pagevec_release(&pvec);
+ spin_lock_irq(&zone->lru_lock);
+ }
+ }
+ zone->nr_inactive += pgmoved;
+ pgdeactivate += pgmoved;
+ if (buffer_heads_over_limit) {
+ spin_unlock_irq(&zone->lru_lock);
+ pagevec_strip(&pvec);
+ spin_lock_irq(&zone->lru_lock);
+ }
+
+ pgmoved = 0;
+ while (!list_empty(&l_active)) {
+ page = lru_to_page(&l_active);
+ prefetchw_prev_lru_page(page, &l_active, flags);
+ if (TestSetPageLRU(page))
+ BUG();
+ BUG_ON(!PageActive(page));
+ list_move(&page->lru, &zone->active_list);
+ pgmoved++;
+ if (!pagevec_add(&pvec, page)) {
+ zone->nr_active += pgmoved;
+ pgmoved = 0;
+ spin_unlock_irq(&zone->lru_lock);
+ __pagevec_release(&pvec);
+ spin_lock_irq(&zone->lru_lock);
+ }
+ }
+ zone->nr_active += pgmoved;
+ spin_unlock_irq(&zone->lru_lock);
+ pagevec_release(&pvec);
+
+ mod_page_state_zone(zone, pgrefill, pgscanned);
+ mod_page_state(pgdeactivate, pgdeactivate);
+}
+
Index: linux-2.6-git/mm/vmscan.c
```

```
-----
--- linux-2.6-git.orig/mm/vmscan.c
+++ linux-2.6-git/mm/vmscan.c
@@ -103,12 +103,6 @@ struct shrinker {
long nr; /* objs pending delete */
};
```

```

_/*
- * From 0 .. 100. Higher means more swappy.
- */
-int vm_swappiness = 60;
-static long total_memory;
-
-static LIST_HEAD(shrinker_list);
-static DECLARE_RWSEM(shrinker_rwsem);

@@ -590,200 +584,31 @@ static void shrink_cache(struct zone *zo
}

_/*
- * This moves pages from the active list to the inactive list.
- *
- * We move them the other way if the page is referenced by one or more
- * processes, from rmap.
- *
- * If the pages are mostly unmapped, the processing is fast and it is
- * appropriate to hold zone->lru_lock across the whole operation. But if
- * the pages are mapped, the processing is slow (page_referenced()) so we
- * should drop zone->lru_lock around each page. It's impossible to balance
- * this, so instead we remove the pages from the LRU while processing them.
- * It is safe to rely on PG_active against the non-LRU pages in here because
- * nobody will play with that bit on a non-LRU page.
- *
- * The downside is that we have to touch page->_count against each page.
- * But we had to alter page->flags anyway.
- */
-static void
-refill_inactive_zone(struct zone *zone, int nr_pages)
-{
- int pgmoved;
- int pgdeactivate = 0;
- int pgscanned;
- LIST_HEAD(l_hold); /* The pages which were snipped off */
- LIST_HEAD(l_inactive); /* Pages to go onto the inactive_list */
- LIST_HEAD(l_active); /* Pages to go onto the active_list */
- struct page *page;
- struct pagevec pvec;
- int reclaim_mapped = 0;
- long mapped_ratio;
- long distress;
- long swap_tendency;
-
- lru_add_drain();
- spin_lock_irq(&zone->lru_lock);
- pgmoved = isolate_lru_pages(nr_pages, &zone->active_list,
- &l_hold, &pgscanned);
- zone->pages_scanned += pgscanned;

```

[PATCH 11/14] page-replace-move-refill.patch

```
- zone->nr_active -= pgmoved;
- spin_unlock_irq(&zone->lru_lock);
-
- /*
- * `distress' is a measure of how much trouble we're having reclaiming
- * pages. 0 -> no problems. 100 -> great trouble.
- */
- distress = 100 >> zone->prev_priority;
-
- /*
- * The point of this algorithm is to decide when to start reclaiming
- * mapped memory instead of just pagecache. Work out how much memory
- * is mapped.
- */
- mapped_ratio = (read_page_state(nr_mapped) * 100) / total_memory;
-
- /*
- * Now decide how much we really want to unmap some pages. The mapped
- * ratio is downgraded - just because there's a lot of mapped memory
- * doesn't necessarily mean that page reclaim isn't succeeding.
- *
- * The distress ratio is important - we don't want to start going oom.
- *
- * A 100% value of vm_swappiness overrides this algorithm altogether.
- */
- swap_tendency = mapped_ratio / 2 + distress + vm_swappiness;
-
- /*
- * Now use this metric to decide whether to start moving mapped memory
- * onto the inactive list.
- */
- if (swap_tendency >= 100)
- reclaim_mapped = 1;
-
- while (!list_empty(&l_hold)) {
- cond_resched();
- page = lru_to_page(&l_hold);
- list_del(&page->lru);
- if (page_mapped(page)) {
- if (!reclaim_mapped ||
- (total_swap_pages == 0 && PageAnon(page)) ||
- page_referenced(page, 0)) {
- list_add(&page->lru, &l_active);
- continue;
- }
- }
- list_add(&page->lru, &l_inactive);
- }
-
- pagevec_init(&pvec, 1);
- pgmoved = 0;
```

[PATCH 11/14] page-replace-move-refill.patch

```
- spin_lock_irq(&zone->lru_lock);
- while (!list_empty(&l_inactive)) {
- page = lru_to_page(&l_inactive);
- prefetchw_prev_lru_page(page, &l_inactive, flags);
- if (TestSetPageLRU(page))
- BUG();
- if (!TestClearPageActive(page))
- BUG();
- list_move(&page->lru, &zone->inactive_list);
- pgmoved++;
- if (!pagevec_add(&pvec, page)) {
- zone->nr_inactive += pgmoved;
- spin_unlock_irq(&zone->lru_lock);
- pgdeactivate += pgmoved;
- pgmoved = 0;
- if (buffer_heads_over_limit)
- pagevec_strip(&pvec);
- __pagevec_release(&pvec);
- spin_lock_irq(&zone->lru_lock);
- }
- }
- zone->nr_inactive += pgmoved;
- pgdeactivate += pgmoved;
- if (buffer_heads_over_limit) {
- spin_unlock_irq(&zone->lru_lock);
- pagevec_strip(&pvec);
- spin_lock_irq(&zone->lru_lock);
- }
-
- pgmoved = 0;
- while (!list_empty(&l_active)) {
- page = lru_to_page(&l_active);
- prefetchw_prev_lru_page(page, &l_active, flags);
- if (TestSetPageLRU(page))
- BUG();
- BUG_ON(!PageActive(page));
- list_move(&page->lru, &zone->active_list);
- pgmoved++;
- if (!pagevec_add(&pvec, page)) {
- zone->nr_active += pgmoved;
- pgmoved = 0;
- spin_unlock_irq(&zone->lru_lock);
- __pagevec_release(&pvec);
- spin_lock_irq(&zone->lru_lock);
- }
- }
- zone->nr_active += pgmoved;
- spin_unlock_irq(&zone->lru_lock);
- pagevec_release(&pvec);
-
- mod_page_state_zone(zone, pgrefill, pgscanned);
```

[PATCH 11/14] page-replace-move-refill.patch

```
- mod_page_state(pgdeactivate, pgdeactivate);
-}
-
-/*
* This is a basic per-zone page freer. Used by both kswapd and direct reclaim.
*/
static void
shrink_zone(struct zone *zone, struct scan_control *sc)
{
- unsigned long nr_active;
unsigned long nr_inactive;

atomic_inc(&zone->reclaim_in_progress);

- /*
- * Add one to `nr_to_scan' just to make sure that the kernel will
- * slowly sift through the active list.
- */
- zone->nr_scan_active += (zone->nr_active >> sc->priority) + 1;
- nr_active = zone->nr_scan_active;
- if (nr_active >= sc->swap_cluster_max)
- zone->nr_scan_active = 0;
- else
- nr_active = 0;
-
- zone->nr_scan_inactive += (zone->nr_inactive >> sc->priority) + 1;
- nr_inactive = zone->nr_scan_inactive;
- if (nr_inactive >= sc->swap_cluster_max)
- zone->nr_scan_inactive = 0;
- else
- nr_inactive = 0;
+ nr_inactive = (zone->nr_inactive >> sc->priority) + SWAP_CLUSTER_MAX;
+ nr_inactive &= ~(SWAP_CLUSTER_MAX - 1);

+ sc->nr_to_scan = SWAP_CLUSTER_MAX;
sc->nr_to_reclaim = sc->swap_cluster_max;

- while (nr_active || nr_inactive) {
- if (nr_active) {
- sc->nr_to_scan = min(nr_active,
- (unsigned long)sc->swap_cluster_max);
- nr_active -= sc->nr_to_scan;
- refill_inactive_zone(zone, sc->nr_to_scan);
- }
-
- if (nr_inactive) {
- sc->nr_to_scan = min(nr_inactive,
- (unsigned long)sc->swap_cluster_max);
- nr_inactive -= sc->nr_to_scan;
- shrink_cache(zone, sc);
- if (sc->nr_to_reclaim <= 0)
```

```

- break;
- }
+ while (nr_inactive >= SWAP_CLUSTER_MAX) {
+ nr_inactive -= SWAP_CLUSTER_MAX;
+ shrink_cache(zone, sc);
+ if (sc->nr_to_reclaim <= 0)
+ break;
+ }

- throttle_vm_writeout();
-
atomic_dec(&zone->reclaim_in_progress);
+
+ throttle_vm_writeout();
+ }

/*
@@ -1245,7 +1070,6 @@ static int __init kswapd_init(void)
for_each_pgdat(pgdat)
pgdat->kswapd
= find_task_by_pid(kernel_thread(kswapd, pgdat, CLONE_KERNEL));
- total_memory = nr_free_pagecache_pages();
hotcpu_notifier(cpu_callback, 0);
return 0;
}
Index: linux-2.6-git/include/linux/mm_page_replace.h
=====
--- linux-2.6-git.orig/include/linux/mm_page_replace.h
+++ linux-2.6-git/include/linux/mm_page_replace.h
@@ -45,8 +45,6 @@ static inline void page_replace_activate
}
void page_replace_reinsert(struct zone *, struct list_head *);

-int isolate_lru_pages(int, struct list_head *, struct list_head *, int *);
-
static inline void
add_page_to_active_list(struct zone *zone, struct page *page)
{
Index: linux-2.6-git/include/linux/mmzone.h
=====
--- linux-2.6-git.orig/include/linux/mmzone.h
+++ linux-2.6-git/include/linux/mmzone.h
@@ -144,7 +144,6 @@ struct zone {
struct list_head active_list;
struct list_head inactive_list;
unsigned long nr_scan_active;
- unsigned long nr_scan_inactive;
unsigned long nr_active;
unsigned long nr_inactive;
unsigned long pages_scanned; /* since last reclaim */
Index: linux-2.6-git/mm/page_alloc.c

```

```
-----  
--- linux-2.6-git.orig/mm/page_alloc.c  
+++ linux-2.6-git/mm/page_alloc.c  
@@ -2010,7 +2010,6 @@ static void __init free_area_init_core(s  
INIT_LIST_HEAD(&zone->active_list);  
INIT_LIST_HEAD(&zone->inactive_list);  
zone->nr_scan_active = 0;  
- zone->nr_scan_inactive = 0;  
zone->nr_active = 0;  
zone->nr_inactive = 0;  
atomic_set(&zone->reclaim_in_progress, 0);  
@@ -2161,7 +2160,7 @@ static int zoneinfo_show(struct seq_file  
"\n high %lu"  
"\n active %lu"  
"\n inactive %lu"  
- "\n scanned %lu (a: %lu i: %lu)"  
+ "\n scanned %lu"  
"\n spanned %lu"  
"\n present %lu",  
zone->free_pages,  
@@ -2171,7 +2170,6 @@ static int zoneinfo_show(struct seq_file  
zone->nr_active,  
zone->nr_inactive,  
zone->pages_scanned,  
- zone->nr_scan_active, zone->nr_scan_inactive,  
zone->spanned_pages,  
zone->present_pages);  
seq_printf(m,  
-
```

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

• **References:**

- ◆ **[PATCH] vm: page-replace and clockpro**
◇ From: Peter Zijlstra

- Prev by Date: **[PATCH 6/9] clockpro-clockpro.patch**
- Next by Date: **Re: [PATCH 11 of 20] ipath - core driver, part 4 of 4**
- Previous by thread: **Re: [PATCH 6/9] clockpro-clockpro.patch**
- Next by thread: **[PATCH 12/14] page-replace-rotate.patch**
- Index(es):
 - ◆ **Date**
 - ◆ **Thread**