

[PATCH 1/9] clockpro-nonresident.patch

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-12/msg08500.html>

- *From:* Peter Zijlstra <a.p.zijlstra@xxxxxxxxxx>
 - *Date:* Fri, 30 Dec 2005 23:42:44 +0100
-

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

Originally started by Rik van Riel, I heavily modified the code to suit my needs.

The nonresident code approximates a clock but sacrifices precision in order to accomplish faster lookups.

The actual datastructure is a hash of small clocks, so that, assuming an equal distribution by the hash function, each clock has comparable order.

TODO:

- remove the ARC requirements.

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

```
include/linux/swap.h | 32 +++++
init/main.c | 2
mm/Makefile | 3
mm/nonresident.c | 391 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
4 files changed, 427 insertions(+), 1 deletion(-)
```

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

=====

```
--- /dev/null
+++ linux-2.6-git/mm/nonresident.c
@@ -0,0 +1,391 @@
+/*
+ * mm/nonresident.c
+ * (C) 2004,2005 Red Hat, Inc
+ * Written by Rik van Riel <riel@xxxxxxxxxx>
+ * Released under the GPL, see the file COPYING for details.
+ * Adapted by Peter Zijlstra <a.p.zijlstra@xxxxxxxxxx> for use by ARC
+ * like algorithms.
+ *
+ * Keeps track of whether a non-resident page was recently evicted
+ * and should be immediately promoted to the active list. This also
+ * helps automatically tune the inactive target.
```

[PATCH 1/9] clockpro-nonresident.patch

```
+ *
+ * The pageout code stores a recently evicted page in this cache
+ * by calling nonresident_put(mapping/mm, index/vaddr)
+ * and can look it up in the cache by calling nonresident_find()
+ * with the same arguments.
+ *
+ * Note that there is no way to invalidate pages after eg. truncate
+ * or exit, we let the pages fall out of the non-resident set through
+ * normal replacement.
+ *
+ *
+ * Modified to work with ARC like algorithms who:
+ * - need to balance two FIFOs;  $|b1| + |b2| = c$ ,
+ *
+ * The bucket contains four single linked cyclic lists (CLOCKS) and each
+ * clock has a tail hand. By selecting a victim clock upon insertion it
+ * is possible to balance them.
+ *
+ * The first two lists are used for B1/B2 and a third for a free slot list.
+ * The fourth list is unused.
+ *
+ * The slot looks like this:
+ * struct slot_t {
+ * u32 cookie : 24; // LSB
+ * u32 index : 6;
+ * u32 listid : 2;
+ * };
+ *
+ * The bucket is guarded by a spinlock.
+ */
+#include <linux/swap.h>
+#include <linux/mm.h>
+#include <linux/cache.h>
+#include <linux/spinlock.h>
+#include <linux/bootmem.h>
+#include <linux/hash.h>
+#include <linux/prefetch.h>
+#include <linux/kernel.h>
+
+#define TARGET_SLOTS 64
+#define NR_CACHELINES (TARGET_SLOTS*sizeof(u32) / L1_CACHE_BYTES)
+#define NR_SLOTS (((NR_CACHELINES * L1_CACHE_BYTES) - sizeof(spinlock_t) - 4*sizeof(u8) -
sizeof(u32)) / sizeof(u32))
+#if 0
+#if NR_SLOTS < (TARGET_SLOTS / 2)
+#warning very small slot size
+#if NR_SLOTS <= 0
+#error no room for slots left
+#endif
+#endif
+#endif
+#endif
```

[PATCH 1/9] clockpro-nonresident.patch

```
+
+#define BUILD_MASK(bits, shift) (((1 << (bits)) - 1) << (shift))
+
+#define LISTID_BITS 2
+#define LISTID_SHIFT (sizeof(u32)*8 - LISTID_BITS)
+#define LISTID_MASK BUILD_MASK(LISTID_BITS, LISTID_SHIFT)
+
+#define SET_LISTID(x, flg) ((x) = ((x) & ~LISTID_MASK) | ((flg) << LISTID_SHIFT))
+#define GET_LISTID(x) (((x) & LISTID_MASK) >> LISTID_SHIFT)
+
+#define INDEX_BITS 6 /* ceil(log2(NR_SLOTS)) */
+#define INDEX_SHIFT (LISTID_SHIFT - INDEX_BITS)
+#define INDEX_MASK BUILD_MASK(INDEX_BITS, INDEX_SHIFT)
+
+#define SET_INDEX(x, idx) ((x) = ((x) & ~INDEX_MASK) | ((idx) << INDEX_SHIFT))
+#define GET_INDEX(x) (((x) & INDEX_MASK) >> INDEX_SHIFT)
+
+#define COOKIE_MASK BUILD_MASK(sizeof(u32)*8 - LISTID_BITS - INDEX_BITS, 0)
+
+struct nr_bucket
+{
+ spinlock_t lock;
+ u8 hand[4];
+ u32 cycle;
+ u32 slot[NR_SLOTS];
+} ____cacheline_aligned;
+
+/* The non-resident page hash table. */
+static struct nr_bucket * nonres_table;
+static unsigned int nonres_shift;
+static unsigned int nonres_mask;
+
+/* hash the address into a bucket */
+static struct nr_bucket * nr_hash(void * mapping, unsigned long index)
+{
+ unsigned long bucket;
+ unsigned long hash;
+
+ hash = (unsigned long)mapping + 37 * index;
+ bucket = hash_long(hash, nonres_shift);
+
+ return nonres_table + bucket;
+}
+
+/* hash the address and inode into a cookie */
+static u32 nr_cookie(struct address_space * mapping, unsigned long index)
+{
+ unsigned long hash;
+
+ hash = 37 * (unsigned long)mapping + index;
+
+}
```

[PATCH 1/9] clockpro-nonresident.patch

```
+ if (mapping && mapping->host)
+ hash = 37 * hash + mapping->host->i_ino;
+
+ return hash_long(hash, sizeof(u32)*8 - LISTID_BITS - INDEX_BITS);
+}
+
+DEFINE_PER_CPU(unsigned long[4], nonres_count);
+
+/*
+ * remove current (b from 'abc'):
+ *
+ * initial swap(2,3)
+ *
+ * 1: -> [2],a 1: -> [2],a
+ * 2: -> [3],b 2: -> [1],c
+ * 3: -> [1],c * 3: -> [3],b
+ *
+ * 3 is now free for use.
+ *
+ * @nr_bucket: bucket to operate in
+ * @listid: list that the deletee belongs to
+ * @pos: slot position of deletee
+ * @slot: possible pointer to slot
+ *
+ * returns pointer to removed slot, NULL when list empty.
+ */
+static u32 * __nonresident_del(struct nr_bucket *nr_bucket, int listid, u8 pos, u32 *slot)
+{
+ int next_pos;
+ u32 *next;
+
+ if (slot == NULL) {
+ slot = &nr_bucket->slot[pos];
+ if (GET_LISTID(*slot) != listid)
+ return NULL;
+ }
+
+ --__get_cpu_var(nonres_count[listid]);
+
+ next_pos = GET_INDEX(*slot);
+ if (pos == next_pos) {
+ next = slot;
+ goto out;
+ }
+
+ next = &nr_bucket->slot[next_pos];
+ *next = xchg(slot, *next);
+
+ if (next_pos == nr_bucket->hand[listid])
+ nr_bucket->hand[listid] = pos;
+out:

```

[PATCH 1/9] clockpro-nonresident.patch

```
+ BUG_ON(GET_INDEX(*next) != next_pos);
+ return next;
+}
+
+static inline u32 * __nonresident_pop(struct nr_bucket *nr_bucket, int listid)
+{
+ return __nonresident_del(nr_bucket, listid, nr_bucket->hand[listid], NULL);
+}
+
+/*
+ * insert before (d before b in 'abc')
+ *
+ * initial set 4 swap(2,4)
+ *
+ * 1: -> [2],a 1: -> [2],a 1: -> [2],a
+ * 2: -> [3],b 2: -> [3],b 2: -> [4],d
+ * 3: -> [1],c 3: -> [1],c 3: -> [1],c
+ * 4: -> [4],nil 4: -> [4],d * 4: -> [3],b
+ *
+ * leaving us with 'adbc'.
+ *
+ * @nr_bucket: bucket to operator in
+ * @listid: list to insert into
+ * @pos: position to insert before
+ * @slot: slot to insert
+ */
+static void __nonresident_insert(struct nr_bucket *nr_bucket, int listid, u8 *pos, u32 *slot)
+{
+ u32 *head;
+
+ SET_LISTID(*slot, listid);
+
+ head = &nr_bucket->slot[*pos];
+
+ *pos = GET_INDEX(*slot);
+ if (GET_LISTID(*head) == listid)
+ *slot = xchg(head, *slot);
+
+ ++__get_cpu_var(nonres_count[listid]);
+}
+
+static inline void __nonresident_push(struct nr_bucket *nr_bucket, int listid, u32 *slot)
+{
+ __nonresident_insert(nr_bucket, listid, &nr_bucket->hand[listid], slot);
+}
+
+DEFINE_PER_CPU(u32, nonres_cycle);
+static DEFINE_PER_CPU(u32, nonres_delay);
+
+static void __nonresident_rotate(struct nr_bucket *nr_bucket)
```

[PATCH 1/9] clockpro-nonresident.patch

```
+{
+ u32 nr_cycle = __sum_cpu_var(u32, nonres_cycle) & ~((1 << nonres_shift) - 1);
+ u32 * slot;
+ while (nr_bucket->cycle != nr_cycle) {
+ ++__get_cpu_var(nonres_delay);
+ nr_bucket->cycle += (1 << nonres_shift);
+ slot = __nonresident_pop(nr_bucket, NR_b1);
+ if (slot)
+ __nonresident_push(nr_bucket, NR_free, slot);
+ }
+}
+
+/*
+ * Remembers a page by putting a hash-cookie on the @listid list.
+ *
+ * @mapping: page_mapping()
+ * @index: page_index()
+ * @listid: list to put the page on (NR_b1, NR_b2 and NR_free).
+ * @listid_evict: list to get a free page from when NR_free is empty.
+ *
+ * returns the list an empty page was taken from.
+ */
+int nonresident_put(struct address_space * mapping, unsigned long index, int listid, int listid_evict)
+{
+ struct nr_bucket *nr_bucket;
+ u32 cookie;
+ unsigned long flags;
+ u32 *slot;
+ int evict = NR_free;
+
+ prefetch(mapping->host);
+ nr_bucket = nr_hash(mapping, index);
+
+ spin_lock_prefetch(nr_bucket); // prefetchw_range(nr_bucket, NR_CACHELINES);
+ cookie = nr_cookie(mapping, index);
+
+ spin_lock_irqsave(&nr_bucket->lock, flags);
+ __nonresident_rotate(nr_bucket);
+ slot = __nonresident_pop(nr_bucket, evict);
+ if (!slot) {
+ evict = listid_evict;
+ slot = __nonresident_pop(nr_bucket, evict);
+ }
+ BUG_ON(!slot);
+ SET_INDEX(cookie, GET_INDEX(*slot));
+ cookie = xchg(slot, cookie);
+ __nonresident_push(nr_bucket, listid, slot);
+ spin_unlock_irqrestore(&nr_bucket->lock, flags);
+
+ return evict;
+}
```

[PATCH 1/9] clockpro-nonresident.patch

```
+
+/*
+ * Searches a page on the first two lists, and places it on the free list.
+ *
+ * @mapping: page_mapping()
+ * @index: page_index()
+ *
+ * returns listid of the list the item was found on with NR_found set if found.
+ */
+int nonresident_get(struct address_space * mapping, unsigned long index)
+{
+ struct nr_bucket * nr_bucket;
+ u32 wanted;
+ int j;
+ unsigned long flags;
+ int ret = 0;
+
+ if (mapping)
+ prefetch(mapping->host);
+ nr_bucket = nr_hash(mapping, index);
+
+ spin_lock_prefetch(&nr_bucket); // prefetch_range(nr_bucket, NR_CACHELINES);
+ wanted = nr_cookie(mapping, index) & COOKIE_MASK;
+
+ spin_lock_irqsave(&nr_bucket->lock, flags);
+ __nonresident_rotate(nr_bucket);
+ j = nr_bucket->hand[NR_b1];
+ do {
+ u32 *slot = &nr_bucket->slot[j];
+ if (GET_LISTID(*slot) != NR_b1)
+ break;
+
+ if ((*slot & COOKIE_MASK) == wanted) {
+ slot = __nonresident_del(nr_bucket, NR_b1, j, slot);
+ __nonresident_push(nr_bucket, NR_free, slot);
+ ret = NR_b1 | NR_found;
+ break;
+ }
+
+ j = GET_INDEX(*slot);
+ } while (j != nr_bucket->hand[NR_b1]);
+ spin_unlock_irqrestore(&nr_bucket->lock, flags);
+
+ return ret;
+}
+
+unsigned int nonresident_total(void)
+{
+ return (1 << nonres_shift) * NR_SLOTS;
+}
+
```

[PATCH 1/9] clockpro-nonresident.patch

```
+unsigned int nonresident_estimate(void)
+{
+ u32 count, cycle, delay, diff;
+
+ preempt_disable();
+ count = __sum_cpu_var(u32, nonres_count[NR_b1]);
+ cycle = __sum_cpu_var(u32, nonres_cycle);
+ delay = __sum_cpu_var(u32, nonres_delay);
+ preempt_enable();
+
+ diff = cycle - delay;
+
+ if (diff > count)
+ return 0;
+
+ return count - diff;
+}
+
+/*
+ * For interactive workloads, we remember about as many non-resident pages
+ * as we have actual memory pages. For server workloads with large inter-
+ * reference distances we could benefit from remembering more.
+ */
+static __initdata unsigned long nonresident_factor = 1;
+void __init nonresident_init(void)
+{
+ int target;
+ int i, j;
+
+ /*
+ * Calculate the non-resident hash bucket target. Use a power of
+ * two for the division because alloc_large_system_hash rounds up.
+ */
+ target = nr_all_pages * nonresident_factor;
+ target /= (sizeof(struct nr_bucket) / sizeof(u32));
+
+ nonres_table = alloc_large_system_hash("Non-resident page tracking",
+ sizeof(struct nr_bucket),
+ target,
+ 0,
+ HASH_EARLY | HASH_HIGHMEM,
+ &nonres_shift,
+ &nonres_mask,
+ 0);
+
+ for (i = 0; i < (1 << nonres_shift); i++) {
+ spin_lock_init(&nonres_table[i].lock);
+ for (j = 0; j < 4; ++j)
+ nonres_table[i].hand[j] = 0;
+
+ for (j = 0; j < NR_SLOTS; ++j) {
```

[PATCH 1/9] clockpro-nonresident.patch

```
+ nonres_table[i].slot[j] = 0;
+ SET_LISTID(nonres_table[i].slot[j], NR_free);
+ if (j < NR_SLOTS - 1)
+ SET_INDEX(nonres_table[i].slot[j], j+1);
+ else /* j == NR_SLOTS - 1 */
+ SET_INDEX(nonres_table[i].slot[j], 0);
+ }
+ }
+
+ for_each_cpu(i) {
+ for (j=0; j<4; ++j)
+ per_cpu(nonres_count[j], i) = 0;
+ }
+ }
+
+static int __init set_nonresident_factor(char * str)
+{
+ if (!str)
+ return 0;
+ nonresident_factor = simple_strtoul(str, &str, 0);
+ return 1;
+ }
+
+__setup("nonresident_factor=", set_nonresident_factor);
```

Index: linux-2.6-git/include/linux/swap.h

```
----- linux-2.6-git.orig/include/linux/swap.h
+++ linux-2.6-git/include/linux/swap.h
@@ -152,6 +152,31 @@ extern void out_of_memory(gfp_t gfp_mask
/* linux/mm/memory.c */
extern void swapin_readahead(swp_entry_t, unsigned long, struct vm_area_struct *);
```

```
/* linux/mm/nonresident.c */
#define NR_b1 0
#define NR_b2 1
#define NR_free 2
#define NR_lost 3
+
#define NR_listid 3
#define NR_found 0x80000000
+
+
+extern int nonresident_put(struct address_space *, unsigned long, int, int);
+extern int nonresident_get(struct address_space *, unsigned long);
+extern unsigned int nonresident_total(void);
+extern unsigned int nonresident_estimate(void);
+extern void nonresident_init(void);
+
+DECLARE_PER_CPU(unsigned long[4], nonres_count);
+DECLARE_PER_CPU(u32, nonres_cycle);
+
```

[PATCH 1/9] clockpro-nonresident.patch

```
+ #define __sum_cpu_var(type, var) ({ __typeof__(type) sum = 0; \
+ int cpu; \
+ for_each_cpu(cpu) sum += per_cpu(var, cpu); \
+ sum; })
+
+
+/* linux/mm/page_alloc.c */
extern unsigned long totalram_pages;
extern unsigned long totalhigh_pages;
@@ -288,6 +313,13 @@ static inline swp_entry_t get_swap_page(
#define has_swap_token(x) 0
#define disable_swap_token() do { } while(0)

+/* linux/mm/nonresident.c */
+#define nonresident_put(w,x,y,z) 0
+#define nonresident_find(x,y) 0
+#define nonresident_count(x) 0
+#define nonresident_total() 0
+#define nonresident_init() do { } while (0)
+
+#endif /* CONFIG_SWAP */
+#endif /* __KERNEL__ */
+#endif /* _LINUX_SWAP_H */
Index: linux-2.6-git/init/main.c
```

```
=====
--- linux-2.6-git.orig/init/main.c
+++ linux-2.6-git/init/main.c
@@ -46,6 +46,7 @@
#include <linux/unistd.h>
#include <linux/rmap.h>
#include <linux/mempolicy.h>
+#include <linux/swap.h>
#include <linux/key.h>
#include <net/sock.h>

@@ -509,6 +510,7 @@ asmlinkage void __init start_kernel(void
}
#endif
vfs_caches_init_early();
+ nonresident_init();
mem_init();
kmem_cache_init();
setup_per_cpu_pageset();
Index: linux-2.6-git/mm/Makefile
```

```
=====
--- linux-2.6-git.orig/mm/Makefile
+++ linux-2.6-git/mm/Makefile
@@ -12,7 +12,8 @@ obj-y := bootmem.o filemap.o mempool.o
readahead.o slab.o swap.o truncate.o vmscan.o \
prio_tree.o page_replace.o $(mmu-y)
```

[PATCH 1/9] clockpro-nonresident.patch

-obj-\$(CONFIG_SWAP) += page_io.o swap_state.o swapfile.o thrash.o
+obj-\$(CONFIG_SWAP) += page_io.o swap_state.o swapfile.o thrash.o \
+ nonresident.o
obj-\$(CONFIG_HUGETLBFS) += hugetlb.o
obj-\$(CONFIG_NUMA) += mempolicy.o
obj-\$(CONFIG_SPARSEMEM) += sparse.o
-

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

- **Follow-Ups:**

- ◆ **Re: [PATCH 1/9] clockpro-nonresident.patch**

- ◆ From: Marcelo Tosatti

- **References:**

- ◆ **[PATCH] vm: page-replace and clockpro**

- ◆ From: Peter Zijlstra

- Prev by Date: **[PATCH 7/9] clockpro-remove-old.patch**
- Next by Date: **Re: [PATCH 12 of 20] ipath - misc driver support code**
- Previous by thread: **[PATCH 7/9] clockpro-remove-old.patch**
- Next by thread: **Re: [PATCH 1/9] clockpro-nonresident.patch**
- Index(es):

- ◆ **Date**

- ◆ **Thread**