
Subject: [PATCH 1/2] mm: nonresident page tracking
Posted by [Peter Zijlstra](#) on Thu, 26 Jul 2007 17:25:05 GMT
[View Forum Message](#) <> [Reply to Message](#)

From: Rik van Riel <riel@redhat.com>

Track non-resident pages through a simple hashing scheme. This way the space overhead is limited to 1 u32 per page, or 0.1% space overhead and lookups are one cache miss.

Aside from seeing whether or not a page was recently evicted, we can also take a reasonable guess at how many other pages were evicted since this page was evicted.

TODO: make the entries unsigned long, currently we're limited to $1^{32} \times \text{NUM_NR} \times \text{PAGE_SIZE}$ bytes of memory. Event though this would end up being 1008 TB of memory, I suspect the hash function to go crap at around 4 to 16 TB.

Signed-off-by: Rik van Riel <riel@redhat.com>

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

```
Documentation/kernel-parameters.txt | 4
include/linux/nonresident.h          | 35 +++++++
init/main.c                         | 2
mm/Kconfig                          | 4
mm/Makefile                         | 1
mm/nonresident.c                    | 173 +++++++++++++++++++++++++++++++++++++
mm/swap.c                           | 3
mm/vmscan.c                         | 3
8 files changed, 225 insertions(+)
```

Index: linux-2.6/mm/nonresident.c

=====

--- /dev/null

+++ linux-2.6/mm/nonresident.c

@@ -0,0 +1,173 @@

+/*

+ * mm/nonresident.c

+ * (C) 2004,2005 Red Hat, Inc

+ * Written by Rik van Riel <riel@redhat.com>

+ * Released under the GPL, see the file COPYING for details.

+ *

+ * 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.

+ *

+ * The pageout code stores a recently evicted page in this cache

```

+ * by calling remember_page(mapping/mm, index/vaddr, generation)
+ * and can look it up in the cache by calling recently_evicted()
+ * 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.
+ */
#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>
+
+/* Number of non-resident pages per hash bucket. Never smaller than 15. */
+#if (L1_CACHE_BYTES < 64)
+#define NR_BUCKET_BYTES 64
+#else
+#define NR_BUCKET_BYTES L1_CACHE_BYTES
+#endif
+#define NUM_NR ((NR_BUCKET_BYTES - sizeof(atomic_t))/sizeof(atomic_t))
+
+struct nr_bucket
+{
+ atomic_t hand;
+ atomic_t cookie[NUM_NR];
+} ____cacheline_aligned;
+
+/* The non-resident page hash table. */
+static struct nr_bucket *nonres_table;
+static unsigned int nonres_shift;
+static unsigned int nonres_mask;
+
+static struct nr_bucket *nr_hash(void *mapping, unsigned long index)
+{
+ unsigned long bucket;
+ unsigned long hash;
+
+ hash = hash_ptr(mapping, BITS_PER_LONG);
+ hash = 37 * hash + hash_long(index, BITS_PER_LONG);
+ bucket = hash & nonres_mask;
+
+ return nonres_table + bucket;
+}
+
+static u32 nr_cookie(struct address_space *mapping, unsigned long index)

```

```

+{
+ unsigned long cookie;
+
+ cookie = hash_ptr(mapping, BITS_PER_LONG);
+ cookie = 37 * cookie + hash_long(index, BITS_PER_LONG);
+
+ if (mapping && mapping->host) {
+  cookie *= 37;
+  cookie += hash_long(mapping->host->i_ino, BITS_PER_LONG);
+ }
+
+ return (u32)(cookie >> (BITS_PER_LONG - 32));
+}
+
+unsigned long
+nonresident_get(struct address_space *mapping, unsigned long index)
+{
+ struct nr_bucket *bucket;
+ unsigned long distance = ~0UL;
+ u32 wanted;
+ int i;
+
+ if (!mapping)
+  return distance;
+
+ prefetch(mapping->host);
+ bucket = nr_hash(mapping, index);
+
+ prefetch(bucket);
+ wanted = nr_cookie(mapping, index);
+
+ for (i = 0; i < NUM_NR; i++) {
+  if ((u32)atomic_cmpxchg(&bucket->cookie[i], wanted, 0) == wanted) {
+   /* Return the distance between entry and clock hand. */
+   distance = atomic_read(&bucket->hand) + NUM_NR - i;
+   distance %= NUM_NR;
+   distance += (bucket - nonres_table);
+   goto out;
+  }
+ }
+
+out:
+ return distance;
+}
+
+u32 nonresident_put(struct address_space *mapping, unsigned long index)
+{
+ struct nr_bucket *bucket;

```

```

+ u32 cookie;
+ int cur_hand;
+ int hand;
+
+ prefetch(mapping->host);
+ bucket = nr_hash(mapping, index);
+
+ prefetchw(bucket);
+ cookie = nr_cookie(mapping, index);
+
+ /* Atomically find the next array index. */
+ do {
+   cur_hand = atomic_read(&bucket->hand);
+   hand = cur_hand + 1;
+   if (unlikely(hand == NUM_NR))
+     hand = 0;
+ } while (atomic_cmpxchg(&bucket->hand, cur_hand, hand) != cur_hand);
+
+ /* Statistics may want to know whether the entry was in use. */
+ return atomic_xchg(&bucket->cookie[hand], cookie);
+}
+
+unsigned long nonresident_total(void)
+{
+   return NUM_NR << nonres_shift;
+}
+
+/*
+ * 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 = 100;
+void __init nonresident_init(void)
+{
+   int target;
+   int i;
+
+   /*
+    * 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) / 100;
+   target /= (sizeof(struct nr_bucket) / sizeof(atomic_t));
+
+   nonres_table = alloc_large_system_hash("Non-resident page tracking",
+     sizeof(struct nr_bucket),
+     target,

```

```

+ 0,
+ HASH_EARLY,
+ &nonres_shift,
+ &nonres_mask,
+ 0);
+
+ for (i = 0; i < (1 << nonres_shift); i++)
+ atomic_set(&nonres_table[i].hand, 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/include/linux/nonresident.h
=====
--- /dev/null
+++ linux-2.6/include/linux/nonresident.h
@@ -0,0 +1,35 @@
+#ifndef _LINUX_NONRESIDENT_H_
+#define _LINUX_NONRESIDENT_H_
+
+
+#ifdef __KERNEL__
+
+#ifdef CONFIG_MM_NONRESIDENT
+
+extern void nonresident_init(void);
+extern unsigned long nonresident_get(struct address_space *, unsigned long);
+extern u32 nonresident_put(struct address_space *, unsigned long);
+extern unsigned long nonresident_total(void);
+
+#else /* CONFIG_MM_NONRESIDENT */
+
+static inline void nonresident_init(void) { }
+static inline
+unsigned long nonresident_get(struct address_space *, unsigned long, int)
+{
+ return 0;
+}
+
+static inline u32 nonresident_put(struct address_space *, unsigned long)
+{
+ return 0;
+}

```



```

#include <linux/notifier.h>
#include <linux/init.h>
+#include <linux/nonresident.h>

/* How many pages do we try to swap or page in/out together? */
int page_cluster;
@@ -365,6 +366,7 @@ void __pagevec_lru_add(struct pagevec *p
}
VM_BUG_ON(PageLRU(page));
SetPageLRU(page);
+ nonresident_get(page_mapping(page), page_index(page));
add_page_to_inactive_list(zone, page);
}
if (zone)
@@ -392,6 +394,7 @@ void __pagevec_lru_add_active(struct pag
}
VM_BUG_ON(PageLRU(page));
SetPageLRU(page);
+ nonresident_get(page_mapping(page), page_index(page));
VM_BUG_ON(PageActive(page));
SetPageActive(page);
add_page_to_active_list(zone, page);
Index: linux-2.6/mm/vmscan.c
=====
--- linux-2.6.orig/mm/vmscan.c
+++ linux-2.6/mm/vmscan.c
@@ -37,6 +37,7 @@
#include <linux/delay.h>
#include <linux/kthread.h>
#include <linux/freezer.h>
+#include <linux/nonresident.h>

#include <asm/tlbflush.h>
#include <asm/div64.h>
@@ -402,6 +403,7 @@ int remove_mapping(struct address_space

if (PageSwapCache(page)) {
swp_entry_t swap = { .val = page_private(page) };
+ nonresident_put(mapping, page_index(page));
__delete_from_swap_cache(page);
write_unlock_irq(&mapping->tree_lock);
swap_free(swap);
@@ -409,6 +411,7 @@ int remove_mapping(struct address_space
return 1;
}

+ nonresident_put(mapping, page_index(page));
__remove_from_page_cache(page);

```

```
write_unlock_irq(&mapping->tree_lock);
```

```
__put_page(page);
```

```
Index: linux-2.6/mm/Kconfig
```

```
=====
```

```
--- linux-2.6.orig/mm/Kconfig
```

```
+++ linux-2.6/mm/Kconfig
```

```
@@ -158,6 +158,10 @@ config RESOURCES_64BIT
```

```
help
```

```
This option allows memory and IO resources to be 64 bit.
```

```
+config MM_NONRESIDENT
```

```
+ bool "Track nonresident pages"
```

```
+ def_bool y
```

```
+
```

```
config ZONE_DMA_FLAG
```

```
int
```

```
default "0" if !ZONE_DMA
```

```
Index: linux-2.6/Documentation/kernel-parameters.txt
```

```
=====
```

```
--- linux-2.6.orig/Documentation/kernel-parameters.txt
```

```
+++ linux-2.6/Documentation/kernel-parameters.txt
```

```
@@ -1167,6 +1167,10 @@ and is between 256 and 4096 characters.
```

```
nomce [IA-32] Machine Check Exception
```

```
+ nonresident_factor= [KNL] Scale the size of the nonresident history
```

```
+ The default is 100, which equals the total
```

```
+ memory size.
```

```
+
```

```
noreplace-paravirt [IA-32,PV_OPS] Don't patch paravirt_ops
```

```
noreplace-smp [IA-32,SMP] Don't replace SMP instructions
```

```
--
```

```
Containers mailing list
```

```
Containers@lists.linux-foundation.org
```

```
https://lists.linux-foundation.org/mailman/listinfo/containers
```
