

[RFC] Demand faulting for large pages

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-08/1472.html>

From: Adam Litke (agl_at_us.ibm.com)

Date: 08/05/05

To: linux-kernel@vger.kernel.org
Date: Fri, 05 Aug 2005 10:21:38 -0500

Below is a patch to implement demand faulting for huge pages. The main motivation for changing from prefaulting to demand faulting is so that huge page allocations can follow the NUMA API. Currently, huge pages are allocated round-robin from all NUMA nodes.

The default behavior in SLES9 for i386 is to use demand faulting with NUMA policy-aware allocations. To my knowledge, this continues to work well in practice. Thanks to consolidated hugetlb code, switching the behavior requires changing only one fault handler. The bulk of the patch just moves the logic from `hugelb_pfault()` to `hugetlb_pte_fault()`.

Diffed against 2.6.13-rc4-git4

Signed-off-by: Adam Litke <agl@us.ibm.com>

```
fs/hugetlbf/inode.c | 5 -
include/linux/hugetlb.h | 2
mm/hugetlb.c | 140 ++++++-----
mm/memory.c | 7 --
4 files changed, 83 insertions(+), 71 deletions(-)
diff -upN reference/fs/hugetlbf/inode.c current/fs/hugetlbf/inode.c
--- reference/fs/hugetlbf/inode.c
+++ current/fs/hugetlbf/inode.c
@@ -79,10 +79,7 @@ static int hugetlbf_file_mmap(struct fi
     if (!(vma->vm_flags & VM_WRITE) && len > inode->i_size)
         goto out;

- ret = hugetlb_pfault(mapping, vma);
- if (ret)
- goto out;
+ ret = 0;
+ if (inode->i_size < len)
+     inode->i_size = len;
out:
diff -upN reference/include/linux/hugetlb.h current/include/linux/hugetlb.h
```

Linux-Kernel: [RFC] Demand faulting for large pages

```
--- reference/include/linux/hugetlb.h
+++ current/include/linux/hugetlb.h
@@ -25,6 +25,8 @@ int is_hugepage_mem_enough(size_t);
unsigned long hugetlb_total_pages(void);
struct page *alloc_huge_page(void);
void free_huge_page(struct page *);
+int hugetlb_fault(struct mm_struct *mm, struct vm_area_struct * vma,
+ unsigned long address, int write_access);

extern unsigned long max_huge_pages;
extern const unsigned long hugetlb_zero, hugetlb_infinity;
diff -upN reference/mm/hugetlb.c current/mm/hugetlb.c
--- reference/mm/hugetlb.c
+++ current/mm/hugetlb.c
@@ -277,18 +277,20 @@ int copy_hugetlb_page_range(struct mm_struct
    unsigned long addr = vma->vm_start;
    unsigned long end = vma->vm_end;

- while (addr < end) {
+ for (; addr < end; addr += HPAGE_SIZE) {
+ src_pte = huge_pte_offset(src, addr);
+ if (!src_pte || pte_none(*src_pte))
+ continue;
+
+     dst_pte = huge_pte_alloc(dst, addr);
+     if (!dst_pte)
+         goto nomem;
- src_pte = huge_pte_offset(src, addr);
- BUG_ON(!src_pte || pte_none(*src_pte)); /* prefaulted */
+ BUG_ON(!src_pte);
+     entry = *src_pte;
+     ptepage = pte_page(entry);
+     get_page(ptepage);
+     add_mm_counter(dst, rss, HPAGE_SIZE / PAGE_SIZE);
+     set_huge_pte_at(dst, addr, dst_pte, entry);
- addr += HPAGE_SIZE;
+ }
    return 0;

@@ -329,63 +331,6 @@ void zap_hugepage_range(struct vm_area_struct
    spin_unlock(&mm->page_table_lock);
}

-int hugetlb_prefault(struct address_space *mapping, struct vm_area_struct *vma)
-{
- struct mm_struct *mm = current->mm;
- unsigned long addr;
- int ret = 0;
-
- WARN_ON(!is_vm_hugetlb_page(vma));
- BUG_ON(vma->vm_start & ~HPAGE_MASK);
```

Linux–Kernel: [RFC] Demand faulting for large pages

```
– BUG_ON(vma->vm_end & ~HPAGE_MASK);
–
– hugetlb_pfault_arch_hook(mm);
–
– spin_lock(&mm->page_table_lock);
– for (addr = vma->vm_start; addr < vma->vm_end; addr += HPAGE_SIZE) {
– unsigned long idx;
– pte_t *pte = huge_pte_alloc(mm, addr);
– struct page *page;
–
– if (!pte) {
– ret = -ENOMEM;
– goto out;
– }
– if (!pte_none(*pte))
– hugetlb_clean_stale_pgtable(pte);
–
– idx = ((addr - vma->vm_start) >> HPAGE_SHIFT)
– + (vma->vm_pgoff >> (HPAGE_SHIFT - PAGE_SHIFT));
– page = find_get_page(mapping, idx);
– if (!page) {
– /* charge the fs quota first */
– if (hugetlb_get_quota(mapping)) {
– ret = -ENOMEM;
– goto out;
– }
– page = alloc_huge_page();
– if (!page) {
– hugetlb_put_quota(mapping);
– ret = -ENOMEM;
– goto out;
– }
– ret = add_to_page_cache(page, mapping, idx, GFP_ATOMIC);
– if (!ret) {
– unlock_page(page);
– } else {
– hugetlb_put_quota(mapping);
– free_huge_page(page);
– goto out;
– }
– }
– add_mm_counter(mm, rss, HPAGE_SIZE / PAGE_SIZE);
– set_huge_pte_at(mm, addr, pte, make_huge_pte(vma, page));
– }
–out:
– spin_unlock(&mm->page_table_lock);
– return ret;
–}
–
int follow_hugetlb_page(struct mm_struct *mm, struct vm_area_struct *vma,
    struct page **pages, struct vm_area_struct **vmas,
```

Linux–Kernel: [RFC] Demand faulting for large pages

```
        unsigned long *position, int *length, int i)
@@ -433,3 +378,76 @@ int follow_hugetlb_page(struct mm_struct

        return i;
    }
+
+int hugetlb_pte_fault(struct mm_struct *mm, struct vm_area_struct *vma,
+ unsigned long address, int write_access)
+{
+ int ret = VM_FAULT_MINOR;
+ unsigned long idx;
+ pte_t *pte;
+ struct page *page;
+ struct address_space *mapping;
+
+ WARN_ON(!is_vm_hugetlb_page(vma));
+ BUG_ON(vma->vm_start & ~HPAGE_MASK);
+ BUG_ON(vma->vm_end & ~HPAGE_MASK);
+ BUG_ON(!vma->vm_file);
+
+ pte = huge_pte_alloc(mm, address);
+ if (!pte) {
+ ret = VM_FAULT_SIGBUS;
+ goto out;
+ }
+ if (!pte_none(*pte))
+ goto flush;
+
+ mapping = vma->vm_file->f_mapping;
+ idx = ((address - vma->vm_start) >> HPAGE_SHIFT)
+ + (vma->vm_pgoff >> (HPAGE_SHIFT - PAGE_SHIFT));
+retry:
+ page = find_get_page(mapping, idx);
+ if (!page) {
+ /* charge the fs quota first */
+ if (hugetlb_get_quota(mapping)) {
+ ret = VM_FAULT_SIGBUS;
+ goto out;
+ }
+ page = alloc_huge_page();
+ if (!page) {
+ hugetlb_put_quota(mapping);
+ ret = VM_FAULT_SIGBUS;
+ goto out;
+ }
+ if(add_to_page_cache(page, mapping, idx, GFP_ATOMIC)) {
+ put_page(page);
+ goto retry;
+ }
+ unlock_page(page);
+ }
```

Linux–Kernel: [RFC] Demand faulting for large pages

```
+ add_mm_counter(mm, rss, HPAGE_SIZE / PAGE_SIZE);
+ set_huge_pte_at(mm, address, pte, make_huge_pte(vma, page));
+flush:
+ flush_tlb_page(vma, address);
+out:
+ return ret;
+}
+
+int hugetlb_fault(struct mm_struct *mm, struct vm_area_struct *vma,
+ unsigned long address, int write_access)
+{
+ pte_t *ptep;
+ int rc = VM_FAULT_SIGBUS;
+
+ spin_lock(&mm->page_table_lock);
+
+ ptep = huge_pte_alloc(mm, address & HPAGE_MASK);
+ if (! ptep) {
+ BUG();
+ goto out;
+ }
+ if (pte_none(*ptep))
+ rc = hugetlb_pte_fault(mm, vma, address, write_access);
+out:
+ spin_unlock(&mm->page_table_lock);
+ return rc;
+}
diff -upN reference/mm/memory.c current/mm/memory.c
--- reference/mm/memory.c
+++ current/mm/memory.c
@@ -933,11 +933,6 @@ int get_user_pages(struct task_struct *t
                || !(flags & vma->vm_flags))
                return i ? -EFAULT;

- if (is_vm_hugetlb_page(vma)) {
- i = follow_hugetlb_page(mm, vma, pages, vmas,
- &start, &len, i);
- continue;
- }
                spin_lock(&mm->page_table_lock);
                do {
                        struct page *page;
@@ -2024,7 +2019,7 @@ int handle_mm_fault(struct mm_struct *mm
                inc_page_state(pgfault);

                if (is_vm_hugetlb_page(vma))
- return VM_FAULT_SIGBUS; /* mapping truncation does this. */
+ return hugetlb_fault(mm, vma, address, write_access);

                /*
                * We need the page table lock to synchronize with kswapd
```

Linux-Kernel: [RFC] Demand faulting for large pages

--

Adam Litke - (agl at us.ibm.com)
IBM Linux Technology Center

-

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/>