

## Re: Active Memory Defragmentation: Our implementation & problems

**Source:** <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-02/0707.html>

---

**From:** Dave Hansen ([haveblue\\_at\\_us.ibm.com](mailto:haveblue_at_us.ibm.com))

**Date:** 02/04/04

To: Alok Mooley <[rangdi@yahoo.com](mailto:rangdi@yahoo.com)>

Date: 03 Feb 2004 21:54:34 -0800

On Tue, 2004-02-03 at 21:09, Alok Mooley wrote:

- > *Yes we have cut & pasted some code from there. But,*
- > *try\_to\_unmap\_one also does lots of other stuff like*
- > */\**
- > *\* Store the swap location in the pte.*
- > *\* See handle\_pte\_fault() ...*
- > *\*/*
- > *which we don't want. Hence we use a separate function.*

If that function is fairly close, you might want to look at modifying it in a way that will make it work for you, but maintain the current semantics for current users. There seems to be a lot more in common than different there. Think long and hard about things that you don't think you need. Will it *\*hurt\**, or is it just a bit superfluous?

- > *Could you also please comment & advise us on our*
- > *problems which are as below: -*
- >
- > *We want to broaden our definition of a movable page, & consider kernel*
- > *pages & file-backed pages also for movement (currently we consider*
- > *only userspace anonymous pages as movable). Do file-backed pages also*
- > *obey the 3GB rule?*

The 3GB rule? file-backed pages are referenced via the page cache which can store arbitrary pages; they don't have to be in low memory.

Moving file-backed pages is mostly handled already. You can do a regular page-cache lookup with `find_get_page()`, make your copy, invalidate the old one, then reread the new one. The invalidation can be done in the same style as `shrink_list()`.

- > *In order to move such pages, we will have to patch macros like*
- > *"virt\_to\_phys" & other related macros, so that the address*
- > *translation for pages moved by us will take place vmalloc style, i.e.,*
- > *via page tables, instead of direct +-3GB. Is it worth introducing such*

## Linux–Kernel: Re: Active Memory Defragmentation: Our implementation & problems

- > *an overhead for address translation (vmalloc does it!)? If no, then is*
- > *there another way out, or is it better to stick to our current*
- > *definition of a movable page?*

Low memory kernel pages are a much bigger deal to defrag. I've started to think about these for hotplug memory and it just makes my head hurt. If you want to do this, you are right, you'll have to alter virt\_to\_phys and company. The best way I've seen this is with CONFIG\_NONLINEAR: <http://lwn.net/2002/0411/a/discontig.php3>

Those lookup tables are pretty fast, and have benefits to many areas beyond defragmentation like NUMA and the memory hotplug projects.

Rather than try to defrag kernel memory now, it's probably better to work on schemes that keep from fragmenting memory in the first place. What kind of situations are causing you the most fragmentation?

We've thought about having many more allocator pools around to help ease fragmentation. I started to code one up that would allocate some higher order pages for each struct address\_space and hand little pages out from that pool instead of going back to the buddy allocator. That way, when you go to free a file's page cache, you get some pretty large contiguous areas instead of a bunch of scattered 0-order pages.

- > *Identifying pages moved by us may involve introducing a new page–flag.*
- > *A new page–flag for per–cpu pages would be great, since we have to*
- > *traverse the per–cpu hot & cold lists in order to identify if a page*
- > *is on the pcp lists.*

We already have pretty sparse utilization of the current page–>flags. See some of the work that Bill Irwin (wli@holomorphy.com) has done in the past. But, I'm not sure you really even need to go there.

If the per–cpu allocator caches are your only problem, I don't see why we can't just flush them out when you're doing your operation. Plus, they aren't *\*that\** big, so you could pretty easily go scanning them. Martin, can we just flush out and turn off the per–cpu hot/cold lists for the defrag period?

This is part of a larger problem that we'll see with memory removal as well. Given any random page in the system, we'll need to be able to see what's being done with it. There are a lot of things that do a alloc\_page() and vm never sees the page again. We'll eventually need ways to account for these and possibly have mechanisms to get some of them back.

This might involve having some new allocator flags for things that can allow themselves to be moved, with the default being for things that either refuse to be moved, or don't know if they can be.

- > *As of now, we have adopted a failure based approach, i.e, we*
- > *defragment only when a higher order allocation failure has taken place*

## Linux–Kernel: Re: Active Memory Defragmentation: Our implementation & problems

- > *(just before kswapd starts swapping). We now want to defragment based*
- > *on thresholds kept for each allocation order. Instead of a daemon*
- > *kicking in on a threshold violation (as proposed by Mr. Daniel*
- > *Phillips), we intend to capture idle cpu cycles by inserting a new*
- > *process just above the idle process.*

I think I'd agree with Dan on that one. When kswapd is going, it's pretty much too late. The daemon approach would be more flexible, allow you to start earlier, and more easily have various levels of aggressiveness.

- > *Now, when we are scheduled, we are sure that the cpu is idle, & this*
- > *is when we check for threshold violation & defragment. One problem*
- > *with this would be when to reschedule ourselves (allow our*
- > *preemption)? We do not want the memory state to change beneath us,*
- > *so right now we are not allowing our preemption.*

It's a real luxury if the state doesn't change underneath you. It's usually worthwhile to try and do it without locking too many things down. Take the page cache, for instance. It does a lot of its work without locks, and has all of the error handling necessary when things collide during a duplicate addition or go away from underneath you. It's a good example of some really capable code that doesn't require a static state to work properly.

—dave

—

To unsubscribe from this list: send the line "unsubscribe linux–kernel" in the body of a message to [majordomo@vger.kernel.org](mailto: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/>