Started Jan 2000 by Kanoj Sarcar <kanoj@sgi.com>

Memory balancing is needed for non __GFP_WAIT as well as for non __GFP_IO allocations.

There are two reasons to be requesting non __GFP_WAIT allocations: the caller can not sleep (typically intr context), or does not want to incur cost overheads of page stealing and possible swap io for whatever reasons.

__GFP_IO allocation requests are made to prevent file system deadlocks.

In the absence of non sleepable allocation requests, it seems detrimental to be doing balancing. Page reclamation can be kicked off lazily, that is, only when needed (aka zone free memory is 0), instead of making it a proactive process.

That being said, the kernel should try to fulfill requests for direct mapped pages from the direct mapped pool, instead of falling back on the dma pool, so as to keep the dma pool filled for dma requests (atomic or not). A similar argument applies to highmem and direct mapped pages. OTOH, if there is a lot of free dma pages, it is preferable to satisfy regular memory requests by allocating one from the dma pool, instead of incurring the overhead of regular zone balancing.

In 2.2, memory balancing/page reclamation would kick off only when the _total_ number of free pages fell below 1/64 th of total memory. With the right ratio of dma and regular memory, it is quite possible that balancing would not be done even when the dma zone was completely empty. 2.2 has been running production machines of varying memory sizes, and seems to be doing fine even with the presence of this problem. In 2.3, due to HIGHMEM, this problem is aggravated.

In 2.3, zone balancing can be done in one of two ways: depending on the zone size (and possibly of the size of lower class zones), we can decide at init time how many free pages we should aim for while balancing any zone. The good part is, while balancing, we do not need to look at sizes of lower class zones, the bad part is, we might do too frequent balancing due to ignoring possibly lower usage in the lower class zones. Also, with a slight change in the allocation routine, it is possible to reduce the memclass() macro to be a simple equality.

Another possible solution is that we balance only when the free memory of a zone _and_ all its lower class zones falls below 1/64th of the total memory in the zone and its lower class zones. This fixes the 2.2 balancing problem, and stays as close to 2.2 behavior as possible. Also, the balancing algorithm works the same way on the various architectures, which have different numbers and types of zones. If we wanted to get fancy, we could assign different weights to free pages in different zones in the future.

Note that if the size of the regular zone is huge compared to dma zone, it becomes less significant to consider the free dma pages while deciding whether to balance the regular zone. The first solution becomes more attractive then.

The appended patch implements the second solution. It also "fixes" two problems: first, kswapd is woken up as in 2.2 on low memory conditions for non-sleepable allocations. Second, the HIGHMEM zone is also balanced, so as to give a fighting chance for replace_with_highmem() to get a HIGHMEM page, as well as to ensure that HIGHMEM allocations do not fall back into regular zone. This also makes sure that HIGHMEM pages are not leaked (for example, in situations where a HIGHMEM page is in the swapcache but is not being used by anyone)

kswapd also needs to know about the zones it should balance. kswapd is primarily needed in a situation where balancing can not be done, probably because all allocation requests are coming from intr context and all process contexts are sleeping. For 2.3, kswapd does not really need to balance the highmem zone, since intr context does not request highmem pages. kswapd looks at the zone_wake_kswapd field in the zone structure to decide whether a zone needs balancing.

Page stealing from process memory and shm is done if stealing the page would alleviate memory pressure on any zone in the page's node that has fallen below its watermark.

watemark[WMARK_MIN/WMARK_LOW/WMARK_HIGH]/low_on_memory/zone_wake_kswapd: These are per-zone fields, used to determine when a zone needs to be balanced. When the number of pages falls below watermark[WMARK_MIN], the hysteric field low_on_memory gets set. This stays set till the number of free pages becomes watermark[WMARK_HIGH]. When low_on_memory is set, page allocation requests will try to free some pages in the zone (providing GFP_WAIT is set in the request). Orthogonal to this, is the decision to poke kswapd to free some zone pages. That decision is not hysteresis based, and is done when the number of free pages is below watermark[WMARK_LOW]; in which case zone_wake_kswapd is also set.

(Good) Ideas that I have heard:
1. Dynamic experience should influence balancing: number of failed requests for a zone can be tracked and fed into the balancing scheme (jalvo@mbay.net)
2. Implement a replace_with_highmem()-like replace_with_regular() to preserve dma pages. (lkd@tantalophile.demon.co.uk)