Zoned Allocator -14- (Kswapd & Kcompactd)

2016-12-01 (http://jake.dothome.co.kr/zonned-allocator-kswapd/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

<kernel v5.0>

Kswapd & Kcompactd

Each node runs kswapd and kcompactd, and it goes to sleep when there is more than a certain amount of free memory. However, when the page allocator tries to allocate the order page and does not meet the low watermark criteria due to the lack of free pages, it wakes up kswapd and kcompactd. kswapd performs page retrieval from its own node, and kcompact performs compaction, and if it meets the high watermark criteria for all nodes, it sleeps itself.

Initialize kswapd

kswapd_init()

mm/vmscan.c

```
01 static int __init kswapd_init(void)
02 {
03         int nid, ret;
04
05         swap_setup();
06         for_each_node_state(nid, N_MEMORY)
07                 kswapd_run(nid);
08         ret = cpuhp_setup_state_nocalls(CPUHP_AP_ONLINE_DYN,
09                                         "mm/vmscan:online", kswapd_cpu_o
nline,
10                                         NULL);
11         WARN_ON(ret < 0);
12         return 0;
13 }
14
15 module_init(kswapd_init)
```

kswapd.

- Prepare kswapd before executing it on line 5 of the code.
- In line 6~7 of the code, run kswapd on all memory nodes.
- In line 8~10 of the code, register that the kswapd_cpu_online() function can be called when the CPU is changed to a CPUHP_AP_ONLINE_DYN state via hot-plug.

swap_setup()

mm/swap.c

```
/*
```

```
2   * Perform any setup for the swap system
3   */
```

```
01 void __init swap_setup(void)
02 {
03         unsigned long megs = totalram_pages >> (20 - PAGE_SHIFT);
04 
05         /* Use a smaller cluster for small-memory machines */
06         if (megs < 16)
07                 page_cluster = 2;
08         else
09                 page_cluster = 3;
10         /*
11          * Right now other parts of the system means that we
12          * _really_ don't want to cluster much more
13          */
14 }
```

Prepare before running kswapd.

- In code lines 3~9, if the total RAM is less than 16M, substitute 2 for page_cluster, otherwise substitute 3.

kswapd_run()

mm/vmscan.c

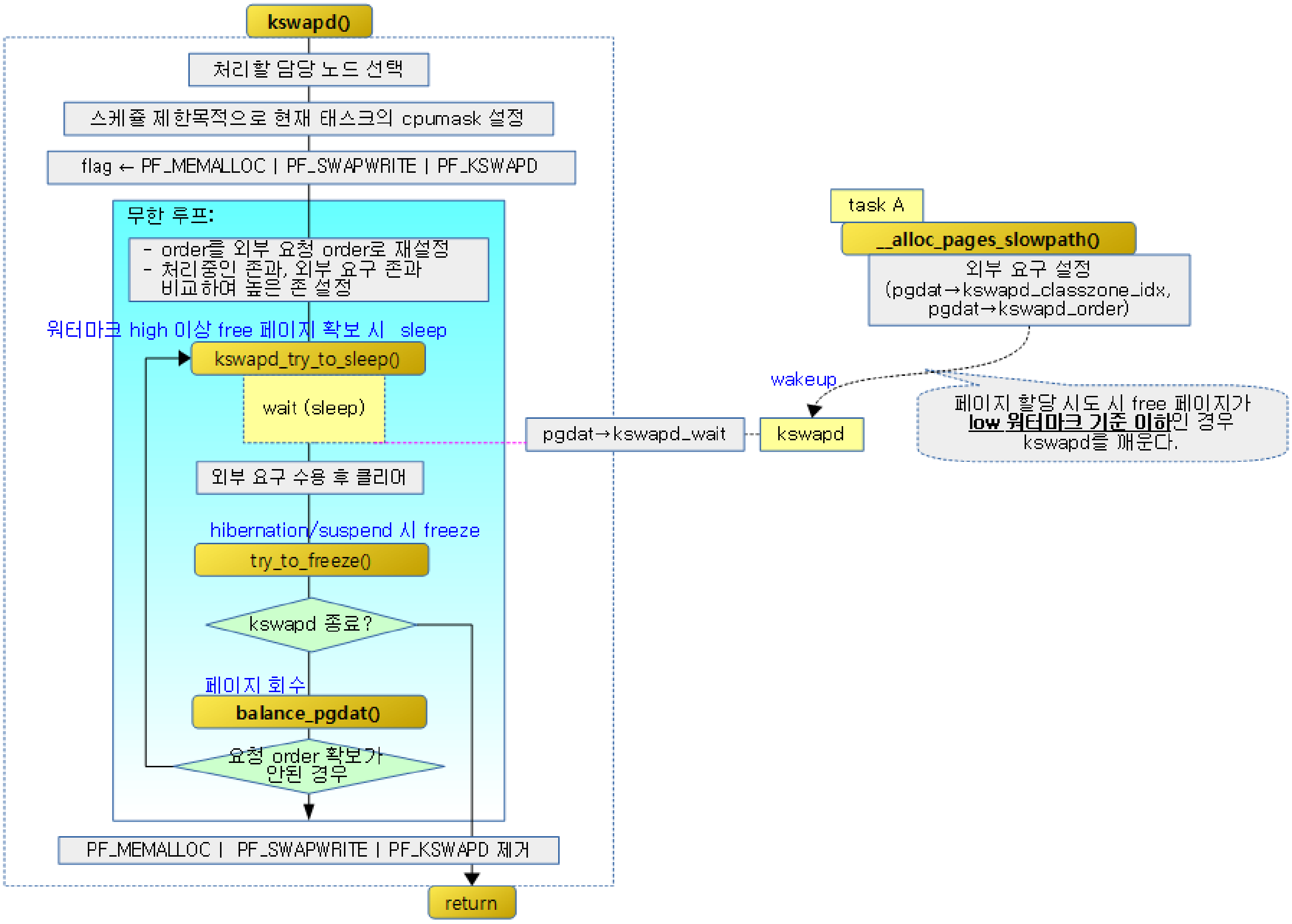
```
1 /*
2  * This kswapd start function will be called by init and node-hot-add.
3  * On node-hot-add, kswapd will moved to proper cpus if cpus are hot-add
  ed.
4  */
```

```
01 int kswapd_run(int nid)
02 {
03         pg_data_t *pgdat = NODE_DATA(nid);
04         int ret = 0;
05 
06         if (pgdat->kswapd)
07                 return 0;
08 
09         pgdat->kswapd = kthread_run(kswapd, pgdat, "kswapd%d", nid);
10         if (IS_ERR(pgdat->kswapd)) {
11                 /* failure at boot is fatal */
12                 BUG_ON(system_state == SYSTEM_BOOTING);
13                 pr_err("Failed to start kswapd on node %d\n", nid);
14                 ret = PTR_ERR(pgdat->kswapd);
15                 pgdat->kswapd = NULL;
16         }
17         return ret;
18 }
```

kswapd thread.

- In code lines 3~7, the kswapd thread on the @nid node returns 0 to skip if it is already running.
- Run the kswapd thread on lines 9~16 of code.

kswapd behavior


kswapd()
처리할 담당 노드 선택
스케쥴 제한목적으로 현재 태스크의 cpumask 설정
flag ← PF_MEMALLOC | PF_SWAPWRITE | PF_KSWAPD
무한 루프:
- order를 외부 요청 order로 재설정
- 처리중인 존과, 외부 요구 존과 비교하여 높은 존 설정
워터마크 high 이상 free 페이지 확보 시 sleep
kswapd_try_to_sleep()
wait (sleep)
외부 요구 수용 후 클리어
hibernation/suspend 시 freeze
try_to_freeze()
kswapd 종료?
페이지 회수
balance_pgdat()
요청 order 확보가 안된 경우
PF_MEMALLOC | PF_SWAPWRITE | PF_KSWAPD 제거
return
pgdat→kswapd_wait
kswapd
wakeup
task A
__alloc_pages_slowpath()
외부 요구 설정
(pgdat→kswapd_classzone_idx,
pgdat→kswapd_order)
페이지 할당 시도 시 free 페이지가
low 워터마크 기준 이하인 경우
kswapd를 깨운다.

(http://jake.dothome.co.kr/wp-content/uploads/2016/12/kswapd-1b.png)

kswapd()

mm/vmscan.c

```
/*
 * The background pageout daemon, started as a kernel thread
 * from the init process.
 *
 * This basically trickles out pages so that we have _some_
 * free memory available even if there is no other activity
 * that frees anything up. This is needed for things like routing
 * etc, where we otherwise might have all activity going on in
 * asynchronous contexts that cannot page things out.
 *
 * If there are applications that are active memory-allocators
 * (most normal use), this basically shouldn't matter.
 */
```

```
static int kswapd(void *p)
{
        unsigned int alloc_order, reclaim_order;
        unsigned int classzone_idx = MAX_NR_ZONES - 1;
        pg_data_t *pgdat = (pg_data_t*)p;
        struct task_struct *tsk = current;

        struct reclaim_state reclaim_state = {
                .reclaimed_slab = 0,
        };
        const struct cpumask *cpumask = cpumask_of_node(pgdat->node_id);

        if (!cpumask_empty(cpumask))
                set_cpus_allowed_ptr(tsk, cpumask);
        current->reclaim_state = &reclaim_state;

        /*
         * Tell the memory management that we're a "memory allocator",
         * and that if we need more memory we should get access to it
         * regardless (see "__alloc_pages()"). "kswapd" should
         * never get caught in the normal page freeing logic.
         *
         * (Kswapd normally doesn't need memory anyway, but sometimes
         * you need a small amount of memory in order to be able to
         * page out something else, and this flag essentially protects
         * us from recursively trying to free more memory as we're
         * trying to free the first piece of memory in the first place).
         */
        tsk->flags |= PF_MEMALLOC | PF_SWAPWRITE | PF_KSWAPD;
        set_freezable();

        pgdat->kswapd_order = 0;
        pgdat->kswapd_classzone_idx = MAX_NR_ZONES;
        for ( ; ; ) {
                bool ret;

                alloc_order = reclaim_order = pgdat->kswapd_order;
                classzone_idx = kswapd_classzone_idx(pgdat, classzone_idx);

kswapd_try_sleep:
                kswapd_try_to_sleep(pgdat, alloc_order, reclaim_order,
                                        classzone_idx);

                /* Read the new order and classzone_idx */
                alloc_order = reclaim_order = pgdat->kswapd_order;
                classzone_idx = kswapd_classzone_idx(pgdat, 0);
                pgdat->kswapd_order = 0;
                pgdat->kswapd_classzone_idx = MAX_NR_ZONES;

                ret = try_to_freeze();
                if (kthread_should_stop())
                        break;

                /*
                 * We can speed up thawing tasks if we don't call balance_pgdat
                 * after returning from the refrigerator
                 */
                if (ret)
                        continue;

                /*
                 * Reclaim begins at the requested order but if a high-order
                 * reclaim fails then kswapd falls back to reclaiming for
                 * order-0. If that happens, kswapd will consider sleeping
                 * for the order it finished reclaiming at (reclaim_order)
                 * but kcompactd is woken to compact for the original
                 * request (alloc_order).
                 */
                trace_mm_vmscan_kswapd_wake(pgdat->node_id, classzone_idx,
                                                alloc_order);
                reclaim_order = balance_pgdat(pgdat, alloc_order, classzone_idx);
                if (reclaim_order < alloc_order)
                        goto kswapd_try_sleep;
        }
```

```
75
76         tsk->flags &= ~(PF_MEMALLOC | PF_SWAPWRITE | PF_KSWAPD);
77         current->reclaim_state = NULL;
78
79         return 0;
80 }
```

The kswapd thread running on each node will reclaim the page in the background if the free page goes below the low watermark for the zone running on each node, and stops recalling the page if the high watermark is above the high watermark.

- In line 5~14 of the code, read the online cpumask running on the request node and set it to the current task.
    - Set the request node's cpumask to cpus_allowed the current task, etc.
- In line 15 of the code, point the reclaim_state of the current task to the initialized reclaim_state struct.
- In line 29 of code, set the PF_MEMALLOC, PF_SWAPWRITE, and PF_KSWAPD to the flags of the current task.
    - PF_MEMALLOC
        - It allows you to assign a task for memory reclaim without watermark restrictions.
    - PF_SWAPWRITE
        - Make a swap write request to anon memory.
    - PF_KSWAPD
        - kswapd task.
- Remove the PF_NOFREEZE flag from line 30 so that you can freeze the current task.
    - See also: freeze (http://jake.dothome.co.kr/freeze) | Qc
- In code lines 32~33, kswapd_order starts from 0 and kswapd_classzone_idx is assigned to MAX_NR_ZONES so that it starts from the top.
- In lines 34~38 of the code, we use an order that kswapd nodes to alloc_order and reclaim_order. Then bring the target zone, the classzone_idx.
- In code lines 40~42, try_sleep: Label. kswapd will attempt to slip.
- In lines 45~46 of the code, apply the order and zone requested from the outside for the alloc_order and reclaim_order.
- Reset the kwapd_order to 47 in code lines 48~0, and substitute the MAX_NR_ZONES so that the kswapd_classzone_idx can be done from the top.
- If there is a freeze request for the current task kswapd at line 50 of the code, try freeze.
- If the current task's KTHREAD_SHOULD_STOP flag bit is set in code lines 51~52, it will break out of the loop and terminate the thread.
- If you have ever been freezed in code lines 58~59, you can continue without running the node balance for faster processing.
- This is the case when I never freeze in line 71 of code. Proceed with page retrieval for order pages and zones.
- If the recall fails in the order requested in code lines 72~73, go to the try_sleep: label to order 0 and retry in that zone.
- If the recall succeeds in the order requested in line 74, the loop repeats.
- Finish processing the kswapd thread on lines 76~79 of code.

kswapd_try_to_sleep()

mm/vmscan.c

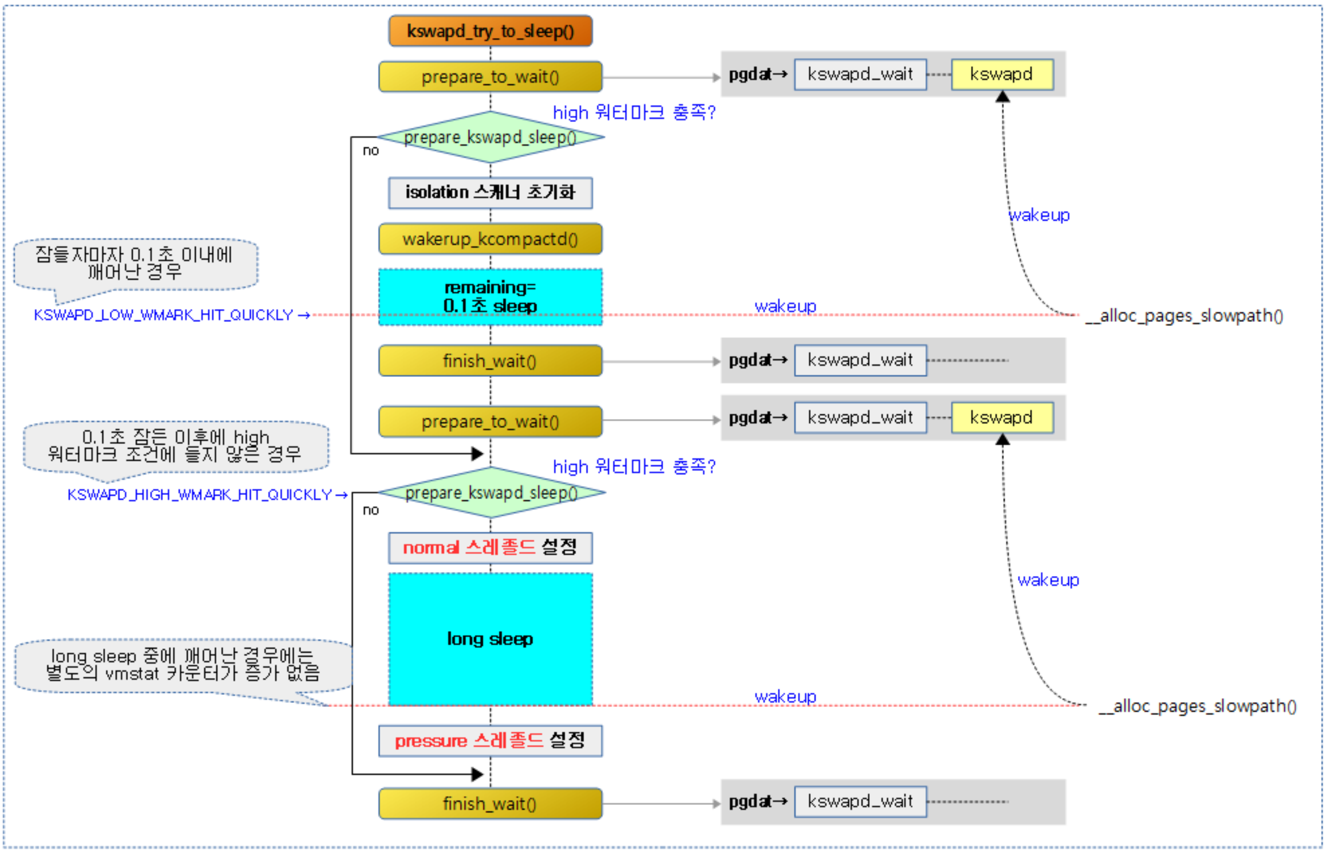
```
static void kswapd_try_to_sleep(pg_data_t *pgdat, int alloc_order, int reclaim_order,
                                unsigned int classzone_idx)
{
        long remaining = 0;
        DEFINE_WAIT(wait);

        if (freezing(current) || kthread_should_stop())
                return;

        prepare_to_wait(&pgdat->kswapd_wait, &wait, TASK_INTERRUPTIBLE);

        /*
         * Try to sleep for a short interval. Note that kcompactd will only be
         * woken if it is possible to sleep for a short interval. This is
         * deliberate on the assumption that if reclaim cannot keep an
         * eligible zone balanced that it's also unlikely that compaction will
         * succeed.
         */
        if (prepare_kswapd_sleep(pgdat, reclaim_order, classzone_idx)) {
                /*
                 * Compaction records what page blocks it recently failed to
                 * isolate pages from and skips them in the future scanning.
                 * When kswapd is going to sleep, it is reasonable to assume
                 * that pages and compaction may succeed so reset the cache.
                 */
                reset_isolation_suitable(pgdat);

                /*
                 * We have freed the memory, now we should compact it to make
                 * allocation of the requested order possible.
                 */
                wakeup_kcompactd(pgdat, alloc_order, classzone_idx);

                remaining = schedule_timeout(HZ/10);

                /*
                 * If woken prematurely then reset kswapd_classzone_idx and
                 * order. The values will either be from a wakeup request or
                 * the previous request that slept prematurely.
                 */
                if (remaining) {
                        pgdat->kswapd_classzone_idx = kswapd_classzone_idx(pgdat, classzone_idx);
                        pgdat->kswapd_order = max(pgdat->kswapd_order, reclaim_order);
                }

                finish_wait(&pgdat->kswapd_wait, &wait);
                prepare_to_wait(&pgdat->kswapd_wait, &wait, TASK_INTERRUPTIBLE);
        }

        /*
51          * After a short sleep, check if it was a premature sleep. If not, then
52          * go fully to sleep until explicitly woken up.
53          */
54         if (!remaining &&
55             prepare_kswapd_sleep(pgdat, reclaim_order, classzone_idx)) {
56                 trace_mm_vmscan_kswapd_sleep(pgdat->node_id);
57 
58                 /*
59                  * vmstat counters are not perfectly accurate and the estimated
60                  * value for counters such as NR_FREE_PAGES can deviate from the
61                  * true value by nr_online_cpus * threshold. To avoid the zone
62                  * watermarks being breached while under pressure, we reduce the
63                  * per-cpu vmstat threshold while kswapd is awake and restore
64                  * them before going back to sleep.
65                  */
66                 set_pgdat_percpu_threshold(pgdat, calculate_normal_threshold);
67 
68                 if (!kthread_should_stop())
69                         schedule();
70 
71                 set_pgdat_percpu_threshold(pgdat, calculate_pressure_threshold);
72         } else {
73                 if (remaining)
74                         count_vm_event(KSWAPD_LOW_WMARK_HIT_QUICKLY);
75                 else
76                         count_vm_event(KSWAPD_HIGH_WMARK_HIT_QUICKLY);
77         }
78         finish_wait(&pgdat->kswapd_wait, &wait);
79 }
```

If the request order and zone for a node are in a state where the free page can be allocated in a balanced manner based on the high watermark, then it should be sleep.

- If there is a freeze request in line 7~8 of the code, exit the function.
- In line 10 of code, add the current task to the kswapd_wait to prepare it for sleep.
- This is done in code sections 19~48 to the request zone and when the free page meets the balanced high watermark criteria for the request order.
    - The compact_blockskip_flush is set in the zone so that the direct-component has recently completed and the compaction can start from the beginning again in that zone. Clear all skip block bits for these zones.
    - wakes up the compactd thread.
    - Sleep 0.1 seconds. If you break it in the middle, make a note of the zone and order that kswapd is processing in the node.
    - Get ready to slip again.
- In code sections 54~71, if you completely slip 0.1 seconds without waking up in the middle, and you still have a free page for the request zone and the request order, and it meets the balanced high watermark standard, it will be processed as follows.
    - Whether or not it is necessary to precisely calculate the NR_FREE_PAGES and so on is called a per-cpu threadsold, and it specifies a normal threaded value for each zone included in the node to specify it as a typical thread resolve.

  - If it's not a request to terminate the thread, it sleeps.
  - The fact that kswapd woke up means that it has become a memory pressure situation. Therefore, this time, we specify that we use the pressure threaded value as the per-cpu threadold value.
- In code sections 72~77, the memory shortage situation came quickly. I woke up as soon as I slipped, and depending on the condition, the relevant counters were handled as follows.
  - If you sleep for 0.1 seconds and the current thread, kswapd, wakes up again due to lack of memory, increments the KSWAPD_LOW_WMARK_HIT_QUICKLY counter.
  - Even after sleeping for 0.1 seconds, it is not possible to sleep because there is not enough memory to meet the high watermark standard. In these cases, the KSWAPD_HIGH_WMARK_HIT_QUICKLY counter is incremented.
- Remove the current task from kswapd_wait in section 78 of the code.

The following figure shows the process of using the kswapd_try_to_sleep() function to slip when kswapd meets the high watermark criteria.



(http://jake.dothome.co.kr/wp-content/uploads/2016/12/kswapd_try_to_sleep-1b.png)

Recall pages until balanced

balance_pgdat()

mm/vmscan.c -1/3-

```
/*
```

```
 * For kswapd, balance_pgdat() will reclaim pages across a node from zones
 * that are eligible for use by the caller until at least one zone is
 * balanced.
 *
 * Returns the order kswapd finished reclaiming at.
 *
 * kswapd scans the zones in the highmem->normal->dma direction.  It skips
 * zones which have free_pages > high_wmark_pages(zone), but once a zone is
 * found to have free_pages <= high_wmark_pages(zone), any page is that zone
 * or lower is eligible for reclaim until at least one usable zone is
 * balanced.
 */
```

```
static int balance_pgdat(pg_data_t *pgdat, int order, int classzone_idx)
{
        int i;
        unsigned long nr_soft_reclaimed;
        unsigned long nr_soft_scanned;
        unsigned long pflags;
        unsigned long nr_boost_reclaim;
        unsigned long zone_boosts[MAX_NR_ZONES] = { 0, };
        bool boosted;
        struct zone *zone;
        struct scan_control sc = {
                .gfp_mask = GFP_KERNEL,
                .order = order,
                .may_unmap = 1,
        };

        psi_memstall_enter(&pflags);
        __fs_reclaim_acquire();

        count_vm_event(PAGEOUTRUN);

        /*
         * Account for the reclaim boost. Note that the zone boost is left in
         * place so that parallel allocations that are near the watermark will
         * stall or direct reclaim until kswapd is finished.
         */
        nr_boost_reclaim = 0;
        for (i = 0; i <= classzone_idx; i++) {
                zone = pgdat->node_zones + i;
                if (!managed_zone(zone))
                        continue;

                nr_boost_reclaim += zone->watermark_boost;
                zone_boosts[i] = zone->watermark_boost;
        }
        boosted = nr_boost_reclaim;

restart:
        sc.priority = DEF_PRIORITY;
        do {
                unsigned long nr_reclaimed = sc.nr_reclaimed;
                bool raise_priority = true;
                bool balanced;
                bool ret;

                sc.reclaim_idx = classzone_idx;

                /*
```

```
49          * If the number of buffer_heads exceeds the maximum allowed
50          * then consider reclaiming from all zones. This has a dual
51          * purpose -- on 64-bit systems it is expected that
52          * buffer_heads are stripped during active rotation. On 32-bit
53          * systems, highmem pages can pin lowmem memory and shrinking
54          * buffers can relieve lowmem pressure. Reclaim may still not
55          * go ahead if all eligible zones for the original allocation
56          * request are balanced to avoid excessive reclaim from kswapd.
57          */
58         if (buffer_heads_over_limit) {
59                 for (i = MAX_NR_ZONES - 1; i >= 0; i--) {
60                         zone = pgdat->node_zones + i;
61                         if (!managed_zone(zone))
62                                 continue;
63
64                         sc.reclaim_idx = i;
65                         break;
66                 }
67         }
```

Prioritize from 12 to 1 and proceed with page reclamation and compaction, until the free page meets the high watermark criteria in a balanced manner to the requested order and zone.

- Set may_unmap=11 so that the page mapped to the scan_control structure prepared in line 15~1 can be unmapped.
- This is the point at line 17 of code where we start calculating the psi of the current task due to insufficient memory.
    - 참고: PSI – Pressure Stall Information (https://www.kernel.org/doc/Documentation/accounting/psi.txt) | kernel.org
- Increment the PAGEOUTRUN counter on line 20 of code.
- Travers the zones below classzone_idx requested in code lines 27~36, add up the watermark boost values, and assign them to boosted and nr_boost_reclaim. In addition, the zone boost value is also assigned to zone_boosts[].
- On line 38~40 of code, the restart: label is. Set the priority to the first value (12) and try again.
- In code lines 46~67, use classzone_idx as the zone index value to be reclaimed, but substitute the available top-level zone if the buffer_heads_over_limit is set.

mm/vmscan.c -2/3-

```
01 .       /*
02          * If the pgdat is imbalanced then ignore boosting and preserve
03          * the watermarks for a later time and restart. Note that the
04          * zone watermarks will be still reset at the end of balancing
05          * on the grounds that the normal reclaim should be enough to
06          * re-evaluate if boosting is required when kswapd next wakes.
07          */
```

```
                balanced = pgdat_balanced(pgdat, sc.order, classzone_idx);
                if (!balanced && nr_boost_reclaim) {
                        nr_boost_reclaim = 0;
                        goto restart;
                }

                /*
                 * If boosting is not active then only reclaim if there are no
                 * eligible zones. Note that sc.reclaim_idx is not used as
                 * buffer_heads_over_limit may have adjusted it.
                 */
                if (!nr_boost_reclaim && balanced)
                        goto out;

                /* Limit the priority of boosting to avoid reclaim writeback */
                if (nr_boost_reclaim && sc.priority == DEF_PRIORITY - 2)
                        raise_priority = false;

                /*
                 * Do not writeback or swap pages for boosted reclaim. The
                 * intent is to relieve pressure not issue sub-optimal IO
                 * from reclaim context. If no pages are reclaimed, the
                 * reclaim will be aborted.
                 */
                sc.may_writepage = !laptop_mode && !nr_boost_reclaim;
                sc.may_swap = !nr_boost_reclaim;
                sc.may_shrinkslab = !nr_boost_reclaim;

                /*
                 * Do some background aging of the anon list, to give
                 * pages a chance to be referenced before reclaiming. All
                 * pages are rotated regardless of classzone as this is
                 * about consistent aging.
                 */
                age_active_anon(pgdat, &sc);

                /*
                 * If we're getting trouble reclaiming, start doing writepage
                 * even in laptop mode.
                 */
                if (sc.priority < DEF_PRIORITY - 2)
                        sc.may_writepage = 1;

                /* Call soft limit reclaim before calling shrink_node. */
                sc.nr_scanned = 0;
                nr_soft_scanned = 0;
                nr_soft_reclaimed = mem_cgroup_soft_limit_reclaim(pgdat, sc.order,
                                                sc.gfp_mask, &nr_soft_scanned);
                sc.nr_reclaimed += nr_soft_reclaimed;

                /*
                 * There should be no need to raise the scanning priority if
                 * enough pages are already being scanned that that high
                 * watermark would be met at 100% efficiency.
                 */
                if (kswapd_shrink_node(pgdat, &sc))
```

```
64                                      raise_priority = false;
```

- In line 8~12 of the code, if the node is not in a balanced state and is in the process of boosting, reset the nr_boost_reclaim and then go to the restart: label to start over.
- In line 19~20 of the code, if the node is already balanced and not boosted, it doesn't need to get the page anymore, so it goes to the out label.
- In line 23~24 of the code, it doesn't matter if the priority is low (12~10) during the boost, but from the high rank (9~1), the raise_priority is assigned false so that the priority is no longer higher.
    - Whenever possible, low-priority writebacks are not allowed.
- In line 32~34 of the code, if the laptop is not in power saving mode and is not under boost, set the may_writepage to 1 to allow writeback. And if it's not boosting, set the may_swap and may_shrinkslab to 1 to support swap and slap shrink.
- If swap is enabled on line 42 of the code, and the inactive anon is less than the active anon, it will perform a shrink on the active list to rebalance the active and inactive.
- In line 48~49 of code, in high priority (9~1), the may_writepage is set to 1 to allow writeback.
- On lines 52~56 of code, perform a memcg soft limit reclaim before shrinking the node. The number of scans is assigned to the nr_soft_scanned, and the number of soft-reclaimed pages is added to the nr_reclaimed.
- In lines 63~64 of the code, shrink the node enough for the free page to meet the high watermark criteria. If shrink succeeds, set the raise_priority to false so that it doesn't increase the ranking.

mm/vmscan.c -3/3-

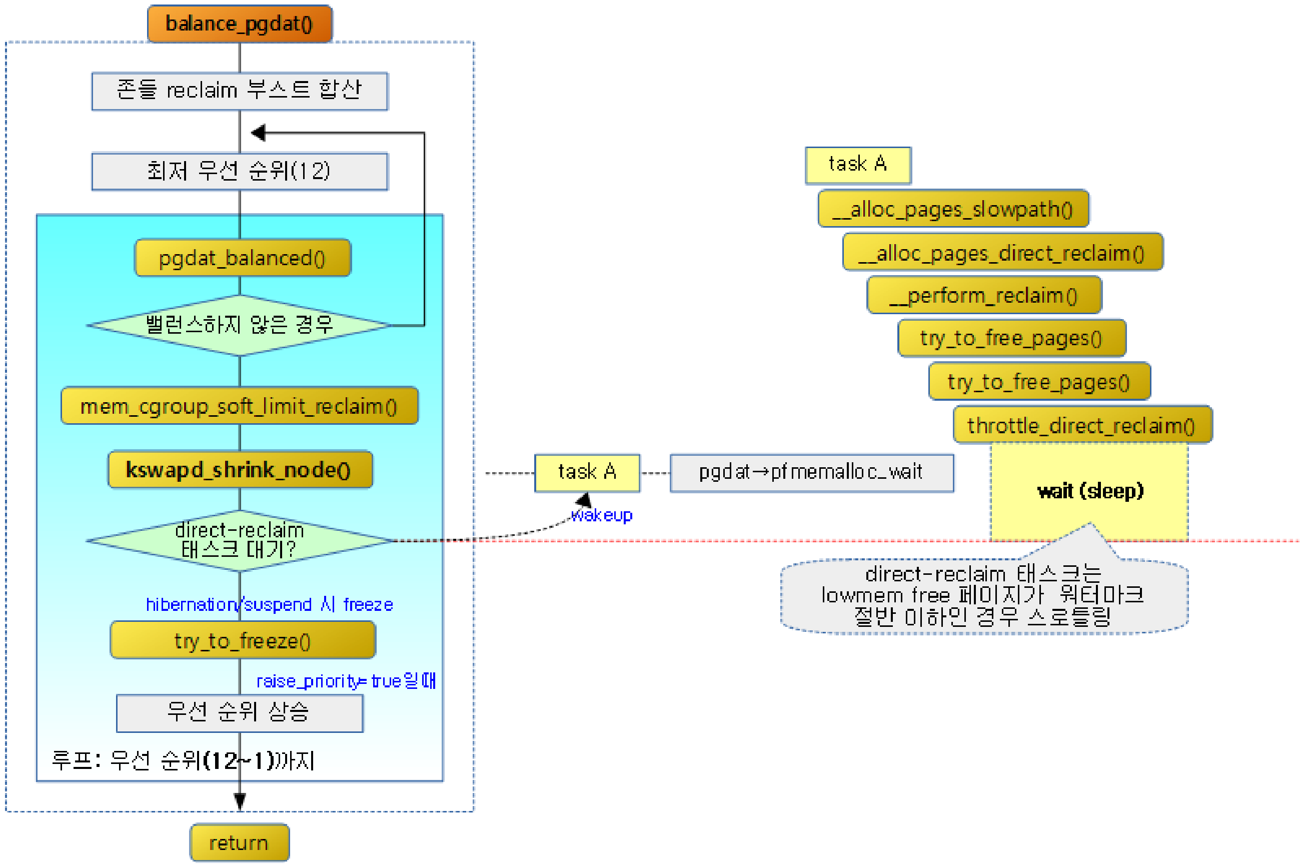
```
01                /*
02                 * If the low watermark is met there is no need for processes
03                 * to be throttled on pfmemalloc_wait as they should not be
04                 * able to safely make forward progress. Wake them
05                 */
06                if (waitqueue_active(&pgdat->pfmemalloc_wait) &&
07                                allow_direct_reclaim(pgdat))
08                        wake_up_all(&pgdat->pfmemalloc_wait);
09
10                /* Check if kswapd should be suspending */
11                __fs_reclaim_release();
12                ret = try_to_freeze();
13                __fs_reclaim_acquire();
14                if (ret || kthread_should_stop())
15                        break;
16
17                /*
18                 * Raise priority if scanning rate is too low or there was no
19                 * progress in reclaiming pages
20                 */
21                nr_reclaimed = sc.nr_reclaimed - nr_reclaimed;
22                nr_boost_reclaim -= min(nr_boost_reclaim, nr_reclaimed);
23
24                /*
25                 * If reclaim made no progress for a boost, stop reclaim as
26                 * IO cannot be queued and it could be an infinite loop in
27                 * extreme circumstances.
```

```
28                  */
29                 if (nr_boost_reclaim && !nr_reclaimed)
30                         break;
31 
32                 if (raise_priority || !nr_reclaimed)
33                         sc.priority--;
34         } while (sc.priority >= 1);
35 
36         if (!sc.nr_reclaimed)
37                 pgdat->kswapd_failures++;
38 
39 out:
40         /* If reclaim was boosted, account for the reclaim done in this pass */
41         if (boosted) {
42                 unsigned long flags;
43 
44                 for (i = 0; i <= classzone_idx; i++) {
45                         if (!zone_boosts[i])
46                                 continue;
47 
48                         /* Increments are under the zone lock */
49                         zone = pgdat->node_zones + i;
50                         spin_lock_irqsave(&zone->lock, flags);
51                         zone->watermark_boost -= min(zone->watermark_boost, zone_boosts[i]);
52                         spin_unlock_irqrestore(&zone->lock, flags);
53                 }
54 
55                 /*
56                  * As there is now likely space, wakeup kcompact to defragment
57                  * pageblocks.
58                  */
59                 wakeup_kcompactd(pgdat, pageblock_order, classzone_idx);
60         }
61 
62         snapshot_refaults(NULL, pgdat);
63         __fs_reclaim_release();
64         psi_memstall_leave(&pflags);
65         /*
66          * Return the order kswapd stopped reclaiming at as
67          * prepare_kswapd_sleep() takes it into account. If another caller
68          * entered the allocator slow path while kswapd was awake, order will
69          * remain at the higher level.
70          */
71         return sc.order;
72 }
```

- In line 6~8 of the code, if there are tasks in the list of pfmemalloc_wait that are waiting during the direct reclaim attempt due to insufficient memory during page allocation, and the node determines that it is okay to direct reclaim, it wakes up all the pending tasks.
    - allow_direct_reclaim(): true if the sum of the free pages under the normal zone is greater than half of the pages plus the min watermark.
- If you wake up after freeze on lines 12~15 of code, or if there is a request to stop the kswapd thread, exit the loop.
- After calculating the reclaimed pages and nr_boost_reclaim in lines 21~30 of the code, if they are not being boosted and there are no pages reclaimed, they are out of the loop.
- Repeat the loop until the highest priority, increasing the priority from line 32~34 of code. If there are no pages reclaimed, or if you don't want to increase priority, repeat the loop without priority

increase.
- If no pages are retrieved from code lines 36~37 until after the loop is completed, increment the kswapd_failures counter.
- In code lines 39~60, the out: label. If it was boosted on the first attempt, it will wake up kcompactd. It also updates the watermark boost value.
- This is the point at line 64 of code that ends the PSI calculation of the current task due to lack of memory.

The figure below shows that if task A drops below the pfmemalloc watermark threshold during the direct page reclaim, it will be throttled until the page is recalled by kswapd, that is, the direct page recall will be paused for a while to reduce the CPU load.



(http://jake.dothome.co.kr/wp-content/uploads/2016/12/balance_pgdat-1b.png)

Wake up Kswapd

wake_all_kswapds()

mm/page_alloc.c

```
static void wake_all_kswapds(unsigned int order, gfp_t gfp_mask,
                             const struct alloc_context *ac)
{
        struct zoneref *z;
        struct zone *zone;
        pg_data_t *last_pgdat = NULL;
        enum zone_type high_zoneidx = ac->high_zoneidx;

        for_each_zone_zonelist_nodemask(zone, z, ac->zonelist, high_zoneidx,
                                        ac->nodemask) {
                if (last_pgdat != zone->zone_pgdat)
                        wakeup_kswapd(zone, gfp_mask, order, high_zoneidx);
                last_pgdat = zone->zone_pgdat;
        }
}
```

Wakes up the kswpad of the relevant node in the zonelist that the alloc context points to.

wakeup_kswapd()

mm/vmscan.c

```
/*
 * A zone is low on free memory or too fragmented for high-order memory.  If
 * kswapd should reclaim (direct reclaim is deferred), wake it up for the zone's
 * pgdat.  It will wake up kcompactd after reclaiming memory.  If kswapd reclaim
 * has failed or is not needed, still wake up kcompactd if only compaction is
 * needed.
 */
```

```
void wakeup_kswapd(struct zone *zone, gfp_t gfp_flags, int order,
                   enum zone_type classzone_idx)
{
        pg_data_t *pgdat;

        if (!managed_zone(zone))
                return;

        if (!cpuset_zone_allowed(zone, gfp_flags))
                return;
        pgdat = zone->zone_pgdat;
        pgdat->kswapd_classzone_idx = kswapd_classzone_idx(pgdat,
                                                           classzone_idx);
        pgdat->kswapd_order = max(pgdat->kswapd_order, order);
        if (!waitqueue_active(&pgdat->kswapd_wait))
                return;

        /* Hopeless node, leave it to direct reclaim if possible */
        if (pgdat->kswapd_failures >= MAX_RECLAIM_RETRIES ||
            (pgdat_balanced(pgdat, order, classzone_idx) &&
             !pgdat_watermark_boosted(pgdat, classzone_idx))) {
                /*
                 * There may be plenty of free memory available, but it's too
                 * fragmented for high-order allocations.  Wake up kcompactd
                 * and rely on compaction_suitable() to determine if it's
                 * needed.  If it fails, it will defer subsequent attempts to
                 * ratelimit its work.
                 */
                if (!(gfp_flags & __GFP_DIRECT_RECLAIM))
                        wakeup_kcompactd(pgdat, order, classzone_idx);
                return;
        }

34         trace_mm_vmscan_wakeup_kswapd(pgdat->node_id, classzone_idx, order,
35                                       gfp_flags);
36         wake_up_interruptible(&pgdat->kswapd_wait);
37 }
```

If you run out of memory while trying to allocate an order page in a given zone, it wakes up the kswapd task.

- In line 6~7 of the code, if it is not a valid zone, it exits the function because there is no page to process.
- If the node in the zone requested in line 9~10 of the code is not authorized through cgroup cpuset, it will give up processing and exit.
- In code lines 11~14, make a request to kswapd by specifying the zone and order.
- In line 15~16 of the code, if kswapd is already working, the function exits.
- In line 19~32 of the code, only if the following conditions are met and direct-reclaim is not allowed, only kcompactd will be awakened and the function will exit.
    - If there are more than MAX_RECLAIM_RETRIES (16) page reclaim failures via KSWAD
    - If the node is already balanced and not boosting.
- At line 36 of code, wake up the kswapd task.

current_is_kswapd()

include/linux/swap.h

```
static inline int current_is_kswapd(void)
{
        return current->flags & PF_KSWAPD;
}
```

Returns true if the current task is kswapd.

kcompactd

Initialize kcompactd

kcompactd_init()

```
static int __init kcompactd_init(void)
{
        int nid;
        int ret;

        ret = cpuhp_setup_state_nocalls(CPUHP_AP_ONLINE_DYN,
                                        "mm/compaction:online",
                                        kcompactd_cpu_online, NULL);
        if (ret < 0) {
                pr_err("kcompactd: failed to register hotplug callbacks.\n");
                return ret;
        }

        for_each_node_state(nid, N_MEMORY)
                kcompactd_run(nid);
16              return 0;
17      }
18      subsys_initcall(kcompactd_init)
```

Initialize it to use kcompactd.

- In line 6~12 of the code, register that the kcompactd_cpu_online() function can be called when the CPU is changed to a CPUHP_AP_ONLINE_DYN state via hot-plug.
- In lines 14~15 of the code, run kcompactd on all memory nodes.

kcompactd_run()

mm/compaction.c

```
 1      /*
 2       * This kcompactd start function will be called by init and node-hot-add.
 3       * On node-hot-add, kcompactd will moved to proper cpus if cpus are hot-added.
 4       */
```

```
01      int kcompactd_run(int nid)
02      {
03              pg_data_t *pgdat = NODE_DATA(nid);
04              int ret = 0;
05
06              if (pgdat->kcompactd)
07                      return 0;
08
09              pgdat->kcompactd = kthread_run(kcompactd, pgdat, "kcompactd%d", nid);
10              if (IS_ERR(pgdat->kcompactd)) {
11                      pr_err("Failed to start kcompactd on node %d\n", nid);
12                      ret = PTR_ERR(pgdat->kcompactd);
13                      pgdat->kcompactd = NULL;
14              }
15              return ret;
16      }
```

kcompactd thread.

- In code lines 3~7, if the kcompactd thread on the @nid node is already running, it returns 0 to skip.
- Run the kcompactd thread on lines 9~14 of code.

kcompactd behavior

kcompactd()

mm/compaction.c

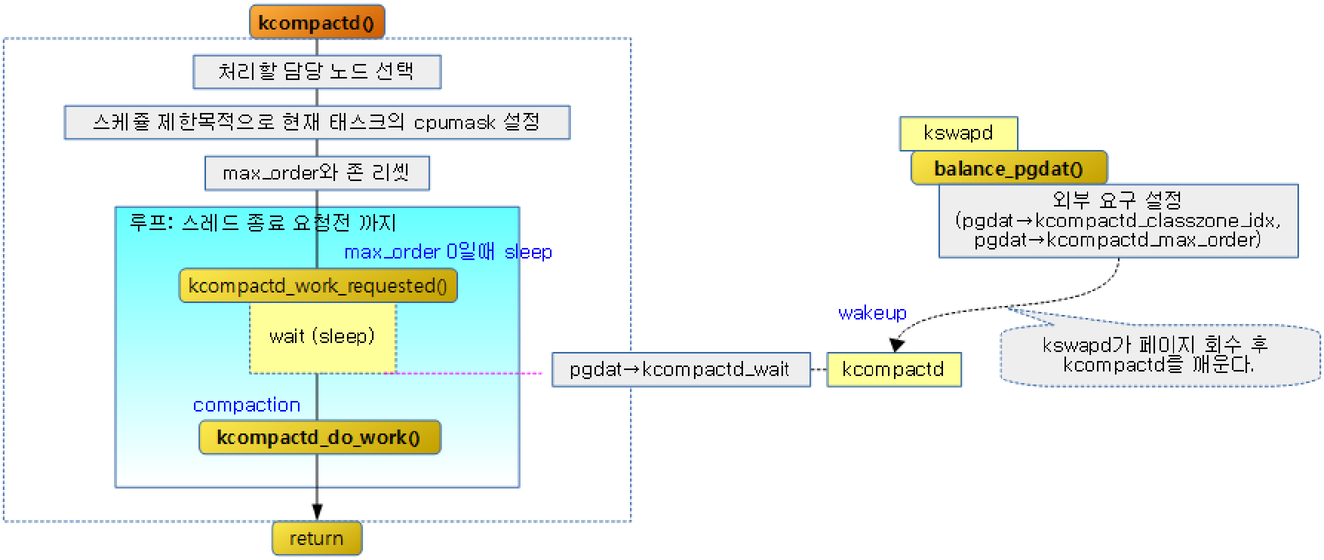
```
 1      /*
 2       * The background compaction daemon, started as a kernel thread
 3       * from the init process.
 4       */
```

```
01      static int kcompactd(void *p)
02      {
03              pg_data_t *pgdat = (pg_data_t*)p;
04         struct task_struct *tsk = current;
05 
06         const struct cpumask *cpumask = cpumask_of_node(pgdat->node_id);
07 
08         if (!cpumask_empty(cpumask))
09                 set_cpus_allowed_ptr(tsk, cpumask);
10 
11         set_freezable();
12 
13         pgdat->kcompactd_max_order = 0;
14         pgdat->kcompactd_classzone_idx = pgdat->nr_zones - 1;
15 
16         while (!kthread_should_stop()) {
17                 unsigned long pflags;
18 
19                 trace_mm_compaction_kcompactd_sleep(pgdat->node_id);
20                 wait_event_freezable(pgdat->kcompactd_wait,
21                                 kcompactd_work_requested(pgdat));
22 
23                 psi_memstall_enter(&pflags);
24                 kcompactd_do_work(pgdat);
25                 psi_memstall_leave(&pflags);
26         }
27 
28         return 0;
29 }
```

The kcompactd thread running on each memory node is in a sleepy state, and when kswapd is called after recalling the page, it wakes up and performs compaction in the background and sleeps again.

- In lines 3~9 of the code, specify a CPU bitmask so that the current kcompactd thread can only work on the CPUs included in the requesting node.
- Remove the PF_NOFREEZE flag from line 11 of code so that you can freeze the task.
  - See also: freeze (http://jake.dothome.co.kr/freeze) | Qc
- In code lines 13~14, reset the maximum order and zone of kcompactd.
- In line 16~26 of code, unless there is an exit request, it will continue to loop and sleep, and then if it is woken up by an external request, it will perform a compaction.

The following diagram shows how the kcompactd thread works.



(http://jake.dothome.co.kr/wp-content/uploads/2016/12/kcompactd-1.png)

kcompactd_do_work()

mm/compaction.c

```
static void kcompactd_do_work(pg_data_t *pgdat)
{
        /*
         * With no special task, compact all zones so that a page of requested
         * order is allocatable.
         */
        int zoneid;
        struct zone *zone;
        struct compact_control cc = {
                .order = pgdat->kcompactd_max_order,
                .total_migrate_scanned = 0,
                .total_free_scanned = 0,
                .classzone_idx = pgdat->kcompactd_classzone_idx,
                .mode = MIGRATE_SYNC_LIGHT,
                .ignore_skip_hint = false,
                .gfp_mask = GFP_KERNEL,
        };
        trace_mm_compaction_kcompactd_wake(pgdat->node_id, cc.order,
                                                        cc.classzone_idx);
        count_compact_event(KCOMPACTD_WAKE);

        for (zoneid = 0; zoneid <= cc.classzone_idx; zoneid++) {
                int status;

                zone = &pgdat->node_zones[zoneid];
                if (!populated_zone(zone))
                        continue;

                if (compaction_deferred(zone, cc.order))
                        continue;

                if (compaction_suitable(zone, cc.order, 0, zoneid) !=
                                                        COMPACT_CONTINUE)
                        continue;

                cc.nr_freepages = 0;
                cc.nr_migratepages = 0;
                cc.total_migrate_scanned = 0;
                cc.total_free_scanned = 0;
                cc.zone = zone;
                INIT_LIST_HEAD(&cc.freepages);
                INIT_LIST_HEAD(&cc.migratepages);

                if (kthread_should_stop())
                        return;
                status = compact_zone(zone, &cc);

                if (status == COMPACT_SUCCESS) {
                        compaction_defer_reset(zone, cc.order, false);
                } else if (status == COMPACT_PARTIAL_SKIPPED || status == COMPACT_COMPLETE) {
                        /*
                         * Buddy pages may become stranded on pcps that could
                         * otherwise coalesce on the zone's free area for
                         * order >= cc.order.  This is ratelimited by the
                         * upcoming deferral.
                         */
                        drain_all_pages(zone);
58
59                          /*
60                           * We use sync migration mode here, so we defer like
61                           * sync direct compaction does.
62                           */
63                          defer_compaction(zone, cc.order);
64                  }
65
66                  count_compact_events(KCOMPACTD_MIGRATE_SCANNED,
67                                       cc.total_migrate_scanned);
68                  count_compact_events(KCOMPACTD_FREE_SCANNED,
69                                       cc.total_free_scanned);
70
71                  VM_BUG_ON(!list_empty(&cc.freepages));
72                  VM_BUG_ON(!list_empty(&cc.migratepages));
73          }
74
75          /*
76           * Regardless of success, we are done until woken up next. But remember
77           * the requested order/classzone_idx in case it was higher/tighter than
78           * our current ones
79           */
80          if (pgdat->kcompactd_max_order <= cc.order)
81                  pgdat->kcompactd_max_order = 0;
82          if (pgdat->kcompactd_classzone_idx >= cc.classzone_idx)
83                  pgdat->kcompactd_classzone_idx = pgdat->nr_zones - 1;
84  }
```

Perform a kcompactd_max_order compaction to the kcompactd_classzone_idx zone specified in the node.

- In line 9~17 of code, prepare the compact_control for use in kcompactd as follows:
  - .order specifies the externally requested order.
  - Specify the externally requested zone index in the .classzone_idx.
  - Use MIGRATE_SYNC_LIGHT in .mode.
  - Use the skip hint.
- Increment KCOMPACTD_WAKE counter at line 20 of code.
- From the lowest zone in code lines 22~27 to the requested zone, the invalid zone is skipped.
- In line 29~30 of code, skip the zone with the compaction grace condition.
- Skip on code lines 32~34 if the zone is not appropriate for compaction.
- In line 36~42 of code, initialize the members that will contain the compaction_control result in order to perform the compaction.
- In line 44~45 of the code, if the request is to terminate the thread, exit the function.
- Perform a compaction on John on line 46 and find out the result.
- On lines 48~49 of code, if the compaction succeeds, the suspend counter is reset.
- In line 50~64 of the code, if there is no desired order by the time the compaction is completed, the per-CPU cache is reclaimed, and the grace limit is increased.
- Update the KCOMPACTD_MIGRATE_SCANNED counter and KCOMPACTD_FREE_SCANNED counter on code lines 66~69.
- If the external request order is less than or equal to the progress order in line 80~81, reset the max_order to 0 so that it does not wake up next time.
- If the external request zone is larger or equal to the progress zone in code lines 82~83, the next starting zone is reset to the highest zone.

Wake up Kcompatd

wakeup_kcompactd()

mm/compaction.c

```
01 void wakeup_kcompactd(pg_data_t *pgdat, int order, int classzone_idx)
02 {
03         if (!order)
04                 return;
05 
06         if (pgdat->kcompactd_max_order < order)
07                 pgdat->kcompactd_max_order = order;
08 
09         if (pgdat->kcompactd_classzone_idx > classzone_idx)
10                 pgdat->kcompactd_classzone_idx = classzone_idx;
11 
12         /*
13          * Pairs with implicit barrier in wait_event_freezable()
14          * such that wakeups are not missed.
15          */
16         if (!wq_has_sleeper(&pgdat->kcompactd_wait))
17                 return;
18 
19         if (!kcompactd_node_suitable(pgdat))
20                 return;
21 
22         trace_mm_compaction_wakeup_kcompactd(pgdat->node_id, order,
23                                                         classzone_idx);
24         wake_up_interruptible(&pgdat->kcompactd_wait);
25 }
```

Wake up the kcompactd thread.

- In line 3~4 of code, if the order value is 0, it exits the function without waking kcompactd.
- If the @order is greater than kcompactd_max_order in line 6~7, update the kcompactd_max_order.
- In line 9~10 of the code, if the @classzone_idx is less than kcompactd_classzone_idx, update the kcompactd_classzone_idx.
- In lines 16~17 of code, if kcompactd is already awake, it exits the function.
- If there is a node that does not have an effect even if you proceed with the compaction on line 19~20 of the code, the function will be exited.
- Wake kcompactd on line 24 of code.

kcompactd_node_suitable()

mm/compaction.c

```
01 static bool kcompactd_node_suitable(pg_data_t *pgdat)
02 {
03         int zoneid;
04         struct zone *zone;
05         enum zone_type classzone_idx = pgdat->kcompactd_classzone_idx;
06 
07         for (zoneid = 0; zoneid <= classzone_idx; zoneid++) {
08                 zone = &pgdat->node_zones[zoneid];
09 
10                 if (!populated_zone(zone))
11                         continue;
12 
13                 if (compaction_suitable(zone, pgdat->kcompactd_max_order, 0,
14                                         classzone_idx) == COMPACT_CONTINUE)
15                         return true;
16         }
17 
18         return false;
19 }
```

Returns whether there are any zones that would have a compaction effect on any of the available zones of the node requested to perform kcompactd.

- Travers the available zones up to the kcompactd_classzone_idx of the nodes requested in code lines 5~11.
- If any of the traversing zones in line 13~15 are judged to have a compaction effect using a kcompactd_max_order value, it returns true.
- In line 18 of the code, it returns false because it does not see the compaction effect for all zones on that node.

guitar

swapper_spaces[] array

mm/swap_state.c

```
struct address_space swapper_spaces[MAX_SWAPFILES];
```

swap_aops

mm/swap_state.c

```
/*
 * swapper_space is a fiction, retained to simplify the path through
 * vmscan's shrink_page_list.
 */
static const struct address_space_operations swap_aops = {
        .writepage      = swap_writepage,
        .set_page_dirty = swap_set_page_dirty,
#ifdef CONFIG_MIGRATION
        .migratepage    = migrate_page,
#endif
};
```

address_space_operations Struct

include/linux/fs.h

```
struct address_space_operations {
        int (*writepage)(struct page *page, struct writeback_control *wbc);
        int (*readpage)(struct file *, struct page *);

        /* Write back some dirty pages from this mapping. */
        int (*writepages)(struct address_space *, struct writeback_control *);

        /* Set a page dirty.  Return true if this dirtied it */
        int (*set_page_dirty)(struct page *page);

        /*
         * Reads in the requested pages. Unlike ->readpage(), this is
         * PURELY used for read-ahead!.
         */
        int (*readpages)(struct file *filp, struct address_space *mapping,
                        struct list_head *pages, unsigned nr_pages);

        int (*write_begin)(struct file *, struct address_space *mapping,
                                loff_t pos, unsigned len, unsigned flags,
                                struct page **pagep, void **fsdata);
        int (*write_end)(struct file *, struct address_space *mapping,
                                loff_t pos, unsigned len, unsigned copied,
                                struct page *page, void *fsdata);

        /* Unfortunately this kludge is needed for FIBMAP. Don't use it */
        sector_t (*bmap)(struct address_space *, sector_t);
        void (*invalidatepage) (struct page *, unsigned int, unsigned int);
        int (*releasepage) (struct page *, gfp_t);
        void (*freepage)(struct page *);
        ssize_t (*direct_IO)(struct kiocb *, struct iov_iter *iter);
        /*
         * migrate the contents of a page to the specified target. If
         * migrate_mode is MIGRATE_ASYNC, it must not block.
         */
        int (*migratepage) (struct address_space *,
                        struct page *, struct page *, enum migrate_mode);
        bool (*isolate_page)(struct page *, isolate_mode_t);
        void (*putback_page)(struct page *);
        int (*launder_page) (struct page *);
        int (*is_partially_uptodate) (struct page *, unsigned long,
                                        unsigned long);
        void (*is_dirty_writeback) (struct page *, bool *, bool *);
        int (*error_remove_page)(struct address_space *, struct page *);

        /* swapfile support */
        int (*swap_activate)(struct swap_info_struct *sis, struct file *file,
                                sector_t *span);
        void (*swap_deactivate)(struct file *file);
};
```

consultation

- Zoned Allocator -1- (Physics Page Assignment - Fastpath) (http://jake.dothome.co.kr/zonned-allocator-alloc-pages-fastpath) | Qc
- Zoned Allocator -2- (Physics Page Assignment - Slowpath) (http://jake.dothome.co.kr/zonned-allocator-alloc-pages-slowpath) | Qc
- Zoned Allocator -3- (Buddy Page Allocation) (http://jake.dothome.co.kr/buddy-alloc) | Qc

- Zoned Allocator -4- (Buddy Page Terminated) (http://jake.dothome.co.kr/buddy-free/) | Qc
- Zoned Allocator -5- (Per-CPU Page Frame Cache) (http://jake.dothome.co.kr/per-cpu-page-frame-cache) | 문c
- Zoned Allocator -6- (Watermark) (http://jake.dothome.co.kr/zonned-allocator-watermark) | 문c
- Zoned Allocator -7- (Direct Compact) (http://jake.dothome.co.kr/zonned-allocator-compaction) | 문c
- Zoned Allocator -8- (Direct Compact-Isolation) (http://jake.dothome.co.kr/zonned-allocator-isolation) | 문c
- Zoned Allocator -9- (Direct Compact-Migration) (http://jake.dothome.co.kr/zonned-allocator-migration) | 문c
- Zoned Allocator -10- (LRU & pagevec) (http://jake.dothome.co.kr/lru-lists-pagevecs) | 문c
- Zoned Allocator -11- (Direct Reclaim) (http://jake.dothome.co.kr/zonned-allocator-reclaim) | 문c
- Zoned Allocator -12- (Direct Reclaim-Shrink-1) (http://jake.dothome.co.kr/zonned-allocator-shrink-1) | 문c
- Zoned Allocator -13- (Direct Reclaim-Shrink-2) (http://jake.dothome.co.kr/zonned-allocator-shrink-2) | 문c
- Zoned Allocator -14- (Kswapd & Kcompactd) (http://jake.dothome.co.kr/zonned-allocator-kswapd) | Sentence C – Current post

13 thoughts to “Zoned Allocator -14- (Kswapd & Kcompactd)”

KIM MOON-SEOP
2018-01-14 19:26 (http://jake.dothome.co.kr/zonned-allocator-kswapd/#comment-175251)

Thanks for the detailed explanation

RESPONSE (/ZONNED-ALLOCATOR-KSWAPD/?REPLYTOCOM=175251#RESPOND)

MOON YOUNG-IL (HTTP://JAKE.DOTHOME.CO.KR)
2018-01-15 17:13 (http://jake.dothome.co.kr/zonned-allocator-kswapd/#comment-175264)

I hope my study has helped you a little. Happy New Year!.

RESPONSE (/ZONNED-ALLOCATOR-KSWAPD/?REPLYTOCOM=175264#RESPOND)

ILSEOP HWANG
2018-06-27 17:19 (http://jake.dothome.co.kr/zonned-allocator-kswapd/#comment-178529)

It was a great help. It's been a long time since a site has been described so well in Korean.
I have a question for you.
kzalloc(alloc_size, GFP_KERNEL); The result of the ZONEINFO is null, as shown below. (It's a mobile device and has 2G RAM)
Node 0, zone DMA
pages free 6985
min 1342
low 4204
high 4540
——————————————————-
cat proc/sys/vm/min_free_kbytes : read 5368

Question.
1. If you change the following changes, the malloc error will be improved, but is it correct to increase it manually like this?
echo 107216 > /proc/sys/vm/min_free_kbytes
2. If not, what can I do to free up malloc memory?
I appreciate it.

RESPONSE (/ZONNED-ALLOCATOR-KSWAPD/?REPLYTOCOM=178529#RESPOND)

MOON YOUNG-IL (HTTP://JAKE.DOTHOME.CO.KR)
2018-06-27 21:13 (http://jake.dothome.co.kr/zonned-allocator-kswapd/#comment-178532)

Nice to meet you.

If you used the GFP_KERNEL flag in the kzalloc() function, it would behave as follows for kernel memory allocation:
Depending on
the alloc_size – if it's 8K or less, it will allocate slub memory for kmalloc in units to the power of 2, and – if it's above 8K,
the buddy system will allocate pages right away.

현재 시스템 메모리 상태를 보니 남은 메모리 여분이 약 28M이고, 약 20M 미만으로 내려가면 kswapd를 통해 백그라운드에서 메모리를 확보하라고 한 상태입니다.
위의 상태라면 보통 alloc_size가 약간 큰 페이지를 할당하려고 한 것 같습니다.
28M의 여분이 있다 하더라도 버디시스템에서 연속된 큰 페이지들이 모자란 상태인 듯 합니다.
..
메모리를 지속적으로 할당하고 다시 풀어주고를 반복하는 경우 황일섭님이 설정하신 것 같이 워터마크 기준을 높이면 메모리가 shortage 나기 전에 compaction 및 reclaim 등을 통해 메모리가 다시 확보되니 대부분 해결이 됩니다. (물론 지속적으로 할당을 시도하는 상황에서 할당 해제되는 메모리가 계속 모자라지는 버그 또는 설계가 잘못된 demon이 없다는 가정입니다)

그런데 휴대폰이라면 사용자가 메모리가 큰 게임 등을 구동하는 경우 메모리 관리 앱을 사용하여 정리하곤 하는데 그 와는 다른 상황인가 보네요?
참고로 휴대폰이 아니고 1년 365일 계속 동작해야 하는 임베디드 시스템인 경우에는 메모리가 줄어들지 않도록 충분히 잘 설계하는 것으로 회피합니다.

감사합니다.

응답 (/ZONNED-ALLOCATOR-KSWAPD/?REPLYTOCOM=178532#RESPOND)

황일섭
2018-06-28 13:50 (http://jake.dothome.co.kr/zonned-allocator-kswapd/#comment-178554)

답변 감사합니다.

응답 (/ZONNED-ALLOCATOR-KSWAPD/?REPLYTOCOM=178554#RESPOND)

KEVIN LEE
2019-05-08 17:42 (http://jake.dothome.co.kr/zonned-allocator-kswapd/#comment-212866)

좋은 글 감사합니다.
근데 하나 궁금한 게 있습니다.
메모리를 회수하고 나서 storage device로 swap partition(예를 들어 /swap)으로 보내줘야 할 것 같은데
이와 관련된 동작이 언제 어디서 일어나는지 알 수 있을까요??

응답 (/ZONNED-ALLOCATOR-KSWAPD/?REPLYTOCOM=212866#RESPOND)

KEVIN LEE
2019-05-08 20:08 (http://jake.dothome.co.kr/zonned-allocator-kswapd/#comment-212896)

swap_writepage()를 통해 swap partition으로 쓰여지는 것으로 이해되는데요, 문제는 이 함수에 printk()를 심어놔도 trigger되는 경우가 없다는 것입니다.
제 컴퓨터에 swap partition이 따로 잡혀있지는 않고 swapfile만 존재하고 커널 버전은 5.0.5입니다.
fio benchmark를 통해 약 20gb를 쓰는데도(제 DRAM용량은 16GB입니다) trigger되는 경우가 없습니다. 혹시 조언을 받을 수 있을까요?
혹시, mmap()등을 통해 anonymous로 매핑 후에 실험을 해야 할까요?

응답 (/ZONNED-ALLOCATOR-KSWAPD/?REPLYTOCOM=212896#RESPOND)

문영일 (HTTP://JAKE.DOTHOME.CO.KR)

2019-05-09 17:03 (http://jake.dothome.co.kr/zonned-allocator-kswapd/#comment-213087)

안녕하세요?

swap을 동작시키려면 유저 레벨에서 할당하는 anon 메모리에 기록을 해야 합니다.
그냥 유저 레벨 application을 작성할 때 반복 루프 내에서 malloc()을 사용하시고 memset()으로 아무 값이나 기록해보시면 알 수 있을 것입니다.
참고: https://linuxize.com/post/create-a-linux-swap-file (https://linuxize.com/post/create-a-linux-swap-file)

감사합니다.

응답 (/ZONNED-ALLOCATOR-KSWAPD/?REPLYTOCOM=213087#RESPOND)

권용범
2021-10-13 22:52 (http://jake.dothome.co.kr/zonned-allocator-kswapd/#comment-306072)

안녕하세요.

오타로 보이는 부분입니다.
1. "코드 라인 3~7에서 @nid 노드의 kswapd 태스크가 지정되지 않은 경우 0을 반환한다" (지정되지 않은 -> 지정된, 이미 실행중인)
2. "코드 라인 3~7에서 @nid 노드의 kcompactd 태스크가 지정되지 않은 경우 0을 반환한다." (지정되지 않은 -> 지정된, 이미 실행중인)
3. "다음 그림은 kswapd_try_to_sleep() 함수를 통해 kswapd가 high 워터마크 기준을 충족하면 슬립하는 과정을 보여준다" 그림에서
0.1초 잠든 이후에 high 워터마크 조건에 들지 않은 경우
(KSWAPD_LOW_WMARK_HIT_QUICKLY -> KSWAPD_HIGH_WMARK_HIT_QUICKLY)
4. "그리고 부트트 중이 아니면" (부트트 -> 부스트)
5. "아래 그림은 task A에서 direct 페이지 회수를 진행 중에 pfmemalloc 워터마크 기준 이하로…" 그림에서
"밸런스 하지 않은 경우" 의 화살표 목적지가 "존들 reclaim 부스트 합산 아래", "최저 우선 순위" 위가 되어야 할 것 같습니다.

그리고 질문이 있습니다.
"요청한 order에서 회수가 실패한 경우 order 0 및 해당 존에서 다시 시도하기 위해 try_sleep: 레이블로 이동한다"
balance_pgdat의 반환값이 sc.order 인데 이 부분을 변경하는 코드가 kswapd_shrink_node 함수에서 nr_reclaimed 가 compact_gap 반환값이상
일때 order를 0으로 변경하는 경우이고 이 경우는 실패라기 보다는 order의 2배 이상으로 충분히 회수해서 reclaim 보다는 compaction을
수행하려는 의도처럼 보여지는데요. 제가 뭘 놓친건가요?

응답 (/ZONNED-ALLOCATOR-KSWAPD/?REPLYTOCOM=306072#RESPOND)

문영일 (HTTP://JAKE.DOTHOME.CO.KR)
2021-10-14 14:54 (http://jake.dothome.co.kr/zonned-allocator-kswapd/#comment-306074)

안녕하세요? 권용범님.

1번부터 5번까지 잘못된 오자는 모두 수정하였습니다. 세심히 봐주셔서 감사합니다. ^^

잘 아시는 것처럼 kswapd는 보통 슬립상태에 있다가, 메모리 부족 상황에서 깨어나서 동작합니다. 그런 후 메모리 확보가 완료되면 다시 잠들게됩니다.

if (reclaim_order < alloc_order) 코드에서 alloc_order는 사용자가 최근에 할당을 원하는 order 페이지이고, reclaim_order는 kswapd에 의해 회수한 페이지 order 중 가장 큰 order 입니다. 이를 비교하는 이유는 당연히 원하는 만큼 order를 확보했는지 비교하고, 확보가 완료된 경우 새로운 alloc order 요청을 수행하기 위해 다시 루프를 돌것이고, 확보를 실패한 경우 kswapd_try_sleep: 레이블로 다시 back 하여 기존 요청을 계속 수행합니다. 루프 내에서 0.1초씩 sleep 하는 이유는 reclaim 반복만으로는 빠르게 회수되지 않는 페이지를 계속 시도하여 cpu performance를 떨어뜨리므로, kswap를 잠시 슬립하고, 이 때 compaction에 의해 확보가 될 수 있는 기대 역시 합니다. 감사합니다.

응답 (/ZONNED-ALLOCATOR-KSWAPD/?REPLYTOCOM=306074#RESPOND)

권용범
2021-10-14 16:35 (http://jake.dothome.co.kr/zonned-allocator-kswapd/#comment-306075)

“reclaim_order는 kswapd에 의해 회수한 페이지 order 중 가장 큰 order 입니다” 이 부분에서 헤매고 있습니다. 원래 질문의 요지이기도 하구요.
코드상에서 reclaim_order는 balance_pgdat 가 반환하는 sc.order인데 alloc_order로 설정된 sc.order 가 변경되는 부분을 위질문에서 언급한 부분외에서는
못찾고 있습니다. 회수한 페이지 order중 가장 큰 order로 reclaim_order를 설정하는 부분이 코드상 어느 부분인가요?
감사합니다.

응답 (/ZONNED-ALLOCATOR-KSWAPD/?REPLYTOCOM=306075#RESPOND)

문영일 (HTTP://JAKE.DOTHOME.CO.KR)
2021-10-15 09:23 (http://jake.dothome.co.kr/zonned-allocator-kswapd/#comment-306076)

권용범님이 고심하는 부분을 코드를 다시 살펴보니 제가 reclaim_order에 대해 잘못 해석했다는 것을 알았습니다.
reclaim_order는 회수 페이지 order 중 가장 큰 order라고 했는데, 이를 정정합니다. ^^;

direct-reclaim과 kswapd는 워터마크를 기준으로 서로 경쟁하는 관계입니다.
kswapd가 회수한 페이지가 compact_gap() 만큼 즉 워터마크(high) + 원하는 alloc_order의 2배 이상 충분히 확보한 경우 sc->order를 0으로 낮춥니다.
이렇게 0으로 낮추고 kswapd_try_sleep: 레이블을 통해 kswapd_try_to_sleep()으로 진행

할 때 의미가 있군요.
kswapd는 cost가 높으므로 direct-reclaim 쪽과 경쟁을 회피하기 위해 일부러 order를 0으로 내리면 밸런스 기준이 낮아지므로 곧바로 sleep하도록 도움을 주는 상황입니다.

감사합니다.

응답 (/ZONNED-ALLOCATOR-KSWAPD/?REPLYTOCOM=306076#RESPOND)

권용범
2021-10-15 17:18 (http://jake.dothome.co.kr/zonned-allocator-kswapd/#comment-306078)

친절한 답변 감사합니다.

응답 (/ZONNED-ALLOCATOR-KSWAPD/?REPLYTOCOM=306078#RESPOND)

댓글 남기기

이메일은 공개되지 않습니다. 필수 입력창은 * 로 표시되어 있습니다

댓글

이름 *

이메일 *

웹사이트

댓글 작성

Freeze (hibernation/suspend) (http://jake.dothome.co.kr/freeze/)

numa_policy_init() (http://jake.dothome.co.kr/numa_policy_init/)