Making Huge Pages Effective

Aayush Desai
DAIICT
201801060@daiict.ac.in

Rutwa Rami
DAIICT
201801205@daiict.ac.in

Mitesh Koradia
DAIICT
201801017@daiict.ac.in

***Abstract*—To increase multiprogramming, Virtualisation concept is used in operating systems. To implement this we use page tables which are themselves stored into the memory. So for every memory read that take place first virtual address is converted to physical address using page tables. To make this translation faster TLB is used. But even after using TLB the address translation remains a performance bottleneck for modern systems. The effect of this can be reduced by huge page allocation. By large page allocation more data can be kept in memory which increases the hit rate of the TLB. But further we will see how fragmentation of the memory doesn't allow us to use huge pages.**

I. Introduction

A. Unmovable Pages

Kernel pages are known as unmovable pages. These pages are mostly for the objects handled by the kernel such as inodes etc. These pages are directly mapped into the physical memory which helps in faster access. So they can not be moved in the memory and also can not be swapped into the disk once they are mapped.

B. Movable Pages

These are the user level pages which are generally requested by the applications. They are movable because they are maintained by reference. The virtual address is converted into a physical address with the help of page tables. So if the pages are swapped into the disk then they can again be allocated at different memory locations.

C. Two way Classification of Memory

There are two types of memory allocation kernel and user level pages.

- The slab allocator is responsible for allocating pages to kernel objects.
- Buddy allocator provides pages to the slab allocator and also to the user space.
- If there is a lack of pages in one memory space then it can take from the other memory space.
- Fig-1 shows how the memory is allocated.

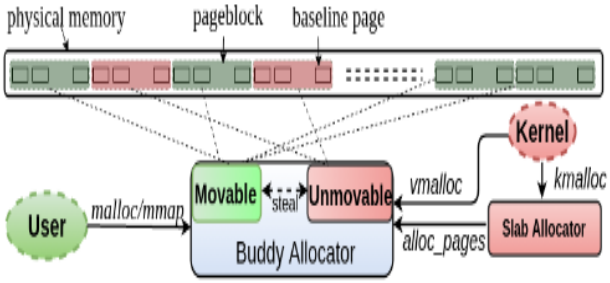

Fig. 1. Physical memory management in Linux. The slab allocator allocates kernel objects while the buddy alloca- tor serves pages to the slab allocator (from the unmovable region) and to the user space (from the movable region).

II. Things affecting usage of huge pages

A. Fragmentation-via-pollution:

- This occurs when memory is fragmented due to sparsely distributed unmovable pages.
- Kernel only knows about movable and unmovable pages. It does not know about which blocks have movable and unmovable pages both.
- So whenever there is a fallback then the block which is at the top is taken by the memory space.
- So due to this pure movable or pure unmovable page blocks get converted into hybrid blocks(blocks having both movable and unmovable pages).
- With time these hybrid blocks increase in number in the memory. Due to this memory gets fragmented and most of the huge page allocation requests fail.

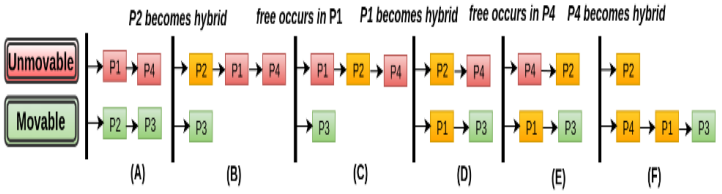

Fig. 2. Two-way classification based anti-fragmentation leads to fragmentation via pollution because the buddy allocator cannot reuse hybrid pageblocks during fallbacks. For example, pollution of P1 can be avoided by reusing P2 during C→D. For clarity, hybrid pageblocks are colored yellow in this figure, but the Linux kernel treats them as either red or green, depending on which region they belong to.

B. Latency-inducing unsuccessful migration:

This occurs when the kernel migrates pages to make continuous space for huge pages but is unsuccessful due to unmovable pages.

- In current implementation the success of huge page allocation requests depends on the location of the unmovable pages.
- Like in fig-3 (A) migrate scanner is pointing to P1 while free scanner is pointing to P3. So pages of P1 can be transferred to P3 and we can get continuous large space free.
- But in the case of fig-3 (B), the migrate scanner is pointing to P1 which is a hybrid block and the free scanner is pointing to P3. Two way classification doesn't know that there is an unmovable page in P1, so it transfers remaining unmovable pages to P3. But at last it encounters an unmovable page which it can not transfer. Now when the migrate scanner goes to P2 then it will not be able to transfer pages as P3 is already full with the pages of P1. So the time it took to transfer pages from P1 block is wasted because we were unable to get huge free space.

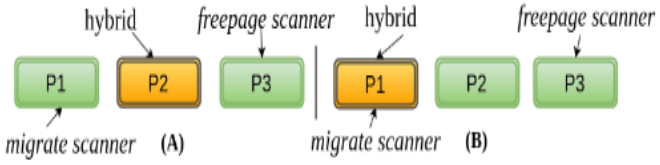

Fig. 3. The location of unmovable pages affects the out- come of compaction in the two-way classification approach. In this case, Linux can allocate a huge page only in scenario A while Illuminator can allocate huge page in both A and B.

III. Proposed Solution : Illuminator

A. Three way Classification:

In this solution we have added the functionality to keep track of hybrid blocks along with movable and unmovable blocks. In fig-4 the yellow color blocks are the hybrid blocks. Due to this the memory management unit has a clear idea that the red blocks have only the unmovable pages , green blocks have only movable pages and yellow blocks have both movable and unmovable pages.

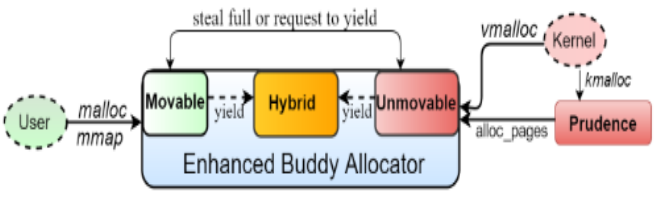

Fig. 4. Illuminator explicitly manages hybrid pageblocks in a different region to prevent fragmentation via pollu- tion. Prudence helps Illuminator by minimizing callbacks to alloc-pages with timely reclamation of deferred objects.

B. Preventing Fragmentation via Pollution:

Three Way classification helps to keep unmovable pages in a close area. When there is a fallback then memory is taken from hybrid blocks until the request can be fulfilled. Then if still memory is not available then it is stolen from another memory type. This ensures that the memory is not fragmented unnecessarily. It also provides a mechanism to remove page blocks from the hybrid region.

- In the following diagram fig-5 first the P2 blocks become hybrid and then P1 block is becoming hybrid.
- Now if the unmovable page is freed from the block P2 then the block is returned to the movable section.
- This helps in clustering of the unmovable pages at one place.

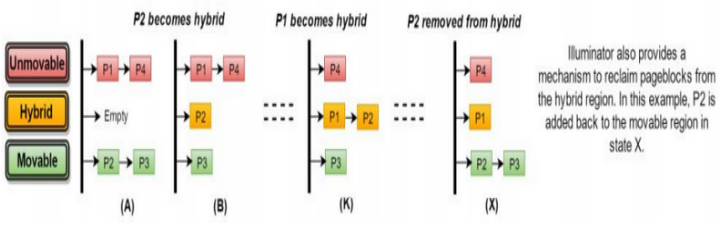

Fig. 5. Illuminator improves page clustering by explicitly managing hybrid pageblocks. Once P2 is yielded to the hybrid region during A→B, it is reused until state K. P1 is added to the hybrid region only when P2 fails to allocate memory.

C. Eliminating LIU migration:

- As now the memory management unit has the knowledge of hybrid blocks, so the migrate scanner just skips the hybrid blocks in the way.
- As hybrid blocks would not help us to fetch the huge space, skipping them helps in the unnecessary transfer of blocks to the free pages.
- This reduces the time taken by the compaction algorithm.

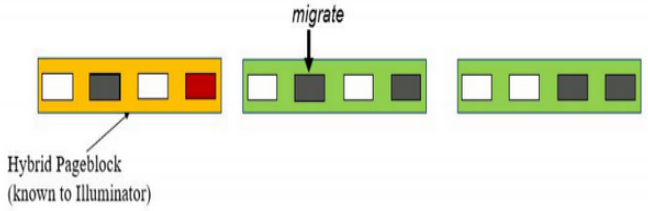

Fig. 6. Migrate pointer is skipping hybrid blocks and migrating from movable blocks.

D. Eliminating susceptibility to page locations:

In fig-3 (B) the Two way classification was not successful in allocating a huge page because it was not having knowledge of hybrid blocks. But in three way classification, the migrate scanner has the information about the hybrid blocks and they are just skipped.So block P1 would be skipped and the pages of P2 can be transferred to P3 and the huge page allocation

request is successful. In this way we have removed the dependence of compaction on spatial distribution of unmovable pages.

IV. Experiment

- We have implemented the Two way classification and Three way classification of memory in c++.
- Around 1 lakh allocation and deallocations were performed.
- 4KB was the maximum page size taken that the memory can allocate.
- Number of allocations was produced with higher probability than deallocation because we wanted to check the performance of the illuminator under high load conditions.
- We generated the memory size of the request randomly, but the probability of a huge page was kept high so that actual performance of the illuminator can be measured.

V. Result

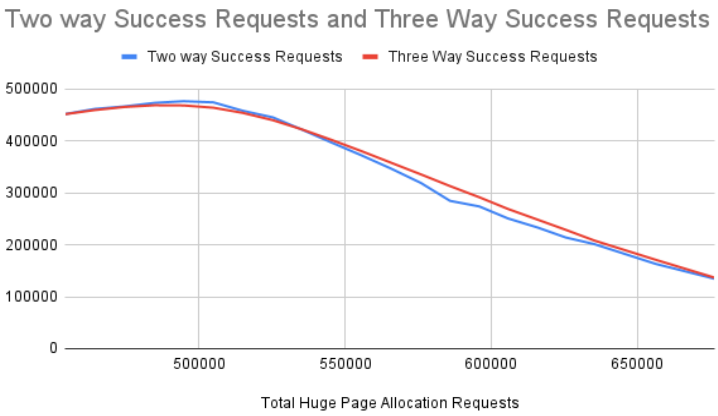

Fig. 7. The graph compares the success huge page request of Three way and Two way when the total huge page request increases

We compared both Two way classification and Three Way classification algorithm under normal and high load condition and got following results:

1) In the long run Three way classification helps in clustering of the unmovable pages. Due to this when the system is loaded with 65% to 75% huge page allocation requests then Illuminator gives better results as the degree of fragmentation is reduced.
2) In Two way classification memory management unit does not have knowledge of the hybrid blocks which lead to unnecessary transfer of blocks. Which lead to unusual CPU spikes. But in three way classification the hybrid blocks are just skipped in the compaction algorithm. This majorly reduces the time for this algorithm as unnecessary work is reduced.
3) In normal load condition Illuminator gave similar results as given by the Two way classification. In normal condition as most of the memory is free, most of the huge page allocation requests are successful. But Illuminator overpowers Two way classification when there is high load condition.

References

[1] Ashish Panwar, Aravinda Prasad, and K. Gopinath. 2018. Making Huge Pages Actually Useful. In ASPLOS '18: 2018 Architectural Sup- port for Programming Languages and Operating Systems, March 24–28, 2018, Williamsburg, VA, USA. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3173162.3173203