**An Approach Towards Improvement of Contiguous Memory Allocation Linux Kernel: A Review**

|  |  |  |
| --- | --- | --- |
| **Article Info** |  | **ABSTRACT** (10 PT) |
| ***Article history:***  Received  Revised  Accepted |  | The demand of contiguous memory allocation has been expanded in day-to-day life in all the devices. It is achieved in existing systems by using various reservation techniques. There are various other methods to achieve the goal of contiguous memory allocation in linux kernel such as, Input Output Memory Management Units (IOMMU’s) & Scatter/Gather Direct Memory Access (DMA), but these are the hardware solutions. However, the configuration of additional hardware's increases the cost & power consumption of the system. It is very difficult to access Contiguous Memory Allocator (CMA) in low end devices which are unable to provide real contiguous memory. This will be the main disadvantage of the existing system. The motivation behind the study is to review existing Contiguous Memory Allocation (CMA) approaches, which utilize the maximum amount of available memory space. |
| ***Keywords:***  Contiguous Memory Allocator  Reservation Techniques  Scatter/Gather Direct Memory access  Input-output Memory Management Units |
| *This is an open access article under the* [*CC BY-SA*](https://creativecommons.org/licenses/by-sa/4.0/) *license.* |
| ***Corresponding Author:***  Anmol Suryavanshi,  Department of Computer Science & Engineering,  Oriental University, Indore  SVKM’s Institute of Technology, Near Mumbai Agra Road, Behind Gurudwara,Dhule, Maharashtra, India 424001  Email: [eranmol89@gmail.com](mailto:eranmol89@gmail.com)  [Anmol.suryavanshi@svkm.ac.in](mailto:Anmol.suryavanshi@svkm.ac.in) | | |

1. **INTRODUCTION**

Nowadays, the requirement for physical contiguous memory allocation is extremely demanded, particularly in low-end 32-bit devices as the currently available solutions are not sufficient. The frequently used method is reservation technique. Even though it serves memory allocation in good manner, yet it can seriously downgrade the memory usage. Although there are some H/w solutions such as Scatter/Gather DMA & IOMMU for this issue. But the cost of this additional H/w is more for the low end 32 bit devices. CMA is a Linux S/w product that targets to resolve memory allocation as well as efficient memory utilization issues. There are different devices on embedded frameworks that have no Scatter-Gather DMA or IO map facility and contiguous memory blocks for allocation [1]. They incorporate devices like cameras, Hardware video decoders-encoders etc. However, such devices regularly require large memory which makes systems inadequate. Some embedded devices force extra necessities on the buffers, for example they can just operate on buffers allocated in specific memory bank (if one or more memory bank available in the system) or buffers allocated to a specific memory limit. We have seen an enormous rise in the development of embedded devices recently (particularly in the V4L region) and there were various such type of drivers incorporate their own memory allocation code. A large portion of them use bootmem-based strategies [2].

CMA structure is an effort to bind together contiguous memory allocation systems and give a straightforward API to device drivers, while remaining as adjustable and modular as could really be expected [3]. While keeping contiguous allocation, it should be possible without genuine punishment with CMA which has emphasis on memory proficiency. It reserves large memory region during boot time & let the region be utilized for contiguous allocation [1]. If the contiguous memory may not get fully utilizes the reserved memory, let the memory should be made available for 2nd-class clients, otherwise that memory could be wasted. The pages allocated to the 2nd-class clients were required for contiguous memory allocation, it get moves or disposes the pages & utilizes them for contiguous memory allocation [2]. We can refer figure 1 for the detailed block diagram structure of CMA.

Diagram, timeline

Description automatically generated.

Figure 1. Block diagram of Contiguous Memory Allocator

In CMA, the movable pages are \_2nd-class client\_. The reserved area can be allocated for movable pages & movable pages can be moved or disposed of if the contiguous allocation requires them [1][2]. As mentioned in figure 1, the CMA works with specific CMA memory which will be useful for multimedia & non multimedia operations. The CMA reserves the memory at boot time. Despite the fact that it is utilized to take care of the memory assignment issue of the driver, the driver doesn't straightforwardly call the interface of CMA module, however by implication utilizes CMA benefits through DMA planning structure. Toward the start, the idea of CMA region was worldwide. Through bit design boundaries and order line boundaries, the portion can find the beginning location and size of worldwide CMA region in memory (Note: Global here implies for all drivers). Also, in the instatement time, call DMA\_ conTIguous\_ Reserve capacity to save the predefined memory locale for worldwide CMA region. Human instinct is avaricious, as are drivers. Before long, a few drivers need to eat alone and are reluctant to impart CMA to different drivers. Consequently, there are two sorts of CMA regions: Global CMA region for everybody to share, while per gadget CMA can be utilized by at least one assigned drivers. Right now, the order line boundaries are not all that proper, so the idea of saved memory hub in gadget tree is presented. Obviously, for similarity, the bit upholds the CMA order line boundary.

In CMA, the movable pages are \_2nd-class client\_. The reserved area can be allocated for movable pages & movable pages can be moved or disposed of if the contiguous allocation requires them [1][2]. As mentioned in figure 1, the CMA works with specific CMA memory which will be useful for multimedia & non multimedia operations. The CMA reserves the memory at boot time. Despite the fact that it is utilized to take care of the memory assignment issue of the driver, the driver doesn't straightforwardly call the interface of CMA module, however by implication utilizes CMA benefits through DMA planning structure. Toward the start, the idea of CMA region was worldwide. Through bit design boundaries and order line boundaries, the portion can find the beginning location and size of worldwide CMA region in memory (Note: Global here implies for all drivers). Also, in the instatement time, call DMA\_ conTIguous\_ Reserve capacity to save the predefined memory locale for worldwide CMA region. Human instinct is avaricious, as are drivers. Before long, a few drivers need to eat alone and are reluctant to impart CMA to different drivers. Consequently, there are two sorts of CMA regions: Global CMA region for everybody to share, while per gadget CMA can be utilized by at least one assigned drivers. Right now, the order line boundaries are not all that proper, so the idea of saved memory hub in gadget tree is presented. Obviously, for similarity, the bit upholds the CMA order line boundary.

Diagram

Description automatically generated

Figure 2. Workflow of Contiguous Memory Allocator

The figure 2 expresses the workflow of CMA. The Contiguous memory allocation (CMA) is required for I/O devices that can just work with contiguous physical memory. On systems with an I/O memory management unit (IOMMU), this would not be an issue on the grounds that IOMMU can map to non-contiguous memory locations to contiguous locations of physical memory. Additionally, a few gadgets can do Scatter/gather DMA Preferably, all I/O gadgets ought to be intended to one or the other work behind an IOMMU or ought to be fit for Scatter/gather DMA. Sadly, this isn't the situation and there are gadgets that require physically contiguous buffers. There are two different ways for a device driver to dispense a contiguous buffer:

The device driver can allocate a chunk of physical memory at boot-time. This is reliable because most of the physical memory would be available at boot-time. However, if the I/O device is not used, then the allocated physical memory is just wasted [5].

A chunk of physical memory can be allocated on demand, but it may be difficult to find a contiguous free range of the required size. The advantage, though, is that memory is only allocated when needed.

CMA solves this exact problem by providing the advantages of both approaches with none of their downsides. The basic idea is to make it possible to migrate allocated physical pages to create enough space for a contiguous buffer. More information on how CMA works can be found here [4].

1. **LITERATURE REVIEW**

Sean J Park et al. [1] (2019) introduced a GCMA that assures low latency, perfect allocation & efficient memory area. The GCMA uses memory reservation technique which increases efficient memory utilization by sharing memory region with immediately discardable data[1][2].

Sean J Park et al. (2018) have introduced alternative approach for CMA and called it as Guaranteed CMA (GCMA). But it degrades the system performance and uses transcendent memory as it does not utilize memory effectively. The authors had studied over existing solutions i.e., the MMU like Hardware Scatter/Gather DMA or IOMMU. In their work, authors stated that hardware solutions are too much costly for low-end devices. As a result, it imposes function overhead and it can’t achieve real contiguous memory. The research also states that the buddy allocators are restricted ones[2].

J. Corbet et al. (2017) have introduced five level page tables which are originated from four level page tables. It does not support on all platforms and increase only linear addresses size. The study states that the results are having emphasis on translation look-a-side buffer (TLB) and the huge pages which require fewer lookups can easily work. The data stored when a virtual memory address maps itself to the physical address in a page table. The virtual addresses are indexed via indexed pages and the page frame numbers are yielded for linked physical page. The array used for this purpose mostly unusual and wasteful. The 32-bit systems are mostly don’t use all the available virtual address[3].

C. Lameter (2017) has introduced method in user space for contiguous physical memory allocation of multiple page frames. It has a limitation that it needed only pre allocated memory for mmap[4].

Y.Kwon et al. (2016) have introduced efficiency in huge page management system and coordinated while improving decency and execution, the Ingens decreases tail-latency and bloat. But it has a drawback that huge pages will reserve the memory for special purpose only. The first class resources are being maintained contiguity, tracks utilization and access memory frequency for huge pages. Ingens improves huge page allocation by eliminating plague in the current systems[5].

J. Stultz [1] (2013) described the Android ION subsystem which is considered for allotting & sharing buffers between H/w gadgets & user spaces to empower zero copy memory sharing amongst the different devices. The approach allows centralize allocation of different "types" of memories or "heaps"[1]. The ION plays a vital role through central buffer allocator & manager who handles cache maintenance for DMA[1]. But it allocates & shares data between devices using existing CMA for user space allocations [6].

J. Jeong et al. [5] (2013) have introduced rental memory allocation by reusing device reserved memory [5]. The proposed approach states that the rental memory returning time cannot be less than simple rental memory management. This prototype shows minimum return time in synthetic’s & android’s based workload [5]. Also it gives analytical improvement in the performance as compare with existing simple management policy[5]. The research gap in the system is the device memory needs to reserve, so that it can be reused/rented for other purpose[7].

A. Basu et al. (2013) have introduced efficiency in Virtual memory for big server with segmentation approach. The authors propose an approach to eliminate the inefficiency of the memory caused due to memory reservation. When device is in an idle state, the reserved memory space of the device can be used for different purposes. This approach needs minimal reallocating associated memory space. It only reduces the TLB misses[8].

D. Magenheimer (2013) has introduced compression-based technique in Linux which compresses and decompress pages into RAM and does not support page cache pages. The kernel compression should have a byte sequences in memory, compresses it and afterwards keeps the compressed version in the RAM until data needed in future again. We can’t read or write any one byte in compressed form. When we need data from compressed one then we can decompress the data to find individual byte to directly access it again[9].

M. Nazarewicz (2012) has introduced CMA in Linux systems. This approach reserves memory by using mem parameter and mem block/bootmem and booting time of system. It assigns buffer space to device when needed. The memory might be wasted in idle state. The approach provides an API for allocating reserved pool. This method migrates pages for allocation [10].

T. Zeng (2012) introduced ION memory manager for android. This approach suggests the specific hardware needs or to combat fragmentation, the ION managing more than one memory pools which are set aside a boot time. Some of the hardware blocks have special memory requirements like GPU, display controller and camera. IONS heaps are the representation of ION memory pool. Android devices are configured with various sets of ION heaps as per the device’s memory requirement. But the underlying Linux kernel implementation is same [11].

J. Jeong [4] (2012) initiated memory reservation based on device. The authors have stated the on-demand reservations method i.e. eCache. The eCache method takes very less time for user latency due to reservation time. It also maximizes the efficiency system memory [4]. It also reduces read I/O operation & increase the launch performance of the application. With this approach, memory efficiency of the system maximizes due to nature of eviction-based memory allocation [4]. But the memory reservation technique was Static memory allocation [12].

M. Ferdman et al. [3] (2012) have made study on data centers and workloads require efficient power and space. In the research, the authors have analyzed micro architectural behavior of the huge type of scale out workload for the performance counter. This approach identifies & analyze the important sources of power inefficiencies in a instruction fetch, core micro architectural memory system organization [3]. This analysis gives efficiently working scale-out workloads which required for optimizing an instruction fetching path for multi megabyte instructions working sets & reducing the core aggressiveness’s & end level cache capacity to clear up an memory area [3]. But in this study, there is no solution for physical contiguous memory allocation failure issue in workloads [13].

D. Magenheimer (2011) has defined the motivation behind transcendent memory (tmem) is gives the Kernel with the capacity to use memory. This research shows two existing tmem frontends, frontswap and cleancache. These two primary kernel memory types are impactful over memory allocation pressure. These two methods are complementary to each other. The cleancache works on cleanly mapped pages which are reclaimed by the kernel. The frontswap works on anonymous pages which are to be swapped out by kernel. When cleancache\_get executes successfully then disk read gets avoided. When frontswap\_put executes successfully, a swap device read-write operation get avoided. The tmem is faster as compared to disk paging/swapping. But this method can't specify, now and again can't follow, and can't directly address[14].

T. Barr et al. (2011) had designed a mechanism for speculative address translation. The author shows that the specific devices can raise latency through nested paging but the nested paging gives some performance penalty. While using huge pages, reduces the TLB misses frequency. But this method specifically applies because the hypervisor needs to control over guests physical address space permission. Speculative address translation allows the results of huge pages. But it requires the Special device named as SpecTLB [15].

Robert Love (2010) has introduced the techniques of Memory Zones, different page types and physical to virtual translation technique. In this paper, the authors discussed various mechanisms of obtaining memory such as, Page allocators and the Slab allocators. To obtain the memory is not easy in the kernel because it depends on the allocation process which relies on certain conditions [16].

J. Corbet et al. (2010) have introduced special class of memory which is not used by kernel directly. The author has introduced “Cleancache” and “Frontswaps” methods which has properties like transcendent memory. The CleanCache method provides a place where it can store pages and removes them if the space requires by another process, if not then it keeps it around. The Frontswaps works as an emergency safety valve when guest requires too much memory to work with. But it is transcendent memory type and memory cannot be used by kernel directly [17].

1. **PROBLEM IDENTIFICATION**

Physical contiguous memory can be significant for different processing conditions such as low-end embedded systems & high-end devices [2]. In any case, existing allocators all have various impediments. For instance, Scatter/Gather DMA & IOMMU like H/w-based arrangements increments the cost & power utilization, buddy system is useless in case of fragmentation. The reservation approaches decrease efficient memory usage & CMA is excessively slowest [2].

Along with the above-mentioned drawbacks of CMA, following are other drawbacks in existing CMA in linux kernel.

**3.1. Allocation failure**

CMA may fail to allocate contiguous memory due to below mentioned facts.

3.1.1 Large Size Allocation

It fails due to fragmentation in Virtual memory.

3.1.2 Direct Pinning

The movable pages may get pinned by any kernel thread for a while. The migration may get fail in case of movable pages which are already pinned by someone else while migrating them [1]. As a result, the pages needs to be unpinned as soon as possible, otherwise the contiguous allocation may get in virtual memory [1].

3.1.3 Indirect Pin

If one movable page is dependant on an object, the object may increase reference counts of a movable pages for getting assurance the is it safe to use the page [1]. The page which hasn’t been free to be used by contiguous allocation when a movable page needs get migrate for a contiguous memory allocation [1].

As a result, contiguous allocation may get fails.

**3.2. High cost**

Contiguous memory allocation of CMA can have high cost due to Function overhead issue. When allocation is success in physical memory it will be mapped with virtual contiguous memory and return to the CPU to use [1].

1. **PROPOSED METHODOLOGY**

The proposed approach resolves the drawbacks exists in the CMA. As mentioned in figure 3, It will be initiated through the design and development of proposed device driver for CMA allocation and redirect all the CMA call through that driver. The driver itself will be implemented in such a way that it will allocate/de-allocate memory from CMA reserved area but will not mapped it to virtual memory at the time of CMA allocation and when virtual memory is required from user/application. It will provide the interface to remap that allocated physical area to virtual area.

Diagram, text

Description automatically generated

Figure 3. Proposed Approach

Reserve CMA memory at boot time will be a same as existing CMA, but while allocating CMA, the memory will utilize the above-mentioned approach, i.e. the proposed new driver method and the driver will allocate CMA memory using DMA\_KERNEL\_NO\_MAPPING attribute. It will not be mapped physical memory to virtual memory at the time of CMA allocation, but when application or the user of CMA required virtual memory, driver will provide the API to remap the physical memory to virtual at run time. Therefore, latency and allocation failure issues may be resolved in existing CMA through implementing 32-bit android device for this approach. The proposed driver can be developed with the help of this Beagle-bone Board.

1. **EXPECTED OUTCOMES**
2. To reduce the latency and allocation failure issue in existing CMA by creating one driver in Linux kernel, which will be used to allocate and deallocate CMA memory.
3. When virtual memory (dereference pointer) is required, then only it will mapped with virtual memory (ultimately it will improve latency), otherwise the physical memory will be directly used by device without mapped to virtual memory.
4. By using DMA\_KERNEL\_NO\_MAPPING attribute in DMA API, I will improve allocation failure issue, reduce workload of system to mapped and unmapped memory and latency issue in CMA.
5. This solution will be useful in many applications like GPU/VPU for 4K Set Top box or Android TV, which requires large contiguous memory for performance and features with limited RAM**.**

**5.1 Beagle bone YouTube 4K Video Latency**

This paper is evaluating the latency probability of CMA & proposed Approach using the 32-bit Beagle bone device with YouTube 4K Video. When a YouTube 4k Video played by 32-bit Beagle bone device generally allocates 128 CMA pages 600 times with kernel threads & keeping the allocated memory as more as it can to utilize it again for forthcoming buffer stream. Hence, only the first buffer stream of video played requires CMA allocation.

For conducting this study, one may configure the system which releases a memory allocation for a 4k Video very quickly against subsequent buffer stream.

Without subsequent tasks, every configuration demonstrates a 4K video latency of about 16 seconds. There is no difference between the latencies of the videos played using the CMA & the Proposed Approach. Because the latency is dominated by the YouTube 4K Video application, not by contiguous memory allocation.

Chart, line chart

Description automatically generated

Figure 4(a). YouTube 4K Video Latency

The expected evaluations are shown in figure 4. The CMA illustrates an extremely slower YouTube 4K Video latency while the Proposed Approach has a much better latency. At its lowest, the CMA needs about 100 secs to play a 4K Video Stream while a Proposed Approach requires 16 secs. The latency of the CMA lies between 4 milliseconds and 10 seconds, while that of the GCMA lies between 900 microseconds and 550 milliseconds. This study suggests that contiguous memory allocation using the CMA gives very high & unpredictable latency in a real time application.

Chart, line chart

Description automatically generated

Figure 4(b). Contiguous Memory Allocation

1. **CONCLUSION**

The proposed approach will remove all the drawbacks of existing CMA like large size memory allocation failure, direct pinning issue, indirect pinning issue in linux kernel. This approach will also improve latency of the CMA system in linux kernel. This approach will remove extra expenses of the hardware’s like Scatter/Gather DMA and IOMMU required to allocate contiguous memory in 32-bit devices. It will also overcome drawback of Buddy System has in fragmentation. Hence improved CMA will be supposed to give better results in those existing systems. This study will be help to resolve contiguous memory related issues in low end 32-bit devices in the future.

**REFERENCES**

1. S. Park, M. Kim and H. Yeom, "GCMA: Guaranteed Contiguous Memory Allocator", IEEE Transactions on Computers, vol. 68, no. 3, pp. 390-401, 2019. Available: 10.1109/tc.2018.2869169.
2. S. Park, Linuxplumbersconf.org. [Online]. Available: https://linuxplumbersconf.org/event/2/contributions/247/attachments/74/85/gcma\_ksummit2018.pdf.
3. J. Corbet, "Five-level page tables [LWN.net]", Lwn.net, 2017. [Online]. Available: https://lwn.net/Articles/717293/.
4. C. Lameter, "User space contiguous memory allocation for DMA", 2017.[Online].Available: https://www.linuxplumbersconf.org/2017/ocw/proposals/4669.
5. Y. Kwon, H. Yu, S. Peter, C. Rossbach and E. Witchel, "Coordinated and efficient huge page management with ingen", in Proc. 12th USENIX Symp. Operating Syst. Des. Implementation, 2016, pp. 705-721.
6. J. Stultz, "Integrating the ION memory allocator [LWN.net]", Lwn.net, 2013. [Online]. Available: http://lwn.net/Articles/565469/.
7. J. Jeong, H. Kim, J. Hwang, J. Lee and S. Maeng, "Rigorous rental memory management for embedded systems", 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2435227.2435239.
8. A. Basu, J. Gandhi, J. Chang, M. Hill and M. Swift, ", "Efficient virtual memory for big memory servers",", in Proc. ACM SIGARCH Comput. Archit. News, 2013, pp. 237-248.
9. D. Magenheimer, "In-kernel memory compression [LWN.net]", Lwn.net, 2013. [Online]. Available: http://lwn.net/Articles/545244/.
10. M. Nazarewicz, "Contiguous memory allocator", in Proc. LinuxCon Eur., 2012.
11. T. Zeng, "The Android ION memory allocator [LWN.net]", Lwn.net, 2012. [Online]. Available: http://lwn.net/Articles/480055/.
12. J. Jeong, H. Kim, J. Hwang, J. Lee and S. Maeng, "DaaC: Device-reserved memory as an eviction-based file cache", in Proc. of Int. Conf. Compilers Archit. Synthesis Embedded Systems, 2012.
13. M. Ferdman, et al, “Clearing the clouds”, in Proc. Int. Conf. Archit. Support Program. Lang. Operating Syst., 2012, Art. no. 37. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2150976.2150982Silberschatz.
14. D. Magenheimer, "Transcendent memory in a nutshell [LWN.net]", Lwn.net, 2011. [Online]. Available: http://lwn.net/Articles/454795/.
15. T. Barr, A. Cox and S. Rixner, "Spectlb: A mechanism for speculative address translation", in Proc. ACM SIGARCH Comput. Archit. News, 2011, pp. 307-318.
16. R. Love, Linux kernel development. Upper Saddle River, NJ: Addison-Wesley, 2010.
17. J. Corbet, "Cleancache and Frontswap [LWN.net]", Lwn.net, 2010. [Online]. Available: http://lwn.net/Articles/386090/.
18. A. Jaleel, "Memory characterization of workloads using instrumentation-driven simulation", 2007. [Online]. Available: http://www.jaleels.org/ajaleel/publications/SPECanalysis.pdf
19. J. Corbet, A. Rubini and G. Kroah-Hartman, Linux Device Drivers. Sebastopol: O'Reilly Media, Inc., 2007.
20. P. Galvin, G. Gagne and A. Silberschatz, Operating system concepts. Hoboken, N.J.: Wiley, 2010.
21. M. Freedman, "The compression cache: Virtual memory compression for handheld computers", Citeseerx.ist.psu.edu, 2000.[Online]. Available:http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.8782.

**BIOGRAPHIES OF AUTHORS (10 PT)**

|  |  |
| --- | --- |
|  | Anmol Suryavanshi was born in India 1st July 1989. I had received my bachelor’s degree in Information Technology Engineering from KBC North Maharashtra University (NMUJ), Jalgaon, India in 2011. Later I had received his master’s degree from Rajiv Gandhi Technological University, (RGPV), Bhopal, India in 2014. I am currently persuing Ph.D. from Oriental University, Indore (OUI), India since February 2020. |
|  |  |
|  | Dr.(Lt.) Sanjeevkumar Sharma is working as a Pprofessor at Oriental University, Indore (OUI), India. He has received his bachelor’s degree from Bundelkhand University, Jhansi, M.P., India in 1998. He later received his master’s degree from Rajiv Gandhi Technological University, (RGPV), Bhopal, India in 2014Madhav Institute of Technology & Science, Gwalior, india in 2004 & He later reveived PhD from Devi Ahilya Vishwa Vidhyalaya, Indore in the year 2014. |