International Journal of Computer Systems (ISSN: 2394-1065), Volume 02– Issue 12, December, 2015
Available at http://www.ijcsonline.com/

Performance Analysis of Compressed Caching Technique

Aman Kumar[A], Shubham Girdhar[B]

[A,B]*Computer Science and Engineering, MAIT, GGSIP University,New Delhi, India*

Abstract

Compressed Caching (virtual memory compression) is the technique that attempts to reduce the request for paging from secondary storage. There is a huge performance gap in accessing primary memory (RAM) and secondary storage (Disk). Compressed caching technique intercepts the pages to be swapped out, compresses them and stores them in pool allocated in RAM. Hence it tries to fill the performance gap by adding a new level to virtual memory hierarchy. This paper presents the performance analysis of the virtual memory compression on various parameters such as workload, size of RAM. The results are displayed in the form of efficiency graphs to show the increase in performance of physical memory using compressed caching technique over normal operation.

Keywords: *Virtual memory, Zswap, zbud, frontswap, LZO, PSO, SO, pool limit hit..*

I. INTRODUCTION

In virtual memory systems, swapping turns out to be the biggest setback in performance. Disk latency is around four times to that of accessing the RAM. In addition to this throughput also decreases when disk is accessed to fetch the pages. In addition to this, disks are becoming more shared and virtualized, due to which I/O latency factor also comes into play [3].

In compressed caching system, during a page request (page fault), evicted pages are compressed and stored in RAM only. The addresses of virtual memory whose contents are compressed during page faults are redirected to the physical memory pool where compressed pages are stored such that future page requests for compressed pages can be completed from the physical memory itself after decompressing them[4]. The size of the page evicted is reduced by the compression process and hence it requires less space in allocated memory pool and remaining space is returned to the available physical memory pool. In this way, storing a large number of pages in compressed form affects the available memory in small extent only, but it reduces the accesses to slow hard disks during page faults to a great extent. This technique takes advantage of the continuously increasing gap of accessing of main memory and that of hard disks by the fast CPU. This accessing gap accounts for the underutilization of CPU when it requires to access more memory. Reducing the swapping (I/O) activity increases the overall system performance [11].

Although the reduction of accesses due to the compression tends to improve system performance, the reduction of non-compressed memory (main memory not allocated for the compressed cache) tends to worsen it. This inherent tradeoff leads us to the question of how much memory should be used by compressed cache, whose appropriate size to achieve the best performance is dependent on the workload. A compressed cache that adapts its size during the system execution in order to reach a good compromise is called adaptive and one with fixed size is called static. Besides this various factors should also be taken into account towards the use of virtual memory compression such as thrashing, compression latency, low compression ratio which tends to reduce the efficiency of compressed caching technique [12].

One such compressed caching module is zswap that comes with linux kernel 3.5 and above. Zswap provides a write-back cache for the pages to be evicted.

A. *ZSWAP*

Zswap is a linux kernel module that implements compressed caching technique. It intercepts the pages that are to be swapped out and attempts to compress them and store in a pool allocated in RAM. If this process is successful, then the writing of the page back to swap device is either deferred for the later time or in some cases it is not required. This results in I/O reduction to a large extent and increase in performance for virtual memory systems. Actually, zswap tries to steal the CPU cycles that are wasted in swapping pages in/out of memory and performance is improved if it takes less time to access compressed pages from RAM than to access the swap device.

Zswap works on adaptive caching technique, as the compression pool is not preallocated. Rather it increases (upto maximum limit) or decreases on demand. When the compressed pool allocated in RAM reaches its maximum limit, the pages are evicted to write them to the swap device. The selection of pages for eviction is done on the Least Recently Used (LRU) basis. Moreover, the task of managing the compressed pool is assigned to zbud module. zbud is a special-purpose memory allocator used internally by zswap for storing compressed pages. Zbud stores two compressed pages (buddies) instead of one in a frame (space allocated for one physical memory page) in

compression pool. It has got a disadvantage too that it denies the possibility of more compression.

Frontswap is the API with the help of which zswap is integrated into the linux kernel virtual memory hierarchy. Frontswap is a feature of linux kernel that announces the possibility of a device to be used as swap device. Frontswap helps zswap in intercepting the pages to be swapped out for compression and also the page requests for already compressed pages. In this way Frontswap acts as the front end while the zswap works in background.

II. WORKINGARCHITECTURE OF COMPRESSED CACHING TECHNIQUE

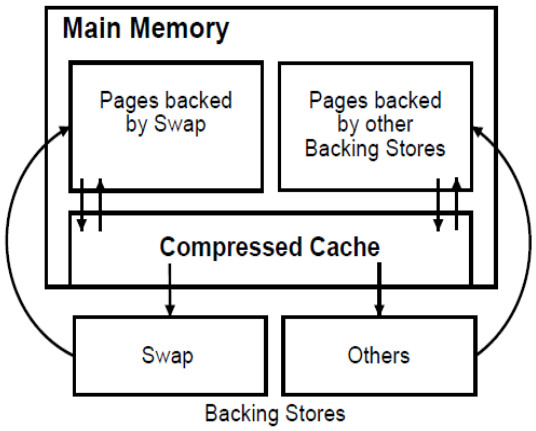

Figure 1 Architecture of 3-level virtual memory system

In adaptive compressed caching technique, almost all the pages backed by a backing store are eligible to be compressed and stored in the compressed cache. Also, as soon as there is no available space in the compressed cache to insert a new page, either the compressed cache allocates more memory for its usage or some compressed pages are freed. When the latter action is taken, the oldest compressed page is released. However, before being released, compressed dirty pages must first be written to the backing store. Only pages backed by swap are written in compressed form, and each compressed page is null-padded to complete the size of a block. Pages that are read in advance from swap are only decompressed if any process faults in them, i.e., they are mapped back by a process page table [8].

Pages requested back by any kernel operation in order to be immediately used are said to be reclaimed. This includes a page reclaimed by a page fault and a page holding block data cached in memory. Reclaimed compressed pages are removed from the compressed cache, decompressed, and their data are placed in newly allocated memory pages. When a reclaimed page is not present in the compressed cache, it is read from the backing store, and decompressed if the backing store is the swap. The page is not added to the compressed cache when read from the backing store [8].

III. TESTING PROCESS

The system used for testing was intel i5 (4th gen) processor with linux kernel 3.10. "stress" is used as a testing tool. Stress is a simple workload generator for POSIX systems. It imposes a configurable amount of CPU, memory, I/O, and disk stress on the system. It is not a benchmark, but is rather a tool designed. The benefit of using stress is that it dispatches working processes and workload is configurable.

Zswap is enabled by passing the the kernel parameter zswap.enabled=1 at boot time. There are two parameters that are configurable for zswap: zswap.compressor and max_pool_percent. Zswap.compressor specifies the compression module (algorithm) used for compression of pages. If it is not specified then zswap uses the default LZO (Lempel-Ziv Oberhumer) algorithm. Second parameter, max_pool_percent tells the maximum size (percentage) of physical memory to be used for allocating pool for compressed pages. The amount of compression pool in preallocated, rather it grows or shrinks on demand and can reach upto the maximum value at the peak loads.

During the testing procedure LZO (default) algorithm was used for compression and maximum pool percent was set to 20. We used "vmstat" as a virtual memory monitoring tool. The outcome of vmstat comprises of swap memory used, swap in/out per second, number of pages swapped in/out etc. In addition to this various parameters of zswap are also recorded such as pool limit hit, stored pages, rejected pages due to poor compression.

Testing was performed on various sizes of RAMs ranging from 256MB to 6GB against various workloads once with zswap enabled and other when it is disabled. The corresponding performances are then compared to plot efficiency graphs of physical memory performance with compressed caching over normal operation. The performance is compared based on two primary parameters pages swapped out (pso) and memory swapped out per second (so).

IV. RESULTS

The performance of compression module is presented for various sizes of physical memory ranging from 256 MB to 6GB. Second and third column of Table I shows the percentage efficiency of physical memory while using the module over its normal operation for parameters pages swapped out(PSO) and memory swapped out (SO) per second respectively. over various sizes of RAM. In addition to these last column shows the performance in terms of workload needed to hit compressed pool limit (for the first time) for various sizes RAM.

TABLE I. EFFICIENCY OF ZSWAP MODULE FORVARIOUS SIZES OF RAM

RAM (GB)	**Efficiency for Page Swapped Out (%)**	**Efficiency for Memory swapped per second (%)**	**Minimum Load for Pool Limit Hit (GB)**
0.25	-3	-11.7	100 (MB)
0.5	13.3	18.3	200 (MB)
1	24.61	26.2	1
2	31.5	30.25	1.5
3	34.26	33.7	2.5
4	38.3	38.6	3.5
5	56.3	53.3	5
6	68.4	70.5	7

Figure 2, 3, 4 shows the performance increase in graphical form for all the three parameters

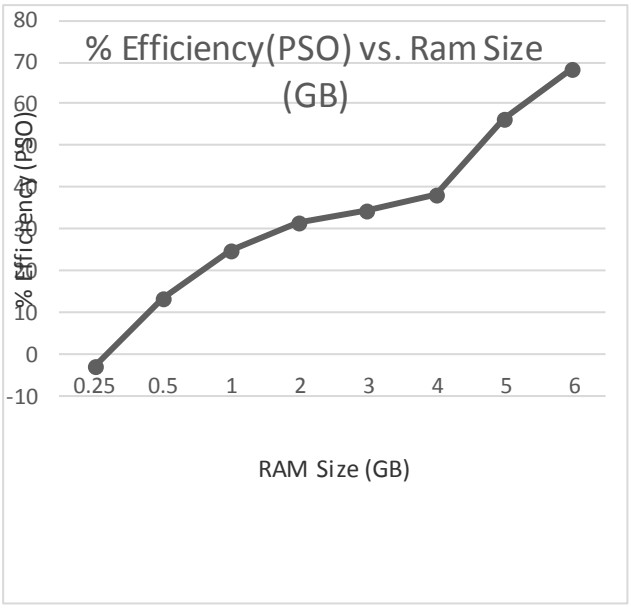

Figure 2 Graphical plot for efficiency in terms of PSO vs. RAM size

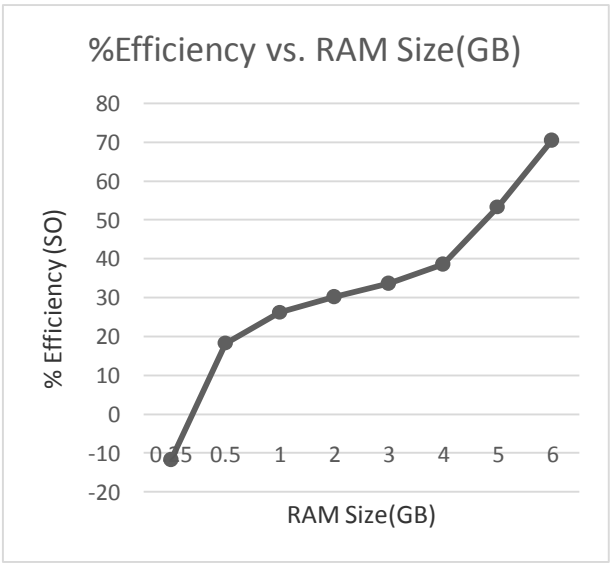

Figure 3 Graphical plot for efficiency in terms of SO vs RAM size

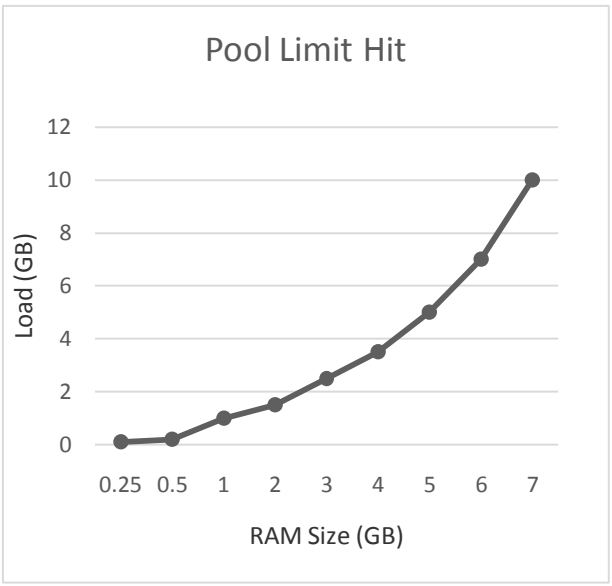

Figure 4 Graphical plot for comparison of efficiency at different RAM sizes based on minimum load required for pool limit hit

V. CONCLUSIONS

After analyzing the results it is concluded that the compression module increases efficiency of physical memory appreciably as number of pages swapped out decreases. Hence the number of disk accesses also decreases and disk latency also decreases. On analyzing pool limit hit data we infer that zswap is working most efficiently over 1GB and can handle appreciable workload. The low performance of the compression module at low RAM sizes is due to thrashing. Since the compression module itself reserves 20 percent of the RAM space for storing compressed pages, therefore at low memory conditions thrashing occurs. Hence the low performance (negative efficiency) at small sizes of RAM.

Hence the compressed caching module proved to be beneficial for Desktop/Laptop users whose system's physical memory lie in the efficiency range. This technique could provide a great improvement in performance even on high loads. Secondly, the users having fast SSDs as secondary storage may improve the life of their disks by reducing the number of write cycles.

ACKNOWLEDGMENT

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our highly respected and esteemed guide Dr.Namita Gupta, Head Of Department, Computer Science and Engineering Department for her valuable guidance, encouragement and help for completing this work. Her useful suggestions for this whole work and co-operative behaviour are sincerely acknowledged.

REFERENCES

[1] Paul R. Wilson, Scott F. Laplan, and YannisSmaragdakis, "The Case for Compressed Caching in Virtual Memory Systems", *Proceedings of USENIX of Annual Technical Conference Monterey*, California, USA: June 6-11, 1999.

[2] Andrew W. Appel and Kai Li, "Virtual memory primitives for user programs", *FourthInternational Conference on ArchitecturalSupport for Programming Languages andOperating Systems (ASPLOS IV)*, Santa Clara, California: April 1991, pp. 96-107.

[3] Fred Douglis, "The compression cache: Using on-line compression to extend physical memory", Proceedings of 1993 Winter USENIX Conference, San Diego, California: January 1993, pp. 519–529.

[4] M. Kjelso, M. Gooch, and S. Jones, "Performance evaluation of computer architectures with main memory data compression", Journal of Systems Architecture 45: 1999, pp. 571-590.

[5] R. Carvera, T. Cortes, and Y Becerra, "Improving Application Performance through Swap Compression", In Usenix '99 – Freenix Refereed Track: 1999.

[6] Ross N Williams, "An extremely fast Ziv-Lempel compression algorithm", Data Compression Conference: April 1991, pp. 362–371.

[7] Walter R.Smith and Robert V. Welland, "A model for address-oriented software and hardware", In 25th Hawaii International Conference on Systems Sciences: January 1991.

[8] Alaa R. Alameldeen and David A. Wood, "Adaptive Cache Compression for High-Performance Processors", proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA-31) Munich, Germany: June 19-23, 2004.

[9] Luca Benini, DavideBruni, Bruno Ricco, Alberto Macii, and Enrico Macii, "An Adaptive Data Compression Scheme for Memory Traffic Minimization in Processor-Based Systems", In Proceedings of the IEEE International Conference on Circuits and Systems, ICCAS-02: May 2002, pp. 866–869.

[10] Toni Cortes, Yolanda Becerra, and Raul Cervera, "Swap Compression: Resurrecting Old Ideas", Software – Practice and Experience Journal, 46(15): December 2000, pp. 567–587.

[11] Michael J. Freedman, "The Compression Cache: Virtual Memory Compression for Handheld Computers", Technical report, Parallel and Distributed Operating Systems Group, MIT Lab for Computer Science, Cambridge: 2000.

[12] S. F., Kaplan, "Compressed Caching and Modern Virtual Memory Simulation", PhD thesis, University of Texas at Austin: 1999.