iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 9 **Issue:** XI **Month of publication:** November 2021

DOI: https://doi.org/10.22214/ijraset.2021.38770

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue XI Nov 2021- Available at www.ijraset.com

Refresh Rate Identification Strategy for Optimal Page Replacement Algorithms for Virtual Memory Management

Gajanan Digambar Gaikwad[1], Gajendra R. Bamnote[2]
[1, 2]*PG Department of Computer science and Engineering, Prof Ram Meghe Institute of Technology & Research, Badnera, Amravati, Maharashtra, India*

Abstract: ***Operating system offers a service known as memory management which manages and guides primary memory. It moves processes between disk and main memory during the execution back and forth. The process in which we provisionally moves process from primary memory to the hard disk so the memory is available for other processes. This process is known as swapping. Page replacement techniques are the methods by which the operating system concludes which memory pages to be swapped out and write to disk, whenever a page of main memory is required to be allocated. There are different policies regarding how to select a page to be swapped out when a page fault occurs to create space for new page. These Policies are called page replacement algorithms. In this paper the strategy for identifying the refresh rate for 'Aging' page replacement algorithm is presented and evaluated.***

Keywords: Aging algorithm, page replacement algorithm, refresh rate, virtual memory management.

I. INTRODUCTION

Operating system offers a service known as memory management which manages and guides primary memory. It moves processes between disk and main memory during the execution back and forth [1]. The process in which we provisionally moves process from primary memory to the hard disk so the memory is available for other processes. This process is known as swapping. A computer can find extra memory than the amount of manually equipped on the system. This extraneous memory is literally called virtual memory and it is indeed a section of hard disk that is set up to imitate the computer's RAM Virtual memory is generally attained with the demand paging.

A memory management method paging is commonly used in which the memory is parted into fixed size pages [2]. Paging is used for accessing data rapidly. Whenever a program requires a page, it could be found in primary memory as if the operating system duplicates a certain number of pages on the main memory from the hard disk. A page table is a data structure, which is used by the virtual memory system in a computer's operating system to find the mapping within the virtual addresses and physical addresses. The accessing process uses virtual addresses, while physical addresses used up by the hardware and most categorically, by the RAM subsystem [3]. Whenever a program attempt to reference a page that is not available in RAM, then the processor take it as an invalid memory reference, or as a page fault and then it relocate the control from the program to the operating system [4].

Page replacement techniques are the methods by which the operating system concludes which memory pages to be swapped out and write to disk, whenever a page of main memory is required to be allocated. A page replacement algorithm hits on a less knowledge about obtaining the pages given by the hardware, and then it tries to elect which pages must be replaced to minimize the total number of page misses, during adjusting it with the cost of primary memory and processor time of the algorithm [5].

The main memory capacity of laptops and mobile devices has been rapidly increasing due to the ever-increasing degree of multitasking and improving quality of multimedia data. For example, the random access memory (RAM) size of the reference was 4 GB in 2018, while it was only 512 MB in 2010, an eight-fold increase in 8 years. The demand for a larger memory capacity is still strong. However, increasing the memory capacity is a challenging issue because it results in higher manufacturing costs and poor energy efficiency [6], [7]. Conventional computing systems, such as personal computers and servers, have achieved larger main memory space than RAM capacity by swapping out infrequently accessed pages to secondary storage devices. However, because the access speed of swap storage devices is usually lower than that of the RAM by up to 100 times, the swap-to-secondary storage scheme is undesirable for consumer electronics, which requires an immediate response to user inputs. In addition, the structure of flash memory, which is being popularly used for storage device in mobile consumer electronics for their high speed, robustness, and small form-factors, is transitioning from single- (SLC) to multi-level cell (MLC), including a triple- and quadruple-level cell (TLC and QLC, respectively) technology [8]. Because the write endurance cycles of TLCs and QLCs are 10-100 times smaller than those of SLCs [9], the frequent small random writes caused from page swap-out adversely impact the life span of mobile devices.

In the last two decades, various in-memory compressed swap schemes have been proposed to accommodate more data than the RAM size and improve the system performance by reducing page swapping I/O operations [10]-[15].

As shown in Figure 1.1, such compressed swap schemes compress cold pages, which are not expected to be accessed in the meantime, and store the compressed pages, called zpages, in a swap page in the compressed swap pool in the RAM. When a zpage stored in the pool is requested from the main memory, it is decompressed and moved back to the main memory.

II. AGING PAGE REPLACEMENT ALGORITHM

The aging algorithm is a descendant of the NFU algorithm, with modifications to make it aware of the time span of use. Instead of just incrementing the counters of pages referenced, putting equal emphasis on page references regardless of the time, the reference counter on a page is first shifted right (divided by 2), before adding the referenced bit to the left of that binary number. For instance, if a page has referenced bits 1,0,0,1,1,0 in the past 6 clock ticks, its referenced counter will look like this: 10000000, 01000000, 00100000, 10010000, 11001000, 01100100. Page references closer to the present time have more impact than page references long ago. This ensures that pages referenced more recently, though less frequently referenced, will have higher priority over pages more frequently referenced in the past. Thus, when a page needs to be swapped out, the page with the lowest counter will be chosen.

Note that aging differs from LRU in the sense that aging can only keep track of the references in the latest 16/32 (depending on the bit size of the processor's integers) time intervals. Consequently, two pages may have referenced counters of 00000000, even though one page was referenced 9 intervals ago and the other 1000 intervals ago. Generally speaking, knowing the usage within the past 16 intervals is sufficient for making a good decision as to which page to swap out. Thus, aging can offer near-optimal performance for a moderate price.

```
t | R-bits (0-5)       | Counters for pages 0-5
00 | [1, 0, 1, 0, 1, 1] | [10000000, 00000000, 10000000, 00000000, 10000000, 10000000]
01 | [1, 1, 0, 0, 1, 0] | [11000000, 10000000, 01000000, 00000000, 11000000, 01000000]
02 | [1, 1, 0, 1, 0, 1] | [11100000, 11000000, 00100000, 10000000, 01100000, 10100000]
03 | [1, 0, 0, 0, 1, 0] | [11110000, 01100000, 00010000, 01000000, 10110000, 01010000]
04 | [0, 1, 1, 0, 0, 0] | [01111000, 10110000, 10001000, 00100000, 01011000, 00101000]
```

III.REFRESH RATE IDENTIFICATION STRATEGY FOR AGING ALGORITHM

The Aging Algorithm is likely the most complex. Aging works by keeping an 8 bit counter and marking whether each page in the page table was used during the last ‘tick’ a time period of evaluation which must be passed in by the user as a ‘refresh rate’, whenever the Aging Algorithm is selected. All refresh rates are in milliseconds on my system, but this relies on the implementation of Python’s “time” module, so it’s possible that this could vary on other systems. For aging, a refresh rate of 0.01 milliseconds is suggested, passed in on the command line as “-r 0.01”, in the 2nd to last position in the arg list. This minimizes the values in testing, and going lower does not positively affect anything.

In order to find a refresh rate that would work well, it is decided to start at 1ms and move 5 orders of magnitude in each direction, from 0.00001ms to 100000ms. To ensure that the results were not biased toward being optimized for a single trace, tried both to confirm that the refresh rate would work will for all inputs. The graphs in figure 1 to figure 4 below show the Page Faults and Disk Writes found during each test.

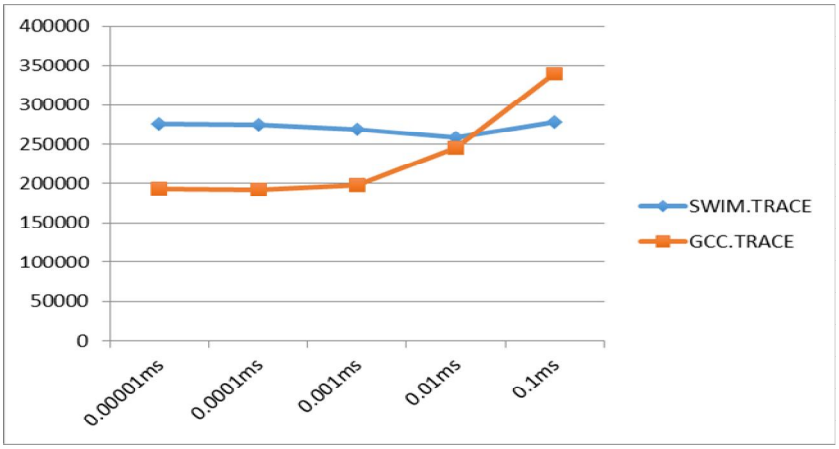


Fig. 1 Aging Algorithm -- Page Faults / Refresh Rate

GCC.TRACE reaches its minimum for page faults at 0.0001ms and SWIM.TRACE reaches its own minimum at 0.01ms. The lines cross at around 0.01ms.

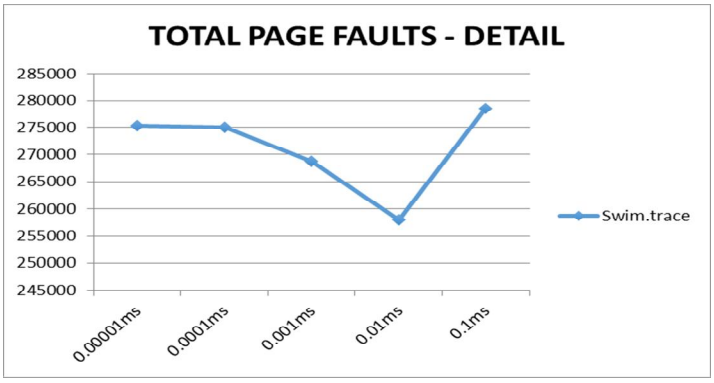


Fig. 2 Aging Algorithm SWIM.TRACE - Total Page Faults

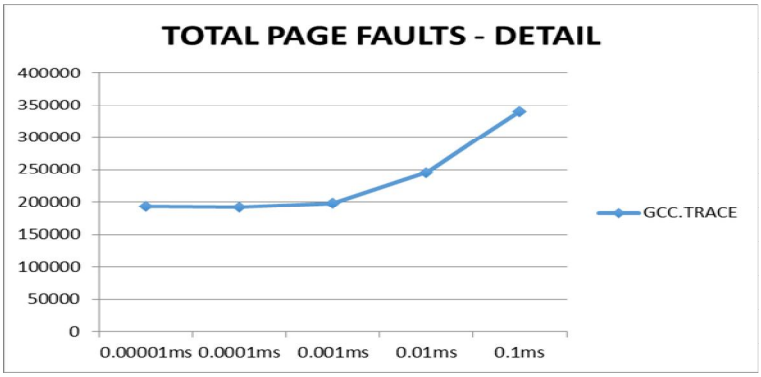


Fig. 3 Aging Algorithm GCC.TRACE - Total Page Faults

Additionally, 0.01ms seems to achieve the best balance, if we need to select a single time for BOTH algorithms.

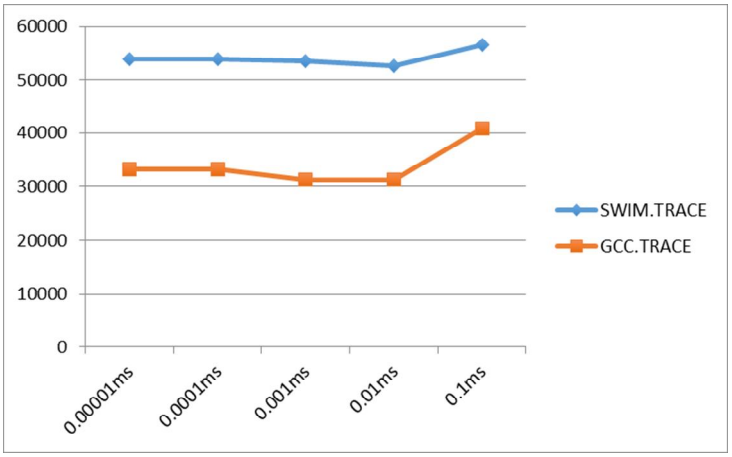


Fig. 4 Aging Algorithm - Disk Writes/ Refresh Rates

The number of disk writes also bottoms out at 0.01ms from SWIM.TRACE. It is relatively constant for GCC.TRACE across all of the different timing options. Because of this, 0.01ms is suggested as the ideal refresh rate. This is because it is optimal for SWIM.TRACE. For GCC.TRACE, it is not the absolute best option, but it is still acceptable, and so this selection will achieve a good balance.

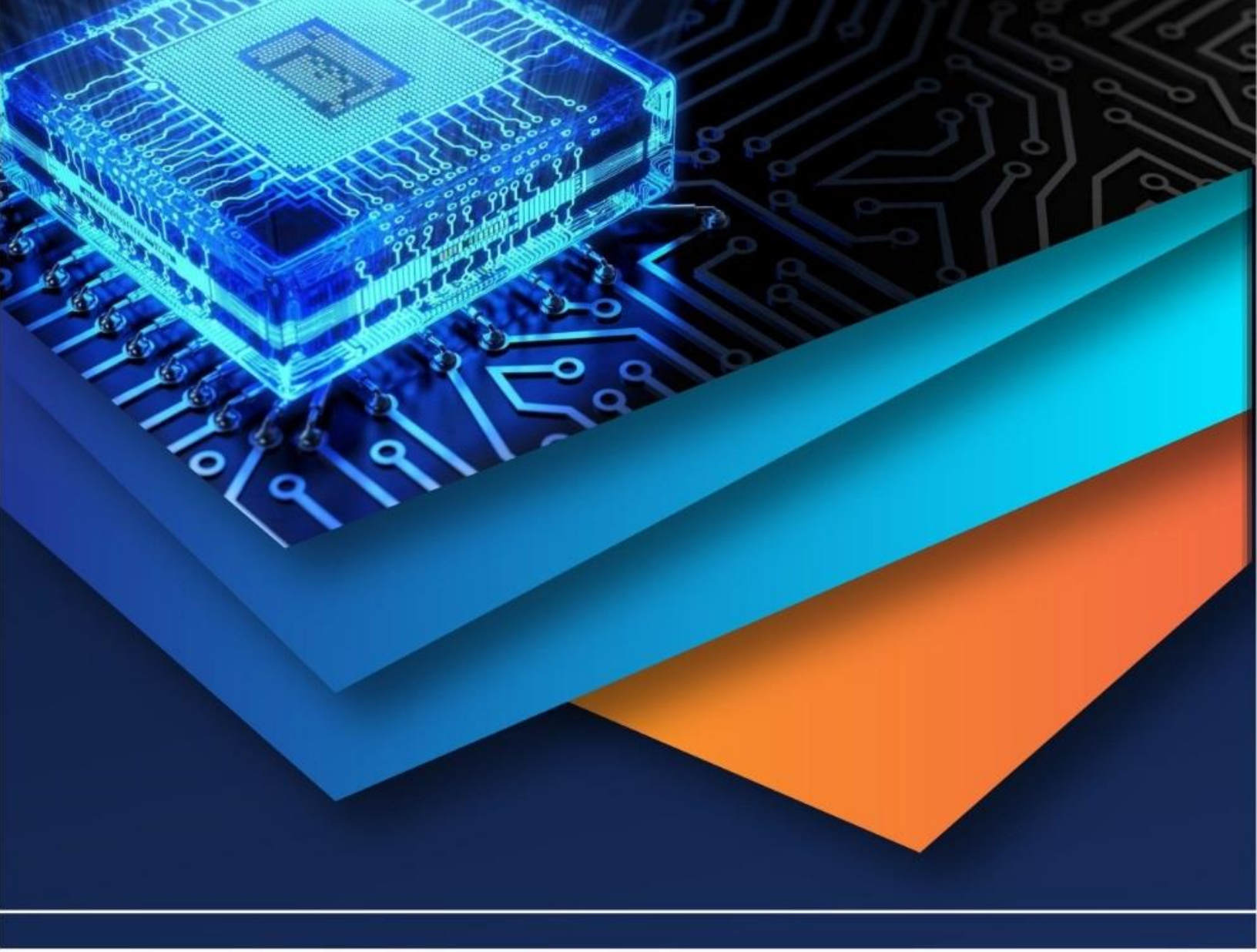
For all tests, a frame size of 8 is chosen, since this small frame size is most sensitive to the algorithm used. At higher frame sizes, all of the algorithms tend to perform better, across the board. So it is wanted to focus on testing at the smallest possible size, preparing for a ‘worst case’ scenario.

IV. CONCLUSIONS

In this paper a strategy for identification of refresh rate for Aging page replacement algorithm is presented. In order to find a refresh rate that would work well, it is decided to start at 1ms and move 5 orders of magnitude in each direction, from 0.00001ms to 100000ms. To ensure that the results were not biased toward being optimized for a single trace, tried both to confirm that the refresh rate would work will for all inputs. GCC.TRACE reaches its minimum for page faults at 0.0001ms and SWIM.TRACE reaches its own minimum at 0.01ms. The lines cross at around 0.01ms. Additionally, 0.01ms seems to achieve the best balance, if we need to select a single time for BOTH algorithms.

REFERENCES

[1] Sanjay kumar Panda and Saurav Kumar Bhai,"An effective Round Robin Algorithm using Min-Max Dispersion Measure",International Journal on Computer Science and Engineering, 4(1), pp 45-53, January 2012.
[2] Sorav Bansal and Dharmendra S Modha CAR: Clock with Adaptive Replacement FAST' 04-3rd USENIX conference on File and Storage Technologies, 2004
[3] S Jiang and X Zhang,"LIRS: An efficient policy to improve Buffer Cache Performance", IEEE transactions on Computers, pp 939-952,2005.
[4] Song Jiang and Xiaodong Zhang,Token Ordered LRU: An effective page replacement policy and its implementation in Linux systems,performance Evaluation 60 5-29,2005.
[5] Abraham Silberschautz Peter B Galvin and Greg Gagne,Operating system concepts(UK:Eiley,2010)
[6] R. Duan, M. Bi, and C. Gniady, "Exploring memory energy optimizations in smartphones," in Proc. Int. Green Comput. Conf. Workshops, Jul. 2011, pp. 1-8.
[7] K. Zhong, D. Liu, L. Liang, X. Zhu, L. Long, Y. Wang, and E. H.-M. Sha, "Energy-efficient in-memory paging for smartphones," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 35, no. 10, pp. 1577-1590, Oct. 2016.
[8] T. Coughlin. Getting solid at FMS. Forbes. Accessed: Aug. 16 2019. [Online]. Available: https://www.forbes.com/sites/tomcoughlin/2019/08/16/getting-solid-at-fms/
[9] T. Yang, H. Wu, and W. Sun, "GD-FTL: Improving the performance and lifetime of TLC SSD by downgrading worn-out blocks," in Proc. IEEE 37th Int. Perform. Comput. Commun. Conf. (IPCCC), Nov. 2018, pp. 1-8.
[10] P. R.Wilson, S. F. Kaplan, and Y. Smaragdakis, "The case for compressed caching in virtual memory systems," in Proc. USENIX Annu. Tech. Conf., 1999, pp. 101-116.
[11] F. Douglis, "The compression cache: Using on-line compression to extend physical memory," in Proc. USENIX Winter Tech. Conf., 1993, pp. 519-529.
[12] R. Cervera, T. Cortes, andY. Becerra, "Improving application performance through swap compression," in Proc. USENIX Annu. Tech. Conf., 1999, pp. 207-218.
[13] R. S. de Castro, A. P. D. Lago, and D. da Silva, "Adaptive compressed caching: Design and implementation," in Proc. 15th Symp. Comput. Archit. High Perform. Comput., Nov. 2003, pp. 10-18.
[14] L. Yang, R. P. Dick, H. Lekatsas, and S. Chakradhar, "Online memory compression for embedded systems," ACM Trans. Embedded Comput. Syst., vol. 9, no. 3, Feb. 2010, Art. no. 27.
[15] I. C. Tuduce and T. Gross, "Adaptive main memory compression," in Proc. USENIX Annu. Tech. Conf., 2005, pp. 237-250.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY