International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2014): 5.611

Proposed D-Range Page Replacement Algorithm

Devansh Dave

K. J. Somaiya College of Engineering Mumbai, India

Abstract*: In real world scenario, workload adaptation and easy implementation is of prime importance. The main idea behind this paper is to introduce a new page replacement algorithm which will take into consideration locality of reference periodically and save the time to search through all pages for reference bits. This gives more performance from traditional LRU after initial cycles and efficient implementation due to use of B tree. The key concept of this algorithm is to maintain range pages according to total no of references periodically which will remain in memory for a cycle. This greatly increases the hit ratio of pages.*

Keywords: Hit ratio; LRU algorithm; FIFO; OPTIMAL; B tree; D-range page replacement algorithm

1. Introduction

In an operating system, virtual memory management is implemented by the help of page replacement algorithms which decide the memory pages to be swapped in or out. We mostly use a two –level memory hierarchy, a faster main memory and a cheaper secondary memory [5]. Pages are brought into main memory only when the executing process demands it, this is known as demand paging. Whenever a page fault occurs the required page is swapped in from secondary memory. Page replacement algorithms help in faster execution of process by increasing the hit ratio and with easy implementation decreases performance overhead.

To manage paging, a page replacement algorithm is needed [4]. It evicts the pages when page table becomes full in such a way that hit ratio is considerably nice. The best-case situation is the one in which the page, that is not going to get used for longest time, be replaced. Since it is practically difficult to implement such a thing, it should keep track of past references or locality of reference.

Practically, in a real world scenario, the page referenced is somewhere around selected pages for certain amount of time. This paper outlines two major ideas- a new algorithm concerning range and to implement it a data structure not used until now and compatible with it, B tree.

2. Previous Work

When some page is selected for replacement and it is again referenced it has to be rewritten into memory, then this results in waiting for I/O completion. A page replacement algorithm simply guesses the next page. Some of them are as follows:

2.1 First In First Out (FIFO):

The FIFO algorithm is the oldest replacement algorithm. The main idea is to replace the longest resident page in main memory. FIFO does not have any past history of page references and takes into consideration of time length rather than usage. This requires queue implementation in which pages are ordered by arrival time. It suffers from Belady's analomy.

2.2 Least Recently Used (LRU)

LRU replaces page which is unused for longest time period [2]. In performance point of view, its results are better than FIFO [1]. The main reason is because it takes into consideration program behavior by storing the usage history of page references with the help of counter. It all lies on the assumption that recent past affects near future. Even though its implementation is possible, it generates a large overhead to system.

2.3 Optimal

This has the best performance in terms of hit ratio, amongst all page replacement algorithms [5]. But its implementation is not realistic. It basically replaces the page which will not be used for longest period of time. This is difficult to implement because in real world scenario there is no knowledge of future strings. Therefore it remains as a standard on which other algorithms are compared.

2.4 B Tree

B-tree is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time [6]. A B-tree is kept balanced by requiring that all leaf nodes be at the same depth. In B-trees, internal (non-leaf) nodes can have a variable number of child nodes within some pre-defined range

B-trees have substantial advantages over alternative implementations when the time to access the data of a node greatly exceeds the time spent processing that data, because then the cost of accessing the node may be amortized over multiple operations within the node. This usually occurs when the node data are in secondary storage such as disk drives [6].

3. Proposed Algorithm (D-RANGE)

The proposed algorithm is completely new as it takes into consideration a new concept of range(R) pages. These are the pages which are referenced most frequently. These pages are never swapped in or out. They remain stationary through a cycle. This algorithm is closely related to locality of reference as we maintain a counter on each page to count total no of references (TNR).

The main idea behind this algorithm is that the pages referenced are mostly the ones closer to the range (R) pages. The main advantage of D-range is that for a particular page to be swapped out there is no need to search all the frames, only the range has to be determined and the pages most referenced always remain in the memory through a cycle.

The Hit ratio considerably increases after primary range cycle. The primary range cycle is the cycle where the range adjusts to the locality of reference.

A range cycle (probably 10 requests) is one after which we change our range pages(R) according to the TNR. This helps in adjusting to the changing locality of reference. Using following steps proposed algorithm can be easily understood-:

STEP 1: Take no of pages (P_T) and no of frames (F) and cycle (C) as input. Eg. P_T=10, F=5, C=10.

STEP 2: Step value (S) = P_T/F (if decimal value then round off) for range pages(R). For n –frames (F) there will be (n-1)/2 range pages (R). Eg. S=10/5=2 range pages.

STEP 3: Start the range cycle.

STEP 4: Swap in and out the pages on the basis of range within it lies. R pages are to be not swapped out through the range cycle.

STEP 5: Maintain a counter on each swapped in page(C) for each range cycle for total no of references (TNR).

STEP 6: After a range cycle, change and assign the R pages according to TNR. The pages most referenced are selected as range pages(R). To check for selection conflicts over range pages, go to Step 7.

STEP 7: If R page is the highest or lowest then decrement and increment by 1 respectively.
If two pages have same TNR, then the one closest to mid value is selected.
If the two pages with the highest TNR are adjacent to each other then increment the lowest of them or the one closest to mid value.

STEP 8: Repeat Step 3 to Step 7 until all pages are traversed.
Example:
Note: In order to represent range pages(R), we use
-- Sign
Reference Strings (total no of pages=10, frames =5)

Table 1: Frame for cycle 1

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1

Table 2: Frame for cycle 2

7	7	7	7	7	7	7	7	7	7
4	4	4	4	4	4	4	4	4	4
					3	3	3	3	3
2	2	2	2	2	2	2	2	2	2
	0	1	1	0	0	0	0	0	0
F	F	F	H	F	F	H	H	H	H

1 CYCLE OVER (10 REQUESTS) (R1=1 AND R2=3)

							7	7	7
3	3	3	3	3	3	3	3	3	3
		2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
F	H	F	H	H	H	H	F	H	H

HIT RATIO 1(FIRST CYCLE) =50%
HIT RATIO 2(SECOND CYCLE) =70%
AVERAGE HIT RATIO=60%

4. Efficient Implementation

There are many data structures like linked list for LRU and circular list for clock algorithm implementation. The D-range algorithm uses B tree, a very efficient data structure which has less searching (pertaining to reference bits) and more of insertion and deletion. B tree reduces the searching by half as it goes down from the root node.

In context of D-range algorithm there are certain points to remember:
1) The range pages will always be in root node or in case of more than five range pages or large reference counts, it can be in non-leaf nodes.
2) In case of large reference counts, the root node will keep on splitting and range pages will be included in the non-leaf nodes.
3) The pages which are to be inserted or deleted are situated in leaf nodes i.e ones which are not range pages.
4) The entire B tree is deleted after a range cycle and the range pages will be set according to the counter.

Whenever each page is referenced, a counter has to be maintained on each swapped in page. Maintaining a counter counts a lot of overhead but it is a very efficient way to increase the hit ratio. The overhead of a counter will not be as high as keeping in LRU (64-bit) because in D-range algorithm it will be maintained only for a particular predetermined range cycle and not until the process completes its execution. The counter will reset after each range cycle.

5. Experimental Results

Experimental results are depicted to analyze proposed algorithm (with respect to page faults and hit ratio). The main aim is to prove that, even with less implementation overhead than other page replacement algorithms in context of searching and replacing, it has same or higher hit ratio. The results have been conducted with the help of JAVA programming .For this purpose, 20 random sequences with length 20, were imposed to LRU, Optimal, FIFO and D-range. In this experiment, 3-frame and 5-frame were

analyzed and outcomes have been reported.

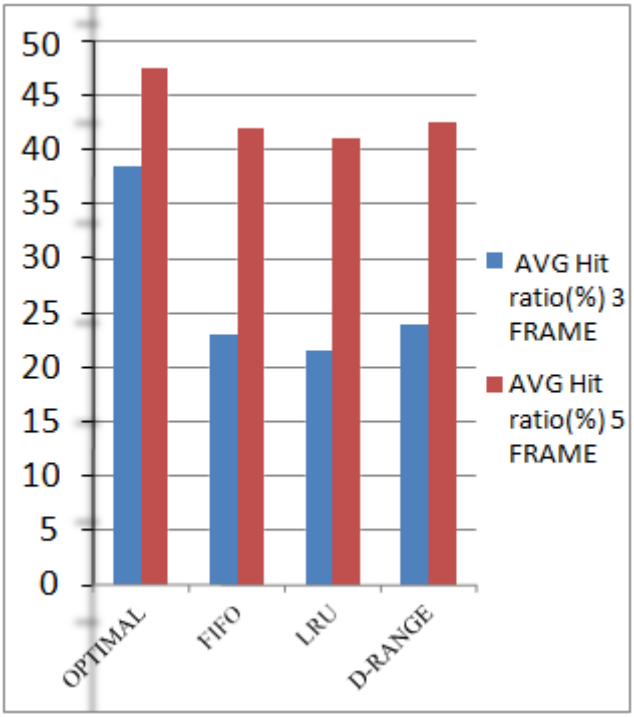

Figure 1: Graphical results for various page replacement algorithms based on hit ratio

6. Future Work

In real world scenario, the reference strings are very large, alternate range pages will take up a lot of space in main memory and cause space wastage. For such a situation, more space has to be left out between range pages and apply LRU/CLOCK algorithm over such various separate frames to decide which to replace.

The division of a big frame size in main memory into smaller frames with the help of range pages helps maintaining locality of reference and reduces implementation overhead due to tracking of less number of reference bits. To make an algorithm pertaining to such a situation and explaining the hardware implementation by the new data structure, B tree is precisely the future work.

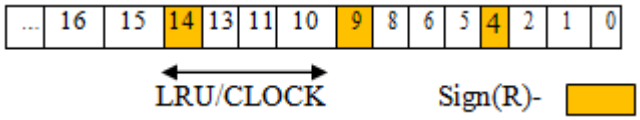

Figure 2: D-range for large references

7. Conclusion

As the operating systems grow day by day, there is an utmost need for workload adaption by page replacement algorithms and less implementation overhead. Algorithms revolving around locality of reference keeps increasing hit ratio slowly.

This paper introduced D-range which even after maintaining a considerable hit ratio has less implementation overhead than other algorithms by introducing a new data structure B tree.

References

[1] O'Neil, J. E., O'Neil, E. P., Weikum, G., "An Optimality Proof of the LRU-K Page Replacement Algorithm", Journal of the ACM, Vol. 46, No. 1, pp. 92- 112, January 1999.
[2] Ali Khosrozadeh, Sanaz Pashmforoush, Abolfazl Akbari, Maryam Bagheri, Neda Beikmahdavi., "Presenting a Novel Page Replacement Algorithm Based on LRU" , Journal of Basic and Applied Scientific Research , 2(10)10377-10383, 2012.
[3] Kim, K., Park, K., "Least Popularity – Per – Byte Replacement Algorithm for a Proxy Cache ", IEEE, pp. 780 – 787, 2001.
[4] S.M. Shamsheer Daula, Dr. K.E Sreenivasa Murthy, G Amjad Khan., "A Throughput Analysis on page replacement algorithm", International Journal of Engineering Research and Applications (IJERA), ISSN: 2248-9622, Vol. 2, Issue 2, pp.126-13, Mar-Apr 2012.
[5] "A Comparison of Page Replacement Algorithms" Amit S. Chavan, Kartik R. Nayak, Keval D. Vora, Manish D. Purohit and Pramila M. Chawan. IACSIT International Journal of Engineering and Technology, Vol.3, No.2, April 2011.
[6] "B tree" retrieved from https://en.wikipedia.org/wiki/B-tree.
[7] "A Study of Page Replacement Algorithms", Anvita Saxena International Journal of Engineering Research and General Science Volume 2, Issue 4, June-July, 2014 ISSN 2091-2730