A Relative Reference Responsive LRU based Page Replacement Algorithm for NAND Flash Memory

Hitha Paulson
Research Scholar, Bharathiar University, Coimbatore, Tamilnadu, India.
Assistant Professor, Little Flower College, Guruvayoor, Kerala, India.
E-mail: hitha@littleflowercollege.edu.in

Dr.R. Rajesh
Associate Professor, Department of Computer Science, CHRIST (Deemed to be University), Bangalore, India. E-mail: r.rajesh@christuniversity.in

Received November 15, 2020; Accepted December 18, 2020
ISSN: 1735-188X
DOI: 10.14704/WEB/V18I1/WEB18027

Abstract

The acceptance of NAND flash memories in the electronic world, due to its non-volatility, high density, low power consumption, small size and fast access speed is hopeful. Due to the limitations in life span and wear levelling, this memory needs special attention in its management techniques compared to the conventional techniques used in hard disks. In this paper, an efficient page replacement algorithm is proposed for NAND flash based memory systems. The proposed algorithm focuses on decision making policies based on the relative reference ratio of pages in memory. The size adjustable eviction window and the relative reference based shadow list management technique proposed by the algorithm contribute much to the efficiency in page replacement procedure. The simulation tool based experiments conducted shows that the proposed algorithm performs superior to the well-known flash based page replacement algorithms with regard to page hit ratio and memory read/write operations.

Keywords

NAND Flash Memory, LRU, Page Replacement Memory, Flash-DBSim.

Introduction

The innovations in mobile technology contributed to promising evolutions in computer memory architecture. High density flash memory chips are extensively used as the main storage media for consumer electronics. New era devices like smartphones and portable mobile devices are equipped with NAND flash memory as their storage component (Kwon, O., & Koh, K, 2010). Non-volatile nature, fast speed of access, less power consumption, lightweight, small size with high data density and low cost are the

exceptional qualities of NAND flash memories (Luo, J et al., 2015). These features of flash memories are highly commendable for a competent memory system but certain irregularities of the Flash memory components need to be addressed cautiously.

Shorter life spans, higher write operation cost and out-of-place updates are the major challenging issues before flash memory researchers (Liu, C et al., 2018). Shorter life span challenges put forth the wear levelling issues of memory cells. Higher write operation cost demanded, eviction of more clean pages than dirty pages. The out-of-place update problem based on the block erases granularity of flash memory called out more concern on limiting write operations. Because of this asymmetric behaviour, the state of art page replacement algorithms used for conventional hard disks stands inappropriate for flash systems (Lee, H et al., 2014). The modified versions of the Least Recently Used (LRU) algorithm are the solution candidates that emerged first. But they cannot adapt themselves to all the asymmetric properties of flash memory. Hence the necessity to keep up a high page hit ratio for different work volume, by reducing the number of write operations remained challenging (Lin, M et al., 2016).

In this paper, the Relative Reference Responsive LRU based Page Replacement algorithm for NAND Flash Memory is proposed. It maintains a cold LRU page list, hot LRU page list and a shadow list for effective decision making in page replacement strategies. Each page in the hot and cold list is associated with a staycount and callcount parameter which tracks the reference count of the page with respect to its age in the buffer list. The shadow list keeps track of the eviction history, which helps for the self-adjustments in the eviction algorithm, based on reference from the shadow list. The adjustable eviction window in the hot list and the size adjustable property of the shadow list adds more optimality to page replacement decisions. The proposed page replacement algorithm calculates and monitors the relative reference ratio of pages for better decision making. Experimental results received from the simulated environment shown that the proposed algorithm is a better alternative to the state-of-art algorithms used for flash page replacement.

This paper is framed as follows: an overview of flash memory implementation and motivation in doing this work is explained in first section. Next sections present the popular page replacement algorithms exist for the flash memory page replacement, the proposed RRR-LRU algorithm and analyses the experimental setup and results.

Background Study

The mechanical based architecture of Hard Disk Drives (HDD) made them inappropriate for the new era devices like smartphones, portable media devices etc. The main drawback

reported for the hard disks is its higher form factor. Even though HDDs come up with large density, due to mechanical part implementation, hard disks cannot be manufactured smaller than a particular limit. This peculiarity made HDDs more suitable for servers and desktop applications than portable devices (Rizvi, S.S., & Chung, T.S., 2010). This mechanical nature made them more vulnerable to errors and noise during read/write operations. These memory concerns led to a more compatible and reliable memory component called flash memory. Even though flash memories have highly reported asymmetricity in I/O operations and life span issues, they got wide acceptance due to their small size, low cost, low power consumption and fast access speed. The wide acceptability of Solid State Drives (SSD) with flash memory implementation shows this influence.

Overview of Flash Memory Implementation

Two common Flash memory types are there, NAND flash memory and NOR flash memory. NAND and NOR flash memories differ very much in their characteristics. In many of the aspects, NOR flash memory performed better than NAND flash memory. But because of lower cost and faster write/erase speed, NAND flash gained more advantage over NOR flash (Yuan, Y et al., 2019). Table 1 summarises the characteristics of memory.

Table 1 NAND Flash vs. NOR Flash

Characteristic	NAND Flash	NOR Flash
Cost	Low	High
Capacity	High	Low
Read Speed	Slower	Faster
Write Speed	Faster	Slower
Erase Speed	Faster	Slower
Data retention	Lower	Higher
Life Span	10 times more than NOR	Less
Access	Sequential in a page	Random

NAND flash memory is organised as blocks. Each block comprises of many pages. Read, write and erase are the operations that can be performed. The granularity of an erase operation is a block and that of read/write operation is pages. NAND flash follows erase before the write operation. So, if we want to update a page in memory, we need to erase a full block in memory. The cost of erase operation is greater than the write operation and the cost of the write operation is greater than the read operation. An out-of-place scheme is followed by NAND flash for better implementation of the update operation. As per this scheme, if we want to write a page that is modified, it is written to a new memory page and the old one gets added to garbage (Huang, Q et al., 2019). We should also take care of

the wear leveling factor of each cell since the number of erase operations determines the life factor of the memory block.

Motivation

The conventional page replacement algorithms concentrated on increasing the page hit ratio, but the asymmetric features of NAND flash memory demanded more optimisation in various operations like read, write and erase. Recently many page replacement algorithms were proposed by the researchers for flash memory systems but, many of the algorithms always gave preference to clean page replacement and considered dirty pages as a second choice. This repetitive pattern of algorithms reduced system performance. The motivation for working in this paper is to develop an algorithm, which not only increases the hit ratio but also lessens the read/write overhead. The proposed algorithm met the expected outcome with a better page hit ratio and a reduced number of I/O operations.

Related Work

Conventional page replacement algorithms developed for hard disks are not suitable for flash memory implementations. With an intension to lessen the write/erase overheads and to improve the hit ratio various algorithms are designed and this section details the popular flash page replacement algorithms.

A Clean First LRU (CFLRU) algorithm was proposed by Park et al for NAND flash based storage components (Park, S.Y et al., 2006). This algorithm maintains an LRU list with a logical, clean first region window at the LRU end and a working set region at the MRU end. This algorithm classifies the pages as clean and dirty pages. The pages on which only the read operation is performed are called clean pages and the page which is modified and needs to be written back to memory while the replacement is called dirty pages. The algorithm chooses the LRU most clean page as a victim if page replacement is needed and LRU most dirty page as victim in the absence of clean pages. The logical barrier separating the working window and the clean first region window gets attuned suitably to adjust the workloads and to increase the hit ratio. The cost of read/write operations, the number of clean/dirty pages evicted and the probability of clean page reference in the future determines the window size. The major drawback found for the algorithm is the inaccuracy in determining the appropriate size of the window for workload adjustments.

Jung et al put forth an LRU based Write Sequence Reordering algorithm and is named as LRU-WSR (Jung, H et al., 2008). The algorithm proposed a method to categorise the pages in memory as hot and cold. Pages that are referenced only once are tagged to be

cold pages and those which are referenced more than once are hot pages. There will be an LRU page list and a cold flag is attached to each page. This flag shows whether the clean/dirty page is cold or not. Pages for replacement are selected from the LRU end. If the victim page is a clean page it will be replaced without checking the cold flag. If it is a dirty page with cold flag zero, then it is selected as victim. If it is a hot dirty page the algorithm makes it cold and places it to MRU position. The algorithm focused on delaying the eviction of dirty pages. Even though the overall performance of the algorithm seems to be good, it ignores the access frequency of clean pages and it adversely affects the hit ratio.

Li et al proposed a CCF-LRU algorithm for NAND Flash memory (Li, Z et al., 2009). The algorithm maintains two LRU lists. The mixed LRU list includes dirty pages and hot clean pages. Cold clean pages are kept in the list2. As and when a reference occurs to cold pages list that page gets moved to mixed list. When a re-reference from mixed list occurs that page gets moved to the MRU end. During the replacement procedure, List2 LRU end is scanned first. If that list is empty, the LRU of the mixed list is considered. If it is a cold dirty page then it will be selected as victim. Hot dirty pages are marked as cold dirty and kept at MRU end. If it is hot clean, it is tagged as cold and moved to List2. This process goes on until a victim is finalised. The algorithm aims at giving a second chance to many pages before they get replaced. The major drawback of the algorithm is its higher preference for cold clean pages for eviction, which may lead to degradation of IO performance.

The Adaptive Double LRU (AD-LRU) proposed by P. Jin et al aims at evicting the least frequently and least recently used page during the time of page replacement (Jin, P et al., 2012). The algorithm maintains a cold LRU and Hot LRU page lists. Replacement occurs from cold list LRU and clean pages are preferred. A minimum lower count parameter is maintained to adjust the size of the cold region. Eviction occurs from cold window till it maintains a size greater than or equal to lower count. Otherwise, eviction from the hot region occurs. Referenced pages from the cold LRU list moves to the hot LRU list, and the pages from the hot LRU list moves to the MRU end during hotlist reference. The main issue identified in the algorithm is its avoidance of cold dirty pages during eviction which may lead to unnecessary memory wastage. The method of determining a minimum lower count value is not well defined for different workloads.

Based on the above popular LRU based flash page replacement algorithms, many page replacement algorithms for flash memory were devised in recent years. But, many of them

concentrated only on certain evaluating criteria, so they failed in contributing to overall system performance.

Proposed Algorithm

In this section, a Relative Reference Responsive page replacement algorithm is presented for NAND flash based memories. It is called RRR-LRU. The main aim of this algorithm is to improve the performance of flash based page replacement algorithms by accomplishing a higher page hit ratio and to reduce the number of the page write operations. To achieve this goal, the proposed algorithm tracks a relative reference value for each page in the memory buffer by recording, the total number of page references processed and the number of references that occurred to a particular page in that period. Various decision making policies are also followed for better optimisation.

As given in figure 1, the RRR-LRU algorithm maintains three LRU page lists, called cold LRU list (cdLRU), hot LRU list (htLRU) and shadowlist (sdLRU). If ‘s’ is the size of the main memory buffer allotted in terms of the number of pages, then the maximum possible size of the cdLRU and htLRU together is 's'. The given equation indicates the size variations in cold and hot LRU lists.

$$0 \leq |\text{cdLRU}| + |\text{htLRU}| \leq s \qquad (1)$$

|cdLRU| represents the size of the cdLRU list and |htLRU| represents the size of the htLRU list in terms of the number of pages. The shadow list is a size adjustable list, whose size ranges between 0 and s, but fixed to a minimum of one third the allotted buffer size(s) once it reaches that count. The list size gets adjusted with respect to the request responsive parameter.

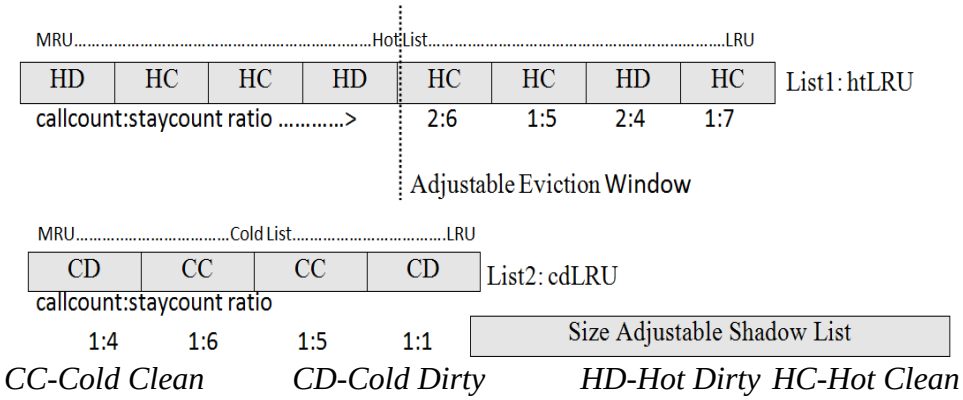

Figure 1 RRR-LRU Implementation

The cdLRU list consists of clean and the dirty pages which are newly accessed from memory and the list maintains pointers to access pages in most recently and least recently

used reference order. The htLRU list consists of clean and dirty pages that are referenced for the second time when they are in the cdLRU list. So the pages in the cdLRU list are pages that are referenced only once and those in the htLRU list consist of pages that are referenced more than once. Thus the frequently accessed pages are maintained in the htLRU list. The htLRU list maintains a logical adjustable eviction window, from which the victim is selected during the page replacement process. The sdLRU list keeps track of the metadata related to the evicted pages from the cdLRU or htLRU list and the list helps in identifying whether the recently evicted pages are referenced again in the near future. The size of the shadow list is adjustable.

As mentioned in the previous sections the main aim of the RRR-LRU algorithm is to lessen the page write operations and to increase the page hit ratio to overcome the architectural limitations of NAND Flash memory. To accomplish this goal a parameter called the relative reference ratio of each page is updated and monitored. The staycount and callcount parameters associated with each page are used to evaluate the relative reference ratio. The callcount parameter of a page records the number of times the page is referenced during its lifetime in the memory list and staycount records the total number of page references occurred after that pages arrival to the memory list. Staycount symbolises the age of the page in the main memory. The ratio between these two, the relative reference ratio is a true indicator of its frequency in terms of the total references that occurred. This ratio plays a key role in the decision making of victim selection. Thus, the Relative Reference Ratio of a page = Callcount of the page/ Staycount of the page.

The relative reference ratio of the shadow list is calculated at a fixed reference interval of time. References of about one third of the total allotted buffer size are fixed reference interval used. The number of references that occurred from the shadow list for this interval of time contributes to the relative reference ratio of the shadowlist. The size of the shadow list is determined from the relative reference ratio. If the relative reference ratio of the shadow list is greater than 0.5, it shows the probability of more references from the shadow list and the size of the shadow list is incremented by one. If the ratio is less than or equal to 0.5, the size is decremented by one, keeping a minimum limit of one third the total allotted buffer size.

During a page reference, the page is searched in the htLRU list first. If the page is found in the list it is moved to the MRU end of the htLRU list, indicating its recency. The Callcount of the corresponding referenced page and staycount of all the pages in cdLRU and htLRU list gets incremented by one. If the page is not found in the htLRU list, the

search continues in the cdLRU list. If found, the page is moved to the MRU end of the htLRU list and callcount, staycount get updated.

Procedure: Main ()
Initialise staycount and callcount of all pages to 0,
BUFFMAX to buffersize in pages,
totalbuffercount=0,prioritycount=0,metastaycount=0,metamincount=BUFFMAX/3,shadowsize=0,
Input: Page p, Pageoperation opr

```
1: BEGIN
2: FOR EACH input
3: FOR EACH Page g in htLRU and cdLRU
4:          g.staycount=g.staycount+1
5: END FOR
6: if Page p in htLRU, then
7: move Page p to MRU of htLRU
8:          p.callcount=p.callcount+1
9: else if Page p in cdLRU, then
10:         move Page p to MRU of htLRU
11:         p.callcount=p.callcount+1
12: else if Page p in sdLRU at m,then
13:         if (totalbuffercount=BUFFMAX)
14:         Page v= selectvictim()
15:         windowadjust(v)
16: endif
17:                 metastorecount=metastorecount+1;
18:                 remove m from shadow list.
19:                 Allocate and read page p from memory to
                    pagenode n & update totalbuffercount
20:         ARCS=m.callcount/m.staycount
21:         if(ARCS>=.5 or prioritycount=1)
22:         add page n to MRU of htLRU
23:         n.callcount=2,n.staycount=2
24: else
25:         add page n to MRU of cdLRU
26:         n.callcount=1,n.staycount=1
27: endif
28: else    // page not in
 hotlist,coldlist,shadowlist
29:         if (totalbuffercount=BUFFMAX)
30: Page v= selectvictim()
31:         windowadjust(v)
32: endif
33:         allocate and read page p from memory
 to pagenode n & update totalbuffercount
34:         add page n to MRU of cdLRU
35:         n.callcount=1, n.staycount=1
36: endif
37: update dirtybit if input page opr is 1
38: //adjust shadow list
39: metastaycount=metastaycount+1
40: if(metastaycount=metamin)
41: if((metastorecount/metastaycount)>=.5)
42: prioritycount=1
43:         shadowsize=shadowsize+1,till
                    shadowsize=BUFFMAX
44: else if(shadowsize>BUFFMAX/3)
45:                 shadowsize=shadowsize-1
46: Endif
47:         prioritycount=0
48: Endif
49:         metastaycount=metastorecount=0;
50: Endif
51: END FOR
52: END
```

Procedure: Selectvictim()

```
1: BEGIN
2: if(cdLRU != NULL)
3:          victim=LRU most clean page
4:          If no clean page in cdLRU
5:          victim=LRU most dirty page
6: vflag='c'
7: else if(htLRU !=NULL)
8:          victim=clean page with
            minimum(callcount/staycount) from logical
            eviction window
9:          if no clean page in eviction window then
10:         victim=   dirty page with
            minimum(callcount/staycount) from logical
            eviction window
11: vflag='h'
12: endif
13:add victim to shadow list
14: if(victim.dirtybit=1)
15:         Writeback the page to memory
16: Endif
17: Return victim
18: END
```

Procedure: windowadjust(v)

```
1: BEGIN
2: if(vflag='h' & v.dirty=0 &
            evictionwindowsize>total pages in HL/4)
3:          Decrement evictionwindowsize by 1
4: else if(v.dirty=1 & evictionwindowsize<total pages in
            HL/2)
5:                  Increment evictionwindowsize by 1
6: endif
7: endif
8: END
```

Figure 2 Algorithm RRR-LRU

If the page in the shadow list is referenced, the RRR-LRU algorithm checks the recently calculated relative reference ratio of the shadow list. If the ratio is greater than 0.5 the pages are moved to the MRU end of the hotlist directly irrespective of the dirty bit. The callcount and staycount of the page are set to two, the minimum count in the hot list pages. If the recently calculated relative reference ratio of the shadow list is less than or equal to 0.5 the algorithm checks the relative reference count of the page during its eviction time. If that count is greater than 0.5, the page is also moved to the MRU end of the htLRU list irrespective of the dirty bit. Those pages whose relative reference count is less than 0.5 during its eviction time are moved to the cdLRU list with callcount and staycount set to one. Since the shadowlist maintains only the metadata of the evicted pages, for each page reference from the shadowlist, the RRR-LRU algorithm loads the page data to the main memory.

If the size of the cdLRU list and htLRU list reaches its maximum as per equation (1), a victim selection and deletion procedure is initiated by the RRR-LRU algorithm during the new page references. The RRR-LRU algorithm first checks the page at the LRU end of the cdLRU list. If it is a clean page it is selected as victim. Due to the nature of coldlist the LRU end pages of the cdLRU list always have the minimum relative reference ratio. If the LRU end page is a dirty page the algorithm checks for a clean page in the list and selects the LRU most clean page as victim. If there is no clean page in the list, LRU dirty page is selected as victim. The victim page metadata will be moved to the MRU end of the shadow list and the page is erased from memory. Depending upon the dirty bit of the page evicted, a memory write operation is initiated.

If the cdLRU list is empty, the algorithm checks for the victim page in the htLRU list. htLRU list maintains an adjustable logical eviction window. The victim is selected from the pages belongs to the eviction window. This method ensures the noneviction of the most recent pages even though they have a low relative reference ratio. The eviction window is initialised with a size |htLRU|/4 and varies up to |htLRU|/2 depending upon the nature of the page evicted. When the algorithm checks for the victim in the htLRU list, it checks for a clean page in the eviction window which meets the criteria,

$$\text{Victim } p = \text{page with } \min\{\text{relative reference ratio}\} \quad (2)$$

This method of eviction gives preference to clean pages in the eviction window to remain in the htLRU list if their frequency of reference is more. If no clean page is there in the eviction window, a dirty page that satisfies the criteria (2) is selected. The victim page metadata will be moved to the MRU end of the shadow list and the page is erased from

memory. Depending upon the dirty bit of the page evicted, a memory write operation is initiated. When a page eviction happens from the htLRU list a window adjusts operation is initiated. If the selected victim is a clean page, the eviction window size is decreased by one and if it is a dirty page the eviction window size is increased by one to accommodate clean pages in the logical window. This method ensures less page write operation in memory by giving preference to clean pages. Figure 2 gives an algorithmic representation of the proposed RRR-LRU algorithm.

Performance Evaluation

This section evaluates the performance of the proposed RRR-LRU algorithm. We have used Flash-DBSim, a simulation tool for analysing Flash memory based algorithms (Su, X et al., 2009). Flash-DBSim is a popular memory simulation tool used by many of the researchers for evaluating the performance of algorithms for NAND flash memory. The tool gives a reconfigurable architecture for algorithm implementation and evaluation, along with a large set of pre-recorded traces. The performance of the proposed algorithm is compared with conventional LRU and the popular NAND flash page replacement algorithms CFLRU, LRU-WSR, CCFLRU and ADLRU.

Experimental Setup

The Flash-DBSim simulator was set to a NAND memory configuration of 128MB, with 1024 blocks of 64 pages each and page size set to 2KB. The simulator was set with an erase limitation of 100000, with a read time of 25µs. Since NAND flash memory always reported eight times the read time for write operation (Lee, H et al., 2014) a write time of 200 µs was set. An erase speed of 1500µs/block with a wear leveling threshold of 4 was set in the simulator. Initially, the experiments were conducted with the pre-recorded traces of Flash-DBSim. For better accuracy of testing three types of synthetic traces generated according to Zipf distribution are used for testing.

- Read most traces – with a read/write ratio of 9:1
- Write most traces – with a read/write ratio of 1:9
- Normal traces –with a read/write ratio of 1:1

For optimising the results of the experiment, each of the traces was tested with various page buffer sizes, ranges from 1 to 5 MB. Three performance parameters page hit ratio, page read ratio and page write ratio are evaluated for algorithm comparison. Higher hit ratio with lesser read and write counts were expected.

Result Assessment

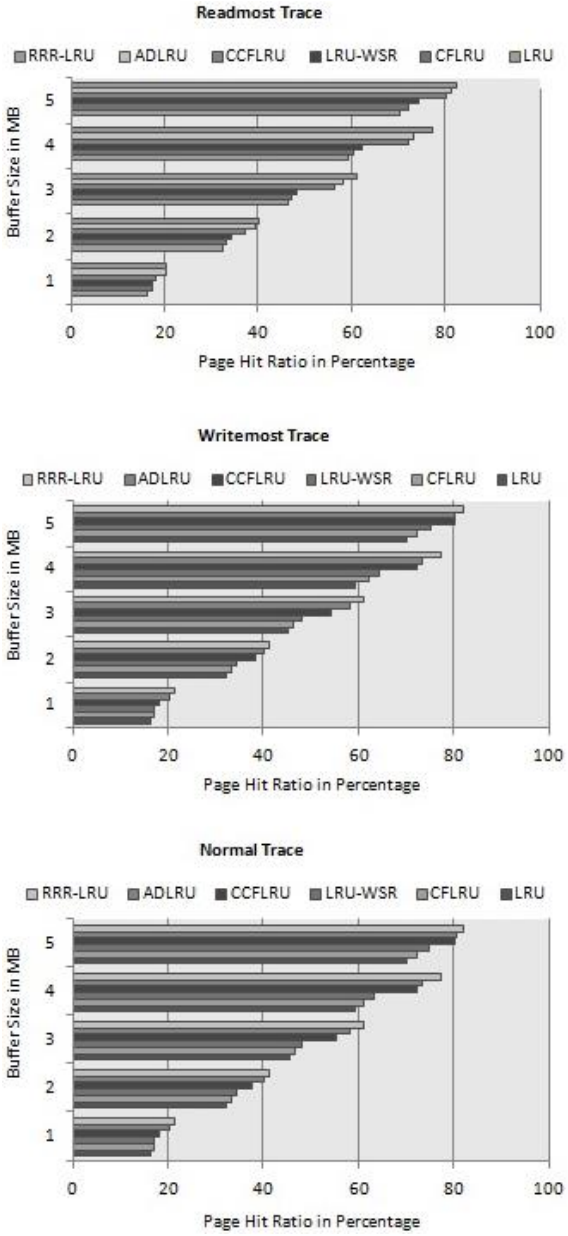

Figure 3 Page hit ratio for three types of traces

Figure 3 shows the reported hit ratio of six flash page replacement algorithms for buffer sizes varying from 1 to 5 MB. The LRU algorithm optimised for page replacement policies of hard disks were also compared for better evaluation and understanding. The other five algorithms are proposed for NAND flash based memory units. From the results, it is obvious that the RRR-LRU algorithm reported a higher hit ratio in all the tested buffer sizes for the three types of traces used. In addition to the decisions concentrated on the replacement of clean pages, the decisions based on the relative reference ratio of the

pages contributed much to the betterment of the hit ratio in RRR-LRU algorithms. The decisions proposed by the algorithm regarding the referenced shadow list pages and the monitoring of the relative reference ratio of the shadow list at a fixed interval of time contributed much to the betterment in the page hit ratio.

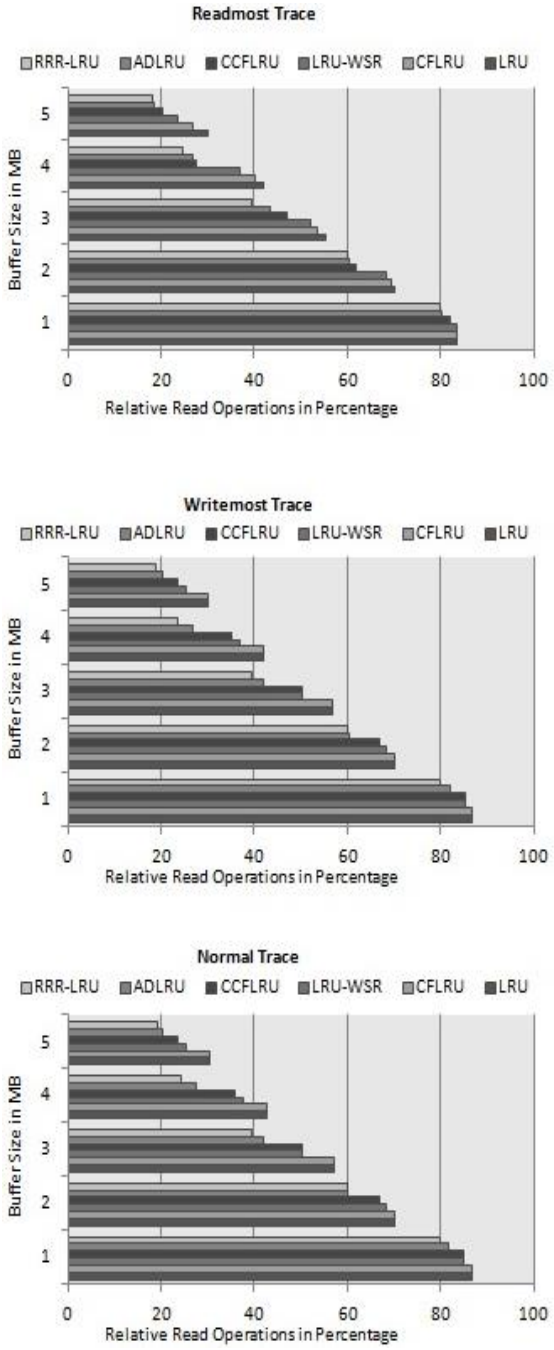

Figure 4 Relative read operations for three types of traces

Figure 4 shows the reported relative read operations percentage of the six page replacement algorithms, with respect to the total number of page references. RRR-LRU

algorithm performed better than the other algorithms compared. The higher page hit ratio reported by the RRR-LRU algorithm contributed to the decrease in page read operations which in effect improved the overall performance of the page replacement mechanism.

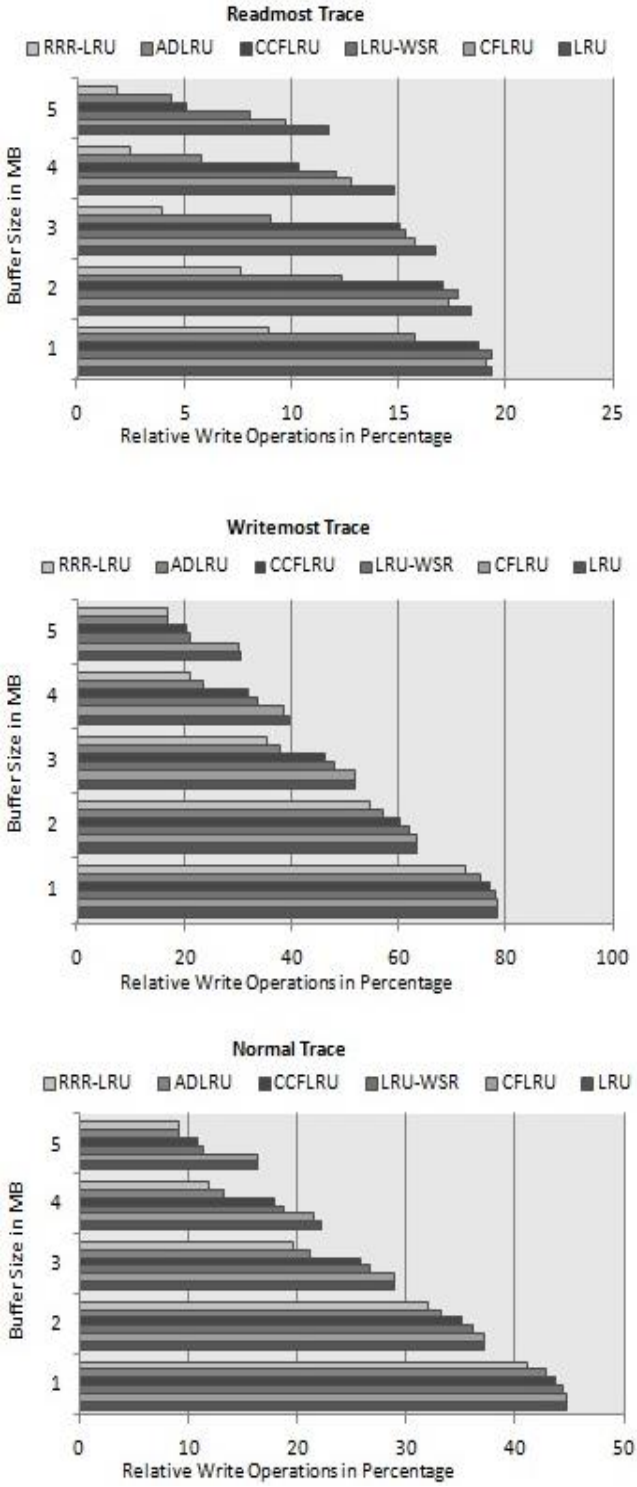

Figure 5 Relative write operations for three types of traces

Figure 5 shows the relative write count percentage of the six page replacement algorithms, with respect to the total number of page references. RRR-LRU algorithm performed better with lesser write count. To lessen the write count of pages, the RRR-LRU algorithm focused on clean page removal, with respect to their relative reference ratio. The adjustable window feature in the htLRU list, preference to the referenced shadow list pages with respect to the relative reference ratio during the eviction and the second chance to remain in the list offered to the LRU dirty pages during eviction helped in lessening the page write operations in memory.

Conclusion

In this paper, a relative reference responsive LRU based page replacement algorithm for flash memory is proposed for NAND based flash devices, which is called RRR-LRU. The algorithm maintains two LRU based lists named cdLRU, the cold page list and htLRU, the hot page list to effectively track the pages based on their frequency in referencing. The cold list and the MRU end of the hotlist indicate the recency in referencing. The shadow list maintained by the algorithm ensures the tracking of pages that are needed but evicted recently. To increase the page hit ratio, the algorithm implements decisions based on the relative reference ratio of the pages and the shadow list. To decrease the number of write operations preference is given to clean pages while eviction, but the decision policy based on relative reference ratio determines who among the clean page is better to evict. A sequence of experiments was conducted in the well versed Flash-DBSim simulator, with different page traces covering the possibilities in buffer allocation and program nature. The experimental outcome shows that the RRR-LRU algorithm performs better than the popular existing flash based page replacement algorithms.

References

Kwon, O., & Koh, K. (2010). Swap space management technique for portable consumer electronics with NAND flash memory. *IEEE Transactions on Consumer Electronics, 56*(3), 1524-1531.

Luo, J., Fan, L., & Tsu, C. (2015). A NAND flash management algorithm with limited on-chip buffer resource. *Computers & Electrical Engineering, 44*, 1-12.

Liu, C., Lv, M., Pan, Y., Chen, H., Li, Y., Li, C., & Xu, Y. (2018). LCR: load-aware cache replacement algorithm for flash-based SSDS. *In IEEE International Conference on Networking, Architecture and Storage (NAS)*, 1-10.

Lee, H., Bahn, H., & Shin, K.G. (2014). Page replacement for write references in NAND flash based virtual memory systems. *Journal of Computing Science and Engineering, 8*(3), 157-172.

Lin, M., Yao, Z., & Xiong, J. (2016). History-aware page replacement algorithm for NAND flash-based consumer electronics. *IEEE Transactions on Consumer Electronics, 62*(1), 23-29. https://doi.org/10.1109/TCE.2016.7448559

Rizvi, S.S., & Chung, T.S. (2010). Flash SSD vs. HDD: High performance oriented modern embedded and multimedia storage systems. *In 2nd International Conference on Computer Engineering and Technology, 7,* V7-297. https://doi.org/10.1109/ICCET.2010.5485421

Yuan, Y., Zhang, J., Han, G., Jia, G., Yan, L., & Li, W, (2019). DPW-LRU: An efficient buffer management policy based on dynamic page weight for flash memory in cyber-physical systems. *IEEE Access, 7,* 58810-58821. https://doi.org/10.1109/ACCESS.2019.2914231.

Huang, Q., Chen, R., Lin, M., Yang, C., Chen, Q., & Li, X. (2019). Clean-First Adaptive Buffer Replacement Algorithm for NAND Flash-based Consumer Electronics. *In IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom),* 1217-1223. https://doi.org/10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00173

Park, S.Y., Jung, D., Kang, J.U., Kim, J.S., & Lee, J. (2006). CFLRU: a replacement algorithm for flash memory. *In Proceedings of the international conference on Compilers, architecture and synthesis for embedded systems,* 234-241.

Jung, H., Shim, H., Park, S., Kang, S., & Cha, J. (2008). LRU-WSR: integration of LRU and writes sequence reordering for flash memory. *IEEE Transactions on Consumer Electronics, 54*(3), 1215-1223.

Li, Z., Jin, P., Su, X., Cui, K., & Yue, L. (2009). CCF-LRU: A new buffer replacement algorithm for flash memory. *IEEE Transactions on Consumer Electronics, 55*(3), 1351-1359.

Jin, P., Ou, Y., Härder, T., & Li, Z. (2012). AD-LRU: An efficient buffer replacement algorithm for flash-based databases. *Data & Knowledge Engineering, 72,* 83-102.

Su, X., Jin, P., Xiang, X., Cui, K., & Yue, L. (2009). Flash-DBSim: a simulation tool for evaluating flash-based database algorithms. *In 2nd IEEE International Conference on Computer Science and Information Technology,* 185-189.

Lee, H., Bahn, H., & Shin, K.G. (2014). Page replacement for write references in NAND flash based virtual memory systems. *Journal of Computing Science and Engineering, 8*(3), 157-172.