Evaluation of Write Sequence Reordering Based Buffer Replacement Algorithms for Flash Memory Based Systems

Arjun Singh Saud[1,*] Hari Sharan Bhatt[2]

[1]Central Department of Computer Science and IT, Tribhuvan University, Kathmandu, Nepal.

[2]Department of Education, Far Western University, Central Campus, Kanchanpur, Nepal.

[*]Email: arjunsaud@cdcsit.edu.np

Abstract

Many page replacement algorithms have been developed for disk-based systems. All of them consider the hit rate as a key performance measure. Flash memory has different characteristics than hard disks, such as asymmetric I/O latency among read, write, and erase operations. The read operation is faster than the write and erase operation, and it does not support in-place updates. Besides, write operations shorten the life of flash memories. Therefore, the number of write counts is also an important factor to be considered for flash-based systems. This paper studied WSR-based algorithms, namely the LRU-WSR and LIRS-WSR, in terms of both hit rate and write count. The trace-driven simulation is performed with four different workloads: random, read-most, write-most, and Zipf traces. The simulation result showed that LIRS-WSR is better than LRU-WSR when locality is not too strong; otherwise, the performance of LRU-WSR is better.

Keywords: Flash memory, Storage system, Page replacement algorithms.

1. Introduction

In contrast with disks, flash memory is an electronic type of memory. It does not have any moveable components, and hence it does not have any rotational latencies, which makes it faster than HDD. There are two types of flash memories: NOR and NAND. NOR flash has better read performance than NAND, whereas NAND has better write performance and offers high cell densities. The life of NOR flash ranges from 10,000 to 100,000 write/erase cycles, and the life of NAND flash is 100,000 to 1,000,000 write/erase cycles [1]. Nowadays, flash memory is used as an alternative to hard disk in many devices like laptops, mobiles, digital cameras, PDAs, etc. as secondary memory. This memory differs from hard drives in that it has asymmetric I/O latency for read, write, and erase operations. The read operation is faster than the write and erase operations, and it does not support in-place updates. To write into any memory cell, it is necessary to erase that cell. Since there are limited numbers of erases

available, erase operations will drain the cell and decrease the life of flash memory. So clever memory management techniques can save the life of flash memory [1–3].

When memory is full and a new page is requested by the CPU, the replacement algorithm replaces one of the pages from memory so that a new page can be loaded into the empty slot of the memory. In the case of disk-based systems, the replacement algorithm should not replace the pages that are frequently used to increase the hit rate, which improves the overall performance of the system. However, in a flash-based system, improving the hit rate is not the only factor that should be considered. While replacing the pages, it is necessary to categorize the clean and dirty pages. Clean pages can be replaced directly from memory when a page fault occurs, and eviction of dirty pages should be delayed because replacement of dirty pages is related to reducing the write count. In this paper, two write sequence reordering (WSR)-based algorithms, namely LRU-WSR [4] and LIRS-WSR [5], are studied and evaluated for flash memory-based systems in terms of hit rate and write count.

The rest of the paper is organized as follows: Section 2 describes the LRU-WSR and LIRS-WSR algorithms. Section 3 explains the literature review and related works. Section 4 describes the research methodology. Section 5 includes the result and discussion, and finally, Section 6 involves the conclusion and future recommendations.

2. WSR Based Algorithms

Write operations require erase operations, and each erase operation decreases the life of flash memory. Due to this reason, replacement algorithms must use some policy to flush the dirty pages to the flash memory. WSR algorithms delay the eviction of dirty pages from cache when page faults occur because keeping dirty pages in cache will reduce the number of erase operations, which will lengthen the life of flash memory. This section briefly describes the LRU-WSR and LISR-WSR replacement policies.

2.1 LRU-WSR

The LRU-WSR algorithm is the first WSR-based buffer replacement algorithm designed for flash-based systems [4]. It is an enhancement of the LRU algorithm so that it can suit the characteristics of flash memory-based systems. LRU considers the recency factor when a page is to be replaced. It assumes that the most recent referenced page will be referenced in the near future. LRU-WSR is an enhancement of the CFLRU [6] algorithm designed for flash-based memory by applying the concept of cold and hot properties to dirty pages. CFLRU replaces the clean pages and keeps dirty pages in memory, whether they are cold or hot. Keeping cold,

dirty pages in cache will decrease the hit ratio, which is not good. For this reason, LRU-WSR uses a cold flag that is set when the page is cold and cleared when the page is hot.

LRU-WSR can be implemented by using the LRU list, which is divided into two parts: MRU and LRU. MRU contains the pages that were accessed recently, and LRU contains the pages that were accessed in the past. Thus, MRU is the head of the list, and LRU is the end of the list. When the page is first referenced, it is inserted at the MRU until the list is full. The LRU list is shown in Figure 1. When the page fault occurs, the page at the LRU end is replaced using the WSR policy. WSR policy uses the concepts of second chance and cold detection. It searches for clean or cold dirty pages for replacement and gives a second chance to hot dirty pages. Therefore, sometime it is also called the cold detection algorithm.

When a page fault occurs, the algorithm starts searching from the end of the list for clean or cold dirty pages. If the page at the end of the list is clean, it is directly replaced from memory without checking its hot/cold status. If the page at the end of the list is cold dirty, it is replaced and the write count is incremented. If the page at the end is hot dirty, it is given a second chance and is inserted at the head of the list by setting its cold flag. Then, the algorithm searches for the next page until the clean or cold dirty page is found.

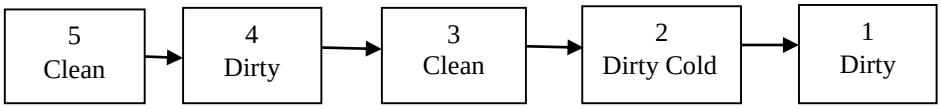

Fig 2.1: LRU-WSR list

2.2 LIRS-WSR

The LIRS-WSR algorithm is an enhancement of the LIRS [7] algorithm made to better fit the flash-based systems. Additionally, it leverages the WSR policy to postpone the eviction of hot, dirty pages. It utilizes a similar idea to the LIRS algorithm. The High Inter-Reference Recency (HIR) page set and the Low Inter-Reference Recency (LIR) page set are the two sets into which the referred pages in LIRS are divided. Each page in memory that contains historical data has a status, either LIR or HIR. Even though some HIR pages may not be physically present in memory, they may nonetheless have entries in the memory that reflect this fact. Memory is divided into two partitions, C_{LIRS} and C_{HIRS}, in a similar way. C_{LIRS} holds the set of LIR pages with the highest likelihood of being accessed again soon, so re-accessing these pages is assured to be hit. Because C_{HIRS} is so small—typically 1% of the memory size—it contains a group of HIR pages with a lower likelihood of being referred in the near future. Accessing these pages

may result in a page fault. HIR resident blocks may be replaced at any time. However, when an LIR block's recency reaches a specific level and an HIR block is visited at a lower recency than the LIR block, the statuses of the two blocks change [5].

Two lists can be used to implement LIRS: HIR list Q, which keeps HIR resident pages, and LIRS stack S, which holds all LIR and HIR pages regardless of residency status. The two LIRS lists are displayed in Figure 2. The front-most page in list Q is always selected as a victim because, as was previously noted, LIRS attempts to evict the HIR page with the biggest recency measure as a victim.

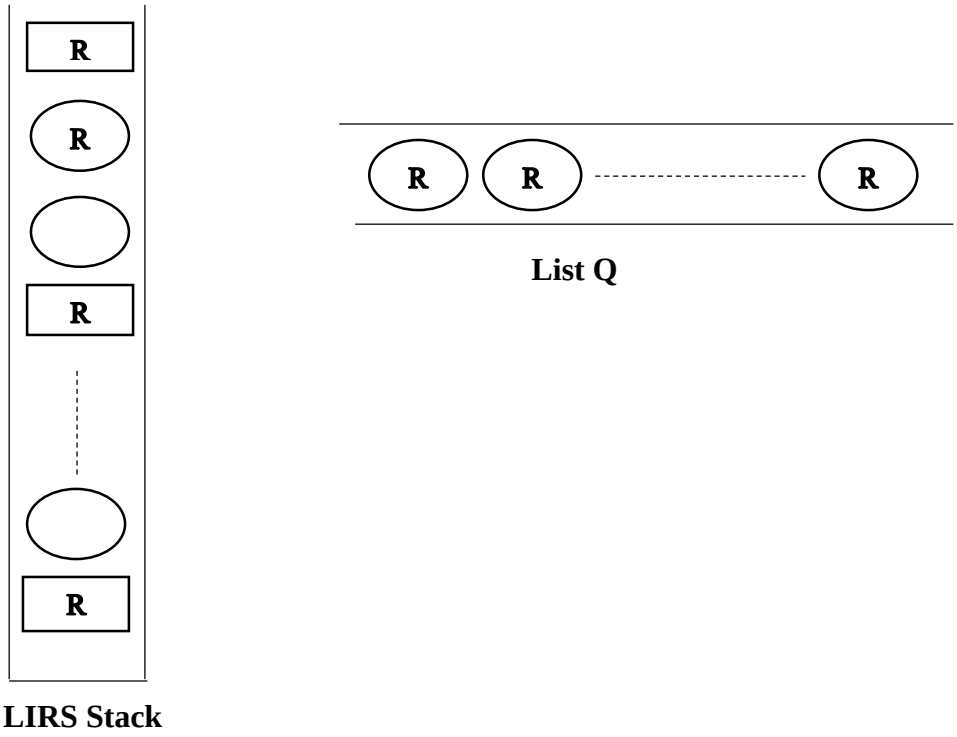

Figure 2: Two list of LIRS algorithm [7]

The main benefit of using stack is that it automatically calculates the recency value. It is not necessary to explicitly calculate the recency of each referenced page. The bottom-most page is always a LIR page with a higher recency value, and the topmost is the most recently referenced page with a zero recency value. The WSR policy is applied to every switch and replacement operation. That is, only clean or cold dirty pages are moved to the HIRQ [5].

3. Literature Review

So many page replacement algorithms have been developed for the flash memory-based system. They all try to save the life of flash memory by keeping dirty pages in memory and maintaining the hit ratio. CFLRU [6] stands for clean first LRU and is the first algorithm

developed for a flash memory-based system. It always tries to evict the clean pages from memory when a page fault occurs. It does not classify the dirty pages as cold or hot when replacing the pages, which causes a decrease in the hit rate.

Clean First Dirty Clustered (CFDC) [8] is an improvement on CFLRU by placing the hot pages at the head of the list and the cold pages at the end of the list so that the algorithm does not have to travel a long distance to replace a clean page. It is somehow similar to the CFLRU in hit rate and write count but reduces the searching cost of clean pages.

LRU-WSR [4] is another improvement on the LRU [9] algorithm with the WSR policy. It divides the dirty pages into two classes: hot and cold, which is lacking in the CFLRU and CFDC algorithms. LRU-WSR always tries to replace clean or cold dirty pages from memory when a page fault occurs. Due to the WSR policy, it maintains both the hit rate and write count.

In CFLRU and CFDC, the priority of staying in memory is always given to the dirty pages, whether they are hot or cold. This process may sometimes decrease the hit rate if the pages are hot in nature. CCF-LRU [10] stands for Cold-Clean-First LRU and improves the LRU-WSR algorithm by giving hot, clean pages another chance to stay in buffer, which improves the hit rate.

LIRS-WSR [5] is another WSR policy for the flash-based system. It is an improvement on the LIRS algorithm with the concept of low inter-reference recency (LIR) and high inter-reference recency (HIR). It tries to replace the pages with high inter-reference recency (HIR) on page fault. For this purpose, it also divides memory into two regions: C_{LIRS} and C_{HIRS}. If the buffer request is for the dirty page, then it is kept in the LIR set; otherwise, it is kept in the HIR set to reduce the number of write operations. It keeps the deeper history of pages as resident and non-resident, which helps in replacement.

A buffer replacement method named AD-LRU [11] is devised for flash-based systems that focuses on lowering write costs while maintaining a high hit ratio. It attempts to include the recency, frequency, and cleanliness of page attributes into the buffer replacement policy. To account for the concept of recency and frequency of page references, AD-LRU uses two LRU queues: the Cold LRU queue and the Hot LRU queue. The Cold LRU queue saves pages that have been cited just once, while the Hot LRU queue keeps track of pages that have been referenced at least twice. These two LRU queues' sizes are dynamically changed in response to variations in reference patterns. A page gets moved to the front of the cold LRU queue when it is first referenced. When a page in the hot LRU queue is chosen as a victim, it is demoted to

the head of the cold LRU queue, and the pages migrate from the cold LRU queue to the head of the hot LRU queue when they are referenced again. The least recently utilized clean page from the cold LRU queue is chosen as a victim during the eviction process. The least recently used clean page in each LRU queue is indicated by a particular pointer called FC (First Clean). The second chance policy is implemented in the event that clean pages are not present in the cold LRU queue.

AALRU [12] is the page replacement policy for SSD-based systems that dynamically senses the workload situation to maintain the hit rate and write count. It uses page-level granularity for data loading and migration and block-level granularity for write back. It uses two buffers for this: the read buffer RB and the write buffer WB. It is mainly used to reduce garbage collection overhead by writing back the dirty pages in a cluster.

CF-ABR [13] is another buffer replacement algorithm designed for flash memory. It creates four lists of referenced pages: L1 contains the pages referenced at first, L2 contains the pages referenced frequently, and H1 and H2 are lists containing the evicted pages from L1 and L2. All the lists are adjusted dynamically according to the reference pattern. During replacement, clean pages from L1 and L2 are selected according to the reference pattern. In the absence of clean pages, dirty pages with zero references will be selected as victims. It utilizes the recency and frequency of pages.

PA-LIRS [14] is the latest algorithm for flash-based systems. It is also an improvement on the LIRS algorithm with the "deep cold detection" policy. It is implemented with the same concept as the LIRS algorithm, which uses LIR stack S and HIR Q. LIR stack S contains all pages, whether they are LIR or HIR, and HIR Q only contains resident HIR pages. It also uses the recency, frequency, and cleanliness properties of pages during replacement. The original LIRS algorithm is designed for disk-based systems and only considers the hit rate. In flash-based systems, reducing the flushing of dirty pages is also important for saving the life of memory. So, PA-LIRS improves the LIRS to adjust it for flash-based systems. During eviction, the algorithm finds a clean page from HIR and, if found, replaces it without checking its deep-cold status. If the page is dirty, its deep cold flag is checked; if it is 0, the page is given a chance to stay in memory, and its deep cold flag is cleared to avoid the excess decrement in hit rate. According to the LIRS, it works well when the size of the C_{HIRS} is 1% of the cache size. In PA-LIRS, it is adjusted according to the workload.

4. Research Methodology

This research work is carried out by following quantitative research methods with the help of trace-driven and simulation approaches. This section discussed details of the research methodology adopted to carry out the research work.

4.1 Dataset Description

In this study, the experiment uses four different forms of synthetic traces: the random trace, the read-most trace, the write-most trace, and the Zipf trace. These are actual memory traces that were produced when actual OS processes were running. Each of the first three traces contains a total of 100,000 page references, which are limited to a set of pages with numbers between 0 and 49,999. In order to produce a good estimate, the total number of page references in the Zipf trace is set to 500,000 but the page numbers still fall within the range [0, 49999]. The specifics of the data sets are displayed in Tables 4.1 to 4.4 [13].

Attributes	**Value**
Total I/O references	100,000
Total Distinct references	43247
Read/Write ratio	50% /50%
Reference Patterns	Uniform

Table 4.1 Random access trace

Attributes	**Value**
Total I/O references	100,000
Total Distinct references	43212
Read/Write ratio	90% /10%
Reference Patterns	Uniform

Table: 4.2 Read-most access trace

Attributes	**Value**
Total I/O references	100,000
Total Distinct references	43182
Read/Write ratio	10% /90%
Reference Patterns	Uniform

Table 4.3 Write-most access trace

Attributes	**Value**
Total I/O references	500,000
Total Distinct references	47023
Read/Write ratio	50% /50%
Reference Locality	20%/80%

Table 4.4 Zipf trace

4.2 Performance Measures

In the case of flash-based systems, a higher hit rate and a lower number of write counts are measures of a better algorithm. Therefore, in this paper, hit ratio and write count are taken as key performance measures.

Hit and Miss Rate

Hit rate (HR) is the rate of finding the page in main memory out of total referenced pages, and miss rate (MR) is the percentage of memory references that are not satisfied from memory and are calculated using Eqs. (1) and 2.

$$MR = \#PF/\#REF * 100 \quad (1)$$

$$HR = 100 - MR \quad (2)$$

Where, HR is the hit rate, MR is the miss rate, #PF is the number of page faults, and #REF is the total number of referenced pages.

Write Counts

The write count is the total number of pages that are propagated to flash memory and is determined by counting the number of memory pages that are written to the flash memory during page replacement.

5. Results and Discussion

Each workload was tested in the LRU-WSR and LIRS-WSR simulators by varying the memory size from 512 to 18192. In the case of the LIRS-WSR algorithm, the size of C_{HIRS} was 1% of the cache size.

Hit Rate Analysis

Figures 5.1 and 5.2 show that the hit rate of LIRS-WSR is better than that of LRU-WSR. In the case of random trace, the hit rate achieved from the LIRS-WSR is 1.06% to 4.78% higher than the LRU-WSR, as demonstrated in figure 5.1. However, in the case of the read-most trace, LIRS-WSR yielded a 5.07% to 14.79% higher hit rate than LRU-WSR, as displayed in figure 5.2. On the other side, there is not much difference between the hit rates of LIRS-WSR and LRU-WSR in the case of write-most trace, as shown in figure 5.3. This is because the workload has a high write locality, and most of the pages are dirty as a result, making it very challenging for the algorithm to discover clean pages to replace. However, LIRS-WSR has a slightly higher hit rate compared to LRU-WSR. In the case of zipf trace, there is a high reference locality, so in this case, the LRU-WSR outperformed the LIRS-WSR. The hit rate gain of LRU-WSR is 13.44% to 42.63% higher than that of LIRS-WSR, as presented in figure 5.4. This is because LRU works well in strong localities, and LRU-WSR is also a variation of LRU with an added WSR policy.

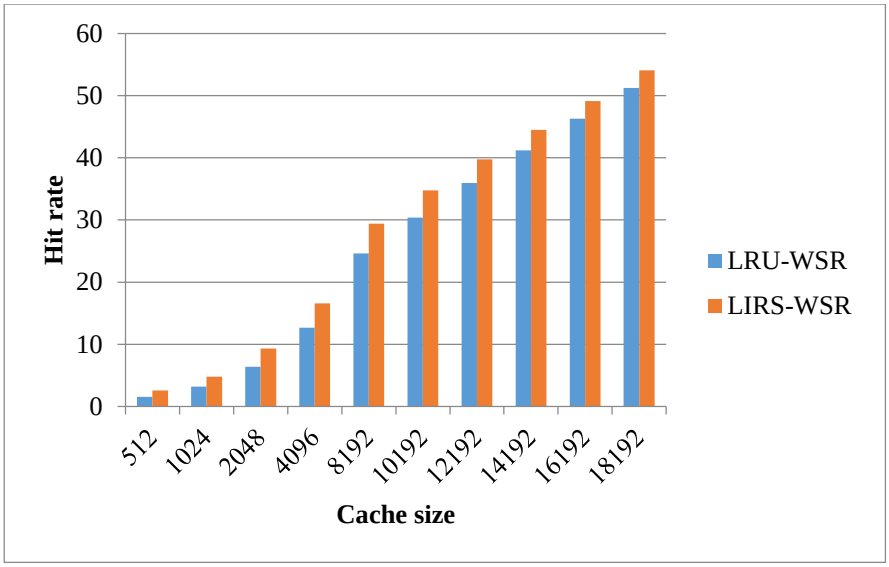

Fig 5.1: Hit rate for random trace

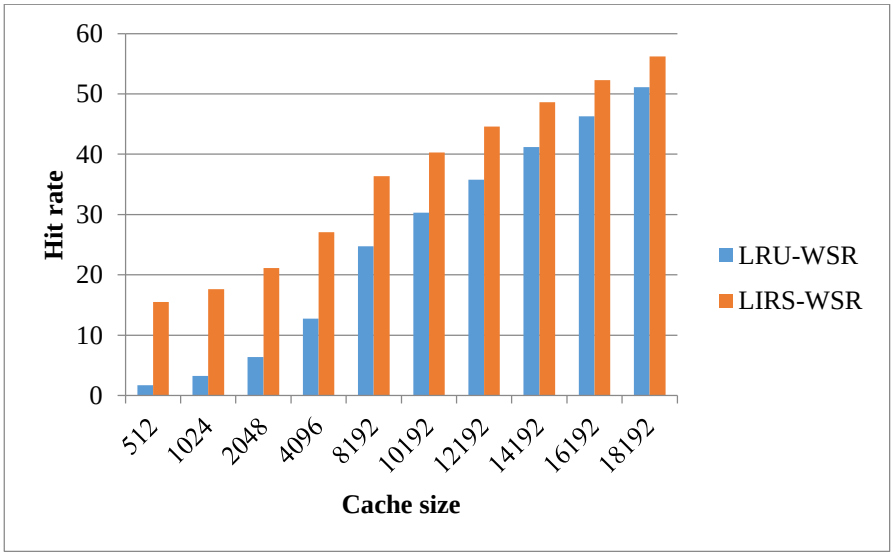

Fig 5.2: Hit rate for read-most trace

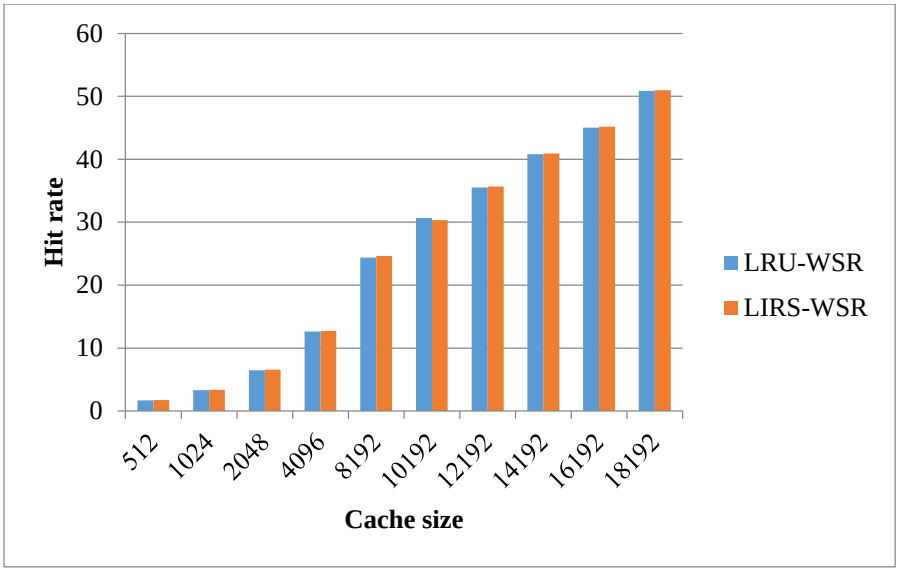

Fig 5.3: Hit rate for write-most

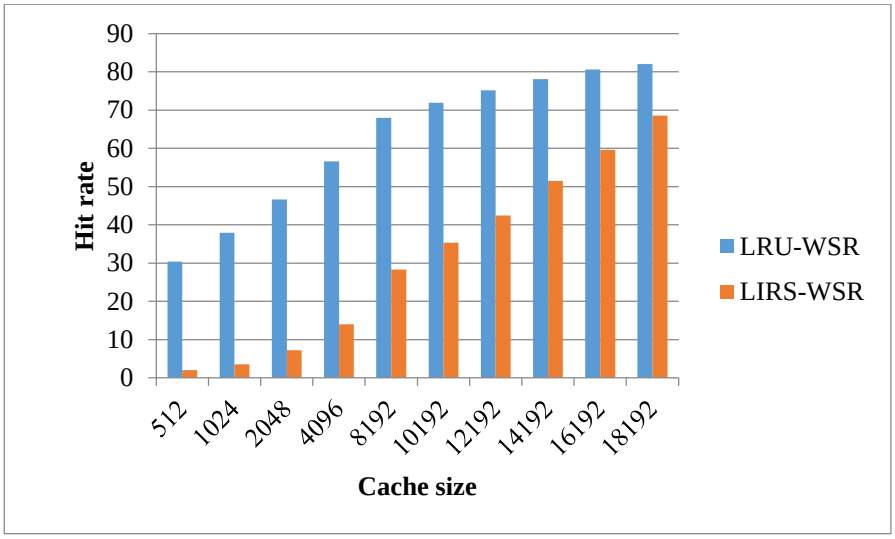

Fig 5.4: Hit rate for

Write Count Analysis

In the case of random, read-most, and write-most traces, the LIRS-WSR achieved a lower write count than that of the LRU-WSR. LIRS-WSR reduces the write count by 3.56% to 45.88%, 56.78% to 79.39.12%, and 0.31% to 6.6%, as displayed in figures 5.5 to 5.7, respectively. The situation is different for Zipf trace. LRU-WSR obtained a lower write count than LIRS-WSR for the trace. LRU-WSR reduces the write count from 26.59% to 49.98%.

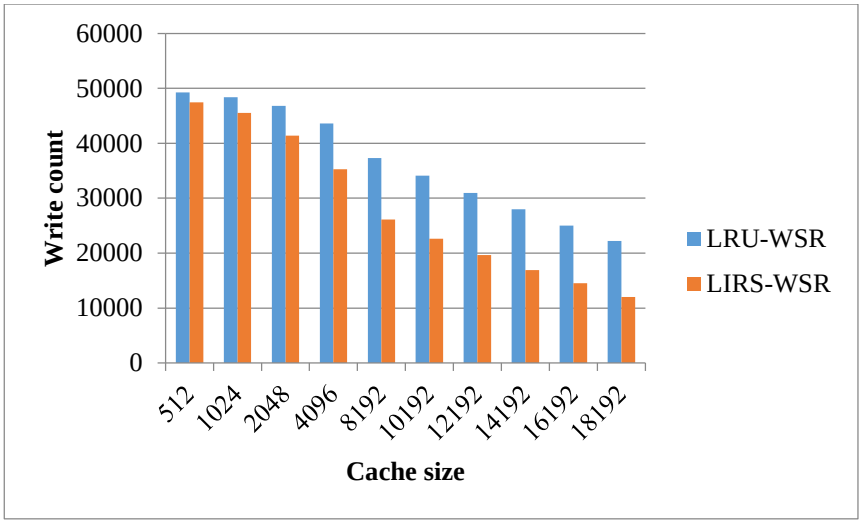

Fig 5.5: write count for random trace

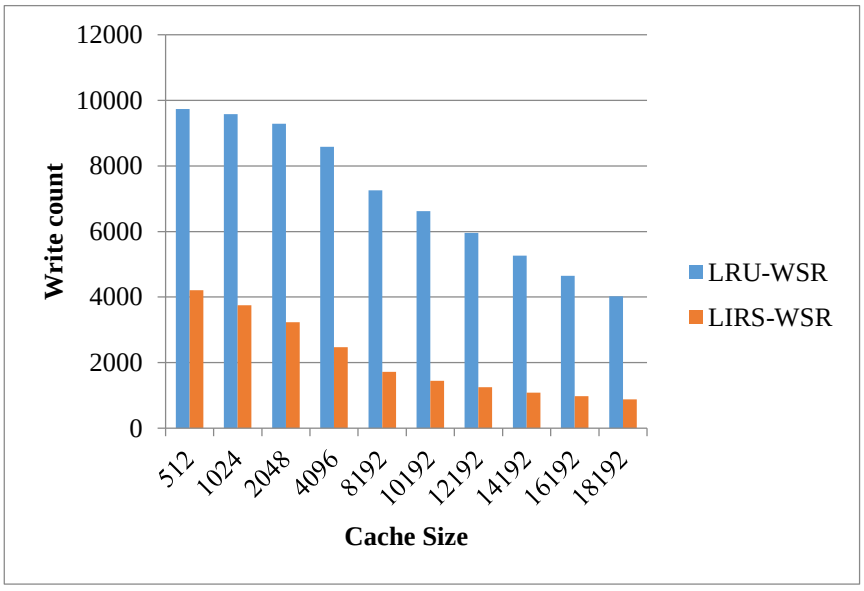

Fig 5.6: write count for read-most trace

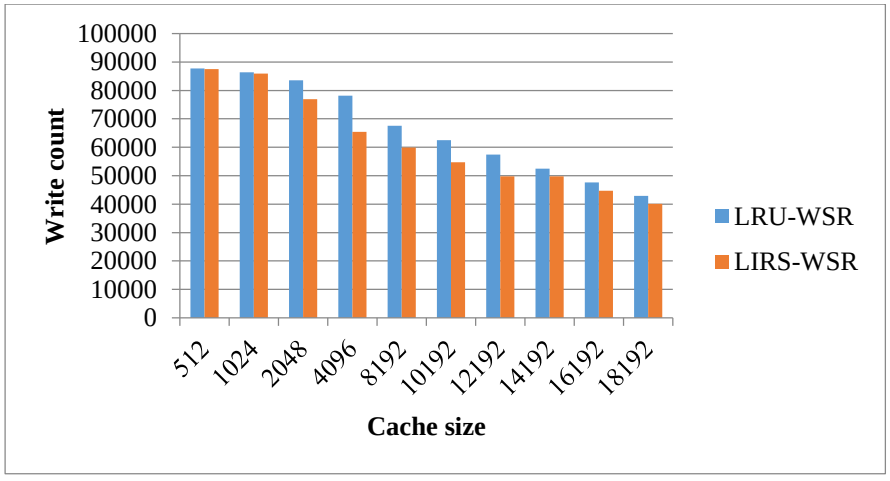

Fig 5.7: write count for write-most trace

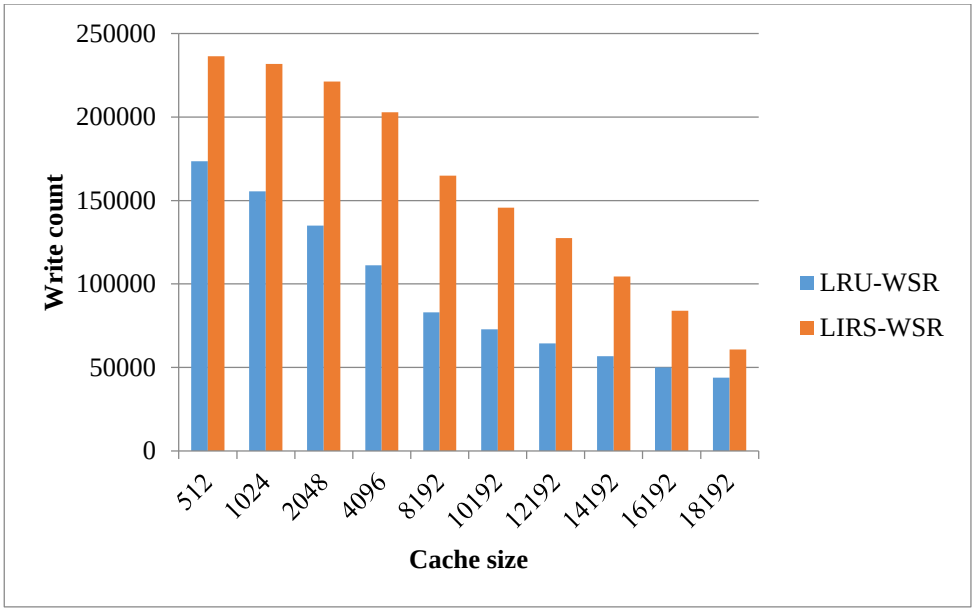

Fig 5.8: write count for zipf trace

6. Conclusion and Recommendations

Traditional page replacement algorithms that have been designed for HDD-based systems are not suitable for flash-based systems. This is because the HDD considers the symmetric nature of I/O. So many page replacement policies have been developed for flash-based systems. This study evaluated the performance of WSR-based page replacement policies with respect to random, read-most, write-most, and zipf traces.

From the experimental results presented in Section 5, we concluded that LISR-WSR policy is suitable for workloads with weak localities of reference, whereas LRU-WSR policy outperforms LIRS-WSR for workloads with strong localities.

In both page replacement policies, the WSR policy is applied to the dirty hot pages only, but this idea can be extended to the clean hot pages by giving them one chance to stay in cache. In the LIRS-WSR algorithm, the size of C_{HIRS} is 1% of the total memory size. This parameter can be adjusted dynamically according to the workload.

7. References

[1] Tal, A. (2003). Two Technologies Compared: NOR vs. NAND White Paper (2003). M-Systems: Flash Disk Pioneers. Retrieved from https://www.maltiel-consulting.com/Nonvolatile_Memory_NOR_vs_NAND.pdf.

[2] Assar, M., Nemazie, S., & Estakhri, P. (1995). Flash memory mass storage architecture incorporation wear leveling technique, Retrieved from https://patentimages.storage.googleapis.com/44/7a/9f/f5050b1288023b/US5479638.pdf.

[3] Bez, R., Camerlenghi, E., Modelli, A., & Visconti, A. (2003). Introduction to flash memory. *Proceedings of the IEEE*, *91*(4), 489-502. DOI: 10.1109/JPROC.2003.811702.

[4] 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. DOI: 10.1109/TCE.2008.4637609

[5] Jung, H., Yoon, K., Shim, H., Park, S., Kang, S., & Cha, J. (2007, August). LIRS-WSR: Integration of LIRS and writes sequence reordering for flash memory. In *Proceedings of the International Conference on Computational Science and Its Applications* (pp. 224-237). Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/978-3-540-74472-6_18.

[6] Park, S. Y., Jung, D., Kang, J. U., Kim, J. S., & Lee, J. (2006, October). CFLRU: a replacement algorithm for flash memory. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems* (pp. 234-241). ACM, Seoul, Korea. DOI: https://doi.org/10.1145/1176760.1176789.

[7] Jiang, S., & Zhang, X. (2002). LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. *ACM SIGMETRICS Performance Evaluation Review*, *30*(1), 31-42. DOI: https://doi.org/10.1145/511399.511340.

[8] Ou, Y., Härder, T., & Jin, P. (2009). CFDC: a flash-aware replacement policy for database buffer management. In *Proceedings of the fifth international workshop on data management on new hardware* (pp. 15-20). Springer, Berlin, Heidelberg. DOI: https://doi.org/10.1007/978-3-642-15576-5_33.

[9] Tanenbaum, A. & Bos, H. (2014). Modern Operating Systems, 4th Edition. Pearson, London, UK.

[10] 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. DOI: 10.1109/TCE.2009.5277999.

[11] 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. DOI: https://doi.org/10.1016/j.datak.2011.09.007.

[12] Yao, Y., Kong, X., Zhou, J., Xu, X., Feng, W., & Liu, Z. (2019). An advanced adaptive least recently used buffer management algorithm for SSD. *IEEE Access*, *7*, 33494-33505. DOI: 10.1109/ACCESS.2019.2904639

[13] 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 *proceedings of the 2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking* (pp. 1217-1223). IEEE, Xiamen, China. DOI: 10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019. 00173.

[14] Wang, F., Jiang, X., Huang, J., & Chen, F. (2020). Pa-LIRS: An adaptive page replacement algorithm for NAND flash memory. *Electronics*, *9*(12), 2172. DOI: https://doi.org/10.3390/electronics9122172.