[1]Shweta

[1]P. K. Singh

Firefly Optimization-Based Buffer Replacement Algorithm to Improve Nand Flash Memory Performance

Abstract: - Because of its shock tolerance, nonvolatility, low power consumption, and fast I/O speed, flash memory is often used for storage in consumer applications and, more recently, in the business computer environment. State drives SSDs to have not-in-place updates and asymmetric I/O latency, with write/erase operations being substantially slower than read operations. In this paper, we propose a buffer replacement technique FBRA that uses the firefly algorithm (FA) to precisely predict whether the pages residing in the buffer is hot or cold. The firefly optimization estimates hot fitness value of each page in the buffer in order to accurately classify them as hot or cold by characterizing precisely the temporal and spatial locality. To increase the hit ratio and SSD buffer use, the frequently used pages should stay longer in buffer compared to other pages. The proposed approach outperforms the existing SSD buffer management strategies regarding buffer hit ratio, write count, and runtime. Trace-driven simulation is done using FlashDB Simulator to support the performance of our approach surpasses various traditional buffer replacement policies.

Keywords: Flash Memory, Firefly Algorithm, Buffer Replacement, Write count, Hit Ratio.

I.INTRODUCTION

NAND flash memory is used in PMPs and PDAs as additional storage (personal digital assistants). NAND flash memory's great density, stress resilience, low power consumption, and fast access time make it appealing [1]. Academic and industrial researchers are increasingly considering flash memory as a next-generation storage medium. Due to technology improvements, the cost per gigabyte of flash memory has dropped by 50% annually, and its capacity is expanding just as quickly. Flash memory is helpful for enterprise-scale server storage. Because of its small size, lightweight, resistance to shock, and low power consumption, NAND flash memory is commonly employed in embedded systems[2][3]. Due to capacity and lower price, flash-memory-based SSDs are replacing magnetic drives in corporate computing. NAND flash memory is a write-once, bulk-erase media structured in blocks with a set number of pages[4]. NAND flash I/O operations vary from the magnetic disc. NAND flash memory has three fundamental operations: read, write, and erase[5]. Read and write are page-based, whereas erase is block-based. The three operations have varying latencies.

Buffer replacement may streamline the I/O sequence and minimize disc access, increasing overall storage system performance. The LRU[6][7] policy evaluates the location of references. For buffer replacement, most storage systems employ LRU or extended LRU algorithms[8]. The read and write performance of flash memory I/O is asymmetric. As a result, the implementation of buffer replacement algorithms in consumer electronics using NAND storage should be reviewed. These algorithms should look at the hit ratio and how to reduce write counts while significantly increasing read counts.

However, several limitations still limit SSD performance. The first is its "erase-before-write" operating technique. An SSD unit often has several blocks,each storing many pages. Page is the unit of write and read, whereas block isthe unit of erase. Rewriting a single page requires erasing the whole block comprising it. Because block erasing takes time, it reduces SSD performance. Thesecond problem is SSD endurance. SSD program/erase cycles are restrictedto about 1,000,000, depending on NAND flash type. A block's use is limited.The concerns have been addressed extensively. FTL[9][10] stands for Flash Translation Layer, and it is a software layer . The FTL maps logical block addresses(LBA) to physical block addresses (PBA) to manage blocks in flash memory. ADRAM buffer cache may also hold the data and mapping table. Using a buffer has several benefits. If the buffer is filled, the victim pages are flashed. There are many SSD buffer management systems. Least recently used (LRU) is themost often utilized technique

[1]Computer Science and Engineering Department, Madan MohanMalaviya University Of Technology, Gorakhpur, 273010, Uttar Pradesh, India.

*Corresponding author(s). E-mail(s): Shweta20989@gmail.com;Contributing authors: topk.singh@gmail.com;

due to its simplicity and decent performance. However, picking victims solely based on residence duration does not always provide the greatest results. Clean first least recently used (CFLRU) is an LRU upgrade that keeps updated pages in a buffer to decrease write operationson SSD storage.

The following are the major contribution of paper

1. We proposes FBRA which is designed to improve the overall performance of memory. The proposed approach uses firefly algorithm to evaluate the fitness value of pages in buffer to enhance system performance efficiency.

2.We use three types of OLTP traces to compare FBRA with LRU, CFLRU, CCFLRU, LRU-WSR, and PTLRU.

3. We evaluate the various algorithms' hit rate and the number of write operations for every trace. Our experimental findings demonstrate that our approach outperforms others as far as overall performance is concerned.

In Section 2, we briefly describe the existing literature on buffer management. The proposed buffer management strategy for flash memory is presented in Section 3; section 4 shows the performance evaluation. Section 5 draws the paper to a conclusion.

II.RELATED WORK

NOR flash and NAND flash are two types of flash memory. The bus interface design differentiates these two kinds. Moreover, NOR memory erase blocks vary from 64 to 128 Kbytes in size, and NAND flash erase blocks vary from 8to 32 Kbytes. The NOR memory is suitable for code preservation and execute-in-place applications, but the NAND flash is suitable for data storage due toits high density.

Flash memory has asymmetric read, write, and erase latencies. Table I[11]analyses the differentiating features in terms of access time and energy usage(Unlike NOR memory, NAND flash spends more time reading and less time writing, resulting in faster erasing. As indicated in Table I, the overhead of a write operation is approximately 200 times that of a read operation in NOR memory and around 10 times in NAND flash.

Table 1. Access Time of NAND and NOR.

Device	Access Time		
	Read	Write	Erase
NAND	20 µs	28 ms	1.2 sec
NOR	25 µs	250 µs	2 ms

Three kinds of operations are available on NAND flash memory. Read and write operations that is performed on the page level and erase operation that takes place in block. Whenever NAND flash memory runs low on free space, the dirty block is erased to make space for the valid pages. Therefore, the granularity for the read and write operation is page, and block for erase operation.

NAND flash memory, Linux, and the flash translation layer (FTL) comprise our NAND flash memory storage system architecture. FTL act like interface between the standard file system and NAND flash memory storage devices. As seen in fig 1 the architecture of flash memory which depicts the functionalities of FTL [14–16], which offers address translation and garbage collector, hides NAND flash memory's fundamental properties and emulates it as a block device.

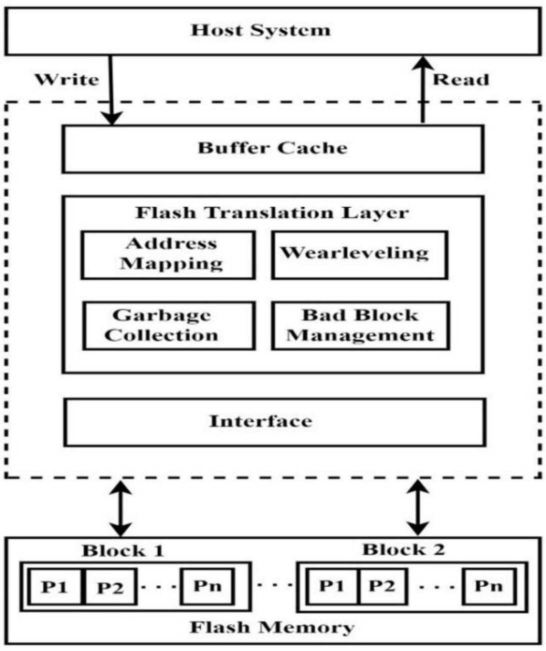


Figure 1 Architecture of flash memory.

I.1 Buffer Replacement Algorithm

Several researchers designed classic buffer replacement techniques to improve the hit ratio and hence I/O speed. Due to the fact that pagereferences reflect temporal locality, many of these works concentrate on their frequency or recency. Algorithms in this category regard write and read costs equally.

Clean first LRU termed as CFLRU[12] was the first algorithm designed for flash memory, for flash memory. CFLRU keeps a page list in LRU order and splitsit into two regions: working and clean-first. Pages inside the working area haverecently been referred and will likely be cited again soon. Pages in the clean- first area have not been mentioned in a long time and are good candidates as victim pages.

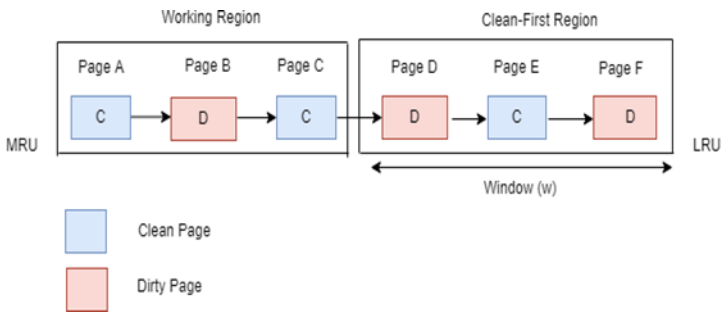


Figure 2 Working of CFLRU

As shown in fig 2. pages are accessed in following order F,E,D,C,B and Aand window size (w) is 3.Even though the least recently page accessed is F but CFLRU selects clean page E as victim. However, it has certain flaws. As aconsequence of cold page cache pollution, CFLRU does not properly analyzethe access frequency of buffered pages. Second, finding an acceptable windowsize for diverse workloads and buffer sizes is not straightforward.

To decrease write requests to flash memory, Jung et al. devised the LRU- WSR buffer replacement technique[13]. Cold detection technique in LRU-WSR is designed to identify cold dirty pages. Each buffer page has a cold flag added. A dirty page that was selected as a target has the cold flag activated.If it isn't set, the page is relocated to the buffer's MRU position, and thena new page is chosen from the LRU. If activated, the page is saved to flash memory as the victim. Whether a page seems clean or not, it willbe victimized. On visit, the buffer list dirty page is transferred to the MRUplace and its cold tag is deleted. As shown in fig 3.there is a flag bit attached to each page and the victim is selected for eviction based on the flag.

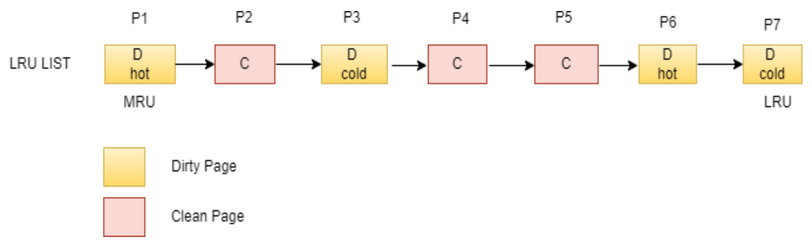


Figure. 3.Working of LRU-WSR

Li et al introduced a novel page replacement technique dubbed CCF-LRU[14]. CCF-LRU maintains both the mixed and cold clean LRU page lists. Cold clean pages are the only ones on the mixed LRU page list. CCF-LRU firstscans the cold clean LRU page list, then evicts the pages in LRU order. Then it scans the mixed LRU page list, identifying the most frequently referred page.It will activate the cold flag and shift the page to MRU inside the mixed list. Assuming it's a cold clean page, it'll be put towards the cold-clean list. A cold dirty page will be chosen as the victim.

PT-LRU [15]manages the buffer using threeLRU lists: cold clean, cold dirty, and mixed. After the first access, the pageis moved to a cold clean list and a mixed LRU list. When no new access page is available, the policy preferably tried to evict the LRU page of cold clean list. Cold dirty pages will be evicted if list is empty. When picking avictim from a mixed list, hot clean pages are evacuated first, followed by hotdirty pages.

III.FIREFLY OPTIMIZATION BUFFER REPLACEMENT ALGORITHM (FBRA)

A. **Structure:** The proposed approach adopts the firefly algorithm to calculate the prominence of buffer pages, which will help to categorize them into cold or hot pages. The frequently used pages are known as hot; otherwise, they are cold. Our buffer management algorithm tries to stay hot pages in buffer to increase the buffer hit ratio. The structure of our proposed approach consists of three lists i.e., working, reserved, and victim.
Working lists are used for indexing and have as many items as buffers. It keeps all requested pages without affecting their referred order. Once the buffer is filled, new requests must be processed. The selected victim page is removed from the buffer to victim list and then it is deleted from working list.
The reserved list keeps in the buffer pages of working list with a high HF value. Reserved list works like LRU. The victim list maintains the pages who have low HF in working pages. The size of victim list and reserved list is half the buffer size respectively. The victim list is a typical LRU list, where the pages having low frequency is flushed. When victim list page access frequency is higher than reserved list, then the pages in victim list are hotter. Firefly algorithm calculates PH fitness for each working list page. Pages with higher PH fitness are reserved, while others are chosen as victim. This study uses the FBRA method to categorize hot and cold logical buffer pages. The suggested technique stores projected hot pages in the buffer to increase hit ratio and buffer use.

Working List (W_L) : the working list consists of an equal number of entries as in a buffer of size N . This list contains N frequently referenced of pages ordered according to the access. When there is no space in the buffer, the space is made for further requests by evicting the victim pages. The victim is chosen from the victim list, deleted from there, and from the working list. For example in fig there are six entries (N=6)where each entry consists of three tuple such as $P_{N:}$ Page Number, AF: access frequency and HF_v: Hot fitness value. The most frequently used page is P_4 whose HF_v is 18.5.

Reserved List (R_L): The reserved list contains pages with the half-highest HF_v on the working-list .

Victim List (V_L): The victim-list stores the working-half-lowest list's HF values' logical pages.

The structure of proposed approach has been depicted in fig 4.

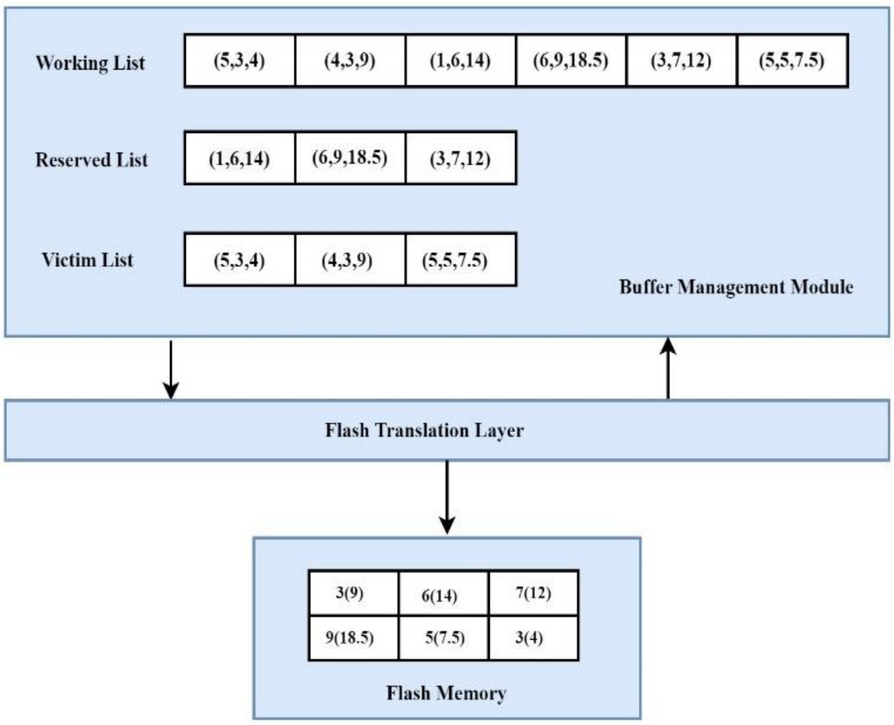


Figure 4.Structure of Proposed Approach

The pages in the victim list are hotter than those in the reserved list when the access frequency of the victim list exceeds that of the reserved list. The new HF_V value of each page in the working-list is then calculated using the Firefly algorithm. After then, as per changes reserved-list and victim-list are updated.

B. **Hot Fitness value (HF_V):** The primary goal of the proposed approach is enhancing the buffer management policies by maximizing the number of buffer hit ratio, which is done with the help of HF_V of pages with firefly algorithm.

Suppose there are N buffer pages, the HF_V of page is based on AF of page having a maximum frequency, its own AF, and the distance from the page with the maximum frequency in the buffer to it. This is how spatial and temporal locality is perceived. For instance, in Figure 5, the ith element in the working list is indicated as WLi. In WLp, page number 6 is identified as having AF=10. The pages maintained in the working list are in access order sequence; hence the adjacent pages will likely be associated with the request. In fig W_{Lp} is the most frequently referenced page.Pages close to it will be accessible with WLp since working list pages are kept in the order of access.The probability of W_{Li} access ($i \neq p$) is find with the help of distance from $W_{Lp.}$

For example, in fig.5. $W_{L(p-1)}$ is closer to W_{Lp} than W_{L2} therefore $W_{L(p-1)}$ has better chance than W_{L2} to be accessed in future together with $W_{Lp.}$

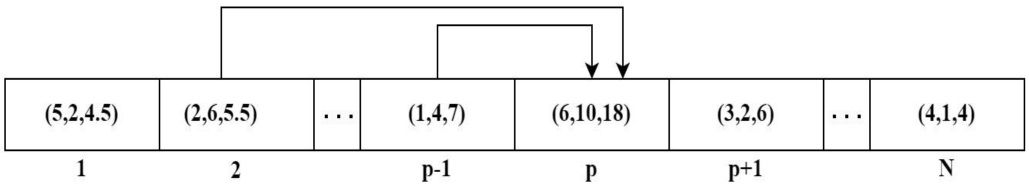


Figure.5. Demonstration of Working List.

Growth rate of AF of page in working list also affected by its own AF. In other words, a page will likely be viewed again in the near future if it is now being accessed often, which is referred to as temporal locality. For example if temporal locality is taken into consideration, since AF of W_{L2} is greater than $W_{L(p-1)}$, the probability of W_{L2} is higher than $W_{L(p-1)}$ to be accessed in future.

C. Firefly Algorithm (FA): Winged beetles or insects called fireflies emit light and flicker during the night. The term "bioluminescence" refers to a chemically produced light that emanates from the lower abdomen but is neither infrared nor ultraviolet in frequency. To attract possible mates or prey, they especially use the flashlight. Additionally, the flashing light serves as an indicator system to alert the firefly to potential predators.The Firefly Algorithm (FA), a metaheuristic algorithm inspired by the bioluminescent communication phenomenon and the flashing feature of fireflies, was created by Yang [16].The Firefly Algorithm was developed under the following presumptions:

1. A firefly is unisexual
2. The more sparkling firefly will gravitate towards the less sparkling firefly since attractiveness is inversely correlated with brightness. But the attraction is diminished when the distance between the two fireflies grew.
3. When both fireflies are bright, they will travel randomly. New solutions are generated by random walk and firefly attraction [17][18]. The brightness of the firefly should be connected to the problem's relevant goal function. They can divide into smaller groups due to their allure, and each group converges around the local models. According to [19][20], Firefly Algorithm is appropriate for optimization situations.

Based on the findings of earlier research, many researchers have indicated that FA, invented by Yang in 2008, is a highly strong approach to handle limited optimization issues and NP-hard problems. FA [21] has been used extensively to solve continuous mathematical functions. The algorithm must only use basic arithmetic and reasoning in applied mathematics. Since FA's behavior is straightforward, it may be used to solve continuous mathematical functions.

FA is efficient and outperforms traditional techniques based on ordinary stochastic test functions. The program relies on the firefly's worldwide communication to function. It may, therefore, concurrently discover both the global and local optimum. FA primarily employs genuine random numbers. Fireflies operate independently of one another, making parallel implementation possible. FA is one of the methods lately utilized by academics to find solutions to optimization issues in dynamic environments.

Using the firefly algorithm generally involves the following steps:

1. Initialize the parameters of the firefly algorithm, the population of fireflies, and the number of iterations.

2. Examine each firefly's fitness function.

3. Use the initial fitness value to determine the early light intensity level.

4. Use the equation of movement to update each firefly's movement.

5. To get the optimal firefly value, compare each top firefly candidate to the fitness function's value.

6. Perform iterations up to their maximum or until a firefly with a satisfactory fitness function exists.

The attractiveness of the firefly is

$$E_i = E_{max} e^{-Tf_i} \tag{1}$$

Where, E_i is expected access frequency, E_{max} is maximum access frequency, T is the Temporal locality factor and f_i is access frequency of D_i.

$$\Psi = \Psi_o e^{-Tf_i} \tag{2}$$

Where, Ψ is personal learning and Ψ_o is weight of personal learning when $f_i = 0$

$$R_i^{t+1} = R_i^t + \Psi(E_i - f_i) + \Phi \frac{1}{|H - i|} \tag{3}$$

Where Φ is social learning, H is position of page of largest frequency and R_i^{t+1} is location of fireflies i.

The HF$_V$ value is calculated using equation

$$HFv_i = HF_i + R_i^{t+1} \tag{4}$$

In our proposed approach, the following algorithms 1 and 2 are for buffer write and the victim selection procedure.

Algorithm 1 Write management in buffer

```
Initialization-
$L_P$ : Logical Page, P: Page, $B_{SSD}$ :SSD_Buffer, $RL_P$ :Requested Logic Page
$R_L$ : Reserved List, $W_F$ : Write Frequency, TOP: Header, $V_L$ :Victim List
$F_{SSD}$ : SSD_Flash, $V_P$ :Victim Page, $W_L$ :Working List

If($L_P \in B_{SSD}$ )
            Search $RL_P$ in $W_L$
            If ($L_P \in P_L$)
              {
                $M_{Selectp}$ ();
                $W_F$++;
                 Keep $L_P$ on the TOP of $P_{L;}$
              }
            Else( $L_P \in V_{L)}$
              {
              $M_{Selectp}$ ();
              $W_F$++;
               Keep $L_P$ on the TOP of $P_{L;}$
              }
Else( $L_P \notin B_{SSD}$ )
           If ($L_P \in F_{SSD}$)
              {
               $V_P$ = $M_{Select\ Vp}$ ();
                Flush $V_P \rightarrow F_{SSD;}$
                Load $L_P$ from $F_{SSD;}$
                Keep $L_P$ on the TOP of $V_L$ ;
               }
            Else($L_P \notin F_{SSD}$ )
                {
                  If ($W_L$! = FULL)
                     {
                       If ($R_L$! = FULL)
                       Keep $L_P$ on the TOP of $R_L$ ;
                       Else ($V_L$! = FULL)
                       Keep $L_P$ on the TOP of $V_L$ ;
                     }
                  Else ($W_L$ == FULL)
                     {
                       $V_P$ = $M_{Select\ Vp}$ ();
                        Flush $V_P \rightarrow F_{SSD}$ ;
                        Keep $L_P$ on the TOP of $V_L$ ;
                       }
}
```

Algorithm 2 Victim selection algorithm

```
Initialization-
$L_P$ : Logical Page, N: Buffer Size, ϒ= Predicted Hot Fitness value, $f_c$ : frequency count
$S\_R_L$ : Sum of $f_c$ of pages in $R_{L,}$ $S\_V_L$ : Sum of $f_c$ of pages in $V_L$
$V_P$: Victim Page.

If ($S\_R_L$ >$S\_V_L$)
```

```
     {
        Goto tail of $V_L$;
        $M_{Select\ Vp}$ ();
     }
Else
     {
        Perform FFA in $W_L$ ;
        Maintain 2/L $L_P$ with large ϒ value in the $R_L$ ;
        Maintain remaining 2/L $L_P$ in the $V_L$ ;
        Goto tail of $V_L$;
        $M_{Select\ Vp}$ ();
     }
```

IV.PERFORMANCE EVALUATION:

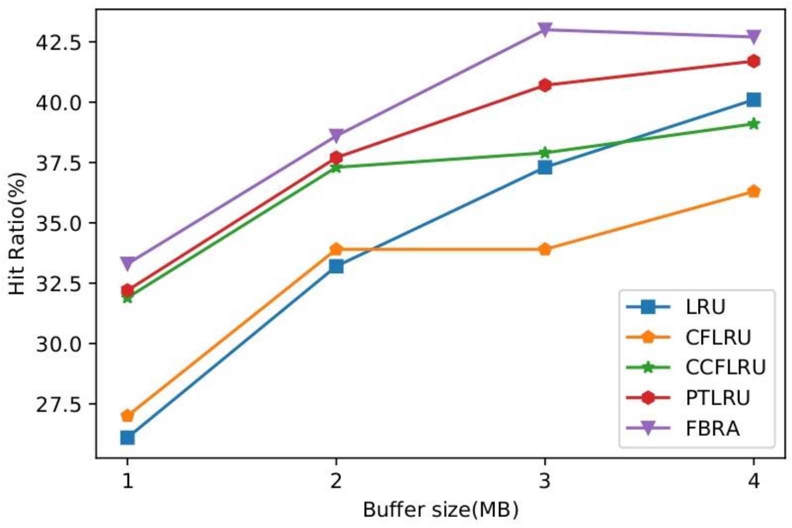
The experiments in this section compare FBRA against various policies, such as LRU, CCFLRU, CFLRU, LRU-WSR, and PTLRU, regarding write count and buffer hit ratio. Performance metrics are obtained by executing synthetic traces along with real workloads collected from corporate systems. Three types of traces are utilized to compare buffer management techniques. The Storage Performance Council made them accessible from an OLTP storage institution (SPC)[22][23].

Flash-DBSim[24], a configurable simulation environment, is used to conduct the simulation tests. For investigating flash-based buffer replacement techniques, Flash- DBSim is employed since it is reusable, adjustable, and versatile. A block can only be erased 100,000 times.

An "x%/y%" locality shows that x% of all references are conducted actively in y% of all pages. Table 2. shows that x percent of all references are extensively completed in y percent of all pages.

Table 2. Three types of OLTP Traces

Traces	No. of Request	R/W Ratio	Locality
T1	3,000,000	30%/70%	70%/30%
T2	3,000,000	50%/50%	50%/50%
T3	3,000,000	10%/90%	80%/20%



.

Figure. 6 Hit Ratio for trace T1

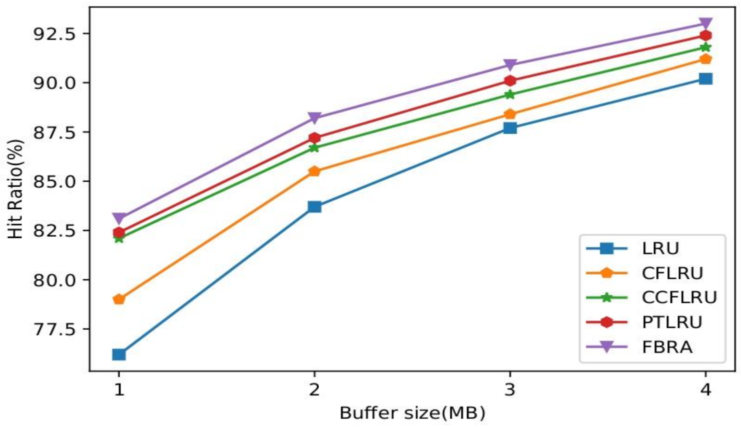


Figure. 7 Hit Ratio for trace T2.

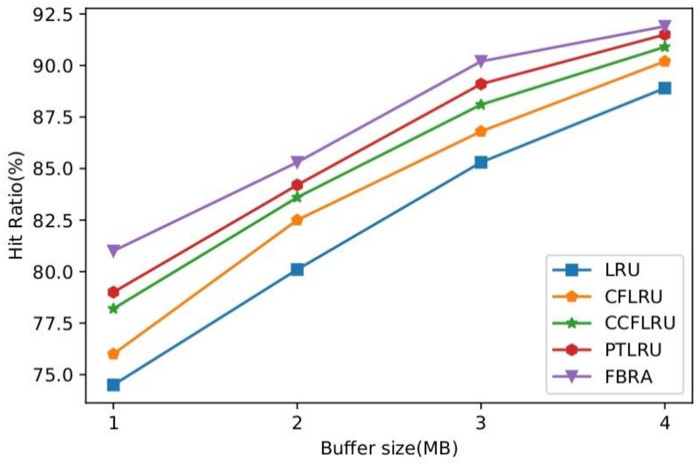


Figure. 8 Hit Ratio for trace T3.

A comparison of buffer replacement techniques is shown in Fig 6,7 and 8. In most circumstances, FBRA outperforms other strategies in terms of hit ratio. As FBRA considers both spatial and temporal locality, it efficiently evict pages that are not expected to be accessed subsequently as well as pages with low frequency and situated near page with the highest frequency. The suggested system considers temporal and spatial localization of pages, which increases the hit ratio. The nature of the workloads has a significant amount of control over the buffer hit ratio.

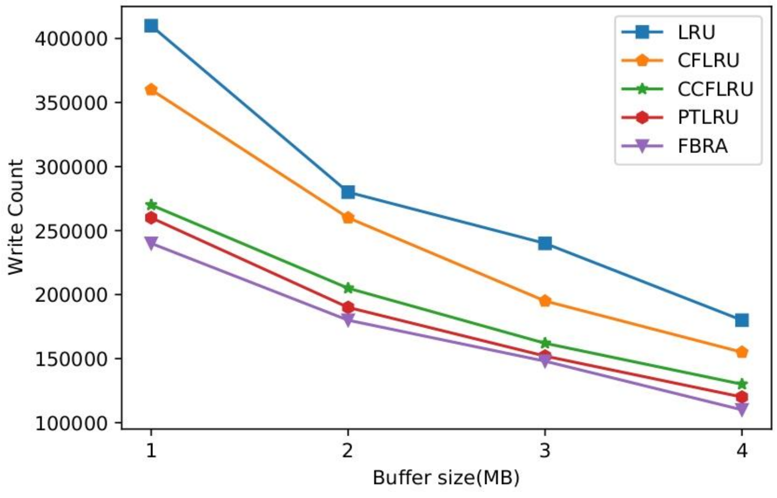


Figure. 9 Write count for trace T1

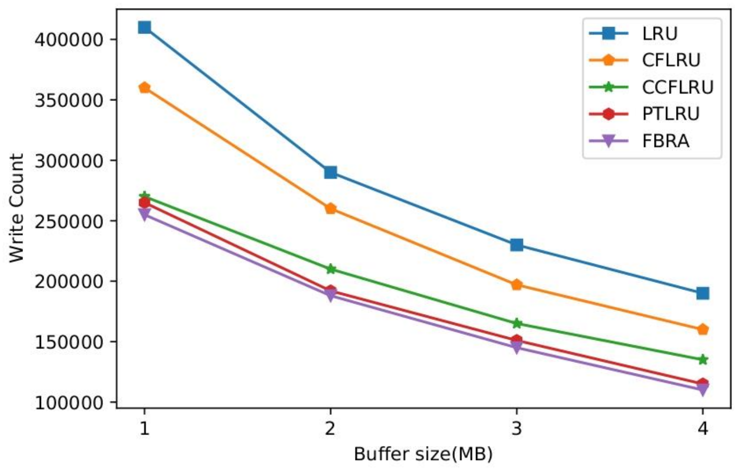


Figure. 10 Write count for trace T2.

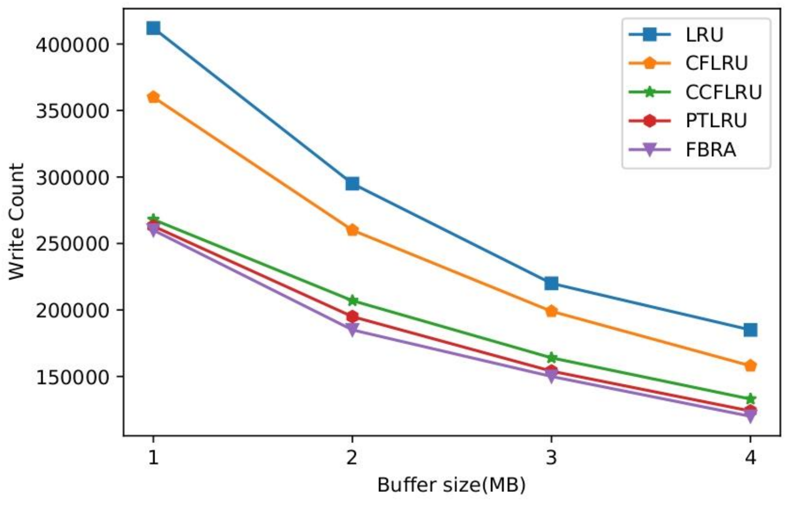


Figure. 11 Write count for trace T3.

Fig 9,10 and 11 shows the experimental results of FBRA and other existing buffer replacement algorithms in context of number of write operation performed. The write count is calculated using number of writes occurred during page replacement. To extend the life of SSD number of write operations must be decreased. The suggested approach consistently decreases the write count by taking into account the HF_v of the page.

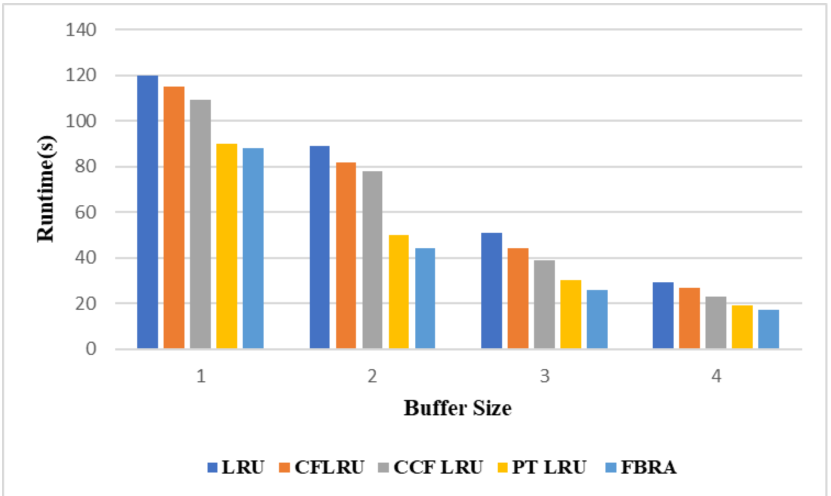


Figure. 12 Runtime for trace T1.

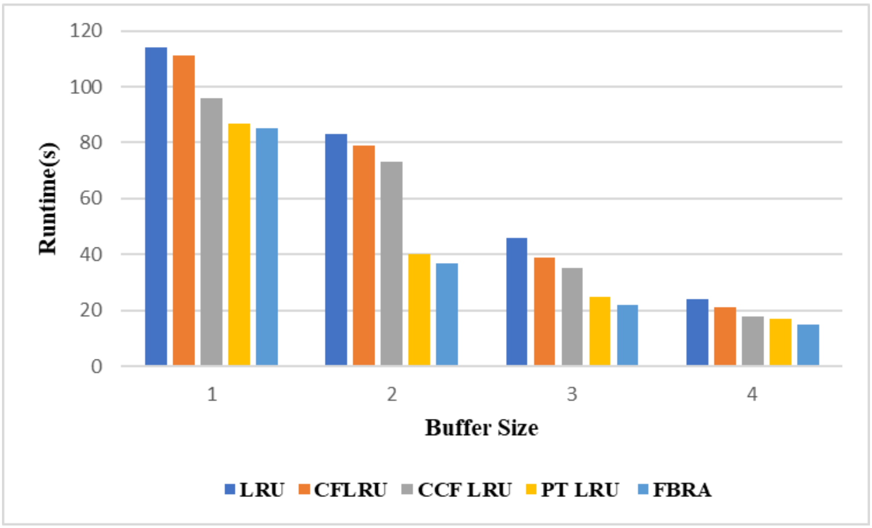


Figure. 13 Runtime for trace T2.

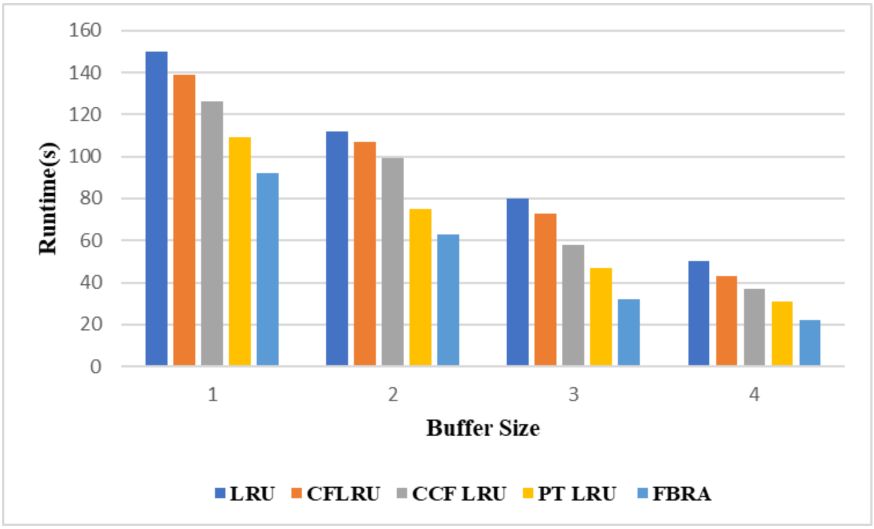


Figure. 14 Runtime for trace T3.

Figure 12,13 and 14 displays algorithm runtimes of five buffer replacement policies for various buffer sizes in which FBRA has minimal runtime. The graph above shows that FBRA takes less time than others existing approaches because of lower write count. Due to the asymmetric I/O latencies, the variations in the number of write operations to flash memory are not especially significant. As a result, lowering the number of write operation is the main goal of the buffer replacement techniques.

Table 3. shows the comparative analysis of FBRA and other existing approaches by comparing the median of hit ratio and write count. For given traces our approach FBRA outperforms better than other policies.

Table 3. Comparison table of improvement in FBRA.

Traces	% of improvement in FBRA (hit ratio)				% of reduction in FBRA (write count)			
	LRU	CFLRU	CCFLRU	PTLRU	LRU	CFLRU	CCFLRU	PTLRU
T1	18.7	15.1	5.5	2.7	39.2	34.6	22.7	19.0
T2	6.0	3.5	2.3	1.1	37.5	25.9	13.0	4.7
T3	8.9	4.9	4.2	3.6	38.7	26.9	9.5	5.0

V.CONCLUSION

Write operations take more access time than read operations in NAND flash-based storage devices, while erase operations are substantially lower than write operations. Preferably, the buffer replacement methods must be developed to result in fewer of write operations in order to obtain a high degree of performance. The proper management of buffer improves the efficiency system performance. To tackle the shortcomings of current buffer replacement algorithms, we presented FBRA which employs the firefly algorithm to predict the fitness value of pages in buffer. This fitness value helps to categorize buffer pages into hot or cold. The hot pages are kept into buffer to increase the hit ratio this directly effects the performance of system. Trace driven simulation presents that FBRA surpasses existing approaches in context of hit ratio, write count and runtime.

REFERENCES

[1] Y. Wei, D. Shin, Nand flash storage device performance in linux file system, in: 2011 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT), IEEE, 2011, pp. 574–577.

[2] L. Wu, N. Xiao, F. Liu, Y. Du, S. Li, Y. Ou, Dysource: A high performance and scalable nand flash controller architecture based on source synchronous interface, in: Proceedings of the 12th ACM International Conference on Computing Frontiers, 2015, pp. 1–8.

[3] H. Sun, G. Chen, J. Huang, X. Qin, W. Shi, Calmwpc: A buffer management to calm down write performance cliff for nand flash-based storage systems, Future generation computer systems 90 (2019) 461–476.

[4] S. Nie, Y. Zhang, W. Wu, C. Zhang, J. Yang, Dir: Dynamic request interleaving for improving the read performance of aged ssds, in: 2019 IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA),IEEE, 2019, pp. 1–6.

[5] J. Kwak, J. Lee, D. Lee, J. Jeong, G. Lee, J. Choi, Y. H. Song, Galru: A group-aware buffer management scheme for flash storage systems, IEEE Access 8 (2020) 185360–185372.

[6] R. Kumaar, A. Sharma, M. Bhaskar, Reference table based cache design using lru replacement algorithm for last level cache, in: 2016 IEEE Region 10 Conference (TENCON), IEEE, 2016, pp. 2219–2223.

[7] J. O'neil, P. E. O'neil, G. Weikum, The lru-k page replacement algorithm for database disk buffering, Acm Sigmod Record 22 (2) (1993) 297–306.

[8] Y.Wang, Y. Yang, C. Han, L. Ye, Y. Ke, Q.Wang, Lr-lru: A pacs-oriented intelligent cache replacement policy, IEEE Access 7 (2019) 58073–58084.

[9] P. K. Singh, et al., Flash translation layer and its functionalities, in: 2019 IEEE Conference on Information and Communication Technology, IEEE, 2019, pp. 1–5.

[10] Q. Xia, W. Xiao, High-performance and endurable cache management for flash-based read caching, IEEE Transactions on Parallel and Distributed Systems 27 (12) (2016) 3518–3531.

[11] S. Elec, Nand flash memory & smart media data book (2010).

[12] S.-y. Park, D. Jung, J.-u. Kang, J.-s. Kim, J. Lee, Cflru: a replacement algorithm for flash memory, in: Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems, 2006, pp. 234–241.

[13] H. Jung, H. Shim, S. Park, S. Kang, J. Cha, Lru-wsr: integration of lru and writes sequence reordering for flash memory, IEEE Transactions on Consumer Electronics 54 (3) (2008) 1215–1223.

[14] Z. Li, P. Jin, X. Su, K. Cui, L. Yue, Ccf-lru: A new buffer replacement algorithm for flash memory, IEEE Transactions on Consumer Electronics 55 (3) (2009) 1351–1359.

[15] J. Cui,W.Wu, Y.Wang, Z. Duan, Pt-lru: a probabilistic page replacement algorithm for nand flash-based consumer electronics, IEEE Transactions on Consumer Electronics 60 (4) (2014) 614–622.

[16] Yang, X. S. (2008). *Nature-Inspired Metaheuristic Algorithms*. Frome: Luniver Press. ISBN 1-905986-10-6.

[17] Xin She Yang. (2011). Optimization Algorithms. *Comput. Optimization, Methods and Algorithms, SCI 356*. pp. 13–31.

[18] Yang, X.S. (2011). Metaheuristic and Optimization: Algorithm Analysis and Open Problems.*Lecture Notes in National Physical Laboratory, UK.*

[19] Yang, X. S. (2010), Firefly Algorithm, Stochastic Test Functions and Design Optimization. *Int.*

[20] *J. Bio-Inspired Computation*. 2, No. 2, pp.78–84.

[21] Yang, X. S. (2010). Nature-Inspired Metaheuristic Algorithms. *Luniver Press.* Second Edition.

[22] Lukasik, S. and Zak, S. (2009). "Firefly Algorithm for continuous constrained optimization Tasks", *Lecture Notes in Computer Science*. Vol. 5796, pp. 97-106.

[23] Umass trace repository, http://traces.cs.umass.edu/.

[24] Storage networking industry association, https://www.snia.org/.

[25] X. Su, P. Jin, X. Xiang, K. Cui, L. Yue, Flash-dbsim: a simulation tool for evaluating flash-based database algorithms, in: 2009 2nd IEEE International Conference on Computer Science and Information Technology IEEE, 2009, pp. 185–189.

[26] Narayan, Vipul, et al. "7 Extracting business methodology: using artificial intelligence-based method." Semantic Intelligent Computing and Applications 16 (2023): 123.

[27] Narayan, Vipul, et al. "A Comprehensive Review of Various Approach for Medical Image Segmentation and Disease Prediction." Wireless Personal Communications 132.3 (2023): 1819-1848.

[28] Mall, Pawan Kumar, et al. "Rank Based Two Stage Semi-Supervised Deep Learning Model for X-Ray Images Classification: AN APPROACH TOWARD TAGGING UNLABELED MEDICAL DATASET." Journal of Scientific & Industrial Research (JSIR) 82.08 (2023): 818-830.

[29] Mall, Pawan Kumar, et al. "FuzzyNet-Based Modelling Smart Traffic System in Smart Cities Using Deep Learning Models." Handbook of Research on Data-Driven Mathematical Modeling in Smart Cities. IGI Global, 2023. 76-95.

[30] Saxena, Aditya, et al. "Comparative Analysis Of AI Regression And Classification Models For Predicting House Damages İn Nepal: Proposed Architectures And Techniques." Journal of Pharmaceutical Negative Results (2022): 6203-6215.

[31] Kumar, Vaibhav, et al. "A Machine Learning Approach For Predicting Onset And Progression"“Towards Early Detection Of Chronic Diseases “." Journal of Pharmaceutical Negative Results (2022): 6195-6202.