Transactions on Soft Computing

www.tsc.reapress.com

Trans. Soft. Comput. Vol. 1, No. 1 (2025) 1–13.

Paper Type: Original Article

Priority Cache Object Replacement by Using LRU, LFU and FIFO algorithms to Improve Cache Memory Hit Ratio

Davood Akbari Bengar[*]

Department of Computer Engineering, Savadkooh Branch, Islamic Azad University, Savadkooh, Iran; akbari@iausk.ac.ir.

Citation:

Received: 09 January 2025 *Revised: 12 January 2025* *Accepted: 22 January 2025*	Akbari Bengar, D. (2025). Priority cache object replacement by using LRU, LFU and FIFO algorithms to improve cache memory hit ratio. *Transactions on Soft Computing, 1*(1), 1-13.

Abstract

Due to the increasing performance between CPU and cache memory, the use of replacement algorithms is very important in various aspects of high-performance computing environments such as e-commerce systems, cache memory management in microprocessors, object management in operating systems, and iteration strategies in information distribution systems and etc. Database server cache performance is an important issue in e-commerce systems. Managing the entry and exit of cache objects with replacement algorithms can reduce the workload of the database server and improve server performance. The algorithms determine which objects remain in the cache memory and which ones go out to make room for new objects. In this way, the algorithms not only decrease user access time, but also enhance the performance of the system. Most of these algorithms are developed by the famous LRU and LFU schemes and can fix their flaws; but unlike them, they are difficult to implement. This research proposes a priority object replacement algorithm that is easy to implement. The algorithm, called Priority Cache Object Replacement Algorithm (PCORA), is based on prioritizing cache memory objects according to three parameters. Experiments show that the proposed algorithm has a better hit ratio than other algorithms and can effectively improve cache memory performance.

Keywords: Operating systems, Cache memory management, E-commerce systems, Replacement algorithms, Hit ratio, performance.

1 | Introduction

The rapid rise of the World Wide Web has led to an increase in the visibility of web traffic and delays in its access. One of the areas affected by this problem is database servers in systems with e-commerce applications. Database server cache performance is an essential issue in e-commerce systems. Managing the entry and exit

Corresponding Author: akbari@iausk.ac.ir

10.48314/tsc.v1i1.33

of cache objects can reduce the workload of the database server and improve server performance. One of the most effective techniques proposed to solve this problem is web caching. In order to effectively enhance the performance of the cache memory, two types of methods have been proposed: One is to increase the size of the cache memory, which is very expensive. The second method is to design a useful cache memory replacement algorithm. Caching is a popular performance optimization method that is widely used in web storage. If there is not enough cache memory, the load on the servers will increase as the number of users and their requests increases. The insufficient cache memory can cause network traffic problems and network delay [1].

For cache memory, there are three different patterns of web cache memory: Client-side cache memory, server-side cache memory, and proxy cache memory. Client-side cache memory refers to cache memories on the user side that store web page addresses and other information to use to access servers. Server-side cache memory refers to creating a cache memory on the web server side. It aims to reduce the number of server requests, which can reduce server load. Proxy cache memory usually acts as the interface between the user's servers and the central server. When a user sends a request to a central server via a proxy server, the server sends the data to the user according to the main request path.

During this procedure, the proxy server decides whether or not to save a copy in its cache memory, as this data may be requested in the future. Therefore, using an alternative algorithm for server-side cache memory and proxy server to reduce network delay is more important. Many factors affect the efficiency of cache memory, the most important of which are the frequency of access, the last access time, and the size of the cache memory object. For each factor, researchers have proposed appropriate algorithms. When the cache is full, they all have their own algorithm to choose which object to delete. The cache memory replacement policies are generally aimed at enhancing the cache memory hit rate. Hit rate is a standard for measuring the performance of web cache memory replacement algorithms. The higher the hit rate, the better the replacement algorithm [2].

In this research, a cache memory replacement policy with a high hit rate and low delay, which is easy to implement, is proposed to improve cache memory performance. This algorithm, named Priority Cache Object Replacement Algorithm (PCORA), is based on prioritizing cache memory objects according to three parameters. PCORA, which has a priority model, takes advantage of existing cache memory replacement algorithms, which makes the proposed algorithm more accurate and reliable.

The rest of the paper is organized as follows. In Section 2, a review of the past literature is done, and the strengths and weaknesses of each of them are described. Section 3 describes each part of the PCORA algorithm, including the data and its main and sub-functions, separately. In Section 4, the performance of the proposed algorithm is evaluated in terms of hit ratio and compared with other algorithms. Section 5 concludes the proposed algorithm and suggests future research.

2 | Literature Review

In this section, famous algorithms affecting systems and other existing algorithms are reviewed, and their advantages and disadvantages are stated.

When the cache memory is full and a new object needs to be stored, a replacement algorithm should be used to select which object must be deleted to make room for the new object [3]. Cache memory replacement algorithms select which data can stay in the cache memory. An effective cache memory replacement algorithm has fewer errors and can also improve cache memory hit and byte rates [4]. Many academic and industry researchers are trying to find an optimal cache memory replacement policy. According to *Table 1*, conventional cache memory replacement algorithms are mainly divided into five categories [2], [5]:

Table 1. Five conventional cache memory replacement algorithms.

Algorithm	Brief description	Advantages	Disadvantages
RAND	A random number generator to identify an alternative object	It is the simplest algorithm and easy to implement	It is not considered a factor It has an unstable performance Its hit rate is low
LRU	The least-used items are deleted first	It is easy to implement and has a low hit rate	No factor other than the time factor is considered It contains the cache contamination
LFU	Objects of the least frequency are removed first	It prevents cache memory contamination	Only the frequency factor is considered, and other factors are ignored It is difficult to implement
SIZE	Large objects are removed first.	It is easy to implement, preserves small objects first, and has a high cache hit rate.	It first stores small web objects, even if they are not re-accessed It has a low byte hit rate
GDSF	Frequency and launch factors are combined, and the age factor is produced like the time factor.	It covers the weakness of the size algorithm by deleting objects that are no longer accessible to users.	Its computational cost is low, and it has a complex parameter setting It has a low byte hit rate

In the recent LRU algorithm, time is the main factor. The basic idea of this algorithm is that if data is recently requested, it will be more likely to be requested in the future. When the cache memory is full, it deletes less recently mentioned objects. The advantage of this model is that it has a good performance and is very easy to implement on the client side. But this algorithm only considers the last visit and does not care about the number of visits to an object [1], [2]. In the LFU algorithm, researchers consider popularity or the number of visits to the object as the main factor. The basic idea is that the more data you refer to, the more likely you are to refer to it in the future. It is commonly used to store web URLs, and when the cache memory is full, it removes the web object with the least number of visits. The advantage is that it is also easy to implement and works well on the proxy side [6], [7].

In a weight-based algorithm, researchers use weights to decide which object should be cleared from the cache memory. Weight is classified in descending order. Users can directly remove the object with the biggest weight [8]. FPRA is a modification of popular alternative algorithms such as LRU and LFU. This algorithm uses FCM to cluster pages based on three parameters. Clusters of higher freshness, higher frequency, and lower referral rate have higher priority [9]. Clustering allows pages that are more similar to each other to be in the same group [10].

Arya et al. [11] have come up with a new concept for page replacement that is based on reading page blocks from secondary memory. Whenever there is an error on the page, instead of reading only one missing page, the property of the pages equal to the number of frames allocated to that process is restored. In this way, the number of page errors is minimized, which also increases the hit rate.

In [12], a cache memory replacement policy based on the FHPA algorithm is proposed, which restricts the full use of device space with an edge. Considering the heat of the file, the possibility of re-accessing the file cache is evaluated. The cached file with the least chance of being re-accessed will be removed from the cache memory. In [13], a new cache memory replacement policy is proposed to enhance the last-level cache memory effectiveness of embedded processors. Unlike LRU, this algorithm uses the correlation of the distances among the tags in between the cache memory lines to improve accuracy.

Shin et al. [14] presented a new cache memory management scheme specifically for rendering systems. Unlike public purpose computing systems, rendering systems display specific patterns of file access that result in significant disruption to the performance of the buffer cache memory system. To deal with this situation, different input and output sequences of rendered files are collected, and their access patterns are analyzed. The WRP algorithm has two major problems. The first problem of this algorithm occurs when the result of dividing two factors (Novelty and frequency) for one block is equal to another block. The second problem is that it only considers the time interval between the last two accesses and does not consider the previous accesses.

In [15], these two problems and other minor problems are solved by introducing a new function. Anwar et al. [16] proposed a buffer storage replacement algorithm for flash-based storage devices. Being called the LBA, it takes into account the fact that flash memory has asymmetric reading and writing costs. In [17], the impact of the cost of the last-level cache memory in a hybrid memory system was mainly addressed. A cost cache memory replacement policy in shared level cache memory has been proposed to increase its memory performance. CACRP improves cache memory performance by three parameters. The WOLF-ARE algorithm counters network traffic and recognizes popular files. This algorithm considers the frequency and recent information of the files to determine the popularity of the content [18]. The CCF-LRU policy categorizes data sheets that have cold or hot attributes as well as clean or dirty properties. The replacement algorithm concept is to replace as many cold and clean data sheets as possible, especially those that have been referenced only once before switching hot pages [19], [20].

Jiang and Zhang [21] proposed a cache memory algorithm for HetNets with SBSs, MBS, and D2D transfers. They mixed a cache strategy design as an integer linear programming problem to minimize system costs. The RCR algorithm considers the temporary location of traffic. If a cache memory error occurs and the current traffic location is close to the missing rule, a victim rule is replaced with the miss cache rule, and a high value is set for the missing rule to keep the rule in TCAM [22]. The basic idea of the CLOCK-DNV algorithm is to share all the free space in NVM and DRAM, and manage the DRAM space in a page granulation and the NVM space in a block granulation to use temporal and spatial locality [23]. PLRU and PLFU algorithms keep 20% of the top popular movies of each video group in Memcached without choosing them as alternative candidates [24].

Karami and Guerrero-Zapata [25] proposed an ANFIS-based cache memory replacement algorithm to reduce two general cache infection attacks: Incorrect location and location disruption in NDN. This algorithm is very effective in determining and reducing fake content. The AC-CLOCK screen replacement algorithm takes advantage of both DRAM and PCM algorithms. DRAM has an unlimited number of write cycles and shorter writing delays, and PCM has higher density and less static power [26]. WGDSF is based on a weight substitution algorithm that uses a new weight and cost strategy. In addition, it takes advantage of the existing cache memory replacement policy. WGDSF is a developed GDSF algorithm, and its implementation is based on the type of weight document and time based on weight frequency. The weight frequency-based time parameter is a keyword that indicates the popularity of the content, and also has a large number in the cache memory replacement process [27].

LER focuses on the above writing error in STT-RAM cache memories that originated from transactions 0 to 1. The basic idea is to place the input object in a cache memory line that has the least amount of error in the writing operation [28]. Motwani et al. [29] developed a new algorithm that enhances the performance of multi-level cache memory. In the proposed algorithm, the reference value of an object depends on the freshness of the object in the cache memory, along with the novelty characteristics and frequency of the object. Nomura [30] proposed a way to delay the decision to freeze cache memory line layouts to implement the Stubborn strategy more effectively.

Olanrewaju et al. [31] proposed the NB-AWRPDA smart web proxy memory approach, aiming to improve AWRP performance in terms of HR and BHR. By studying the various page replacement algorithms, it should be noted that the LRU algorithm has better results than many other policies, and it is possible to improve it [32]. The HCR algorithm selects and sacrifices a block that has the least probability of error and the minimum number of writes during the write operation from a set of blocks [33]. ICRA is a cache memory replacement policy combined with an indexing policy, and its main innovation is that when the document and page information are read for the first time, an index is created and the indexed data is simultaneously analyzed, sorted, and stored in the cache memory [34]. PR-LRU increases flash memory performance and reduces writing count. This algorithm divides the buffer into a list of victims, cold, hot, and LRUs. Pages from the cold or hot LRU list are placed in the victim LRU list. The victim LRU list prefers to replace clean pages with dirty ones [35].

Chen et al. [36] proposed a lightweight chart transformation method that provides a custom cache for full use of chart-level data reuse. They also propose a mapping method that uses data parallelization and reuse to manage the input of different graphs effectively. The proposed method increases the memory cache efficiency of database servers for e-commerce applications by reducing server overload.

In [37], the subject of identifying fake ideas in e-commerce businesses based on short-term memory is a repetitive deep learning neural network. The test was performed using the standard Yelp product review dataset. A linguistic query and word count dictionary were used to extract additional linguistic features from the review texts, which can help distinguish between real and fake comments. He and Lin [38] proposed a horizontal position design method in a single-user buffer auxiliary relay system and a 3D position design method in a multi-user buffer auxiliary relay system. The positioning system is designed to achieve the maximum system average and optimal speed. The proposed method significantly improves performance in convergence of power, speed, path losses, and energy costs, which can provide higher quality communication services to users in the system and better support for the widespread use of drones.

In general, among the algorithms proposed to solve the problem of speed difference between CPU and cache, the optimal algorithm is the one that has good performance and is easy to implement. This research proposes a priority object replacement algorithm called PCORA, which is based on prioritizing cache memory objects according to three parameters. Experiments show that the proposed algorithm has a better hit rate than other algorithms and can effectively improve cache memory performance.

3 | Priority Cache Object Replacement Algorithm

This section describes each part of the PCORA algorithm, including the data and its main and sub-functions, separately.

PCORA is based on prioritizing cache memory objects and takes advantage of existing cache memory replacement algorithms, which makes it more reliable and more accurate. This method offers an approach that improves the hit rate of the cache memory and uses as much cache space as possible. Three cases happen when users request an object on the network. In the first case, if the requested object is in the cache memory, a hit occurs and the request is answered. In the second case, if the requested object is not in the cache memory and the cache memory capacity is sufficient, an error occurs, and that object will be added to the cache memory. In the third case, if the requested object is not in the cache memory and the cache capacity is not enough, an error occurs, and one of the cache memory objects must be removed to save the new object.

In this case, PCORA uses the priority function to decide which object should be removed from the cache memory. The proposed replacement algorithm is a powerful tool that can increase server performance by considering three factors: Recency, frequency, and input order of each object. In fact, it runs similarly to the LRU and LFU algorithms and prioritizes each object according to both factors. The first factor, RecP (j), is a counter that indicates the freshness of object j in cache memory, and the second factor, FreP (j), is a counter that indicates the number of times object j is requested in cache memory. Based on the two factors, RecP (j) and FreP (j), the value of the object j priority function is calculated by *Eq. (1).*

$$\mathrm{Per\,P_j} = \frac{\mathrm{Rec\,P_j}}{\mathrm{Fre\,P_j}}. \tag{1}$$

An object in the cache memory whose value of the priority function is higher than the others has the highest priority for deletion from the cache memory. The third factor, IOP(j), is the order in which object j enters the cache memory. When the new object j is placed in the cache memory, the priority function factors must be initially quantified. According to the above variables, PCORA is run based on the following five functions:

The main function

In the primary function, when object j is requested, one of the three main function modes of the proposed algorithm occurs.

Algorithm 1. Main function.

```
For each requested object j
    If (the requested object j is not in the cache and the cache
        is not full)
        Call Function 1
    elseif (the requested object j is in the cache)
        Call Function 2
    elseif (the requested object j is not in the cache and the
            cache is full)
        Call Function 3
```

When the cache memory is referred to, the requested object j is not available in the cache memory, and the cache memory is not full, then a Miss occurs, and the requested object must be added to the cache memory. In such cases, *Algorithm 2* is executed.

Algorithm 2. Function 1.

```
For each frame in the cache memory
    If (the frame i is empty) then
        Place the new object in the frame i in the cache memory
        RecP(j)=1
        FreP(j)=1
        IOP(j) = Cache Size
```

Set the RecP (j) factor to 1, FreP (j) to 1, and IOP (j) to the number of cache memory frames. FreP (j) is set to 1 because this means that object j is used once and IOP (j) is equal to the size of the cache memory because object j is stored as the last object in the cache memory, and it should be at the end of the queue like the FIFO algorithm.

At each access to the cache memory, if the requested object j exists in the cache memory, a hit occurs and *Algorithm* 3 is executed:

Algorithm 3. Function 2.

```
If the requested object j is in the cache
For each object i in the cache
If   $i \neq j$
            RecP(i) = RecP(i) +1
RecP(j) =1 and  FreP(j) = FreP(j) +1
```

When the cache memory is accessed, the requested object j is not in the cache memory, and the cache memory is full; in this case, the proposed algorithm updates the PreP value and selects the object with the maximum value of the priority function. If there is more than one object with the maximum value of the priority function, among the objects with the maximum value, an object with the minimum IOP value is selected to be deleted from the cache memory. The object with the maximum value of the priority function is searched top-down in the cache memory. Suppose an error occurs, the algorithm must select object t with the maximum value of the priority function and clear it from the cache memory; in such a case, *Algorithm 3* will be executed.

Algorithm 4. Function 3.

```
Update the PreP
 If there is only one maximum PreP
 Remove the object with maximum PreP (object t)
 Else
 Remove the object with maximum PreP and
 minimum IOP
 For each object(i) in the cache
 If    i ≠ j and i ≠ t

RecP(i) = RecP(i) +1
 RecP(t) = RecP(t) +1  and  FreP(t) = 1
 RecP(j) =1 and  FreP(j) = FreP(j) +1 and  IOP(j) = Cache Size
 For each object(i) in the cache
 If    IOP(i) > IOP(t)

IOP(i) =  IOP(i) – 1
```

One of the crucial concepts in replacement algorithms is their high overhead on systems. The proposed algorithm requires three counters to run, which adds memory overhead to the system: first, the algorithm requires a counter for RecP (j); second, a counter for FreP (j); and third, a counter for IOP (j). The last and maximum space required is the space for PreP (j), which is the value of the priority function of each object in the cache memory. Calculating the value of the priority function of each object reduces memory time and overhead only if the requested object j is not in the cache memory and the cache memory is full. The proposed algorithm for solving this problem considers PreP (j) as an integer to reduce the main costs.

4 | Performance Evaluation

In this section, the performance of the proposed algorithm is evaluated experimentally.

The proposed algorithm was simulated with the C# programming language and compared to six algorithms: LRU [1], [2], FIFO [32], WRP [8], LFU [6], [7], MFU [5], and DWRP [15]. The emulator is designed to execute some address sequences and save instructions for some real applications, and implement various replacement algorithms with different sizes of cache memory. The hit rate obtained depends on the locality of the cache memory access requests, the cache memory size, and the replacement algorithm. The modular design of the emulator allows easy simulation and optimization of the proposed algorithm. An address sequence is a list of thousands of memory addresses generated by a real program running on a processor. The addresses come from executing storage and code retrieval instructions. Some address sequences include instructions for both data and fetch addresses, but only one data cache memory is simulated; therefore, the sequences have only data addresses.

To simulate the proposed algorithm, three sequences derived from the SPEC standards have been used. According to the sequences used, four cache memory sizes are considered, and the emulator is run to evaluate the performance of the proposed algorithm. The emulator runs with four different cache memory sizes for the NNgcc address sequence, and the results are compared to LRU, FIFO, WRP, LFU, MFU, and DWRP algorithms. *Table 2* shows the hit rate values for the NNgcc address sequence simulated by LRU, FIFO, WRP, LFU, MFU, DWRP, and PCORA. When the cache memory size is 1K, the performance of the proposed algorithm is 64.046% better than DWRP. Moreover, when the cache memory size is 0.5K, the PCORA hit rate is 2.322% higher than the best algorithm, the LRU.

In the worst case, when the cache memory size is 3K, it works 0.402% better than the LRU. As shown in *Fig. 1*, PCORA performs significantly better than the other six algorithms. The hit rate obtained from PCORA with NNsixpack address sequence is above 60.24%; this rate is between 52.4% and 58.958% for LRU and between 9.234% and 26.656% for DWRP (See *Table 3*). As shown, as the cache memory size increases, the performance of the proposed algorithm increases. *Fig. 2* shows the PCORA simulation results compared to the other six algorithms, in which it performs better than the others. The third address sequence is NNswim. *Fig. 3* shows the comparison and the result of this sequence. *Table 4* shows the exact hit rates of seven algorithms with cache memory sizes of 0.5K, 1K, 2K, and 3K. The results show that the maximum hit rate is related to the LRU algorithm with the largest cache memory size, which is about 84.082%.

The result for the proposed algorithm is expected to be at least equal to the highest value among the other six algorithms or more. The results in Table 4 showed that the maximum hit ratio for PCORA is about 84.692%. The maximum hit ratio is 0.61% higher than the best hit rate for the LRU and 18.59% higher than the worst hit rate for the MFU. As explained, in all the sequences used, the hit rate of the proposed algorithm is always better than that of other alternative algorithms. The hit rate of the proposed algorithm is due to the priority factors considered in PCORA that never allow the result to be worse than the maximum hit rate of the other six alternative algorithms.

Table 2. A comparison between the hit ratios of different algorithms with four cache sizes (NNgcc).

Cache Size	PCORA%	LRU%	FIFO%	WRP%	LFU%	MFU%	DWRP%
512 (0.5k)	65.864	63.542	61.684	59.042	34.642	46.952	11.93
1024 (1k)	76.256	75.754	72.51	66.584	47.836	48.684	12.21
2048 (2k)	80.618	79.784	77.668	74.442	64.268	50.758	33.948
3072 (3k)	82.244	81.842	80.95	76.874	81.256	51.656	60.19

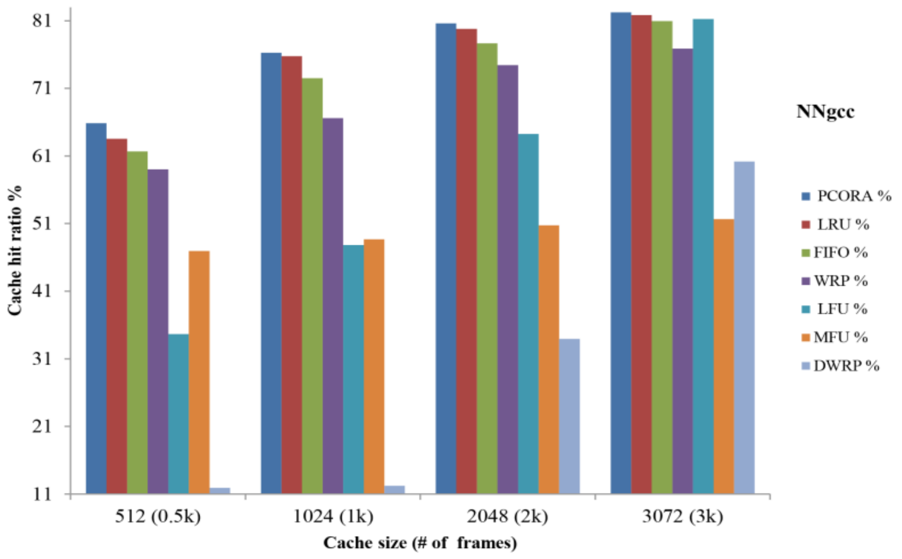

Fig. 1. Performance of different algorithms with different cache sizes for NNgcc.

Table 3. A comparison between the hit ratios of different algorithms with four cache sizes (NNsixpack).

Cache Size	PCORA%	LRU%	FIFO%	WRP%	LFU%	MFU%	DWRP%
512 (0.5k)	53.016	52.4	51.37	49.264	24.244	37.822	9.234
1024 (1k)	55.466	54.92	54.206	53.686	33.564	38.98	9.552
2048 (2k)	58.332	57.8	57.032	57.49	42.446	40.342	13.378
3072 (3k)	60.24	58.958	58.4	59.458	55.808	41.124	26.656

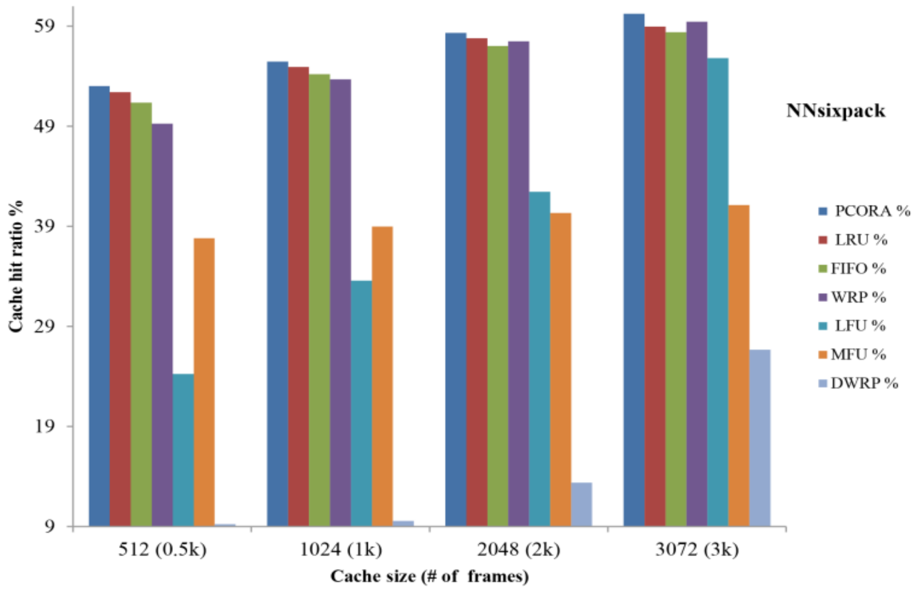

Fig. 2. Performance of different algorithms with different cache sizes for NNsixpack.

Table 4. A comparison between hit ratios of different algorithms with four cache sizes (NNswim).

Cache Size	PCORA%	LRU%	FIFO%	WRP%	LFU%	MFU%	DWRP%
512 (0.5k)	79.696	79.104	76.95	70.294	66.29	50.066	23.12
1024 (1k)	80.974	80.52	79.098	74.34	72.822	53.868	33.036
2048 (2k)	83.438	82.572	81.464	77.052	78.756	59.038	56.296
3072 (3k)	84.692	84.082	83.24	79.638	83.128	66.102	78.652

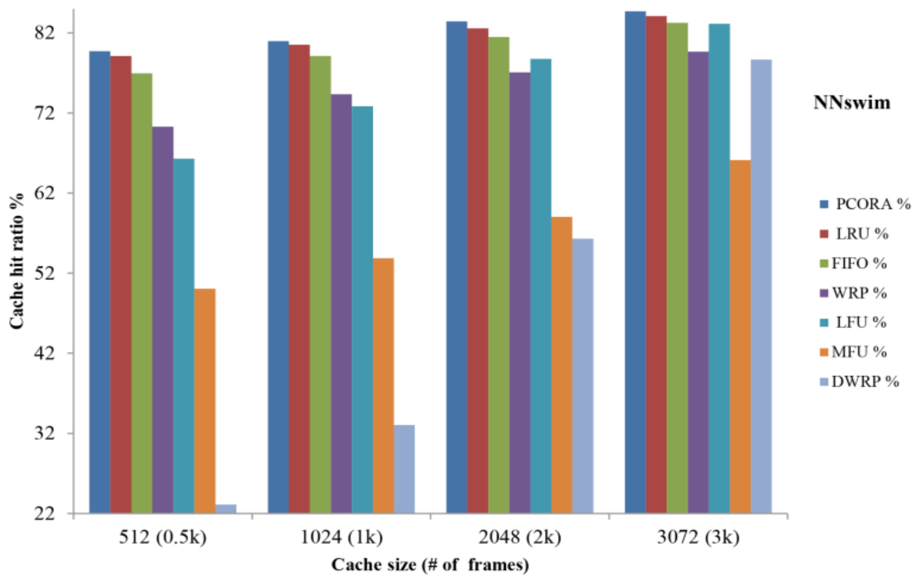

Fig. 3. Performance of different algorithms with different cache sizes for NNswim.

The emulator also calculates the average hit rates of PCORA, LRU, FIFO, WRP, LFU, MFU, and DWRP algorithms with three address sequences. The results are presented in *Tables 5-7*. *Fig. 4* shows that when the NNgcc address sequence is used, the average PCORA performance of 4 is better than that of other algorithms. *Table 5* shows the average of the four hit rates of seven algorithms for the NNgcc address sequence. The performance of the proposed algorithm is 1.015% better than the best algorithm, LRU, and 46.676% better than the worst algorithm, DWRP. As shown in *Figs. 5* and *6*, PCORA performs better than other algorithms; details of which are given in *Tables 6* and *7* for the NNsixpack and NNswim address sequences, respectively.

Table 5. A comparison between the average 4-hit ratios of different algorithms (NNgcc).

PCORA%	LRU%	FIFO%	WRP%	LFU%	MFU%	DWRP%
76.2455	75.2305	73.203	69.2355	57.0005	49.5125	29.5695

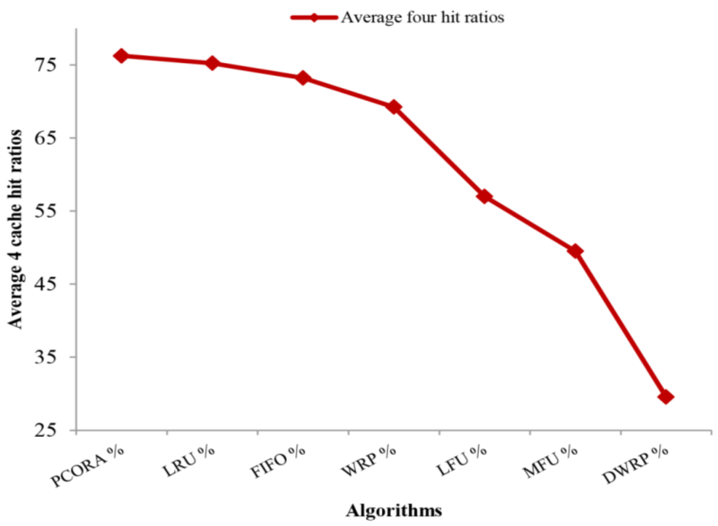

Fig. 4. Average of four performances of different algorithms for NNgcc.

Table 6. A comparison between the average 4-hit ratios of different algorithms (NNsixpack).

PCORA%	LRU%	FIFO%	WRP%	LFU%	MFU%	DWRP%
56.7635	56.0195	55.252	54.9745	39.0155	39.567	14.705

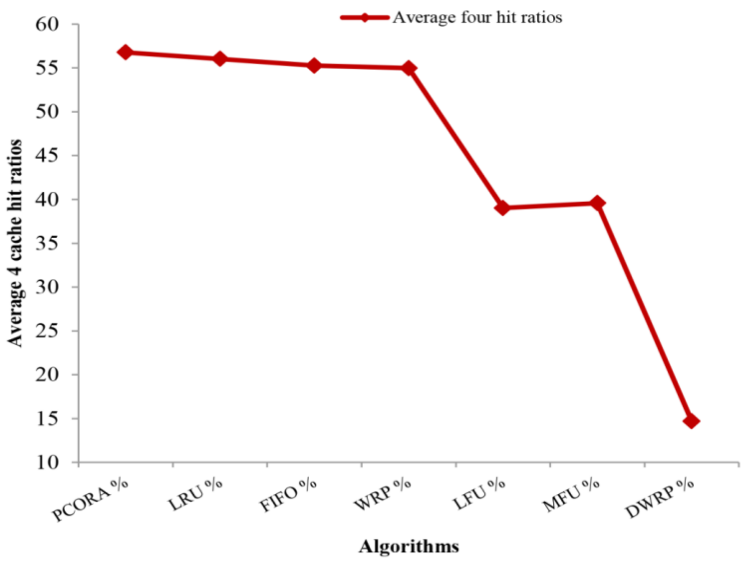

Fig. 5. Average of four performances of different algorithms for NNsixpack.

Table 7. A comparison between the average four-hit ratios of different algorithms (NNswim).

PCORA%	LRU%	FIFO%	WRP%	LFU%	MFU%	DWRP%
82.2	81.5695	80.188	75.331	75.249	57.2685	47.776

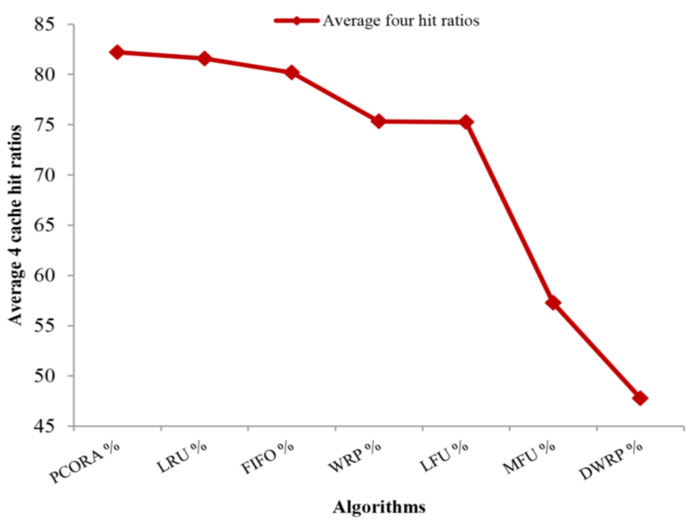

Fig. 6. Average of four performances of different algorithms for NNswim.

5 | Conclusions

In this research, a cache memory replacement algorithm was introduced, and a simulation of cache memory with a sequence of different addresses showed that it was a modification of popular replacement algorithms such as LRU and LFU. It indicated that objects with a smaller priority function value are more likely to be re-requested than others. PCORA was simulated with three address sequences and compared to six popular algorithms. The PCORA simulation result with four different cache memory sizes showed better performance than the other six algorithms. The algorithm can be simulated with other address sequences and algorithms. It should be noted that other parameters and factors that indicate the properties of objects in the cache memory can be considered for the proposed algorithm. For example, considering the size and cost of each object in the cache memory makes PCORA perform better for applications such as web storage. The proposed design is also suitable for deciding on centralized cache memory, but not for a distributed solution.

Conflict of Interest

The authors declare that they have no conflict of interest.

Ethical approval

This article does not contain any studies with human participants or animals performed by any of the authors.

References

[1] Kushwah, J. S., & Tamrakar, S. (2017). An extensive review of web caching techniques to reduce cache pollution. *Imperial Journal of Interdisciplinary Research*, *6*(13), 111–118. https://rntujournals.aisect.org/assets/upload_files/articles/3b39342f735878fb7ee90c670784dc89.pdf

[2] Wang, Y., Yang, Y., Han, C., Ye, L., Ke, Y., & Wang, Q. (2019). LR-LRU: A PACS-oriented intelligent cache replacement policy. *IEEE Access*, *7*, 58073–58084. https://doi.org/10.1109/ACCESS.2019.2913961

[3] Wang, Y. L., Kim, K. T., Lee, B., & Youn, H. Y. (2018). A novel buffer management scheme based on particle swarm optimization for SSD. *The Journal of Supercomputing*, *74*(1), 141–159. https://doi.org/10.1007/s11227-017-2119-2

[4] Ooka, A., Eum, S., Ata, S., & Murata, M. (2018). Compact CAR: Low-overhead cache replacement policy for an ICN router. *IEICE transactions on communications*, *101*(6), 1366–1378. https://doi.org/10.1587/transcom.2017EBP3299

[5] Tailor, P. M., & Morena, R. D. (2017). A survey of database buffer cache management approaches. *International Journal of Advanced Research in Computer Science*, *8*(3), 409–414. https://www.academia.edu/64178716/A_survey_of_database_buffer_cache_management_approaches

[6] Negrão, A. P., Roque, C., Ferreira, P., & Veiga, L. (2015). An adaptive semantics-aware replacement algorithm for web caching. *Journal of Internet Services and Applications*, *6*(1), 4. https://doi.org/10.1186/s13174-015-0018-4

[7] Yang, P., Wang, Q., Ye, H., & Zhang, Z. (2019). Partially shared cache and adaptive replacement algorithm for NoC-based many-core systems. *Journal of Systems Architecture*, *98*, 424–433. https://doi.org/10.1016/j.sysarc.2019.05.002

[8] Samiee, K. (2009). A Replacement algorithm based on weighting and ranking cache objects. *International Journal of Hybrid Information Technology*, *2*(2), 93–104. https://B2n.ir/kk7780

[9] Akbari Bengar, D., Ebrahimnejad, A., Motameni, H., & Golsorkhtabaramiri, M. (2020). A page replacement algorithm based on a fuzzy approach to improve cache memory performance. *Soft Computing*, *24*(2), 955–963. https://doi.org/10.1007/s00500-019-04624-w

[10] Akbari-Bengar, D., Ebrahimnejad, A., Motameni, H., & Golsorkhtabaramiri, M. (2020). Improving cache memory performance based on a fuzzy clustering-based page replacement algorithm by using four features. *Journal of Intelligent & Fuzzy Systems*, *39*(5), 7899–7908. https://doi.org/10.3233/JIFS-201360

[11] Arya, G. P., Prasad, D., & Rana, S. S. (2018). An improved page replacement algorithm using block retrieval of pages. *International Journal of Engineering & Technology*, *7*(4.5), 32–35. https://doi.org/10.14419/IJET.V7I4.5.20004

[12] Wu, H., Luo, Y., & Li, C. (2021). Optimization of heat-based cache replacement in edge computing systems. *The Journal of Supercomputing*, *77*(3), 2268–2301. https://doi.org/10.1007/s11227-020-03356-1

[13] Do, C. T., Choi, H. J., Kim, J. M., & Kim, C. H. (2015). A new cache replacement algorithm for last-level caches by exploiting tag-distance correlation of cache lines. *Microprocessors and microsystems*, *39*(4–5), 286–295. http://dx.doi.org/10.1016/j.micpro.2015.05.005

[14] Shin, D., Cho, K., & Bahn, H. (2020). File type and access pattern aware buffer cache management for rendering systems. *Electronics*, *9*(1), 164. https://www.mdpi.com/2079-9292/9/1/164

[15] Akbari, B. D., Jazayeri, R. H., & Berenjian, G. (2012). An improvement in the WRP block replacement policy involves reviewing and solving its problems. (**In Persian**). *Advances in Computer Research. 3*, 67–75. https://www.sid.ir/paper/328710

[16] Anwar, U., Paik, J. Y., Jin, R., & Chung, T. S. (2017). Log-buffer aware cache replacement policy for flash storage devices. *IEEE transactions on consumer electronics*, *63*(1), 77–84. https://doi.org/10.1109/TCE.2017.7931973

[17] Jia, G., Han, G., Wang, H., & Wang, F. (2018). Cost-aware cache replacement policy in shared last-level cache for hybrid memory-based fog computing. *Enterprise Information Systems*, *12*(4), 435–451. https://doi.org/10.1080/17517575.2017.1295321

[18] Hajiakhondi-Meybodi, Z., Abouei, J., & Raouf, A. H. F. (2018). Cache replacement schemes based on adaptive time window for video on demand services in femtocell networks. *IEEE transactions on mobile computing*, *18*(7), 1476–1487. https://doi.org/10.1109/TMC.2018.2864164

[19] He, J., Jia, G., Han, G., Wang, H., & Yang, X. (2017). Locality-aware replacement algorithm in flash memory to optimize cloud computing for the smart factory of industry 4.0. *IEEE Access*, *5*, 16252–16262. https://doi.org/10.1109/ACCESS.2017.2740327

[20] Talaat, F. M., Ali, S. H., Saleh, A. I., & Ali, H. A. (2020). Effective cache replacement strategy (ECRS) for real-time fog computing environment. *Cluster computing*, *23*(4), 3309–3333. https://doi.org/10.1007/s10586-020-03089-z

[21] Jiang, L., & Zhang, X. (2020). Cache replacement strategy with limited service capacity in heterogeneous networks. *IEEE Access*, *8*, 25509–25520. https://doi.org/10.1109/ACCESS.2020.2970783

[22] Sheu, J. P., & Chuo, Y. C. (2016). Wildcard rules, caching, and cache replacement algorithms in software-defined networking. *IEEE transactions on network and service management*, *13*(1), 19–29. https://doi.org/10.1109/TNSM.2016.2530687

[23] Kang, D. H., Han, S. J., Kim, Y. C., & Eom, Y. I. (2017). CLOCK-DNV: A write buffer algorithm for flash storage devices of consumer electronics. *IEEE transactions on consumer electronics*, *63*(1), 85–91. https://doi.org/10.1109/TCE.2017.014700

[24] Lee, M. C., Leu, F. Y., & Chen, Y. (2015). Pareto-based cache replacement for YouTube. *World wide web*, *18*(6), 1523–1540. https://doi.org/10.1007/s11280-014-0318-9

[25] Karami, A., & Guerrero-Zapata, M. (2015). An ANFIS-based cache replacement method for mitigating cache pollution attacks in Named Data Networking. *Computer networks*, *80*, 51–65. https://doi.org/10.1016/j.comnet.2015.01.020

[26] Kim, S., Hwang, S.-H., & Kwak, J. W. (2018). Adaptive-Classification CLOCK: Page replacement policy based on read/write access pattern for hybrid DRAM and PCM main memory. *Microprocessors and microsystems*, *57*, 65–75. https://doi.org/10.1016/j.micpro.2018.01.003

[27] Ma, T., Qu, J., Shen, W., Tian, Y., Al-Dhelaan, A., & Al-Rodhaan, M. (2018). Weighted greedy dual size frequency-based caching replacement algorithm. *IEEE Access*, *6*, 7214–7223. https://doi.org/10.1109/ACCESS.2018.2790381

[28] Monazzah, A. M. H., Farbeh, H., & Miremadi, S. G. (2016). LER: Least-error-rate replacement algorithm for emerging STT-RAM caches. *IEEE transactions on device and materials reliability*, *16*(2), 220–226. https://doi.org/10.1109/TDMR.2016.2562021

[29] Motwani, A., Swain, D., Motwani, N., Vijan, V., Awari, A., & Dash, B. (2020). Enhancing multi-level cache performance using dynamic r-f characteristics. *Machine learning and information processing* (pp. 277–286). Singapore: Springer Singapore. https://doi.org/10.1007/978-981-15-1884-3_26

[30] Nomura, H. (2020). Experimental investigation of lazy evaluation method in replacement algorithm for long-term re-reference cache management. *Bulletin of networking, computing, systems, and software*, *9*(1), 83–90. http://bncss.org/index.php/bncss/article/view/142

[31] Olanrewaju, R. F., Azman, A. W., & Yaacob, M. (2016). Intelligent web proxy cache replacement algorithm based on adaptive weight ranking policy via dynamic aging. *Indian Journal of Science and Technology*, *9*(36), 1–7. https://doi.org/10.17485/ijst/2016/v9i36/102159

[32] Paulson, H., & Ramachandran, D. R. (2017). Page replacement algorithms–challenges and trends. *International Journal of Computer & Mathematical Sciences IJCMS*, *6*(9), 112–116. https://b2n.ir/nu5851

[33] Priya, B. K., Kumar, S., Begum, B. S., & Ramasubramanian, N. (2019). Cache lifetime enhancement technique using a hybrid cache-replacement policy. *Microelectronics reliability*, *97*, 1–15. https://doi.org/10.1016/j.microrel.2019.03.011

[34] Zhao, Y., Ma, T., Hao, Y., Shen, W., Tian, Y., & Al-Dhelaan, A. (2019). ICRA: index-based cache replacement algorithm for cloud storage. *International Journal of Sensor Networks*, *29*(1), 48–57. https://doi.org/10.1504/IJSNET.2019.097556

[35] Yuan, Y., Shen, Y., Li, W., Yu, D., Yan, L., & Wang, Y. (2017). PR-LRU: A novel buffer replacement algorithm based on the probability of reference for flash memory. *IEEE Access*, *5*, 12626–12634. https://doi.org/10.1109/ACCESS.2017.2723758

[36] Chen, X., Wang, Y., Xie, X., Hu, X., Basak, A., Liang, L., ... & Xie, Y. (2021). Rubik: A hierarchical architecture for efficient graph neural network training. *IEEE transactions on computer-aided design of integrated circuits and systems*, *41*(4), 936–949. https://doi.org/10.1109/TCAD.2021.3079142

[37] Alsharif, N. (2022). Fake opinion detection in an e-commerce business based on a long-short-term memory algorithm. *Soft Computing*, *26*(16), 7847–7854. https://doi.org/10.1109/TCAD.2021.3079142

[38] He, X., & Lin, M. (2022). Reliable auxiliary communication of UAV via relay cache optimization. *Computer Communications*, *186*, 33–44. https://doi.org/10.1016/j.comcom.2021.11.024