International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Ingénierie des Systèmes d'Information
Vol. 29, No. 2, April, 2024, pp. 697-703
Journal homepage: http://iieta.org/journals/isi

Minimizing the Cache Memory Miss Ratio Using Modified Replacement Algorithm (M-CAR)

Salam Ayad Hussein[1*], Mohsin Raad Kareem[2], Dena Nader George[1]

[1] Department of Computer Science, College of Education, Mustansiriyah University, Baghdad 10052, Iraq
[2] Department of Computer Science, College of Basic Education, Mustansiriyah University, Baghdad 10052, Iraq

Corresponding Author Email: salamayad.77@uomustansiriyah.edu.iq

https://doi.org/10.18280/isi.290231

Received: 13 November 2023
Revised: 20 March 2024
Accepted: 25 March 2024
Available online: 25 April 2024

Keywords:
cache memory, replacements algorithms, cache miss, hit ratio, M-CAR

ABSTRACT

Caching is a key method used to close the latency gap between memory and the CPU by using locality in memory accesses. varied cache replacement algorithms have radically varied effects on system performance because they choose which blocks to evict from cache memory in the event of a cache miss. The goal of these replacement strategies is to move closer to the ideal scenario by making the greatest use of the entire cache area, reducing the miss ratio as much as feasible, and obtaining the maximum system performance possible. In this paper, based on clock with adaptive replacement algorithm (CAR), a simple and effective modified algorithm is proposed, namely, modified clock with adaptive replacement (M-CAR), which achieved 91% and 76% hit ratio higher (compared with the conventional CAR method) with datasets that contained 245 and 270 items, respectively. Which is considered to be the most important cache performance criteria. Which means, by default, minimizing the cache miss ratio. As well as the dynamical behavior that has been improved and gained the (M-CAR) that makes it more reliable.

1. INTRODUCTION

Cache memory was developed to close the gap between memory access time and CPU performance speed. Even modern memory technologies remain unable to address this gap, because the computer CPU runs at a high speed, leading to the issue known as the wait state. A wait state is experienced by a computer processor in the case of accessing an external memory or any device which has slow response. This wait state manifests as additional clock cycles allowing the device the time needed for completing a process [1-3]. During this delay, the CPU is unable to perform any operations on data it has not yet received from the memory. M. V. Wilkes created cache memory in 1965, and initially referred to it as the "slave memory", describing it as the second level of a high-speed unconventional memory which produces a zero-wait state [4].

As shown in Figure 1, cache memory is a fast, extremely tiny, zero-wait state memory, placed between the M.M and CPU to act as a bridge for data exchange and storage. Due its small size, a major problem in cache memory is the "cache miss", which took place if the CPU requests an item of data that is not available in the cache. Conversely, a "cache hit" took place if the CPU re-quests an item of data that is available in the cache [5].

Replacement algorithms are required when a cache miss took place and there is no space in the cache to load memory blocks. A replacement algorithm selects the existing block in the cache. Replacement policies/algorithms are used to achieve optimized cache usage. When the cache is full, replacement policies decide what data is replaced to make room for new data currently in use. An efficient algorithm is an algorithm that can take less time, the number of cache misses is low and also balances the cost. In modern embedded systems, applications have become very large and the presence of caches is inevitable. Therefore, newer embedded processors have cache architectures that are just as complex as those of general-purpose processors. Integrated processor caches, especially for mobile devices, have complicated architectures because all three metrics (performance, power and area) must be satisfied simultaneously within certain constraints.

When a cache miss took place, a replacement algorithm is needed if the space demanded to load the memory blocks is not available in the cache. The algorithm searches M.M to identify the required data item and transfer it to the cache, and makes a decision to strategically swap blocks. Such algorithms are referred to as the replacement algorithms [6, 7]. Since replacement algorithms choose a candidate block in the cache for replacement, they need to be implementable in hardware to achieve high operational speed. Based on the principle of replacement, these techniques may be categorized as [6-8]:

A. Optimal: This technique replaces the block least likely to be required in the future. Although this policy of re-placement isn't feasible, it is frequently utilized as a benchmark to assess the effectiveness of other replacement plans.

B. Random: This technique may choose the replacement cache block candidate in a completely arbitrary order that ignores the memory references.

C. First-In, First-Out (FIFO): This technique selects the oldest available block for replacement.

D. Least Recently Used (LRU): This technique is based upon the idea that a block that has recently been referred has a high likelihood of being referred again in the future.

E. Least-Frequently-Used (LFU): This technique uses the software counter related to each cache block to identify and replace the least frequently used block.

F. Second chance algorithm: When a replacement is required, the data exists at the top of the queue (with the longest dwell time) is inspected; if it was referenced while in the cache, it is pushed to the end of the queue and given another opportunity. If not, you'll be evicted.

G. Working set algorithm (WS): The Working Set Replacement technique replaces old data in cache memory using the working set concept, which is defined as the set of blocks used by a process at any given time.

H. Aging algorithm: A method that employs a bit presentation of a zero-counter for each data item in cache, with this counter adding zero for unreferenced blocks per clock tick. When a block has to be replaced, the block with the lowest counter is deleted.

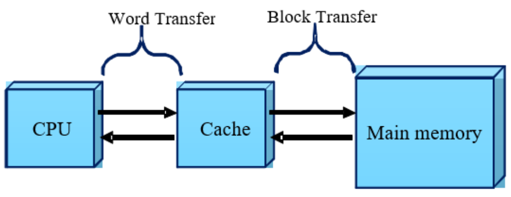


Figure 1. Cache and M.M data transfer

2. RELATED WORK

Numerous studies have developed sophisticated cache replacement algorithms that reduce the miss ratio and speed disparity between the (CPU) and (M.M) access time. Below are some of these algorithms.

2.1 CLOCK algorithm

The page cache entries in this algorithm are structured as a circular buffer (list) resembling a clock, with each data page having an associated reference bit. The hand of the clock scans the clock structure in a circular motion examining the reference bit of each entry until it identifies the buffer's oldest data page. The reference bit of the data is marked in an event where certain data page was referenced. Cache replacement policy is invoked when a page fault is encountered, and the data page is pointed to (identified) by the inspecting hand as the oldest data page [4]. In an event where marking a reference bit of data page, the bit value will be reset to 0, and the next oldest data page is pointed to by the hand. This process continues until a data page with reference bit of 0 is found, in which case that data page is taken away from the buffer, and a new data page is inserted instead and its reference bit value is set to 0 [9-11]. This technique suffers from few disadvantages, including its impracticality (the evicted block may be needed in a short time later), low hit ratio in the presence of unrepeated sequence of data pages, and lack of scan resistance (which means that the entire memory must be searched to choose the best eviction candidate), which makes this technique slow.

2.2 Dueling CLOCK

This represents a policy of adaptive replacement that functions based on the CLOCK method. To achieve a scan-resistant CLOCK, the single necessary modification is for the clock hand to point to the buffer's latest rather than oldest data page [11, 12]. This may be viewed as a method of adaptive replacements using CLOCK structure. The cache will be split into 3 groups: (M1), (M2) and (M3), each to carry out a specific task. This algorithm is commonly implemented as an approach to substitute a small set of (M1), whereas scan-resistant CLOCK is always utilized by small sets (M2 and M3), which are the large sets that may be applied between the scan-resistant CLOCK and traditional CLOCK according to the comparative performance associated with the two former sets (M2 and M3). For specifying the replacement policy that must be applied for (M3), a (10-bit) policy select counter (PSEL) is implemented [12, 13].

The disadvantages of this technique include its technical complexity, its performance efficiency that is only achieved when applied inmulti-level caches (level2 specifically), and its low hit ratio when data pages reside in level1 or level3.

2.3 CLOCK-Pro

The CLOCK-Pro algorithm utilizes CLOCK for the approximation of the reuse distance. A page that is accessed is viewed as "hot" if it has little reuse distance, and "cold" if it has a massive reuse distance. The data items are accessed in a list in which data items that have slight recency level are placed at the top of the list, whereas those that have a larger recency level are placed at end of that list. The circular list is utilized to keep the entries of the cache page [13, 14]. There are three status bits that are associated with these pages; cold or hot indicator, referencing bit, and indicator for every one of the cold pages to specify whether or not a page is in the test phase. The CLOCK-Pro approach has 3-hands [14]. The HANDcold is viewed as the first one that indicates to the last resident cold page, or cold page that is substituted after that. During the process of page substitution, the page which is chosen via the HANDcold will be evicted when the reference bit that is associated with the page is 0. HANDhot is the 2nd hand pointing to the hot page with the maximum recency. If a reference bit associated with the page which has been chosen by HANDhot is 0, the page is considered a cold page [15, 16]. The disadvantages of this technique are its mathematical complexity, and the fact that it only captures the "recency" factor.

2.4 Two queue (2Q)

2Q maintains two detached lists. One of them is managed as a list (LRU), which is named (Hot). And the other as (FIFO), F. The list (F) is divided into two parts (F-in) and (F-out). The list (F-in) comprises data items in (M.M), while the list (F-out) only contains information about data items, but not the actual content. When the data item is accessed for the first time, it is placed data item at the top of the list (F-in). The data item position in the list (F-in) is not changed as long as it remains there. As new data items are used, the list fills up (F-in). In this situation, the final data item in the list (F-in) is fetched next, but the data item information is placed at the beginning of the list (F-out). When the data item is utilized in the list (F-out), space is reclaimed and put at the beginning of the current list. If the list (F-in) is not full, the claim begins at the end of the active list. The data item taken from the active list is not added into any lists since it has not been utilized for a long time.

2.5 LOW inter-reference recency set (LIRS)

(LIRS) dismisses data items based mostly on their Inter-Reference Recency (IRR). The (IRR) of a data item specifies the precise number of other data items transferred between two sequential references to any data item. It is assumed that if the current (IRR) of a data item is considerable, the next (IRR) of the data item may likewise be considerable. It is worth noting that a data item with a high IRR that was picked for deletion may have recently been used.

This method distinguishes between data items of greater IRR (HIR) and data items of lower IRR (LIR). The numbers of data items of (LIR) and (HIR) are chosen so that all (LIR) data items and just a few numbers off (HIR) data items are retained.

2.6 Adaptive replacement cache (ARC)

ARC algorithm links the LFU solution with regulates between the two solutions dynamically. It is an easy to implement with low overhead on systems such as LRU. It employs '4' double linked lists (K1, K2) as the factual cache content. (N1, N2) is an actor as a second level. N1 and N2 contain the data items that have been thrown out from K1 and K2 respectively. thus, an overall number of data items of (2 * C) is required for these lists. (C) would be the number of data items that are present in the then cache. Both lists use LRU replacements, where the data item removed from K2 is placed into N2. (K1, N1) and N2 work in a similar way, except in case of a hit in K1 or N1, the data item is moved to K2. The adaptive nature of this policy makes it possible to change the sizes of lists.

The size of K1 and S2 always adds up to the total number of data items. Whenever there is a hit in N1, the size of K1 increases by '1' and decreases of K2 by '1. In the opposite direction, a hit in N2 will increase the size of K2 by ones, resulting in a decrease of K1 by one. This allows cache to adjust and have more or less frequency or recency depending on the workload.

2.7 CLOCK with adaptive replacement (CAR)

The CAR directory structure is depicted in Figure 2, in which T2, T1, B2, and B1 are the four linked lists that make up the CAR directory. Lists T2 and T1 are linked as CLOCK policy implementers, and B2 and B1 are linked as LRU policy implementers [17-19].

Lists B2 and B1 include history pages that have recently been evicted from the cache, and CLOCKs T2 and T1 contain pages that are currently in the cache. CLOCK T2 records frequency (F), and CLOCK T1 records recency (R). Simple LRU lists can be found in lists B2 and B1. Pages that have been evicted from T1 replace B1, and those removed from T2 replace B2. This algorithm aims to maintain B2 and B1 at nearly equivalent sizes to T2 and T1, respectively. Additionally, the technique prevents |T1|+|B1| from exceeding the cache size. In response to a varying workload, the sizes allocated to CLOCKs T2 and T1 are continuously adjusted. The target size for T1 is increased anytime a hit in B1 is detected; conversely, the T1 target size decreases whenever a hit in B2 is detected. The new pages are placed in T2 or T1, directly behind the clock hands that have been displayed in order to move clockwise. New pages have a page reference bit that is set to "0" [20, 21].

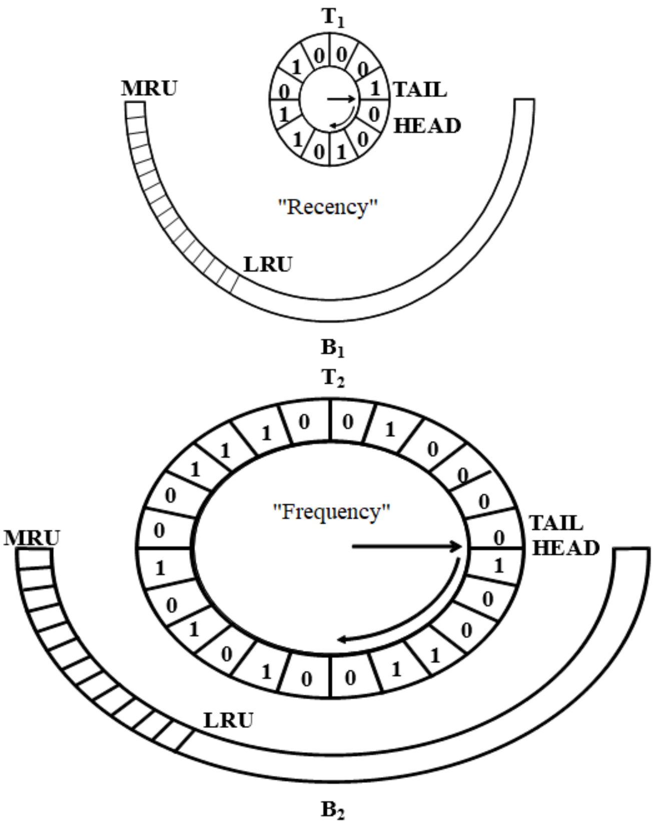


Figure 2. CAR directory structure

Any page in T1 U T2 can be cached by setting the page reference bit for that page to "1". The T1 clock hand moves a page behind T2 clock hand then resets the reference bit of the page to "0" if it meets a page that has a page reference bit of "1". A "0-page" reference bit page is evicted then put at the most recently used position (MRU) in B1 whenever T1 clock hand comes across it. The page reference bit is reset to "0" if the T2 clock hand comes across a page with page reference bit that equals "1". An evicted page is then placed at the MRU position in B2 any time the T2 clock hand comes into contact with a reference page bit with value "1" [22, 23]. The basic CAR algorithm is as follows [5, 11, 17]:

The main disadvantage of CAR is that it does not handle temporal filtering. Therefore, more stringent separation must be imposed between short-term and long-term utility pages for specific workloads. This means that recency (R) and frequency (F) cannot select the best candidate data page for eviction, which results in a lower hit ratio. This will be the focus of the modification proposed in this study [24].

Another common disadvantage in such technique with complex structure is that it cannot be implemented as hardware because it has a partitioned data structure that makes it impossible to implement a hardware clock hand that moves at the required speed [25].

```
INITIALIZATIONS: Set p = 0 and set lists Tl, Bl, B2 and T2 as empty.
CAR(x)
INPUT: requested page x.
if(x is in T2 ∪ Tl) then
/* cacheHit */
Set page reference bit for x to the value of 1.
otherwise /* cacheMiss */ if(|T2|+T1=c)
/* cacheFull, substitute page from the cache */
```

```
replace()
/* replacement of the directory of the cache */
if((x isn't in B2 ∪ B1) and (|B1|+|T1|=c))
Discarding the LRU page in the B 1.
Else if((|T1|+|Bl|+|T2|+|B2|=2c) and (x isn't in B2 ∪ Bl))
Discarding the LRU page in the B2.
End if
End if
/* cache the directory miss */
if(x isn't in B2 ∪ Bl) then
Insert x at Tl tail.
Set the reference bit of page of x to 0.
```

3. THE PROPOSED METHOD (M-CAR)

3.1 Design concept

The proposed technique will concentrate on the concept of "long-term utility pages," which basically depends on:

I. Accurate guessing.
II. Factors updating.
III. Dynamic data blocks repositioning.

The proposed technique structure and parameters will contain:

i. T2, T1, B2, and B1 are the four linked lists that make up the CAR directory.

ii. Lists T2 and T1 are linked as CLOCK policy implementers. iii. B2 and B1 are linked as LRU policy implementers.

iv. Lists B2 and B1 include history pages that have recently been evicted from the cache.

v. CLOCKs T2 and T1 contain pages that are currently in the cache, CLOCK T2 records frequency (F), and CLOCK T1 records recency (R).

vi. Virtual list updated dynamically contains the (MAXF-1) at each cache referencing.

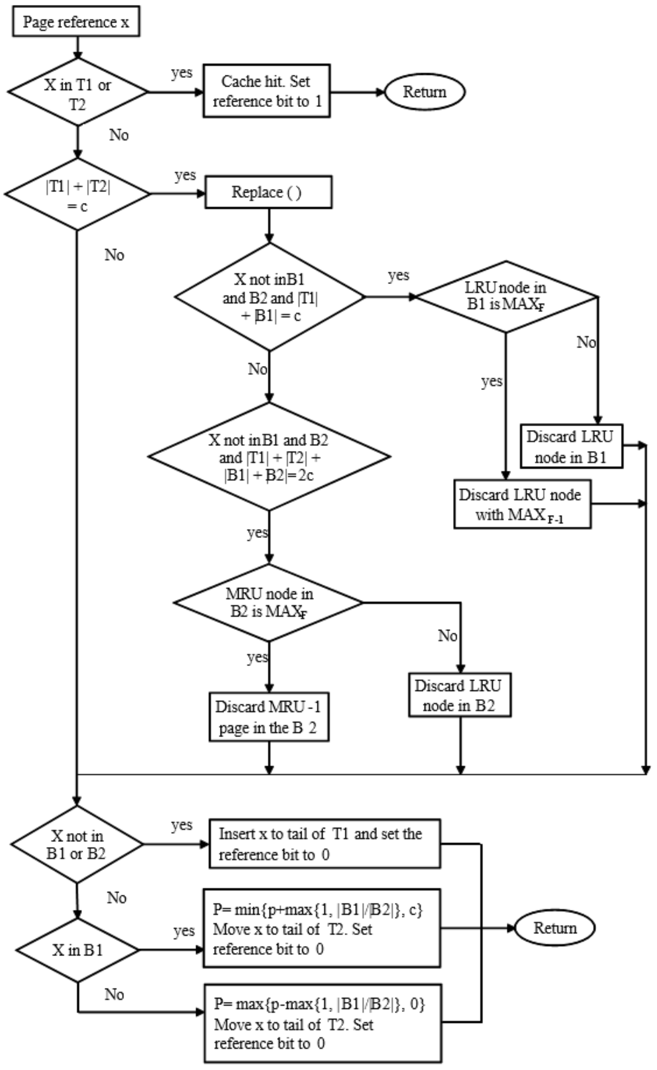
In the proposed method, keeping long-term utility pages for future requests is achieved by giving greater importance to the frequency factor (F) in specific conditions when the cache is full during requesting new data pages. In a traditional CAR algorithm, when a cache miss occurs a data page is normally evicted from T2 (which means that this page was frequently utilized prior its eviction from the cache). In addition, it should be noted that the time complexity in traditional (CAR) and the proposed (M-CAR) will not be affected because both of these techniques are scan resistance. And space complexity also will not be affected because the proposed (M-CAR) maintains the same structure of (CAR). Thus, the previous mentioned factors will not be a noticeable criteria.

The proposed modification will be as follows:

1. When cache miss occurs in T2 and coincides with the data block in the tail of T2 being the maximum frequency (MAXF) data page, the data block with MAXF-1 is evicted from T2 and placed at the most recently used position (MRU) in B2.

2. When a cache miss occurs in B2, the data block that is evicted out of cache from B2 will not be the data block with MAXF even if it located at the end of B2 list. Instead, the data block that positioned before the most recently used position (MRU-1) in B2 will be moved out of cache.

Figure 3 illustrates the concept of the proposed M-CAR technique in a flowchart, in which all the subroutines achieving the different working steps of the proposed system (miss and hit stages) are gathered in one diagram of steps, conditions and actions.



Page reference x
X in T1 or T2
yes
Cache hit. Set reference bit to 1
Return
No
|T1| + |T2| = c
yes
Replace ( )
No
X not in B1 and B2 and |T1| + |B1| = c
yes
LRU node in B1 is MAX_F
No
yes
Discard LRU node in B1
Discard LRU node with MAX_F-1
No
X not in B1 and B2 and |T1| + |T2| + |B1| + B2|=2c
yes
MRU node in B2 is MAX_F
No
yes
Discard MRU -1 page in the B 2
Discard LRU node in B2
X not in B1 or B2
yes
Insert x to tail of T1 and set the reference bit to 0
No
X in B1
yes
P= min{p+max{1, |B1|/|B2|}, c} Move x to tail of T2. Set reference bit to 0
Return
No
P= max{p-max{1, |B1|/|B2|}, 0} Move x to tail of T2. Set reference bit to 0

Figure 3. Flowchart of the proposed M-CAR technique

The M-CAR Pseudocode is given below:

```
INITIALIZATION
Set a pointer p to 0
Initialize four empty lists: T1, B1, B2, and T2
Set a variable c as the maximum cache size
FUNCTION CAR(x):
Check if x is already in the cache x is in (T2 or T1):
Set reference bit of page x to 1
ELSE:
If cache is full, perform cache replacement length(T2) + length(T1) = c:
CALL replace()
IF x is not in (B2 or B1) and If B1 is full length (B1) + length (T1) = c,
 perform replacement:
IF LRU page in B1 has MAXF:
 Discard MAXF-1 page in B2
ELSE:
Discard least recently used page in B1
IF total cache size exceeds limit length (T1) + length (B1) + length (T2) + length (B2) =
 2 * c and x is not in (B2 or B1),
perform replacement:
```

```
Discard least recently used page in B2
IF cache directory miss x is not in (B2 or B1) and length (T1) + length (T2) + length (B1)
 + length (B2) = 2 * c, take action based on MRU page:
 IF page of MRU is MAXF:
Discard MRU-1 page in B2
ELSE:
Append x to T1
Set reference bit of page x to 0
FUNCTION replace():
IF B1 or B2 is full the length (B1) + length (T1) = c OR length (T1) + length (B1) +
 length (T2) + length (B2) = 2 * c, replace the oldest page:
IF length (B1) + length (T1) = c:
cache_to_replace := B1
ELSE:
cache_to_replace := B2
Remove the oldest page from cache_to_replace
```

3.2 Method implementation

The criteria used to judge the performance efficiency in the proposed algorithm is the hit ratio gained by (M-CAR) and original (CAR), because cache memory was basically invented to solve the wait-state problem by keeping a useful data block to achieve the highest reachable hit ratio.

The experimental environment to implement these techniques depended on coding by (c#) on computer system with (O.S: Win-10), CPU: 1.83 GHz (2-CPUs), M.M:3-GB (which has no impact on the obtained results)

The main parameter is the cache size and it will be at the range 20-100 (which will impact on results by the more it increases allows more item's repetitions which causes higher hit ratio).

The two algorithms operated on the same set of input files (data sets), and the performance of the proposed modification is measured using the hit ratio calculated from Equation 1 and presented as the program's final finding:

$$\text{Hit ratio} = \frac{\text{No. of cache hits}}{\text{No. of memory references in data set}} * 100 \qquad (1)$$

First, the method was implemented on Dataset-1 which contains (245) items as follows:

{2, 3, 4, 1, 2, 4, 140, 150, 3, 2, 1, 4, 6, 5, 170, 180, 190, 200, 5, 6, 1, 7, 3, 4, 2, 5, 6, 7, 4, 210, 220, 1, 3, 5, 7, 2, 4, 6, 1, 8, 6, 7, 2, 3, 4, 2, 3, 6, 7, 8, 230, 240, 250, 260, 270, 280, 290, 300, 7, 8, 1, 3, 5, 6, 2, 4, 1, 8, 6, 310, 320, 330, 340, 350, 1, 8, 7, 2, 4, 3, 8, 6, 5, 7, 1, 360, 370, 380, 390, 400, 410, 6, 5, 4, 3, 2, 1, 6, 1, 15, 2, 3, 15, 17, 16, 4, 17, 16, 7, 6, 1, 5, 15, 17, 8, 5, 21, 16, 15, 17, 2, 3, 16, 21, 17, 15, 16, 4, 7, 6, 7, 3, 2, 17, 21, 23, 24, 8, 15, 12, 17, 19, 21, 16, 4, 1, 2, 3, 8, 19, 8, 7, 6, 1, 2, 3, 4, 5, 6, 7, 19, 18, 17, 16, 2, 13, 1, 2, 3, 4, 5, 21, 22, 18, 16, 14, 12, 2, 4, 6, 1, 2, 3, 4, 5, 18, 19, 20, 21, 22, 23, 24, 10, 11, 12, 13, 14, 15, 9, 6, 7, 6, 5, 4, 3, 2, 1, 1, 5, 15, 17, 8, 5, 21, 16, 15, 17, 2, 3, 16, 21, 17, 15, 16, 4, 7, 6, 7, 3, 2, 17, 21, 23, 24, 8, 15, 12, 17, 19, 21, 16, 4, 1, 2, 3, 8}

Implementing the proposed method using Dataset-1 produces the hit ratios shown in Figure 4.

Next, the method was implemented on Dataset-2, which contains (270) items as follows:

{380, 390, 400, 410, 6, 5, 4, 3, 2, 1, 6, 1, 15, 2, 3, 15, 17, 16, 4, 17, 16, 7, 6, 1, 5, 15, 17, 8, 5, 4, 1, 2, 3, 8, 19, 8, 7, 6, 1, 2, 3, 4, 5, 6, 7, 19, 18, 17, 16, 2, 13, 1, 2, 3, 4, 5, 21, 22, 18, 16, 14, 210, 220, 1, 3, 5, 7, 2, 4, 6, 1, 8, 6, 7, 2, 3, 4, 2, 3, 6, 7, 8, 230, 240, 250, 260, 270, 280, 290, 21, 16, 15, 17, 2, 3, 16, 21, 17, 15, 16, 4, 7, 6, 7, 3, 2, 17, 21, 23, 24, 8, 15, 12, 17, 19, 21, 16, 12, 2, 4, 6, 1, 2, 3, 4, 5, 18, 19, 20, 21, 22, 23, 24, 10, 11, 12, 13, 14, 15, 9, 6, 7, 6, 5, 4, 3, 2, 2, 3, 4, 1, 2, 4, 140, 150, 3, 2, 1, 4, 6, 5, 170, 180, 190, 200, 5, 6, 1, 7, 3, 4, 2, 5, 6, 7, 4, 300, 7, 8, 1, 3, 5, 6, 2, 4, 1, 8, 6, 310, 320, 330, 340, 350, 1, 8, 7, 2, 4, 3, 8, 6, 5, 7, 1, 360, 370, 1, 5, 7, 2, 4, 6, 1, 8, 6, 7, 2, 3, 4, 2, 3, 6, 7, 8, 230, 240, 250, 260, 270, 280, 290, 5, 6, 2, 4, 1, 8, 6, 310, 320, 330, 340, 350, 1, 8, 7, 2, 4, 3, 8, 6, 5, 7, 1, 360, 370, 380, 390, 400, 410, 6, 5, 4, 3, 2, 1, 6, 1, 15, 2, 3}

When the proposed method is implemented using Dataeset-2, the results in Figure 5 are obtained.

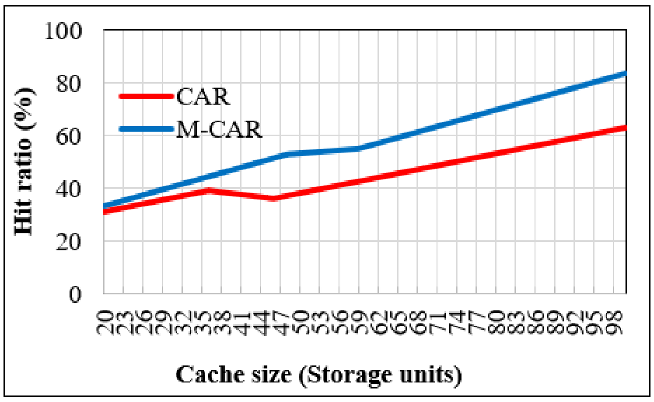


Figure 4. Hit ratio for CAR & M-CAR chart for Data set-1

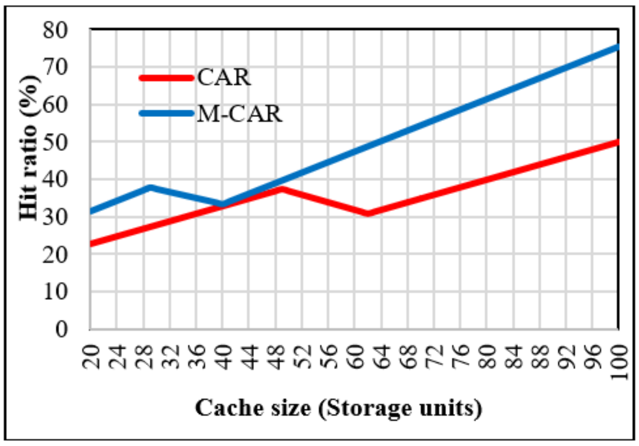


Figure 5. Hit ratio for CAR & M-CAR chart for Data set-2

4. RESULTS AND ANALYSIS

Based on the results obtained in the previous section, the proposed method achieved the following advantages:

A. Numerical results:

1. As seen in Figure 4 and Figure 5, the proposed M-CAR method achieved a hit ratio 91% and 76% higher than that achieved with traditional CAR when the dataset contained 245 and 270 items, respectively. This means that overall, as an average, the proposed M-CAR is capable of achieving a hit ratio 83.5% higher than that achieved using a traditional CAR.

2. While the performance of both methods dropped with the increase in the number of elements in the dataset, that drop was

less pronounced for M-CAR compared with that of the traditional CAR. By keeping the MAXF data block and replacing the MXF-1 block instead, the proposed modification makes a more realistic prediction of future CPU requests.

3. To assess the statistical significance of the increase in hit ratio after applying the (M-CAR) technique, we calculated the hit ratios of the (CAR) and (M-CAR) approaches for each dataset using a two-sample t-test with unequal variances and a two-tailed distribution. The obtained p-values were (1.97403E-10) for dataset-1 and (5.54347E-14) for dataset-2, demonstrating a substantial difference in hit rates between the old and new approaches for both datasets.

B. Results based on working mechanism and structure:

1. (M-CAR) is dynamically higher than (CAR).

2. (M-CAR) is scan resistance which makes it a fast technique.

3. Mathematical and technical complexity in (M-CAR) has not been affected and it is the same as in (CAR).

4. (M-CAR) is more accurate than (CAR) in selecting the candidate cache block for eviction based on two variables (recency and frequency), as well as the necessary modification step (moving MAXF-1 cache blocks in certain instances).

5. CONCLUSIONS

This study proposed a modified CAR (M-CAR) approach that achieves higher performance compared with the original approach through maximizing the hit ratio. The obtained results revealed that the hit ratios achieved using M-CAR are noticeably higher, increasing by an average of 83.5% compared with those obtained using typical CAR. We therefore conclude that the M-CAR is a high-performance cache replacement algorithm, and by default, achieving higher hit ratio means that the miss ratio has been minimized, which is the primary study objective for which it was found. In the same time, as a potential limitation of M-CAR, mathematical complexity (caused by the highly dynamic updating for F, R, MAXF and MAXR counters) might be a challenge for the proposed system that can slow down the overall performance. So, as a future improvement, a third factor can be added to this system, such as the Low Inter-reference Recency Set algorithm (LIRS) that can keep track any changes in its counters without extra time.

ACKNOWLEDGMENT

This research has been supported by the Computer Science Department /College of Education / Mustansiriyah University we strongly express our gratitude to the Department of Computer Science for the continuous and diligent support for us in this work

REFERENCES

[1] Hasoon, J.N., Hassan, R. (2019). Solving job scheduling problem using fireworks algorithm. Journal of Al-Qadisiyah for Computer Science and Mathematics, 11(2): 1-8. https://doi.org/10.29304/jqcm.2019.11.2.557

[2] Supase, S.S., Pansare, J.R. (2023). A robust, preference-based coordinator election algorithm for distributed systems. Ingénierie des Systèmes d'Information, 28(4): 843-851. https://doi.org/10.18280/isi.280405

[3] Balagoni, Y., Rao, R.R. (2018). SAGS: A SLA-aware green scheduling in heterogeneous cloud using hadoop YARN. International Journal of Intelligent Engineering and Systems, 11(6): 108-117. https://doi.org/10.22266/ijies2018.1231.11

[4] Banday, M.T., Khan, M. (2014). A study of recent advances in cache memories. In 2014 International Conference on Contemporary Computing and Informatics (IC3I), Mysore, India, pp. 398-403. https://doi.org/10.1109/IC3I.2014.7019786

[5] Jasim, B.H., AL-Aaragee, A.M.J., Alawsi, A.A.A., Dakhil, A.M. (2023). A heuristic optimization approach for the scheduling home appliances. Bulletin of Electrical Engineering and Informatics, 12(3): 1256-1266. https://doi.org/10.11591/eei.v12i3.3989

[6] Ahmed, T.I.O., Elamin, E.M. (2018). Design strategy of cache memory for computer performance improvement. International Journal of Research, 4(3): 12-17. http://doi.org/10.20431/2454-9436.0403002

[7] Suppiah, Y., Bhuvaneswari, T., Yee, P.S., Yue, N.W., Horng, C.M. (2022). Scheduling single machine problem to minimize completion time. TEM Journal, 11(2): 552-556. https://doi.org/10.18421/TEM112-08

[8] Mlinarić, D. (2020). Challenges in dynamic software updating. TEM Journal, 9(1): 117-128. https://doi.org/10.18421/TEM91-17

[9] Ogrutan, P., Aciu, L.E. (2017). Laboratory works designed for developing student motivation in computer architecture. TEM Journal, 6(1): 3-10. https://doi.org/10.18421/TEM61-01

[10] Al-Atbee, O.Y.K., Abdulhassan, K.M. (2023). A cascade multi-level inverter topology with reduced switches and higher efficiency. Bulletin of Electrical Engineering and Informatics, 12(2): 668-676. https://doi.org/10.11591/eei.v12i2.4138

[11] Zhen, C., Li, K. (2009). Memory management research based on real-time database. In 2009 International Conference on Test and Measurement, Hong Kong, China, pp. 416-419. https://doi.org/10.1109/ICTM.2009.5412904

[12] Prongnuch, S., Sitjongsataporn, S., Wiangtong, T. (2020). A heuristic approach for scheduling in heterogeneous distributed embedded systems. International Journal of Intelligent Engineering and Systems, 13(1): 135-145. https://doi.org/10.22266/ijies2020.0229.13

[13] Raghuvanshi, D. (2018). Memory management in operating system. International Journal of Trend in Scientific Research and Development, 2(5): 2346-2347. https://doi.org/10.31142/ijtsrd18342

[14] Mohialden, Y.M., Hussien, N.M., Hameed, S.A. (2022). Review of software testing methods. Journal La Multiapp, 3(3): 104-112. https://doi.org/10.37899/journallamultiapp.v3i3.648

[15] Salih, N.A.J., Altaie, H.T.R., Al-Azzawi, W.K., Mnati, M.J. (2023). Design and implementation of a driver circuit for three-phase induction motor based on STM32F103C8T6. Bulletin of Electrical Engineering and Informatics, 12(1): 42-50. https://doi.org/10.11591/eei.v12i1.4276

[16] Zaitar, Y. (2022). Analyzing the contribution of ERP systems to improving the performance of organizations. Ingénierie des Systèmes d'Information, 27(4): 549-556. https://doi.org/10.18280/isi.270404

[17] Pappas, C., Moschos, T., Alexoudi, T., Vagionas, C., Pleros, N. (2022). Caching with light: First demonstration of an optical cache memory prototype. In Optical Fiber Communication Conference, San Diego, California, USA, pp. Th4B-3. https://doi.org/10.1364/OFC.2022.Th4B.3

[18] Aalsaud, A., Rafiev, A., Xia, F., Shafik, R., Yakovlev, A. (2018). Model-free runtime management of concurrent workloads for energy-efficient many-core heterogeneous systems. In 2018 28th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), Platja d'Aro, Spain, pp. 206-213. https://doi.org/10.1109/PATMOS.2018.8464142

[19] Jaber, S., Ali, Y., Ibrahim, N. (2022). An automated task scheduling model using a multi-objective improved cuckoo optimization algorithm. International Journal of Intelligent Engineering & Systems, 15(1): 295-304. https://doi.org/10.22266/ijies2022.0228.27

[20] Ilhem, B., Elamin, B.M., Ahmed, L., Salaheddine, B. (2022). 3D modelling of the mechanical behaviour of magnetic forming systems. Bulletin of Electrical Engineering and Informatics, 11(4): 1807-1817. https://doi.org/10.11591/eei.v11i4.3628

[21] Sadiq, A.T., Abdullah, H.S., Ahmed, Z.O. (2018). Solving flexible job shop scheduling problem using meerkat clan algorithm. Iraqi Journal of Science, 59(2A): 754-761. https://doi.org/10.24996/ijs.2018.59.2A.13

[22] Kareem, E.I.A., Hussein, S.A. (2022). Optimal CPU jobs scheduling method based on simulated annealing algorithm. Iraqi Journal of Science, 63(8): 3640-3651. https://doi.org/10.24996/ijs.2022.63.8.38

[23] Salem, R., Abdel-Moneim, W., Hassan, M. (2021). Fast Local Flow-based Method using Parallel Multi-core CPUs Architecture. International Journal of Intelligent Engineering & Systems, 14(4): 1-10. https://doi.org/1010.22266/ijies2021.0831.01

[24] Arumalla, A., Makkena, M.L. (2016). An effective implementation of dual path fused floating-point add-subtract unit for reconfigurable architectures. International Journal of Intelligent Engineering and Systems, 10(2): 40-47. https://doi.org/10.22266/ijies2017.0430.05

[25] Belmahdi, R., Mechta, D., Harous, S. (2021). A survey on various methods and algorithms of scheduling in Fog Computing. Ingénierie des Systèmes d'Information, 26(2): 211-224. https://doi.org/10.18280/isi.260208