Moving Average Re-Reference Interval Prediction

Vinayak Agarwal
New York University
va2083@nyu.edu

Neeraj Yadav
New York University
ny736@nyu.edu

Abstract

As the microprocessors are getting faster, often memory and network communication are bottleneck. Fetching Data from DRAM incurs overhead. The long latencies of DRAM memory accesses can be avoided by storing the data in cache which is much faster. But the cache size is limited. Increasing the cache size often leads to increase in latencies. Moreover, a fully associative cache can be used to add more flexibility but this increases the hardware complexity and power consumption. Hence, an optimal cache replacement policy is a very feasible way to optimize the usage of cache.

We propose Moving Average Re-Reference Interval Prediction (MARRIP) that dynamically chooses an optimal policy depending on the workload between scan resistant Static Re-Reference Interval Prediction (SRRIP)[5] and thrash resistant Bimodal re-reference interval prediction policy (BRRIP)[5]. We evaluate on a set of 10 programs from the SPEC CPU2006, and found out that MARRIP outperfroms DRRIP and LRU on the thrashing mcf program.

Keywords: Replacement, Moving Average, Scan Resistance, Thrashing, Shared Cache

1 Introduction

As accessing data from DRAM is very time consuming and we don't store a lot of data in cache. The only viable option is to only store the meaningful data in cache. We would like to have that data in cache which is frequently accessed by the program and at the same time would like to throw the less frequently used data out of the cache. Hence the term cache replacement develops where researchers are intending to replace the data with lower re-usability. But for this opens many questions before us:

- How are we going to decide what data to throw?
- How are we going to prioritize the data?
- How to handle program which uses data in cyclic patterns, with varied cycle lengths, may be greater than the cache blocks?

We explore these questions in the paper and look at the previous research papers rigorously. We propose a novel method MARRIP. Our algorithm aims to develop a policy that is robust to mixed access patterns like scanning and thrashing patterns. To achieve this, we use a dynamic policy that achieve this by optimally choosing a suitable policy between DRRIP and BRRIP.

2 Literature Survey

Cache replacement policy primarily involves three important steps: block insertion, priority promotion and victim selection. Block insertion decides the position to insert a block into the cache. Priority promotion focuses on placement of the block based on hit. Victim selection decides the block to evict if the cache is fully occupied.

Based on block insertion strategy, [1] divides the cache replacement policies into two categories: Coarse-Grained policies and Fine-Grained policies. Coarse-Grained policies doesn't differentiate the blocks at the time of insertion. They manipulate the priority of block only during hit or miss events. Fine-Grained policies uses historical data and attempts to predict the optimal place for the blocks during insertion. Similar to Coarse-Grained policies, Fine-Grained policies also use suitable priority promotion policy. Coarse-Grained policies are further divided into three categories: Recency-Based policies, Frequency-Based policies and Hybrid Policies. Recency-Based policies prioritize over cache lines based on the their recency of access. Frequency-Based policies prioritize over cache lines based on their frequency of access. Hybrid policies assess the access pattern of cache lines over time and then dynamically choose a suitable coarse grained policy. Their adaptive nature makes them more robust to a mixed access patterns compared to the Recency and Frequency based policies. Fine-Grained policies are categorized further into Classification-Based policies and Reuse Distance-Based policies. Classification-Based policies label the block cache friendly or cache averesely before inserting it into cache, using its past access patterns. Reuse distance-Based policies predict the distance when a cache block is going to be use next and based on that predicted value, the insertion position is decided.

Least Recently Used (LRU) [2] which is a Recency-Based policy, inserts block at the most recently used (MRU) position. On a cache hit, the cache block is moved to the MRU position. Eviction is done at the LRU position. LRU performs well when cache line used recently is likely to be reused again in the near future. LRU suffers from thrashing when the working set exceed the cache size. For example,if cache lines are accessed in cyclic form like

$$(a_1, a_2, ...a_k)^N \quad k > \text{cache size}$$

and the number of elements in cycle are more than cache size. LRU achieves 0 cache hits in the above mentioned pattern.LRU also performs poorly during scans because it keeps the most recently used scans in the cache and evicts the older cache lines which are more likely to be reused.

Segmented LRU [3] which is a Recency-Based policy, divides the cache into two segments: protected segment and probationary segment. A cache line is first inserted in the probationary segment. If a cache line in probationary segment is accessed again then it is promoted to the MRU position of the protected segment. During promotion, if the protected segment overflows then the cache line at the LRU position of the protected segment is demoted to the MRU position of the probationary segment. If there is a hit in the protected segment then the cache line is moved to the MRU position of the protected segment. Eviction is done at the LRU position of the probationary segment. Segmented LRU solves scanning problem of LRU because it places the cache line accessed two or more than two times in protected segment and would evict the useless cache lines more aggressively, since the cache lines are first inserted into the probationary segment whose length is obviously less than the net cache length.

LRU Insertion policy (LIP)[4] which is a Recency-Based policy, uses exactly similar strategy for priority promotion and eviction. It differs from LRU at insertion. LIP inserts an incoming cache line at the LRU position. Hence, it minimizes the time spent by the non reused lines in cache and is thrash resistant. It achieves near optimal hit rates in the cyclic access patterns. For a reference stream with 'k' blocks which repeats itself N times,

$$(a_1, a_2, ...a_k)^N \quad k > \text{P (cache blocks)}$$

LIP achieves a hit rate of $(P-1)/k$, where P is the number of cache blocks and outperforms LRU which has a hit rate of 0.

Bimodal Insertion Policy (BIP)[4] which is a Recency-Based policy, inserts the incoming cache lines at MRU position with a very low probability ϵ and inserts at LRU position with probability $1-\epsilon$. Since most of the times BIP inserts at LRU position, hence it preserves the thrash resistant property of LIP. As BIP also inserts at MRU position, this helps in handling phase changes which LIP fails to do. For example, consider two consecutive reference streams:

$$(a_1, a_2, ...a_k)^N (b_1, b_2, ...b_k)^N \quad k > \text{P (cache blocks)}$$

For the second stream $(b_1, b_2, ...b_k)^N$, LIP performs poorly and achieves a hit rate of 0. This is because LIP doesn't age older lines. Hence, the incoming cache lines of stream $(b_1, b_2, ...b_k)^N$ are never placed in a non-LRU position. When b_i arrives it is inserted at LRU position and as b_{i+1} arrives b_i is evicted. To insert a cache line into non-LRU position, the cache line must have two successive hits which is not the case. For the both streams $(a_1, a_2, ...a_k)^N$ and $(b_1, b_2, ...b_k)^N$, BIP outperforms LIP and achieves a hit rate of $(P-1-\epsilon.[k-P])/k$ which is approximately equal to $(P-1)/k$. Since BIP probabilistically inserts at MRU position, when second stream arrives after few misses the entire cache would be populated by the cache blocks from the second stream.

Static Re-Reference Interval Prediction (SRRIP) [5] which is a Recency-Based policy, assigns a Re-reference Prediction Value (RRPV) to the cache lines. RRPV ranges from 0 to 3 where a low RRPV means that cache line is more likely to be reused next. SRRIP inserts the incoming cache line at RRPV of 2 and on a cache hit it promotes the cache line to a RRPV of 0. Eviction is made at RRPV of 3. If there are no cache line with RRPV of 3, then the RRPV of all cache lines are incremented. SRRIP is scan resistant because it inserts the at RRPV of 2 thus preventing the scan streams (which are likely to be referenced again after a long time) to evict the cache lines with a near re-reference interval. Comparing SRRIP to LRU, LRU can thought of a very optimistic policy which inserts assuming that the cache line is most likely to be reused next. In other words, it can though of as an insert with RRPV of 0. Comparing SRRIP to LIP, LIP is a very pessimistic policy which assumes that inserted line is highly unlikely to be reused next. It can also be considered as an insert with RRPV of 3. Hence SRRIP, is a superior to both of them which inserts at somewhat in middle i.e. RRPV of 2 assuming that the cache line have some intermediate chances to be reused again. However, SRRIP suffers from thrashing.

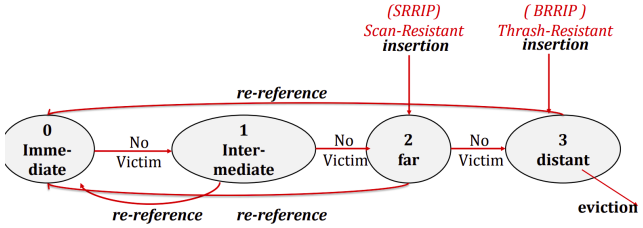

Figure 1. BRRIP, Image source[6]

Bimodal re-reference interval prediction policy (BRRIP)[5] which is a Recency-Based policy, probabilistically makes an insertion decision as shown in Figure 1. It inserts the incoming cache line at RRPV of 2 with probability ϵ and at RRPV of 3 with probability $1-\epsilon$. Eviction and promotion of BRRIP is similar to SRRIP. Since BRRIP inserts at RRPV of 3 mostly, it is thrash resistant. However, for mixed access patterns using BRRIP degrades the performance because it is mostly inserting at RRPV of 3 and would give poor results if there are recency friendly patterns.

Least frequently used (LFU) policy [7] is a Coarse-Grained Frequency based policy. It maintains a frequency counter for each cache line. On insertion, the frequency counter is initialized to 0 and it is incremented on hit. On a cache miss, line with least frequency counter is evicted. LFU performs

poorly in case of mixed access patterns because frequency count can become very large for some cache lines in some access patterns and they might be not used at all in the subsequent accesses, leading to a wastage of cache space.

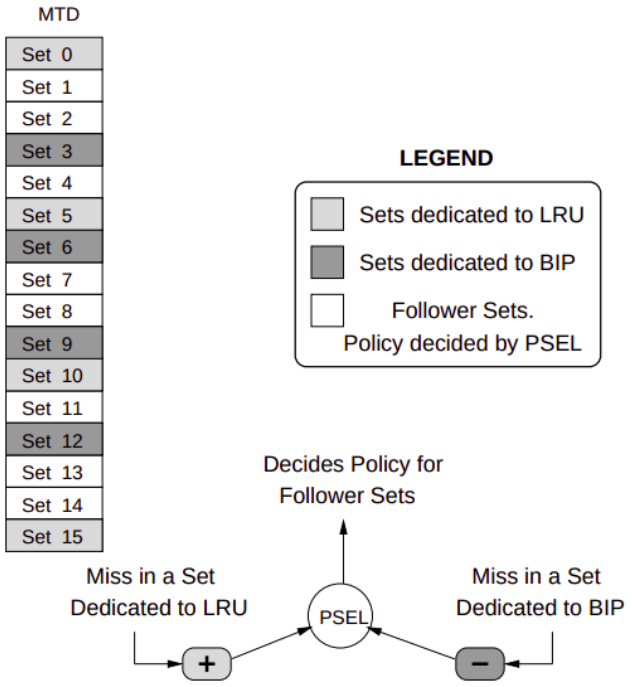

Figure 2. DIP, Set Dueling [4]

Dynamic Insertion Policy (DIP)[4] is a Coarse-Grained Hybrid policy. It dynamically selects the best insertion policy between LRU and BIP. Since LRU is recency friendly and BIP is thrash resistant, DIP achieves benefits from both by optimally choosing the most suitable policy. DIP uses set dueling [8] to achieve that. Set Dueling reduces the hardware overhead uses few sets to capture the behaviour of different policies on the entire cache. In case of DIP, the set dueling mechanism dedicates few sets to LRU and few sets to BIP as shown in Figure 2. These dedicated sets are called set dueling monitors (SDMs) and the remaining sets are called follower sets. When a set dedicated to LRU receives a hit the policy selection (PSEL) saturating counter is incremented whereas when a set dedicated to BIP receives a hit PSEL is decremented. If the MSB of PSEL is 0 then the follower sets uses LRU policy otherwise follower sets uses BIP policy.

Dynamic Re-Reference Interval Policy (DRRIP) [5] is a Coarse-Grained Hybrid policy. DRRIP uses set dueling to dynamically choose between the BRRIP and SRRIP. Since BRRIP is thrash resistant and SRRIP is scan resistant, DRRIP achieves robustness to mixed access pattern by choosing optimal policy between the two.

Hawkeye[8] is a Fine-Grained Classification based policy. It has two main componenets: OPTgen which produces the Belady's optimal solution [9] and a program counter based classifier as shown in figure 3. While Belady's optimal algorithm requires future information to know whether a cache line is good/bad, OPTgen is a practical version of Belady requiring a historical window of 8 × cache size to generate optimal caching decision. OPTgen uses set dueling to reduce the size of historical data. Hawkeye predictor is then trained using the optimal training label generated by OPTgen. Hawkeye Predictor is based on the assumption that cache lines loaded by the same program counter would have similar behaviour. During training, it would look at the label of the cache line loaded by the PC and would assign a positive value to PC if the line loaded is cache friendly and a negative value if it is cache averesely. During inference, a cache line is claimed to be cache friendly or cache averesely based on the program counter that loaded the line.

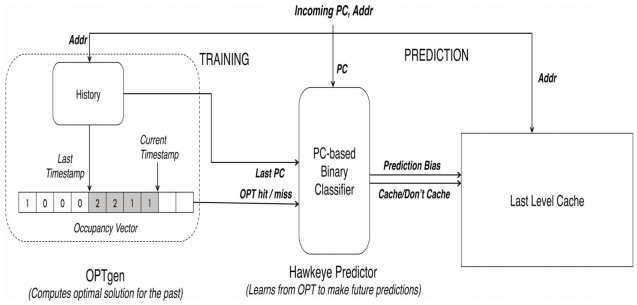

Figure 3. Hawkeye [8]

Zhan shi et al. [9] proposed a Fine grained policy utilising the power of Deep Learning and Machine Learning. Deep Learning models takes a lot of time in training. This can be solved by training the models offline. But in a complex model, inference would involve lots of floating point matrix multiplications and take time in the order of milliseconds while we need to make a decision in the order of nanoseconds. Moreover, the offline trained model won't be able to capture the phase changes entirely. Zhan shi et. al solves this problem by offline training an attention based LSTM model. They derive important features and insights by understanding the change in distribution of weights of attention layers. They exploits the key facts that prediction accuracy is insensitive to the order sequence of control flow and only few elements in control flow can be used instead of full control flow without impacting the accuracy. Based on these insights, they handcrafted features and proposed an online Integer based SVM model. As SVM model is online, it dynamically trains and adapts to phase changes.

LeCaR [10] is an online reinforcement learning algorithm which aims to minimize regret and maximize reward. LeCaR optimally choose eviction policy between two traditional experts, LRU and LFU. LeCaR updates the weights assigned to LRU and LFU dynamically using a fixed learning rate.

CACHEUS [11] builds on LeCaR and uses an adaptive learning rate with better experts, Scan Resistant LRU and Churn Resistant LFU.

3 Proposed Idea: MARRIP

As the offline machine learning models can't adapt to phase changes in the program. Also, the Recency based policies like LRU, LIP, BIP, SRRIP and BRRIP struggles with the mixed accessed patterns. Zhan shi et al. outperformed Hawkeye on several workloads and achieved good results with their LSTM model over mixed access pattern but the model took several milliseconds during inference which forced them to use an Integer based SVM model. Also, LSTM models are state-of-the-art in predicting temporal sequences. We take inspiration from that and use a relatively simpler statistical approach of moving average.

Moving average is traditionally used in time series analysis, prior to LSTM. LSTM model is prone to over-fitting, uses large amount of data to train and incurs extra hardware cost. Moving Average is simpler, fast and require less amount of data.

As DRRIP dynamically chooses between SRRIP and BRRIP using set-dueling. We propose MARRIP- Moving Average Re-Reference Interval, an alternative way to dynamically choose the optimal policy between SRRIP and BRRIP by leveraging the moving average model. We use a fixed sized sliding window mechanism, that counts the number of hits and misses in the window. We then compare the hit rate with the MAHCT (Moving Average Hit Count Threshold). If the hit rate is greater than MAHCT, we use SRRIP. While if the hit rate is less than MAHCT, we use BRRIP. Our hypothesis is that, if the cache lines in the current window are having a higher hit rate than the MAHCT threshold, then it is likely that the incoming cache line is going to be reused again. Hence we make an insertion at RRPV of 2. While if, the current window hit rate is less than the threshold, then we assume that the incoming cache line is less likely to be resused again. Hence, we use BRRIP policy which inserts at RRPV of 3 with probability $1 - \epsilon$.

Eviction is made at the max RRPV value i.e. 3. If there are no cache lines with RRPV of 3, then the RRPV values of all the cache lines are incremented till we have a cache line with RRPV of 3. On a cache hit, the cache line line is promoted to a RRPV of 0.

To make MARRIP robust to the mixed access patterns, we dynamically choose the optimal policy between scan resistant SRRIP and thrash resistant BRRIP. To add more robustness to the algorithm we maintain the windows history. So instead of making a decision using current window, we look at the previous 'k' windows. We call this 'k' as MAHistory. MARRIP lies in fine grained reuse distance prediction algorithm, since it uses historical data to find the optimal insertion place.

4 Experimental Setup

We use ChampSim, a pin based trace-driven x86 simulator for our performance studies. Our LLC is 2048 set and 16 way cache making it a 256KB cache. The replacement policy only applies to the LLC and not L1 or L2 cache. The traces used for the experiments are the SPEC CPU2006 traces, since they cover a wide variety of access patterns. The implementations used to benchmark are the standard implementations of LRU and DRRIP provided with the ChampSim simulator. The MARRIP simulator is based on the DRRIP policy. For the execution of all algorithms for all data sets, a 1 million instruction warmup was performed followed by 10 million instruction execution.

5 Results and Analysis

First, we define a base algorithm and then compare the misses of the algorithm under consideration with respect to the base algorithm using the metric Cache Miss % reduction.

$$\text{Cache Miss \% reduction} = \frac{(M_b - M_1)}{M_b} * 100 \qquad (1)$$

where M_1 is the misses per kilo instructions of algorithm under consideration and M_b is the misses per kilo instructions of base algorithm. We expect to have a large positive value of the metric Cache Miss % reduction.

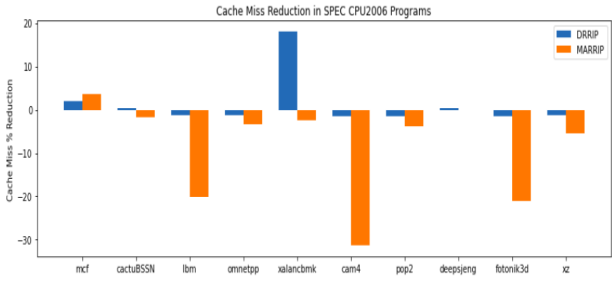

Figure 4. The cache miss % reduction using LRU as the base algorithm.

Figure 4 shows how MARRIP performs when compared to the base algorithm, LRU. We also compare the standard DRRIP implementation, provided in champsim simulator with LRU. Our algorithm beats both DRRIP and LRU in case of mcf program. LRU behaves poorly in case of mcf due to thrashing. We looked at paper [5] and found out that the MPKI of mcf is very high till 3.5 MB of cache size. Since we are using only 256 KB of cache size, the MPKI would be even more. This is because the working set of mcf is quite big and this leads to thrashing in LRU. Here, our algorithms optimally chooses BRRIP and proves to be thrash resistant. Since DRRIP is also thrash resistant. It also does better compared to LRU. For the program xalancbmk, DRRIP outperforms LRU. This indicates that either the program is of scan type or thrashing type. In

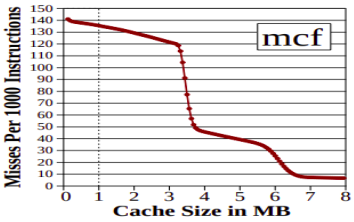

Figure 5. mcf MPKI vs cache size, source[5]

this our algorithm does poor than the DRRIP, this may be due to sub-optimal decisions. Program ibm, xz, pop2 are recency friendly. Since LRU not only outperforms MARRIP but also DRRIP. This is because LRU inserts at the MRU position or in terms of DRRIP at the RRPV value of 0 where as DRRIP and MARRIP inserts at RRPV of 2 or 3 and leads to an early eviction compared to LRU.

Also, MARRIP performs poor as compared to DRRIP for most of the programs. MARRIP outperformed DRRIP on a thrashing program but since other programs are recency friendly, MARRIP is doing worse comapred to DRRIP on these programs. This is because MARRIP is mainly choosing BRRIP policy and inserting the incoming cache line at RRPV of 3 which leads to early eviction, causing poor performance in recency friendly patterns.

We investigate further below by seeing how the algorithm performs when varying its parameters.

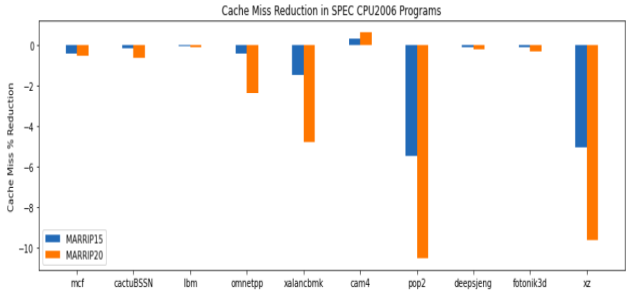

Figure 6. The cache miss % reduction using MARRIP10 as the base algorithm.

MARRIPX represents MARRIP, with X as the MARRIP threshold, that determines at what cache hit rate, the algorithm switches between SRRIP and BRRIP. From the Fig 6, we see that increasing the threshold causes an increase in cache misses except in the case of cam4. Since if the window hits are less than the threshold then insertion is made using BRRIP policy which mainly insert at RRPV of 3. Hence, as we increase the threshold, most of the incoming cache lines would be inserted at the RRPV of 3. Hence a very high threshold doesn't make sense. Also, the programs like xz, pop2 which are recency friendly would impact negatively as they are inserted at a very low priority. While, if the threshold is low then there are chances that the incoming line would be inserted using SRRIP which inserts at RRPV of 2 and gives the cache lines more chances to stay in the cache. Hence the recency friendly cache lines would be favored by an insertion at RRPV of 2 compared to RRPV of 3.

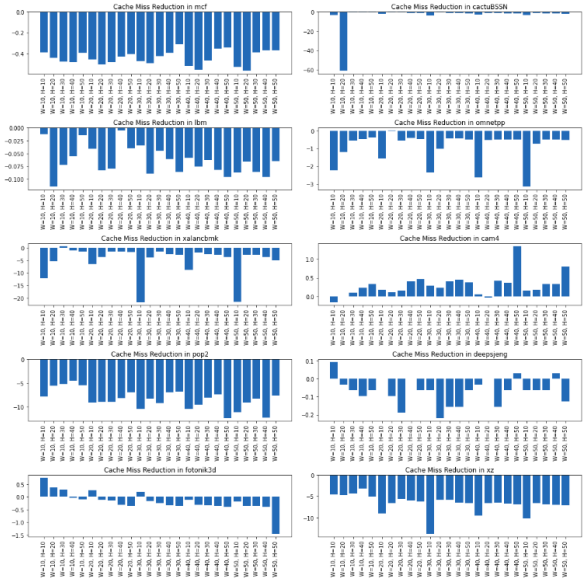

Figure 7. Cache Miss % Reduction by varying window size and history length in SPEC CPU2006 programs. The benchmark uses MARRIP10 W=10, H=50 as baseline.

To further investigate the mode switching, we vary the window and history lengths. In the Fig 7, W indicates the MA Window length and H denotes MAHistory length. We observe that in xalancbmk, if we keep the W constant, H=10 gives the worst performance. At the same time, in fotonik3d, we see that keeping W constant, H=10 gives better performance as compared to other MAHistory values. In cam4, the best performance is reached at W=40, H=50 while in omnetpp, the best performance is reached at W=20, H=20. This indicates that there is no single W=x, H=y value that can give the optimal performance for the algorithm. The performance may vary significantly across programs. It is also possible that the optimal window/history size may vary within a program as well, however it is not possible to test at such granularity.

6 Conclusions

From the experiments performed, we concluded that the MARRIP algorithm outperforms LRU for the thrashing workloads like mcf. Hence, MARRIP is thrash-resistant. This shows a positive hope that MARRIP can handle thrashing program more robustly. This gives an opportunity to use

MARRIP as a thrashing expert in reinforcement learning policies like LeCaR.

MARRIP doesn't do well compared to DRRIP for most of the programs. This shows that MARRIP is not able to optimally choose the correct policy and it is biased towards BRRIP. This shows that MA model is not able to learn the cache behaviour. Though LSTM is too complex to learn the temporal dependence, we need to try a model which is more complex than MA but less than LSTM. Other models like ARMA, ARIMA can be explored to capture the cache behaviour dynamically.

7 References

1 Cache Replacement Policies, Akanksha Jain and Calvin Lin,2019,The University of Texas at Austin.

2 R. L. Mattson, J. Gegsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. IBM Systems Journal, 9(2):78–117, 1970.

3 Ramakrishna Karedla, J. Spencer Love, and Bradley G. Wherry. Caching strategies to improve disk system performance. Computer, (3):38–46, 1994.

4 Moinuddin K. Qureshi, Aamer Jaleel, Yale N. Patt, Simon C. Steely, and Joel Emer. Adaptive insertion policies for high performance caching. In Proc. of the International Symposium on Computer Architecture (ISCA), pages 381–391, 2007.

5 Aamer Jaleel, Kevin B. Theobald, Simon C. Steely Jr., and Joel Emer. High performance cache replacement using re-reference interval prediction (RRIP). In Proc. of the International Symposium on Computer Architecture (ISCA), pages 60–71, 2010b.

6 Modern Memory Systems, Biswabandan Panda www.cse.iitk.ac.in/users/biswap/CS698Y/lectures/L7.pdf

7 Edward Grady Coffman and Peter J. Denning. Operating Systems Theory, vol. 973, Prentice Hall, Englewood Cliffs, NJ, 1973.

8 Akanksha Jain and Calvin Lin. Back to the future: Leveraging belady's algorithm for improved cache replacement. In Proc. of the International Symposium on Computer Architecture (ISCA), June 2016.

9 Laszlo A. Belady. A study of replacement algorithms for a virtual-storage computer. IBM Systems Journal, pages 78–101, 1966.

10 G. Vietri, L. V. Rodriguez, W. A. Martinez, S. Lyons, J. Liu, R. Rangaswami, M. Zhao, and G. Narasimhan. Driving Cache Replacement with ML-based LeCaR. In Proceedings of the USENIX Workshop on Hot Topics in Storage Systems (HotStorage), June 2018.

11 Liana V. Rodriguez and Farzana Yusuf and Steven Lyons and Eysler Paz and Raju Rangaswami and Jason Liu and Ming Zhao and Giri Narasimhan. Learning Cache Replacement with CACHEUS. 19th USENIX Conference on File and Storage Technologies, FAST 2021.