CLOCK-Pro+: Improving CLOCK-Pro Cache Replacement with Utility-Driven Adaptation

Cong Li
Intel Corporation
cong.li@intel.com

ABSTRACT

CLOCK-Pro is the low-overhead approximation of the state-of-the-art cache replacement policy, Low Inter-Reference Recency Set (LIRS). It also improves the static cache space allocation in LIRS with simple heuristics to adapt to LRU-friendly workloads. However, the heuristics do not perform well in certain cases. Inspired by the idea of utility-driven adaptation from another state-of-the-art policy, CLOCK for Adaptive Replacement (CAR), we propose a new CLOCK-Pro+ policy. The new policy directly evaluates the utility of growing the number of cold pages against that of growing hot pages. It then dynamically adjusts the cache space allocation driven by the utility comparison. Experiments are performed on traces from the UMass Trace Repository as well as a synthetic trace drawn from a stack-depth distribution. While sometimes CLOCK-Pro substantially outperforms CAR and sometimes vice versa, the new CLOCK-Pro+ policy consistently performs close to the winner between the two in all the cases.

CCS CONCEPTS

• **Information systems** → *Storage management*; • **Theory of computation** → *Caching and paging algorithms*;

KEYWORDS

Page replacement, CLOCK-Pro, CAR, dynamic cache space allocation

ACM Reference Format:
Cong Li. 2019. CLOCK-Pro+: Improving CLOCK-Pro Cache Replacement with Utility-Driven Adaptation. In *The 12th ACM International Systems and Storage Conference (SYSTOR '19), June 3–5, 2019, Haifa, Israel.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3319647.3325838

1 INTRODUCTION

Performance of cache replacement is crucial to a variety of systems, including, e.g., disk drives [17], file systems [14], middleware [6], databases [18], web servers [3], etc. A wide range of replacement policies have been proposed, e.g., Least-Recently Used (LRU) [4], Least-Frequently Used (LFU) [4], 2Q [11], LRU-K [15], TinyLFU [7, 8], FRD [7, 16], hyperbolic caching [2], LeCaR [19], etc. Most of those policies manipulate the data structure whenever there is a cache hit. The need to serialize the manipulations introduces a lock contention problem. For many policies based on stack data structures, CLOCK-style approximations are proposed to resolve the lock contention problem [1, 5, 9]. When there is cache hit, a reference bit is set for the corresponding page.[1] Further processing is deferred to the stage of a cache miss.

CLOCK-Pro [9] is the low-overhead CLOCK-style approximation to Low Inter-Reference Recency Set (LIRS) [10], one of the state-of-the-art replacement policies. Similar to LIRS, it uses reuse distance as the predictive indicator on the likelihood of a page's future access to discriminate hot pages from cold pages. Different from the static allocation of most cache space to hot pages in LIRS, CLOCK-Pro tracks the accesses of cold pages and the test period termination of cold pages to dynamically allocate the cache space to cold pages against hot pages. The simple heuristics are expected to improve its performance in LRU-friendly workloads where reuse distance is not a good access pattern predictor. While being simple, sometimes the dynamic adaptation does not perform well, but performs even worse when a considerable number of cold page accesses are combined with a large number of cold page test period termination. An ablation study on the effect of adaptation is not available in [9].

CLOCK with Adaptive Replacement (CAR) [1] is the low-overhead approximation to Adaptive Replacement Cache

[1]It is typically done by modern hardware in page replacement for virtual memory management.

```
Variables
  $s$: cache size in number of resident pages
  $\tilde{C_r}$: target number of resident cold pages
  $\tilde{H}$: target number of hot pages
Subroutine Adapt(Event $e$)
  If $e$ is an access on a non-resident cold page then
    $\tilde{C_r} \leftarrow \tilde{C_r} + 1$
  If $e$ is observing the reference bit set on a resident cold page in test
      period then
    $\tilde{C_r} \leftarrow \tilde{C_r} + 1$
  If $e$ is test period termination of a cold page then
    $\tilde{C_r} \leftarrow \tilde{C_r} - 1$
  $\tilde{H} \leftarrow s - \tilde{C_r}$
```

Figure 1: Adaptation algorithm in CLOCK-Pro.

```
Variables
  $s$: cache size in number of resident pages
  $T_1$: recency CLOCK
  $T_2$: frequency CLOCK
  $B_1$: shadow recency list
  $B_2$: shadow frequency list
Subroutine Adapt(Event $e$)
  // Calculate the utility of growing $T_1$ with a hypothetical access in $B_1$
  $U_1 \leftarrow \frac{1}{|B_1|}$
  // Calculate the utility of growing $T_2$ with a hypothetical access in $B_2$
  $U_2 \leftarrow \frac{1}{|B_2|}$
  If $e$ is an access in $B_1$ then
    $|T_1| \leftarrow |T_1| + \min\{1, \frac{U_1}{U_2}\} = |T_1| + \min\{1, \frac{|B_2|}{|B_1|}\}$
    $|T_2| \leftarrow s - |T_1|$
  If $e$ is an access in $B_2$ then
    $|T_2| \leftarrow |T_2| + \min\{1, \frac{U_2}{U_1}\} = |T_2| + \min\{1, \frac{|B_1|}{|B_2|}\}$
    $|T_1| \leftarrow s - |T_2|$
```

Figure 2: Adaptation algorithm in CAR.

(ARC) [13], another state-of-the-art policy. CAR maintains a frequency CLOCK data structure and a recency CLOCK data structure. It measures the utilities to increase the sizes of the two data structures and dynamically adjusts the cache space allocation through utility comparison. Its sophisticated adaptation approach addresses the weakness of CLOCK-Pro, but its frequency CLOCK is less capable in capturing the weak locality without the fine-grained metric of reuse distance in CLOCK-Pro.

Inspired by the idea of CAR, we propose a novel CLOCK-Pro+ policy. The new policy monitors the accesses to the non-resident cold pages to evaluate the utility of growing the number of cold pages. It also observes the accesses to the cold pages demoted from hot pages to evaluate the utility of growing the number of hot pages. The utility comparison is then used to guide the dynamic allocation of the cache space to cold pages against hot pages. We perform empirical evaluation of the policies on traces from the UMass Trace Repository along with a synthetic stack depth distribution (SDD) [4] trace with different cache space configurations. Experimental results indicate that while sometimes CLOCK-Pro outperforms CAR with a substantial margin and sometimes vice versa, the new CLOCK-Pro+ policy consistently performs close to the winner between the two in all the cases. A case study is carried out to demonstrate how the new policy overcomes the weakness of CLOCK-Pro.

2 BACKGROUND

As the low-overhead CLOCK-style approximation to LIRS, CLOCK-Pro [9] uses reuse distance as the fine-grained predictor of future page accesses. The reuse distance of a page is defined as the number of unique pages accessed between its two consecutive accesses. A smaller reuse distance of a page indicates that more likely it will be accessed in the future.

Similar to LIRS, CLOCK-Pro uses an efficient data structure to discriminate cold pages from hot pages. A new page is admitted as a *resident* cold page and is granted with a *test period*. If there is a cache miss, the LRU cold page is evicted and becomes a *non-resident* one.[2] A non-resident cold page is a shadow entry with its meta data kept. If a cold page (either resident or non-resident) in its test period gets accessed, it is promoted to a hot page. If the total number of hot pages is exceeded, the LRU hot page is demoted to a cold page. For any cold pages staying beyond the LRU hot page, their test periods get terminated and the non-resident ones are removed. If the total number of pages tracked goes beyond the limit, the test period of the LRU non-resident cold page is terminated and the page is removed.

Different from LIRS which statically devotes most of its cache space to hot pages, CLOCK-Pro adapts the cache space allocation dynamically. In some LRU-friendly workload, reuse distance may not be a good predictor of page access pattern.[3] Some misses on non-resident pages can be captured by increasing the number of cold pages. To guess whether the workload is LRU-friendly, CLOCK-Pro tracks the accesses to the cold pages and the test period termination of the cold pages. If a cold page (either resident or non-resident) in its test period is (or is observed to have been) accessed, the target cold page number is increased by one. If the test period of a cold page is terminated, the target cold page number is decreased by one. Figure 1 shows the adaptation algorithm. Being simple, the dynamic adaptation does not perform well when a considerable number of non-resident

[2] In CLOCK-style approximation, We only know the approximate order of accesses given the precise information lost.

[3] For example, in a request stream drawn from a stack depth distribution [4], pages accessed are kept in a stack where a page in a smaller stack depth is more likely to be accessed than that in a larger depth. This results in the pattern that new pages are accessed, stay in small depths in the stack, and get accessed again shortly. In the pattern, a new page without a valid reuse distance is very likely to get accessed.

Table 1: CLOCK-Pro vs. CAR on selected configurations.

Trace (# of pages cached)	CLOCK-Pro	CAR
WebSearch1 (131072)	**13.10%**	8.32%
WebSearch1 (262144)	**24.91%**	14.90%
WebSearch1 (524288)	**40.36%**	32.78%
WebSearch2 (262144)	**29.80%**	26.94%
WebSearch2 (524288)	**48.35%**	41.72%
WebSearch3 (262144)	**29.66%**	26.68%
WebSearch3 (524288)	**48.21%**	41.40%
Financial1 (512)	17.78%	**23.17%**
Financial1 (1024)	20.62%	**26.02%**
Financial1 (2048)	24.16%	**29.38%**
Financial1 (4096)	27.58%	**32.61%**
Financial1 (8192)	31.31%	**35.72%**
Financial1 (16384)	34.33%	**38.35%**
SDD (256)	17.10%	**20.40%**
SDD (512)	31.60%	**36.75%**

code page accesses are combined with a large number of cold page termination. [9] does not explicitly quantify the improvement of the heuristic adaptation with an ablation study.

CAR [1] is the CLOCK-style approximation to ARC [13]. CAR maintains two CLOCKs, the recency CLOCK T_1 for pages accessed only once recently and the frequency CLOCK T_2 for pages accessed at least twice. Pages evicted from the two CLOCKs go to the shadow recency list B_1 and the shadow frequency list B_2 respectively with their meta data only. The target CLOCK sizes and the shadow list sizes are adjusted dynamically. The shadow lists are used for evaluating the utilities of increasing the sizes of their corresponding CLOCKs. With a hypothetical access to one of the $|B_1|$ items in the recency list, increasing the size of T_1 by one is likely to convert the access to a hit with probability $\frac{1}{|B_1|}$. Similarly, with a hypothetical access to one of the $|B_2|$ items in the frequency list, increasing the size of T_2 by one is likely to convert the access to a hit with probability $\frac{1}{|B_2|}$. The utility comparison is used to guide the dynamic adjustment of cache space allocation. Given an access in B_1, the target size of T_1 is then increased by $\min\{1, \frac{|B_2|}{|B_1|}\}$. Given an access in B_2, the target size of T_2 is increased by $\min\{1, \frac{|B_1|}{|B_2|}\}$.[4] Figure 2 shows the adaptation algorithm in CAR. With a utility-driven adaptation, CAR learns the dynamic balance between recency and frequency. However, without a fine-grained predictor of reuse distance, its frequency CLOCK is less capable in capturing the weak locality in certain workloads.

Using traces from the UMass Trace Repository along with a synthetic SDD trace, we evaluate the hit ratios of CLOCK-Pro and CAR. Different cache sizes (numbers of cache entries)

[4]The utility concept and the probability calculation are not explicitly called out in either [13] or [1].

```
Variables
  $s$: cache size in number of resident pages
  $\tilde{C}_r$: target number of resident cold pages
  $\tilde{H}$: target number of hot pages
  $C_n$: current number of non-resident cold pages
  $C_d$: current number of cold pages demoted from hot page status
Subroutine Adapt(Event $e$)
  // Calculate the utility of increasing number of cold pages with
  //    a hypothetical access to a non-resident cold page
  $U_{\bar{n}} \leftarrow \frac{1}{C_n}$
  // Calculate the utility of increasing number of hot pages with
  //    a hypothetical access to a demoted page
  $U_{\bar{d}} \leftarrow \frac{1}{C_d}$
  If $e$ is an access on a non-resident cold page then
    $\tilde{C}_r \leftarrow \tilde{C}_r + \min\{1, \frac{U_{\bar{n}}}{U_{\bar{d}}}\} = \tilde{C}_r + \min\{1, \frac{C_d}{C_n}\}$
    $\tilde{H} \leftarrow s - \tilde{C}_r$
  If $e$ is observing the reference bit set on a resident cold page demoted
      from hot page status then
    Clear the demotion bit of the page
    $\tilde{H} \leftarrow \tilde{H} + \min\{1, \frac{U_{\bar{d}}}{U_{\bar{n}}}\} = \tilde{H} + \min\{1, \frac{C_n}{C_d}\}$
    $\tilde{C}_r \leftarrow s - \tilde{H}$
```

Figure 3: Adaptation algorithm in CLOCK-Pro+.

are tried.[5] Table 1 gives a quick glance at the cases where one policy outperforms the other with a relative hit ratio improvement of at least 10%. The results indicate that neither CLOCK-Pro and CAR is the consistent winner. Each of them enjoys its strengths in some cases and suffers from its weakness in other cases.

Adaptation of cache policy parameters has recently been studied in [7, 19, 20]. However, the adaptation methods are not for CLOCK-style policies and there are no trivial adjustments to tailor those methods to CLOCK-style policies. In [20] and [7], a fine-grained comparison of hit ratios is required while hits cannot be accurately counted in CLOCK-style policies. [19] uses the LFU policy as a sub-component which has no known CLOCK-style approximation. We have recently proposed a policy, Dynamic LIRS [12], to bring the idea of utility-driven adaptation into LIRS. However, [12] does not perform the important comparison of its utility-driven adaptation with the simple heuristics used in CLOCK-Pro. The work is not on the CLOCK-style policies as well.

3 IMPROVING CLOCK-PRO WITH UTILITY-DRIVEN ADAPTATION

Inspired by CAR, we propose a new CLOCK-Pro+ policy which employs the utility-driven adaptation mechanism in dynamic allocation of cache space in CLOCK-Pro. With a more sophisticated adaptation approach, CLOCK-Pro+ is

[5]The complete description is located in Section 4.

Table 2: Traces and configurations.

Trace	Requests	Unique pages	# of pages cached	
			Min.	Max.
WebSearch1	3996451	1310273	2048	524288
WebSearch2	17253075	1693344	2048	524288
WebSearch3	16407703	1689882	2048	524288
Financial1	5561703	827801	512	131072
Financial2	13882742	827801	512	131072
SDD	1048576	7578	256	2048

expected to retain the strength of CLOCK-Pro when weak locality dominates and to perform close to CAR when recency dominates.

In the new policy, we perform a direct evaluation of the utilities on increasing the number of hot pages or that of cold pages. Assuming that currently there are C_n non-resident cold pages in their test periods, given a hypothetical access to one of them, increasing the target number of cold pages by one is likely to convert the miss to a hit with probability $\frac{1}{C_n}$. If we attach a demotion bit to each resident cold page, we are able to track whether a page is hit shortly after being demoted from the hot page status. Assuming that currently there are C_d demoted cold pages, given a hypothetical access to one of them, increasing the target number of hot pages by one is likely to prevent the previous inappropriate demotion with probability $\frac{1}{C_d}$. Similar to CAR, when a non-resident cold page is accessed, we compare $\frac{1}{C_n}$ against $\frac{1}{C_d}$ and increase the target number of cold pages by $\min\{1, \frac{C_d}{C_n}\}$. When we observe that a demoted page has been accessed, we compare $\frac{1}{C_d}$ against $\frac{1}{C_n}$ and increase the target number of hot pages by $\min\{1, \frac{C_n}{C_d}\}$. Figure 3 shows the utility-driven adaptation algorithm in CLOCK-Pro+.

There are other incremental operations in addition to the utility-driven adaptation depicted in Figure 3. The demotion bit needs to be set when a hot page is demoted to a cold page.[6] The current number of non-resident cold pages needs to be updated during cold page eviction, termination of the test period of a non-resident cold page, and promotion of a non-resident cold page to a hot page. The number of demoted pages need to be updated in a hot page demotion and in observing a reference to a demoted page. Other than the additional operations described above, the new CLOCK-Pro+ policy follows the original flow of CLOCK-Pro.

Overhead: Comparing with the original CLOCK-Pro policy, the new CLOCK-Pro+ policy introduces two new variables and attaches a bit to each of the resident cold pages. The space overhead is obviously minimum, The additional

[6]Note that the bit gets cleared if the page gets a hit, as is shown in Figure 3. Also note that a demoted page stays beyond the LRU hot page and therefore is not in its test period. Its meta data is purged when it gets evicted.

Table 3: Hit ratios of CLOCK-Pro, CAR, and CLOCK-Pro+ on selected configurations.

Trace (# of pages cached)	CLOCK-Pro	CAR	CLOCK-Pro+
WebSearch1 (131072)	**13.10%**	8.32%	**12.96%**
WebSearch1 (262144)	**24.91%**	14.90%	**24.80%**
WebSearch1 (524288)	**40.36%**	32.78%	**41.66%**
WebSearch2 (262144)	**29.80%**	26.94%	**29.64%**
WebSearch2 (524288)	**48.35%**	41.72%	**48.50%**
WebSearch3 (262144)	**29.66%**	26.68%	**29.52%**
WebSearch3 (524288)	**48.21%**	41.40%	**48.41%**
Financial1 (512)	17.78%	**23.17%**	**22.69%**
Financial1 (1024)	20.62%	**26.02%**	**25.77%**
Financial1 (2048)	24.16%	**29.38%**	**29.15%**
Financial1 (4096)	27.58%	**32.61%**	**32.35%**
Financial1 (8192)	31.31%	**35.72%**	**35.65%**
Financial1 (16384)	34.33%	**38.35%**	**38.31%**
SDD (256)	17.10%	**20.40%**	**19.34%**
SDD (512)	31.60%	**36.75%**	**35.06%**

Table 4: Hit ratios of CLOCK-LIRS, CLOCK-Pro, and CLOCK-Pro+ on representative LRU-friendly cases.

Trace (pages cached)	CLOCK-LIRS	CLOCK-Pro	CLOCK-Pro+
Financial1 (512)	15.08%	*17.78%*	**22.69%**
Financial1 (1024)	19.42%	*20.62%*	**25.77%**
Financial1 (2048)	*25.36%*	24.16%	**29.15%**
Financial1 (4096)	*30.51%*	27.58%	**32.35%**
Financial1 (8192)	*34.24%*	31.31%	**35.65%**
Financial1 (16384)	*37.08%*	34.33%	**38.31%**
SDD (256)	17.00%	17.10%	**19.34%**
SDD (512)	30.95%	*31.60%*	**35.06%**
SDD (1024)	51.55%	**58.08%**	**58.07%**

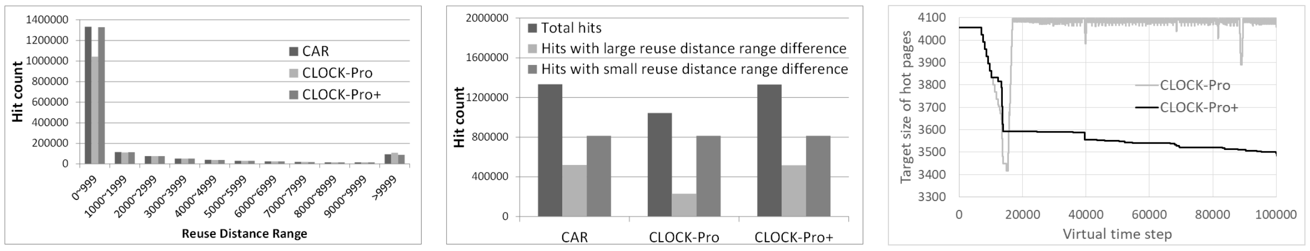
operations in manipulating the variables and the bits are fast. The overhead in execution time is minimum as well.

4 EXPERIMENTS

The storage I/O traces from the UMass Trace Repository[7] are used to evaluate the performance of CAR, CLOCK-Pro, and CLOCK-Pro+ in terms of cache hit ratio. For a fair comparison, we limit the evaluation to the 3 CLOCK-style policies only. In the repository, there are 3 search engine traces and 2 financial online transaction processing traces. While the financial online transaction processing traces are recency-biased, weak locality is commonly observed in the search engine traces. To further evaluate how CLOCK-Pro adapts to LRU-friendly workloads, a synthetic SDD trace is also created following [12].

Table 2 shows the characteristics of the traces as well as the different cache sizes used for evaluation. Here we measure cache size in number of pages cached. We fix the number of shadow cache entries to be the same as the number of pages

[7]Available at http://traces.cs.umass.edu/index.php/Storage/Storage.



(a) (b) (c)

Figure 4: Case study on ’Financial1 (4096)’: (a) hit count histogram with respect to reuse distance ranges; (b) hit count histogram with respect to reuse distance range difference; and (c) curves of target number of hot pages in CLOCK-Pro and CLOCK-Pro+.

cached. The experimental settings in the paper follow those in [12].

Table 3 highlights the performance of the policies on a selected set of configurations in which a substantial performance difference is observed. CLOCK-Pro outperforms CAR with a relative margin of at least 10% given a large cache size on the 3 search traces. CAR outperforms CLOCK-Pro with a relative margin of at least 10% given a small cache size on the first financial trace and the synthetic SDD trace. In all the 15 cases highlighted, CLOCK-Pro+ performs close to the winner. CLOCK-Pro+ substantially improves CLOCK-Pro in the 6 configurations of the first financial trace and the 2 configurations of the synthetic SDD trace. Meanwhile, CLOCK-Pro+ retains the advantage of CLOCK-Pro in the 7 configurations of the search traces, substantially outperforming CAR. The results indicate that when weak locality is dominating, e.g., in those configurations of the search traces, the new CLOCK-Pro+ policy performs close to CLOCK-Pro. When recency is dominating, e.g., in those configurations of the financial traces and the SDD trace, CLOCK-Pro+ performs close to CAR. As a result, it always performs close to the winner between CAR and CLOCK-Pro.

In all the other cases not listed in the table, the performance of CLOCK-Pro+ is also consistent, following closely to that of the winner between CAR and CLOCK-Pro in the individual case. The complete experimental results are plotted in Appendix, in which the performance of CLOCK is also given as the most basic baseline.

Ablation study[8]: We evaluate the effect of adaptation in improving upon the ablated version of CLOCK-Pro without any adaptation. We refer to the baseline policy without adaptation as CLOCK-LIRS. Table 4 shows the hit ratios on a set of representative LRU-friendly configurations. As we see from the table, the simple heuristic adaptation in CLOCK-Pro does not provide a stable improvement. In some cases, it even shows a negative impact. In contrast, the improvement from the utility-driven adaptation in CLOCK-Pro+ is consistent.

Case study: For the configuration of ’Financial1 (4096)’, we analyze each of the cache hits by CLOCK-Pro, CLOCK-Pro+, and CAR, and extract the range of the reuse distances of those hits. Figure 4(a) shows the hit count histogram with respect to different reuse distance ranges. Most hits are concentrated on the lowest range in which CLOCK-Pro generates much less hits than CAR and CLOCK-Pro+ do. Figure 4(b) gives the insight by examining the current reuse distance range versus the previous reuse distance range for each of the hits. The hits are categorized into two categories. The first one is that the two ranges are close. The second category is that the two ranges have a range difference greater than 2 or the previous reuse distance value even does not exist (that is, the hit is the first hit on the second access of the page). The second category indicates that the previous reuse distance is not a good predictor of the current reuse distance. CLOCK-Pro generates much less hits in the second category. Figure 4(c) shows how the target number of hot pages changes in the first 100,000 virtual time steps. For CLOCK-Pro, the target number of hot pages drops at the beginning, but quickly goes up to the maximum value of 4095 (even higher than the initial value of 4096 × 0.99). In fact, there are 382,543 non-resident cold page accesses, 111,244 resident cold page hits observed, but *3,143,452* cases of test period termination of cold pages in the run. The number of test period termination is one magnitude larger, preventing CLOCK-Pro from adapting to the LRU-friendly workload. Instead, for CLOCK-Pro+, there are 102,804 non-resident cold page accesses and 3,780 demoted page hits observed. Therefore the target number of hot pages gets a relatively stable reduction, well adapting to the LRU-friendly workload and overcoming the weakness of CLOCK-Pro.

[8]In an ablation study, a feature in an algorithm is removed to examine how that affects the performance.

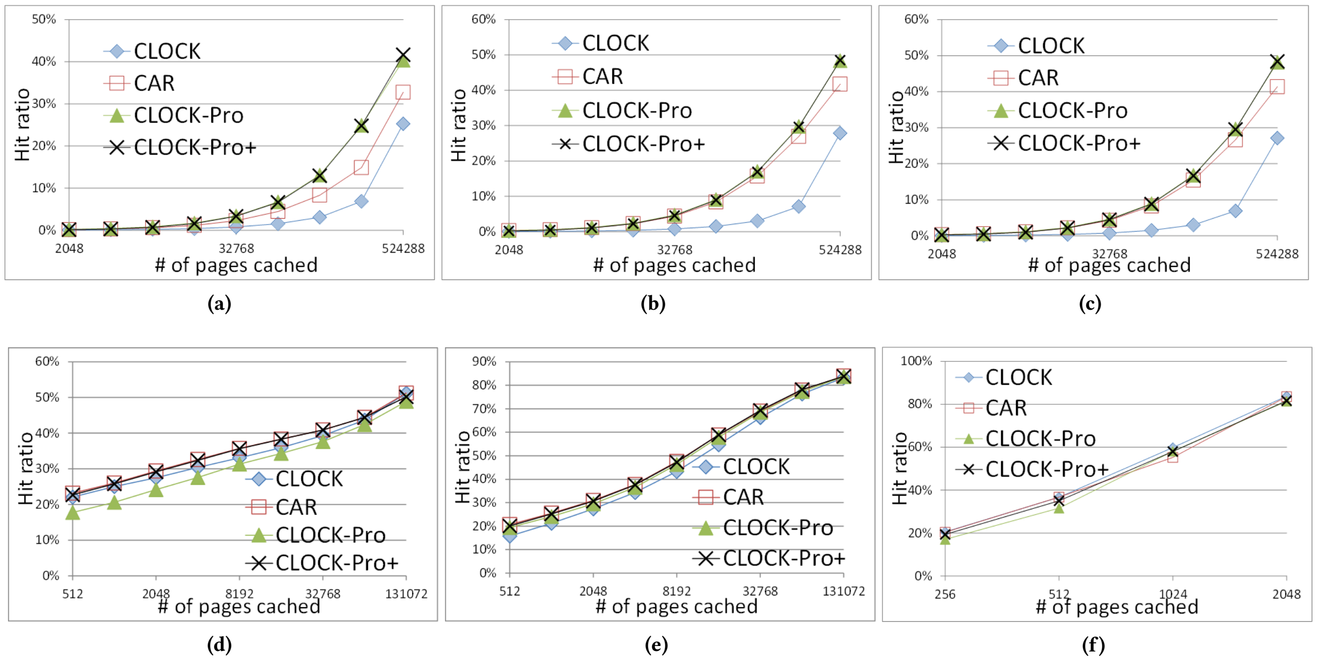


Figure 5: Complete experimental results on the traces of (a) WebSearch1; (b) WebSearch2; (c) WebSearch3; (d) Financial1; (e) Financial2; and (f) synthetic SDD trace.

5 CONCLUSION

In this paper we have proposed a novel improvement to the CLOCK-Pro replacement policy. The new CLOCK-Pro+ policy borrows the idea of utility-driven adaptation from CAR in allocating the cache space to cold pages against hot pages. Experimental results indicate that CLOCK-Pro+ retains the strength of CLOCK-Pro and overcomes its weakness.

6 AVAILABILITY

The code and the datasets used in our experiments (AES encryption password: **Pro+2019!**) are available at

https://www.dropbox.com/s/gzlu4tdpp23htye/CLOCK.zip.

ACKNOWLEDGEMENTS

We thank Jia Bao for many useful discussions regarding the algorithms in this paper. We acknowledge the anonymous reviewers and our shepherd, Roy Friedman, for their valuable comments and criticisms to improve the paper.

APPENDIX: COMPLETE RESULTS

Figure 5 shows the complete experimental results. CLOCK-Pro+ performs consistently close to the winner of the other policies in all the cases.

REFERENCES

[1] Sorav Bansal and Dharmendra S. Modha. 2004. CAR: Clock with Adaptive Replacement. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST '04)*. 187–200.

[2] Aaron Blankstein, Siddhartha Sen, and Michael J. Freedman. 2017. Hyperbolic Caching: Flexible Caching for Web Applications. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 499–511. https://www.usenix.org/conference/atc17/technical-sessions/presentation/blankstein

[3] Pei Cao and Sandy Irani. 1997. Cost-aware WWW Proxy Caching Algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems (USITS'97)*. 193–206.

[4] Jr. Edward G. Coffman and Peter J. Denning. 1973. *Operating Systems Theory*. Prentice Hall Professional Technical Reference.

[5] F. J. Corbató. 1969. A Paging Experiment with the Multics system. In *In Honor of P. M. Morse*. MIT Press, 217–228.

[6] Louis Degenaro, Arun Iyengar, Ilya Lipkind, and Isabelle Rouvellou. 2000. A Middleware System Which Intelligently Caches Query Results. In *Middleware (Lecture*

Notes in Computer Science), Vol. 1795. 24–44.

[7] Gil Einziger, Ohad Eytan, Roy Friedman, and Ben Manes. 2018. Adaptive Software Cache Management. In *Proceedings of the 19th International Middleware Conference, Middleware 2018, Rennes, France, December 10-14, 2018*. 94–106. https://doi.org/10.1145/3274808.3274816

[8] Gil Einziger, Roy Friedman, and Ben Manes. 2017. TinyLFU: A Highly Efficient Cache Admission Policy. *ACM Trans. Storage* 13, 4, Article 35 (Nov. 2017), 31 pages. https://doi.org/10.1145/3149371

[9] Song Jiang, Feng Chen, and Xiaodong Zhang. 2005. CLOCK-Pro: An Effective Improvement of the CLOCK Replacement. In *Proceedings of the 2005 USENIX Annual Technical Conference (ATEC '05)*. 323–336.

[10] Song Jiang and Xiaodong Zhang. 2002. LIRS: An Efficient Low Inter-Reference Recency Set Replacement Policy to Improve Buffer Cache Performance. In *Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '02)*. 31–42.

[11] Theodore Johnson and Dennis Shasha. 1994. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94)*. 439–450.

[12] Cong Li. 2018. DLIRS: Improving Low Inter-Reference Recency Set Cache Replacement Policy with Dynamics. In *Proceedings of the 11th ACM International Systems and Storage Conference (SYSTOR '18)*. ACM, New York, NY, USA, 59–64. https://doi.org/10.1145/3211890.3211891

[13] Nimrod Megiddo and Dharmendra S. Modha. 2003. ARC: A Self-Tuning, Low Overhead Replacement Cache. In *Proceedings of the FAST '03 Conference on File and Storage Technologies (FAST '03)*.

[14] Michael N. Nelson, Brent B. Welch, and John K. Ousterhout. 1988. Caching in the Sprite Network File System. *ACM Trans. Comput. Syst.* 6, 1 (Feb. 1988), 134–154.

[15] Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. 1993. The LRU-K Page Replacement Algorithm for Database Disk Buffering. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD '93)*. 297–306.

[16] Sejin Park and Chanik Park. 2017. FRD: A Filtering based Buffer Cache Algorithm that Considers both Frequency and Reuse Distance. In *Proceedings of the 33rd International Conference on Massive Storage Systems and Technology (MSST '17)*.

[17] Alan J. Smith. 1985. Disk Cache - Miss Ratio Analysis and Design Considerations. *ACM Trans. Comput. Syst.* 3, 3 (Aug. 1985), 161–203.

[18] James Z. Teng and Robert A. Gumaer. 1984. Managing IBM Database 2 Buffers to Maximize Performance. *IBM Systems Journal* 23, 2 (1984), 211–218.

[19] Giuseppe Vietri, Liana V. Rodriguez, Wendy A. Martinez, Steven Lyons, Jason Liu, Raju Rangaswami, Ming Zhao, and Giri Narasimhan. 2018. Driving Cache Replacement with ML-based LeCaR. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/hotstorage18/presentation/vietri

[20] Carl Waldspurger, Trausti Saemundsson, Irfan Ahmad, and Nohhyun Park. 2017. Cache Modeling and Optimization using Miniature Simulations. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX Association, Santa Clara, CA, 487–498. https://www.usenix.org/conference/atc17/technical-sessions/presentation/waldspurger