Improving Adaptive Replacement Cache (ARC) by Reuse Distance

Woojoong Lee, Sejin Park, Baegjae Sung, Chanik Park

Pohang University of Science and Technology, Korea

Buffer caches are used to enhance the performance of file or storage systems by reducing I/O requests to underlying storage media. In particular, multi-level buffer cache hierarchy is commonly deployed on network file systems or storage systems. In this environment, the I/O access pattern on second-level buffer caches of file servers or storage controllers differs from that on upper-level caches.

The reuse distance of a block is an important metric to characterize I/O access pattern. It is defined as the number of requests between two adjacent accesses to a block in an I/O stream. In [1, 2], Zhou et al. showed that the access pattern on second-level buffer caches has a hill-shaped reuse-distance distribution. This implies that two consecutive accesses to a data block have a relatively long temporal distance due to upper-level cache behavior. They also examined the behavior of the access patterns in terms of frequency, and revealed that the more frequently the block is accessed, the larger portion it takes out of total accesses.

For second-level buffer caches, various techniques including frequency-based block prioritizing [1, 2], exclusiveness [3, 4], or multi-level cache coordination [5], have been proposed with consideration of the temporal or frequency pattern of I/O workload on the multi-level cache hierarchy. However, the complexity issue remains still unsatisfactory.

The Adaptive Replacement Cache (ARC) algorithm proposed by Megiddo et al. [6, 7] dynamically balances recency and frequency by using two *Least-Recently-Used* (LRU) queues in response to changing access patterns. ARC is simple to implement and has low computational overhead while performing well across varied workloads. ARC not only outperforms most online algorithms, such as LRU, *Frequency-Based Replacement* (FBR) [8], *Least-Frequently-Used* (LFU) [9], and *Low Inter-reference Recency Set* (LIRS) [10], but is also comparable to offline algorithms LRU-2 [11], *2-Queue* (2Q) [12], and *Least-Recently/Frequently-Used* (LRFU) [13].

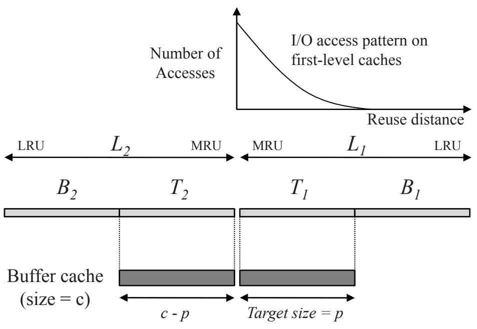
However, because ARC does not take into account the reuse distance of I/O requests, ARC cannot perform efficiently on a second-level cache. Note that the reuse distances of most I/O accesses on the second-level cache will be long so that the recency queue of ARC cannot contribute much to the cache hits. Furthermore, due to the long reuse distance of I/O requests on the second-level cache, most cache hits will be observed near the LRU (not MRU) position of the recency queue in ARC. If cache hits are observed near the LRU position of the recency queue, ARC tries to increase the size of the recency queue in order to capture the recency locality more. Accordingly, the size of the frequency queue decreases. It becomes worse when the second-level cache size is equal to or smaller than the first-level cache size. In this case, the recency queue of ARC contributes nothing to the cache hits. For more details, refer to [6, 7].

In order to solve the problem, we propose an enhanced block replacement algorithm, called RARC (Reuse-distance aware ARC). In order to capture the reuse distance of I/O requests, a history buffer over all I/O requests is maintained. A sliding window of concern on the history buffer will determine which I/O blocks are kept in the recency queue. So, the size of the sliding window is equal to the size of the recency queue in RARC and the sliding window will be dynamically updated reflecting the reuse distance pattern on the second-level cache.

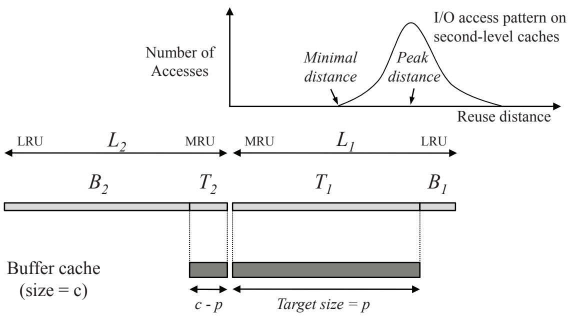
Fig 1-(a) shows a typical buffer cache state of ARC when it is used in the first-level cache. Most block accesses have a reuse distance smaller than the cache size so that the size of T_1 and T_2 can be adjusted efficiently in response to workload changes. On the other hand, in the case of Fig 1-(b) showing a typical buffer cache state of ARC when it is used in the second-level cache, T_1 is extended by the weak temporal locality of second-level I/O accesses while T_2 is reduced by control rules of ARC; the cache size is smaller than the *peak distance* of the workload so that hits in B_1 are increased compared to the former case. Another side effect is only a small portion of the accesses will be captured and moved to T_2; this make the hit ratio in T_2 worse along with the reduced size of T_2.

Fig 1-(c) demonstrates the intuition of the proposed algorithm. Differently from ARC, the newly issued request is not retained by the recency queue, i.e., T_1, but inserted into the history buffer at the MRU position. The history buffer only maintains the block number and its access time. The recency queue is defined as a sliding window where only blocks within the range of the window are loaded into the cache via the background I/Os.

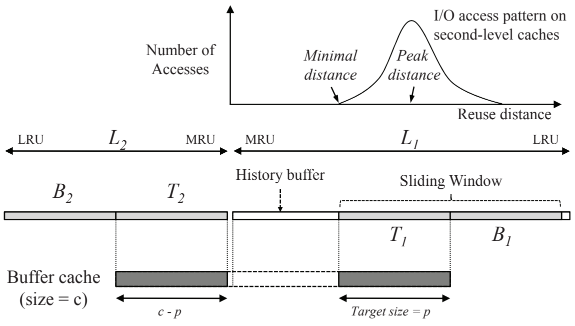
The main challenge in RARC is to capture the reuse distance pattern in order to control the sliding window to maintain I/O blocks of high demand in the cache. Among the reuse distance pattern, the minimal reuse distance plays a key role because the upper-level cache size usually keeps changing for various reasons like memory pressure. The history buffer will be used to estimate the minimal reuse distance.



(a) ARC on the first-level cache



(b) ARC on the second-level cache



(c) RARC on the second-level cache

Figure 1: The problem of ARC and the intuition of the proposed RARC

References

[1] Y. Zhou, Z. Chen, and K. Li, "Second-level Buffer Cache Management," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 7, July, 2004.

[2] Y. Zhou, J.F. Philbin, and K. Li, "Second-level Buffer Cache Management," *In Procedings of the 2001 USENIX Annual Technical Conference*, 2001.

[3] L.N. Bairavasundaram, M. Sivathanu ,A.C. Arpaci-Dusseau, R.H. Arpaci-Dusseau, "X-RAY: A Non-Invasive Exclusive Caching Mechanism for RAIDs," *Proceedings of the 31st annual international symposium on Computer architecture*, p.176, June 19-23, 2004.

[4] T.M. Wong and J. Wilkes, "My Cache or Yours? Making Storage More Exclusive," *In Proceedings of the USENIX Annual Technical Conference*, 2002.

[5] Jiang, S. and Davis, K. and Zhang, X., "Coordinated multilevel buffer cache management with consistent access locality quantification," *IEEE Transactions on Computers*, pp. 95-108, 2007.

[6] N. Megiddo and D.S. Modha, "Arc: A Self-Tuning, Low Overhead Replacement Cache," *Proc. Second USENIX Conf. File and Storage Technologies*, 2003

[7] N. Megiddo , D.S. Modha, "Outperforming LRU with an Adaptive Replacement Cache Algorithm," Computer, v.37 n.4, p.58-65, April 2004

[8] J.T. Robinson and M.V. Devarakonda, "Data cache management using frequency-based replacement," *in Proc. ACM SIGMETRICS Conf.*, pp. 134-142, 1990.

[9] A.V. Aho, P.J. Denning, and J.D. Ullman, "Principles of optimal page replacement," *Journal of the ACM*, vol. 18, no. 1, pp. 80-93, 1971.

[10] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," *in Proc. ACM SIGMETRICS Conf.*, 2002.

[11] E.J. O'Neil, P.E. O'Neil, and G. Weikum, "An optimality proof of the LRU-K page replacement algorithm," *Journal of the ACM*, vol. 46, no. 1, pp. 92-112, 1999.

[12] T. Johnson and D. Shasha, "2Q: A low overhead high performance buffer management replacement algorithm," *in Proc. VLDB Conf.*, pp. 297-306, 1994.

[13] D. Lee, J. Choi, J.-H. Kim, S.H. Noh, S.L. Min, Y. Cho, and C.S. Kim, "LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE Trans. Computers*, vol. 50, no. 12, pp. 1352-1360, 2001.

[14] The SPEC 2000 benchmark: A perlbmk-diffmail trace collected by BYU, `http://tds.cs.byu.edu/$\sim$samples/A2L000024.html`

[15] SNIA IOTTA Repository: Network File System Traces, `http://iotta.snia.org/traces/list/NFS`

[16] BYU Trace Distribution Center: Disk I/O traces, `http://tds.cs.byu.edu/tds`

[17] MSR Cambridge Traces: `ftp://ftp.research.microsoft.com/pub/austind/MSRC-io-traces`

[18] D. Narayanan, A. Donnelly, and A. Rowstron, "Write Off-Loading: Practical Power Management for Enterprise Storage,", *Proc. 6th USENIX Conf. File and Storage Technologies*, 2008.