WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Deutsch
Español
Français
Bahasa Indonesia
日本語
Русский
Simple English
Српски / srpski
Türkçe
中文
Edit links

Cache algorithms

From Wikipedia, the free encyclopedia

This article is about general cache algorithms. For detailed algorithms specific to paging, see Page replacement algorithm. For detailed algorithms specific to the cache between a CPU and RAM, see CPU cache.

In computing, **cache algorithms** (also frequently called **cache replacement algorithms** or **cache replacement policies**) are optimizing instructions—or algorithms—that a computer program or a hardware-maintained structure can follow in order to manage a cache of information stored on the computer. When the cache is full, the algorithm must choose which items to discard to make room for the new ones.

Contents [hide]

1 Overview
2 Examples
3 See also
4 References
5 External links

Overview [edit]

The average memory reference time is[1]

$$T = m * T_m + T_h + E$$

where

T = average memory reference time
m = miss ratio = 1 - (hit ratio)
T_m = time to make a main memory access when there is a miss (or, with multi-level cache, average memory reference time for the next-lower cache)
T_h= the latency: the time to reference the cache when there is a hit
E = various secondary effects, such as queuing effects in multiprocessor systems

There are two primary figures of merit of a cache: The latency, and the hit rate. There are also a number of secondary factors affecting cache performance.[1]

The "hit ratio" of a cache describes how often a searched-for item is actually found in the cache. More efficient replacement policies keep track of more usage information in order to improve the hit rate (for a given cache size).

The "latency" of a cache describes how long after requesting a desired item the cache can return that item (when there is a hit). Faster replacement strategies typically keep track of less usage information—or, in the case of direct-mapped cache, no information—to reduce the amount of time required to update that information.

Each replacement strategy is a compromise between hit rate and latency.

Measurements of "the hit ratio" are typically performed on benchmark applications. The actual hit ratio varies widely from one application to another. In particular, video and audio streaming applications often have a hit ratio close to zero, because each bit of data in the stream is read once for the first time (a compulsory miss), used, and then never read or written again. Even worse, many cache algorithms (in particular, LRU) allow this streaming data to fill the cache, pushing out of the cache information that will be used again soon (cache pollution).[2]

Examples [edit]

Bélády's Algorithm

The *most* efficient caching algorithm would be to always discard the information that will not be needed for the longest time in the future. This optimal result is referred to as Bélády's optimal algorithm or the clairvoyant algorithm. Since it is generally impossible to predict how far in the future information will be needed, this is generally not implementable in practice. The practical minimum can be calculated only after experimentation, and one can compare the effectiveness of the actually chosen cache algorithm.

Least Recently Used (LRU)

Discards the least recently used items first. This algorithm requires keeping track of what was used when, which is expensive if one wants to make sure the algorithm always discards *the* least recently used item. General implementations of this technique require keeping "age bits" for cache-lines and track the "Least Recently Used" cache-line based on age-bits. In such an implementation, every time a cache-line is used, the age of all other cache-lines changes. LRU is actually a family of caching algorithms with members including 2Q by Theodore Johnson and Dennis Shasha,[3] and LRU/K by Pat O'Neil, Betty O'Neil and Gerhard Weikum.[4]

Most Recently Used (MRU)

Discards, in contrast to LRU, the most recently used items first. In findings presented at the 11th VLDB conference, Chou and DeWitt noted that "When a file is being repeatedly scanned in a [Looping Sequential] reference pattern, MRU is the best replacement algorithm."[5] Subsequently other researchers presenting at the 22nd VLDB conference noted that for random access patterns and repeated scans over large datasets (sometimes known as cyclic access patterns) MRU cache algorithms have more hits than LRU due to their tendency to retain older data.[6] MRU algorithms are most useful in situations where the older an item is, the more likely it is to be accessed.

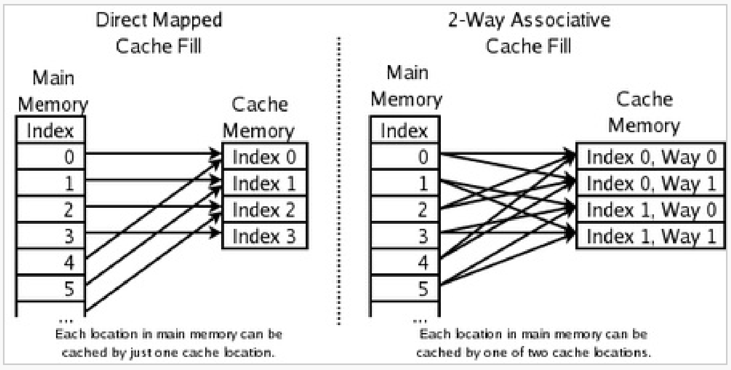
Pseudo-LRU (PLRU)

For CPU caches with large associativity (generally >4 ways), the implementation cost of LRU becomes prohibitive. In many CPU caches, a scheme that almost always discards one of the least recently used items is sufficient. So many CPU designers choose a PLRU algorithm which only needs one bit per cache item to work.

PLRU typically has a slightly worse miss ratio, has a slightly better latency, and uses slightly less power than LRU.

Random Replacement (RR)

Randomly selects a candidate item and discards it to make space when necessary. This algorithm does not require keeping any information about the access history. For its simplicity, it has been used in ARM processors.[7] It admits efficient stochastic simulation.[8]



Which memory locations can be cached by which cache locations

Segmented LRU (SLRU)

An SLRU cache is divided into two segments, a probationary segment and a protected segment. Lines in each segment are ordered from the most to the least recently accessed. Data from misses is added to the cache at the most recently accessed end of the probationary segment. Hits are removed from wherever they currently reside and added to the most recently accessed end of the protected segment. Lines in the protected segment have thus been accessed at least twice. The protected segment is finite, so migration of a line from the probationary segment to the protected segment may force the migration of the LRU line in the protected segment to the most recently used (MRU) end of the probationary segment, giving this line another chance to be accessed before being replaced. The size limit on the protected segment is an SLRU parameter that varies according to the I/O workload patterns. Whenever data must be discarded from the cache, lines are obtained from the LRU end of the probationary segment.[9]"

2-way set associative

Used for high-speed CPU caches where even PLRU is too slow. The address of a new item is used to calculate one of two possible locations in the cache where it is allowed to go. The LRU of the two is discarded. This requires one bit per pair of cache lines, to indicate which of the two was the least recently used.

Direct-mapped cache

Used for the highest-speed CPU caches where even 2-way set associative caches are too slow. The address of the new item is used to calculate the one location in the cache where it is allowed to go. Whatever was there before is discarded.

Least-Frequently Used (LFU)

Counts how often an item is needed. Those that are used least often are discarded first.

Low Inter-reference Recency Set (LIRS)

A page replacement algorithm with an improved performance over LRU and many other newer replacement algorithms. This is achieved by using reuse distance as a metric for dynamically ranking accessed pages to make a replacement decision. The algorithm was developed by Song Jiang and Xiaodong Zhang.

Adaptive Replacement Cache (ARC)

Constantly balances between LRU and LFU, to improve the combined result.[10] ARC improves on SLRU by using information about recently-evicted cache items to dynamically adjust the size of the protected segment and the probationary segment to make the best use of the available cache space.

Clock with Adaptive Replacement (CAR)

Combines Adaptive Replacement Cache (ARC) and CLOCK. CAR has performance comparable to ARC, and substantially outperforms both LRU and CLOCK. Like ARC, CAR is self-tuning and requires no user-specified magic parameters.

Multi Queue (MQ)

By Zhou, Philbin, and Li.[11]

The MQ cache contains multiple LRU queues, Q_0, Q_1, ..., Q_{m-1}. Blocks stay in the LRU queues for a given lifetime, which is defined dynamically by the MQ algorithm to be the maximum temporal distance between two accesses to the same file or the number of cache blocks, whichever is larger. If a block has not been referenced within its lifetime, it is demoted from to Q_i to Q_{i-1} or evicted from the cache if it is in Q_0. Each queue also has a maximum access count; if a block in queue Q_i is accessed more than 2^i times, this block is promoted to Q_{i+1} until it is accessed more than 2^{i+1} times or its lifetime expires. Within a given queue, blocks are ranked by the recency of access, according to LRU.[12]

Other things to consider:

- Items with different cost: keep items that are expensive to obtain, e.g. those that take a long time to get.
- Items taking up more cache: If items have different sizes, the cache may want to discard a large item to store several smaller ones.
- Items that expire with time: Some caches keep information that expires (e.g. a news cache, a DNS cache, or a web browser cache). The computer may discard items because they are expired. Depending on the size of the cache no further caching algorithm to discard items may be necessary.

Various algorithms also exist to maintain cache coherency. This applies only to situation where *multiple* independent caches are used for the *same* data (for example multiple database servers updating the single shared data file).

See also [edit]

- Cache-oblivious algorithm
- Locality of reference
- Distributed cache

References [edit]

1. ^ *a* *b* Alan Jay Smith. "Design of CPU Cache Memories". Proc. IEEE TENCON, 1987. [1]
2. ^ Paul V. Bolotoff. "Functional Principles of Cache Memory". 2007.
3. ^ http://www.vldb.org/conf/1994/P439.PDF
4. ^ O'Neil, Elizabeth J.; O'Neil, Patrick E.; Weikum, Gerhard (1993). "The LRU-K Page Replacement Algorithm for Database Disk Buffering". *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*. SIGMOD '93 (New York, NY, USA: ACM): 297–306. doi:10.1145/170035.170081. ISBN 0-89791-592-5.
5. ^ Hong-Tai Chou and David J. DeWitt. An Evaluation of Buffer Management Strategies for Relational Database Systems. VLDB, 1985.
6. ^ Shaul Dar, Michael J. Franklin, Björn Þór Jónsson, Divesh Srivastava, and Michael Tan. Semantic Data Caching and Replacement. VLDB, 1996.
7. ^ ARM Cortex-R series processors manual
8. ^ An Efficient Simulation Algorithm for Cache of Random Replacement Policy [2]
9. ^ Ramakrishna Karedla, J. Spencer Love, and Bradley G. Wherry. Caching Strategies to Improve Disk System Performance. In Computer, 1994.
10. ^ Nimrod Megiddo and Dharmendra S. Modha. ARC: A Self-Tuning, Low Overhead Replacement Cache. FAST, 2003.
11. ^ Yuanyuan Zhou, James Philbin, and Kai Li. The Multi-Queue Replacement Algorithm for Second Level Buffer

Caches. USENIX, 2002.

12. ^ Eduardo Pinheiro , Ricardo Bianchini, Energy conservation techniques for disk array-based servers, Proceedings of the 18th annual international conference on Supercomputing, June 26-July 01, 2004, Malo, France

External links [edit]

- Definitions of various cache algorithms
- Fully associative cache
- Set associative cache
- Direct mapped cache

- Slides on various page replacement schemes including LRU

Categories: Cache (computing) | Memory management algorithms