Adaptive replacement cache

From Wikipedia, the free encyclopedia

Adaptive Replacement Cache (ARC) is a page replacement algorithm with better performance[1] than LRU (Least Recently Used) developed[2] at the IBM Almaden Research Center. This is accomplished by keeping track of both Frequently Used and Recently Used pages plus a recent eviction history for both. In 2006, IBM was granted a patent for the adaptive replacement cache policy.

Summary [edit]

Basic LRU maintains an ordered list (the cache directory) of resource entries in the cache, with the sort order based on the time of most recent access. New entries are added at the top of the list, after the bottom entry has been evicted. Cache hits move to the top, pushing all other entries down.

ARC improves the basic LRU strategy by splitting the cache directory into two lists, T1 and T2, for recently and frequently referenced entries. In turn, each of these is extended with a *ghost* list (B1 or B2), which is attached to the bottom of the two lists. These *ghost* lists act as scorecards by keeping track of the history of recently evicted cache entries, and the algorithm uses *ghost* hits to adapt to recent change in resource usage. Note that the *ghost* lists only contain metadata (keys for the entries) and not the resource data itself, i.e. as an entry is evicted into a *ghost* list its data is discarded. The combined cache directory is organised in four LRU lists:

1. T1, for recent cache entries.
2. T2, for frequent entries, referenced at least twice.
3. B1, *ghost* entries recently evicted from the T1 cache, but are still tracked.
4. B2, similar *ghost* entries, but evicted from T2.

T1 and B1 together are referred to as L1, a combined history of recent single references. Similarly, L2 is the combination of T2 and B2.

The whole cache directory can be visualised in a single line:

```
. . . [   B1  <-[     T1    <-!->      T2   ]->  B2   ] . .
      [ . . . . [ . . . . . . ! . .^. . . ] . . . . ]
                [   fixed cache size (c)    ]
```

The inner **[ ]** brackets indicate actual cache, which although fixed in size, can move freely across the B1 and B2 history.

L1 is now displayed from right to left, starting at the top, indicated by the **!** marker. **^** indicates the target size for T1, and may be equal to, smaller than, or larger than the actual size (as indicated by **!**).

- New entries enter T1, to the left of **!**, and are gradually pushed to the left, eventually being evicted from T1 into B1, and finally dropped out altogether.
- Any entry in L1 that gets referenced once more, gets another chance, and enters L2, just to the right of the central **!** marker. From there, it is again pushed outward, from T2 into B2. Entries in L2 that get another hit can repeat this indefinitely, until they finally drop out on the far right of B2.

Replacement [edit]

Entries (re-)entering the cache (T1,T2) will cause **!** to move towards the target marker **^**. If no free space exists in the cache, this marker also determines whether either T1 or T2 will evict an entry.

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
Simple English
Edit links

- Hits in B1 will increase the size of T1, pushing **^** to the right. The last entry in T2 is evicted into B2.
- Hits in B2 will shrink T1, pushing **^** back to the left. The last entry in T1 is now evicted into B1.
- A cache miss will not affect **^**, but the **!** boundary will move closer to **^**.

Deployment [edit]

ARC is currently deployed in IBM's DS6000/DS8000 storage controllers.

Sun Microsystems's scalable file system ZFS uses a variant[3] of ARC as an alternative to the traditional Solaris filesystem page cache in virtual memory. It has been modified to allow for locked pages that are currently in use and cannot be vacated.

PostgreSQL used ARC in its buffer manager for a brief time (version 8.0.0), but quickly replaced it with another algorithm, citing concerns over an IBM patent on ARC.[4]

References [edit]

1. **^** One Up on LRU, Usenix :login; August 2003
2. **^** Nimrod Megiddo and Dharmendra Modha, 2010-03-09 archive of the ARC home page, with pointers to several articles
3. **^** comments in Solaris ZFS arc.c source file explains differences with original work
4. **^** Article in Postgresql General Bits, "The Saga of the ARC Algorithm and Patent", published 6 February 2005

External links [edit]

- ARC: A Self-Tuning, Low Overhead Replacement Cache (2003) by Nimrod Megiddo, Dharmendra Modha
- Linux Memory Management Wiki
- Bourbonnais, Roch. ZFS Dynamics
- Python implementation, recipe 576532
- Comparison of LRU, ARC and others
- Implementation in ANSI/C

See also [edit]

- Clock with Adaptive Replacement

Categories: Memory management algorithms | Virtual memory