经典算法的几种现有改进

零、几种request类型：

1.顺序访问。所有的块一个接一个被访问，不存在重访问。

2.循环访问。所有块都按照一定的间隔重复访问

3.时间密集访问。最近被访问的块是将来最有可能被访问的。

4.概率访问。所有块都有固定的访问概率，所有块都互相独立地根据概率被访问。

5.关联访问(Correlated References)，块被首次访问之后，紧接着的短时间内会有数次访问。

一、LFU算法的改进

算法根据数据的历史访问频率来淘汰数据，其核心思想是“如果数据过去被访问多次，那么将来被访问的频率也更高”。

1、LFU*算法

思路：

基于LFU的改进算法，其核心思想是“只淘汰访问过一次的数据”。

且如果所有引用计数为1的数据大小之和都没有新加入的数据那么大，则不淘汰数据，新的数据也不缓存。

2、LFU-Aging算法

思路：

其核心思想是“除了访问次数外，还要考虑访问时间”。

通过平均引用计数来标识时间。当当前缓存中的数据“引用计数平均值”达到或者超过“最大平均引用计数”时，则将所有数据的引用计数都减少。减少的方法有多种，可以直接减为原来的一半，也可以减去固定的值等。

3、LFU*-Aging算法

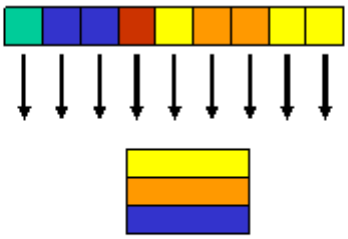
思路：

结合LFU*和LFU-Aging

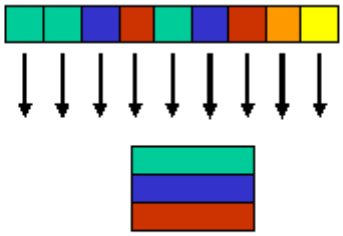
4、Window-LFU算法

思路：

Window-LFU并不记录所有数据的访问历史，而只是记录过去一段时间内的访问历史



样例1



样例2

样例1：黄色访问3次，蓝色和橘色都是两次，橘色更新，因此缓存黄色、橘色、蓝色三个数据块

样例2：绿色访问3次，蓝色两次，暗红两次，蓝色更新，因此缓存绿色、蓝色、暗红三个数据块

对比点	对比
命中率	Window-LFU/LFU-Aging > LFU-*Aging* > *LFU* > *LFU*
复杂度	LFU-Aging > LFU> LFU-*Aging* >*Window-LFU* > *LFU*
代价	LFU-Aging > LFU > Window-LFU > LFU-*Aging* > *LFU*

二、LRU算法的改进

淘汰最长时间未被使用的页面。

存在问题：

- 对冷数据突发性访问抵抗能力差，可能会因此淘汰掉热的文件。热点页面在偶然一个时间节点被其他大量仅访问了一次的页面所取代则造

成浪费。

- 对于大量数据的循环访问抵抗能力查，极端情况下可能会出现命中率0%。（如：循环访问）
- 不能按照数据的访问概率进行淘汰。

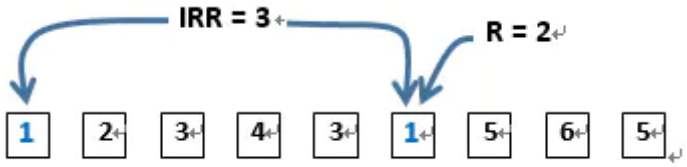
1、LIRS (Low Inter-reference Recency Set)

思路：

通过使用两次访问同一块之间的距离（本距离指中间被访问了多少非重复块）作为一种尺度去动态地将访问块排序

两个衡量参数（不包含重复页数）：

- IRR：一个页面最近两次的访问间隔（页面的访问频度）
- Recency：页面上次访问至今访问了多少其他页。



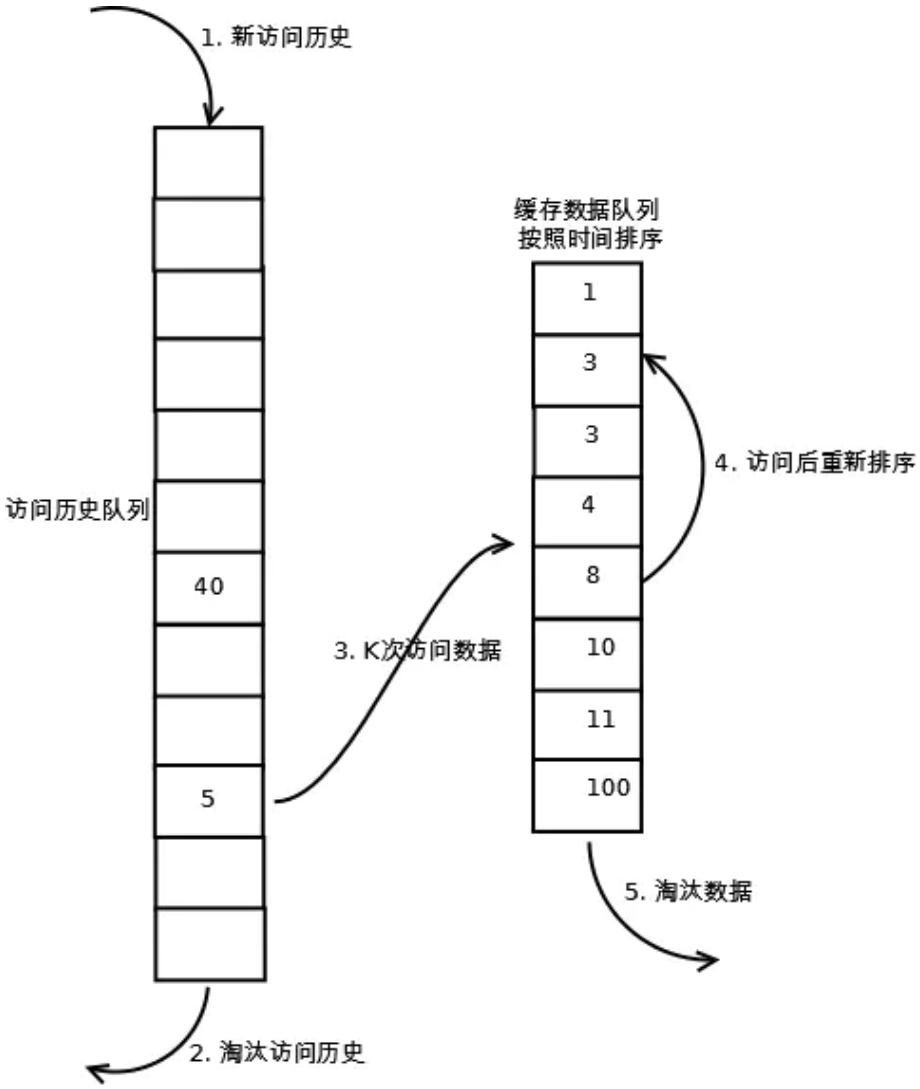
	1	2	3	4	5	6	7	8	9	IRR	R
A	X					X		X		1	1
B			X		X					1	3
C				X						Inf	4
D		X					X			3	2
E									X	Inf	0

替换：替换IRR最大的页面，相同则替换R更大的。降低了最后一次访问信息的优先级。

2、LRU - K算法

思路：

相比于传统的LRU就是LRU - 1，仅访问了一次就能代替别人，**K次访问**才能有替换资格。最后第K次的访问距离。访问距离↑，时间间隔↑，被替换↑。



LRU-2，只有当数据的访问次数达到2次的时候，才将数据放入缓存。当需要淘汰数据时，LRU-2会淘汰第2次访问时间距当前时间最大的数据。

实现步骤：

(1) 数据第一次被访问，加入到访问历史列表；

(2) 如果数据在访问历史列表里后没有达到K次访问，则按照一定规则（FIFO，LRU）淘汰；

(3) 当访问历史队列中的数据访问次数达到K次后，将数据索引从历史队列删除，将数据移到缓存队列中，并缓存此数据，缓存队列重新按照时间排序；

(4) 缓存数据队列中被再次访问后，重新排序；

(5) 需要淘汰数据时，淘汰缓存队列中排在末尾的数据，即：淘汰“倒数第K次访问离现在最久”的数据。

仍然存在的问题和改进：

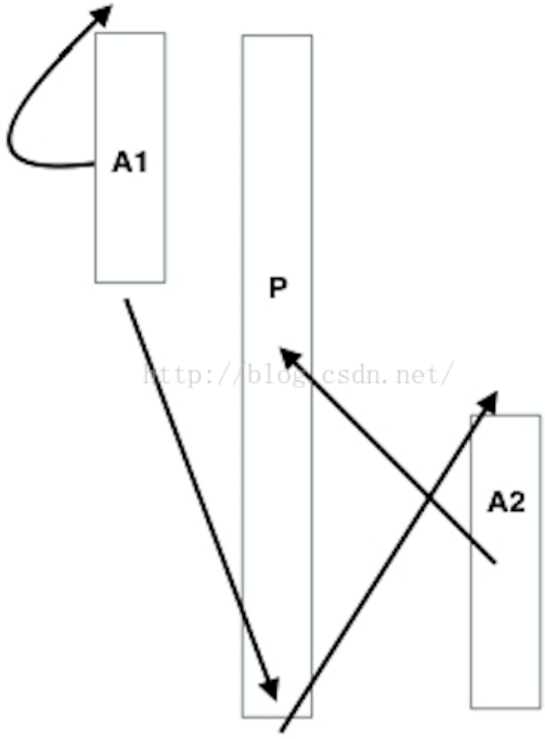
1、针对关联访问的问题：块被首次访问之后，紧接着的短时间内会有数次访问。（如：数据库中同一事务内的select和update会多次扫描相同的块）

提出参数：Correlated References Period（块首次访问后的一段时间）为了避免关联访问的干扰造成对块的错误判断，在这段时间内的多次访问只算作一次访问。只有这段时间后块再次被访问，才算第二次被访问。

2、针对无记忆性的问题：块被替换出cache后，可能很快地再次被访问，由于之前访问记录已丢弃，这样只算作首次访问，之后又很快被替换出cahce后，又再次被访问，这样又只会算作首次访问，如此下来，虽然块被频繁访问，属于hot块，但由于替换出cahce后没有保留访问信息，导致错误判断

提出参数：Reference Retained Information Period（对于替换出cache后的块会继续保留访问信息一段时间）

改进后的实现数据结构：（A1和P被分配cache）



- LRU队列A1。第一次访问的块分配cache后，插入A1队列尾部。在A1中的块被访问时，重新加入队列A1尾部。A1头部出列的块则插入优先级队列P（倒数第二次访问时间初始化为0）。该队列主要实现Correlated References Period，需要根据实际情况设置队列合理固定大小。
- 优先级队列P。优先级队列P以倒数第二次的访问时间进行升序排序。只有当从A1出列的块或者A2重新访问的块可以插入队列P。P中的块被访问时，更新倒数第二次访问时间并重新排序。当需要分配cache的时候，P队列头部的块（倒数第二次访问时间最短，也就是距离最大）替换出cache后插入到A2中。
- FIFO队列A2。负责保存替换出cache的块访问信息。如果A2中的块再次被访问，就更新倒数第二次访问时间，同时分配cache，插入优先级队列P。块从A2出列则删除其历史访问信息。

LRU - K其他的问题：

1. 由于优先级队列的排序操作需要额外的O(logN)的时间复杂度，N为P的大小。
2. A1，P和A2的大小都必须按照实际情况进行配置取最优比例，才能发挥最优性能。
3. 块的访问频率变化响应较慢。这是因为P的比较是按照历史的最后第K次访问距离进行比较。如果块A在P中的时候倒数第K次的距离较少，但经过较长时间才有新的访问，重新更新访问距离后，才会被快速替换出cache。

3、2Q算法

思路：

该算法类似于LRU-2，不同点在于2Q将LRU-2算法中的访问历史队列（不是缓存数据的）改为一个FIFO缓存队列，即：2Q算法有**两个缓存队列**，一个是FIFO队列A1，一个是LRU队列Am。

问题：

A1和Am各自所占cache的比例是关键。

如果A1太小，则检测是否hot块的时间太短，很可能需要较长时间才把hot块加入到Am中。但如果A1太大，则A1会占了原本所属Am的cache，hot块的数量就会减少，会影响cache命中率。

【待补充改进和缺点】

LFU改进算法的参考地址