缓存替换算法研究综述*)

黄 敏 蔡志刚

(西南大学计算机与信息科学学院计算机科学系 重庆 400715)

摘 要 缓存技术作为提高系统性能的重要手段一直是研究的热点。随着网络存储等新技术的出现,存储的层次越来越复杂,原有的简单缓存管理技术已经不合适日趋复杂的应用。频率和时间综合考虑的缓存替换方法、多级缓存的替换技术成为研究的热点。本文综述了缓存替换算法的研究现状,首先介绍传统简单的缓存替换算法及其存在的局限性,而后对单级缓存算法和多级缓存算法当前最新的研究进行了介绍。最后对研究的趋势进行了简单的展望。

关键词 缓存替换算法,多级缓存

1 引言

随着处理器的数度和磁盘 I/O 性能之间的差距越来越大,缓存技术作为缓解这种性能差距的主要技术也一直是工业界和学术界的研究热点。特别是这几年信息技术的迅速发展,需要保存和处理的信息越来越多,对 I/O 系统性能的要求也越来越高。同时伴随着网络和网络存储系统的普遍应用,带来了与原有本地 I/O 系统不同的网络存储系统性能问题。这种情况的出现对缓存技术又提出了新的要求。

缓存替换算法作为缓存技术的核心技术之一自然受到广泛的研究和关注。本文主要对最近几年提出的缓存替换算法进行简单的介绍和总结。文中第 2 节简述以前传统的缓存替换算法,并指出它们存在的主要问题,同时对当前的缓存算法进行简单的分类。第 3 节介绍几种具有代表性的单级缓存算法。第 4 节介绍多级缓存算法。第 5 节对全文进行总结。

2 传统缓存替换算法

传统的缓存替换算法主要利用对数据块的访问频率或者最近的访问时间为标准判断数据块在本次替换操作中应该被替换掉。这些替换算法的判断标准都比较单一,且基本上只对一类访问模式有比较好的效果,对其他的访问模式其效果明显不好。传统的缓存替换算法主要有 LRU、MRU、LFU 三类代表。

LRU(Lease Recently Used)最近最少使用替换算法。该算法是被使用的最为普遍的基本缓存替换算法,其思想是在进行数据替换时将缓存中最近最少使用的数据块替换出缓存。其实现方法也比较简单,一般是采用一个栈结构,将最近被访问的数据块放置到栈定,替换时将栈底的数据块替换掉。该方法实现简单,额外的开销很少,效果明显,在实际系统中绝大多数都是采用该方法。该替换算法的特点是很适合具有高局部性的数据访问模式,但是对于顺序访问扫描这种数据访问模式则表现得很糟糕,在扫描的数据大小超过缓存大小的时候,LRU 就起不到任何缓存作用。

MRU(Most Recently Used)最近被使用替换算法。该算法在进行数据块的替换时,选择最近被访问的数据块进行替换,该算法在循环访问的模式下表现较好,而对于那些具有很强局部性的访问其性能表现很差。

LFU(Least Frequently Used)最少访问频率替换算法。该算法在进行数据块替换时,选择缓存数据中最少被访问的数据块进行替换。该替换算法对那些符合不相关访问模型的工作负载具有比较好的表现,比如随机 B-tree 的查找,但是该算法在那种活动集随时间变化的访问模式中性能很差。

这些传统算法的特点就是实现简单,而且目前被广泛地使用,特别是 LRU 被当前绝大多数缓存系统所采用。但是这些传统算法不具有针对应用变化的自适应调节能力,而随着现在系统中应用的不断变化,对缓存算法的适应性提出了新的要求。同时随着网络应用的普遍,网络存储设备、网络服务器和网络客户机之间的缓存形成了多级缓存的结构,传统的本地单级缓存算法已经不能够适应这种变化,提高多级缓存的性能和利用率也是亟需解决的问题。因此最近几年的缓存算法研究都集中在复杂的本地单级缓存的自适应性算法研究和多级缓存的整体性能上。这些缓存替换算法按照层次可以划分为单级缓存算法和多级缓存算法。单级缓存可以分

*)西南大学青年基金项目(SWNUQ2005014)

为基于频率和时间平衡策略的算法(Balancing Recency and Frequency policy)、基于特殊应用的策略(Application-specific)和基于探测的策略(Detection-based caching)。而多级缓存可以划分为层次感知的策略(Hierarchy-aware caching)和协同操作的策略(Aggressively-collaborative caching)。以下内容会对这几种缓存策略进行具体的介绍。

3 单级缓存替换算法

当前单级缓存算法的研究更加侧重于对各种应用访问模式的自适应能力。因为基于特殊应用策略的缓存替换算法,比如 DBMIN、Application-Controlled File Caching 等一般属于为专有系统特殊的应用场景而定制的,其通用性很有限,所以本文对该内容不进行介绍。

单级缓存替换算法研究中基于频率和时间平衡策略的算法有很多,比如 LRFU、ARC、MQ/2Q 等。以下我们对这些系统进行简单介绍。

3.1 LRFU

LRFU 算法的基本思想是采用一个权值函数,该函数对所有以前的访问进行评估,其评估原则要兼顾 recency(最近访问时间)和 frequency(频繁访问时间)两个特性,并且最近一次访问所占权值最大,越往后其权值越小。该权值被称作 CRF(Combined Recency and Frequency)值,表示该块未来被访问的可能,是进行块替换的依据。该值通过权值函数和记录的上次访问时 CRF 值得到。该算法理论上的模型如下:

$$C_{tbase}(b)=\sum_{i=1}^{k} F(t_{base}-t_{bi});$$

$C_{tbase}(b)$即缓存中数据块 b 在第 base 时间点的 CRF 值,F(x)是权值函数,$\{t_{b1}\ t_{b2}\ t_{b3},\cdots,t_{bk}\}$是块 b 被访问的时间点。但是在实际实现中不可能对全部访问的时间点都记录,这样空间和时间负荷都是不合算的。因此选取一个权值函数变得尤为重要,在该算法中采用函数$(1/p)^{\lambda x}(P>=2)$作为算法的权值函数。该函数主要有如下特性:

$$C_{tbk}(b)=F(0)+F(\pounds)C_{tbk-1}(b)$$

$$\pounds=t_{bk}-t_{bk-1} \qquad (1)$$

因此在实现中只需记录上次访问的时间和上次访问时的 CRF 值即可算出当前时间的 CRF 值,使得算法切实可行。该权值函数还具有以下两个特性:1)如果 λ 为 1 则它和 LRU 算法一致,根据 CRF 替换的和 LRU 算法是同一个块;2)如果 F(x)= C,C 为常数,则算法和 LFU 一致,替换的是同一个块。对于缓存中的任意数据块 b,算法中有一个参数 LAST(b)记录该数据块上次被访问的时间,CRF_{last}(b)记录块 b 上次被访问时的 CRF 值。同时算法中维护一个堆结构,以 CRF 值为基础记录这些块的顺序关系,根节点是具有最小 CRF 值的块所对应的记录项,最缓存替换时选取堆的根节点作为替换的元素。其算法流程如下:

1. 如果当前访问的数据块 b 在缓存中,根据公式(1)更新块 b 的 CRF_{last}值,并记录 LAST(b)为当前的访问时间 Tc(该值是相对值,即访问缓存的次数),然后将堆进行重新排序;

2. 如果该数据块 b 不在缓存中,则从磁盘中读取该块,将 CRF 设置为 F(0),更新 LAST(b)值为当前访问时间 Tc,然后将堆中的根节点替换掉,该节点是所有块中具有最小 CRF 值的块。将新读入的块插入后在调整堆的排序,比较块 b 的 F(Tc − LAST(b)) * CRF_{last}(b))值和它的孩子节点对应的 F(Tc − LAST(a)) * CRF_{last}(a))值进行堆的重排序。

3.2 MQ/2Q 算法

MQ 算法是一种本地缓存替换算法,它是针对两级缓存中的二级缓存设计的,之所以称之为本地缓存算法,因为它无需从一级缓存中得到任何信息。其优势在于不需要对一级中的软件进行任何的修改。

它的主要思想是在缓存中根据数据块不同的访问频率决定其在缓存中保存时间长短。算法如下:

1. 如果请求块 b 在 cache 中命中,则 b 从当前的 LRU 队列中删除,根据它的访问频率放到相应队列 QK 的队尾。即 k 的值根据已访问频率为参数的函数来确定。

2. 如果请求块 b 在 cache 中 miss,MQ 算法从 cache 中,用块 b 替换掉该最低一级的非空队列的第一个块,这种替换只是占用它的空间,块 b 仍然放在队尾。假设块 c 是被块 b 替换的块,则将该块的标示和当前的访问频率插入到队列 Qout 的尾巴。如果 Qout 队列已满,则将最老的标示从队列中删除。

3. 如果当前的请求块 b 在队列 Qout 上有记录,将块 b 读取到 cache 中后,将它的访问频率设置为队列 Qout 上记录的其相应访问频率值 + 1,并且将 b 在队列 Qout 中相应的项删除。这个时候它替换的就不是最低一级的队列的第一个块,而是根据它当前的访问频率计算到应该放入哪个队列中,将该队列的第一个块替换出去。

4. 如果当前的请求块 b 不在队列 Qout 中,则它被读入到 cache 中后,其访问频率被设置为 1。

5. MQ 算法还根据随时间变化的访问次数,将不活跃的块从高级队列低一级的队列中,以此来对队列进行调整,去除那些以前被访问很多次,但最近没有被访问的块,防止它一直存在高一级的队列中,从而兼顾访问频率和访问时间。

具体做法如下 ,每个块有个 expireTime 参数,该时间参数表示逻辑时间,通过访问的次数来进行计算。如果一个块在一个队列中,经过了允许的时间,而没有再被进行任何访问,则将该块移到下一级队列中。在每个块加入到一个队列中,它的 expireTime 被设置为 currentTime + lifeTime,lifeTime

是个可调整的参数，决定在没有任何访问下一个块能在队列中呆多长时间。当每一个访问发生时，所有队列第一个块的 expireTime 都被检查，如果该值小于系统当前的 currentTime 则该块被移到下级队列中，并且重新设置它的 expireTime。

2Q 算法可以考虑成上述算法的特殊情况，即 LRU 队列数 m 为 2 的情况。与 MQ 有所不同的是，2Q 不是采用两个 LRU 队列，而是 1 个 LRU 队列和 1 个 FIFO 队列。2Q 算法没有像 MQ 算法那样，具有将数据块从 Q0 移到 Q2 的调整机制，而且对于 Q1 队列上的元素被替换后也不像 MQ 那样移到 Qout 队列上。

4 多级缓存替换算法

多级缓存算法又可以划分成两类：Hierarchy-aware caching 和 Aggressively-collaborative caching。Hierarchy-aware caching 存储服务器知道在它的前端存在大量的客户端缓存，但不需要对自己 I/O 接口和存储客户端软件有任何改变。主要通过猜测的手段对缓存的冗余等情况进行判断，可以通过数据块的地址、文件语义等信息进行猜测。Aggressively-collaborative caching 修改 I/O 接口和存储客户端软件。又可以分为基于线索的方式 hit-based，比如 TQ，DEMOTE 等，客户端控制的方式 client-controlled，比如张晓东老师的 ULC。这里只介绍 TQ 算法。

TQ 算法中维护两个替换队列，一个是高优先级队列，一个是低优先级队列，高优先级队列主要保存同步和替换的操作相关的数据块，低主要保存上述 1、3 情况下的所涉及到的数据块。因为 2 情况下引发对某个数据块的请求，如果该数据块不在缓存中，则将其添加到高优先级队列上，如果该数据块在低优先级队列上，则将其移到高优先级队列上。这两个队列的大小不固定，随着数据的访问行为而变化。而对 1 引发的数据块请求，如果当前的数据块在高优先级队列，则将其移到低优先级队列。当缓存需要发生替换操作时，如果低优先级队列非空，则从该队列中按照 LRU 算法选择一个数据块进行替换。第 3 种情况，在该算法中被忽略，不影响缓存内容和两个队列数据块顺序。只有在冷启动时，如果缓存不满，在情况 3 时相应的数据块可以添加到低优先级队列。低优先级队列采用 LRU 替换算法管理，而高优先级队列采用文中提出的 LPR- 最新预测读算法。当一个数据块被加入到高优先级队列时，算法中对该数据块下次被读到时间 nextReadPosition 进行预测，当该队列进行数据块替换时，选择 nextReadPosition 值最大的数据块进行替换。nextReadPosition 值由数据块期望的读写距离 (write-to-read distance) 加上当前缓存中的请求数确定，读写距离是该数据块上次被同步或者替换写操作后到随后的因为读请求之间的缓存请求数量决定，而期望读写距离是指该数据块的每次计算后的读写距离的平均值。TQ 算法和 MQ 算法一样保存一个 out 队列，该队列记录最近被替换掉的数据块的读写距离和其他的访问统计信息等。该队列的大小以算法的参数设定，当前该队列中的元素需要进行替换时选择读写距离最大的数据块进行替换。当数据块被添加到缓存中时，TQ 算法检查 out 队列是否包含该数据块信息，如果有，则从中获取该数据块的读写距离确定 nextReadPosition 值，否则将其读写距离设置为无限大。在高优先级队列中追踪每个数据块最后一次替换或者同步写请求以来的缓存请求数，包括向缓存中添加新数据块，或者在缓存中命中的请求数，一旦有读请求命中该数据块，则可以通过跟踪记录的请求数量立即算出该数据块的当前读写距离。然后根据当前计算的读写距离更新每个数据块的平均读写距离，并记录。

5 总结和未来工作介绍

基于 recency/frequency 平衡策略仍然是研究的热点。基于访问模式探测是努力的方向，它将计算机其他的领域的技术结合进缓存替换技术之中。多级缓存技术是研究的重点，同时我们必须考虑其他因素对缓存算法的影响，比如有研究预取技术对各种缓存算法和考虑能耗的缓存管理算法，我们对缓存算法的评估方法是以模拟为主，实测为辅。当前存在的问题包括部分算法实用性较差，CLOCK-pro、ARC 等少数现在被用到实际系统中。

参 考 文 献

1 Agha G. Concurrent Object-Oriented Programming. CACM, 1990,33(9)

2 Zhou Feng, von Behren R, Brewer E. Program Context Specific Buffer Caching with AMP:[UCB Technical Report UCB//CSD-05-1379]. April 2005

3 Lee Donghee, Choi Jongmoo, Kim Jong-Hun, et al. LRFU (least recently/frequently used) replacement policy: A spectrum of block replacement policies. IEEE Transactions on Computers, 2001,50(12):1352～1361

4 Megiddo N, Modha D S. Outperforming LRU with an adaptive replacement cache algorithm. Computer, 2004,37(4):58～65

5 Megiddo N, Modha D S. ARC: A Self-Tuning, Low Overhead Replacement Cache, USENIX File and Storage Technologies (FAST), San Francisco, CA, March, 2003

6 Bansal S, Modha D S. CAR: Clock with Adaptive Replacement, USENIX File and Storage Technologies (FAST), San Francisco, CA, March 31-April 2, 2004

7 Zhou Yuanyuan , Chen Zhifeng, Li Kai . Second-Level Buffer Cache Management. IEEE Tran on parallel and distributed system, 2004,15 (7)

8 Zhou Yuanyuan, Philbin J, Li Kai. The Multi-Queue Replacement Algorithm for Second Level Buffer Caches. In: the Proceedings of USENIX Technical Conference, June 2001

9 Johnson T, Shasha D. 2Q: a low overhead high performance buffer management replacement algorithm. In: Proc. VLDB-20, September 1994

10 Li Kai, Felten E W, Cao Pei. Application-controlled file caching policies. In: Proc. of USENIX Summer 1994 Technical Conference, 1994. 171～182