Frequency-Based Replacement with Reversing Insert and Tail Decay Policy for Cache Replacement*

石大川 <sdc17@mails.tsinghua.edu.cn>

2020 年 3 月 18 日

目录

*https://github.com/sdc17/FBRRD

1 Introduction

本文首先实现了 JT Robinson 和 MV Devarakonda 提出的 Frequency-Based Replacement (FBR) [1]缓存替换算法，该算法与缓存模拟器中已存在的其他 7 种缓存替换算法相比，在 6 个 trace 的测试中取得了 Hit Rate 指标的 5 个第一和 1 个第二，以及平均 Hit Rate 的最高值 17.33%。在此基础上，本文针对 FBR 存在的缓存污染问题，以及尾部权值过大导致 Cache 块无法短时间内被换出的问题，提出了改进的 FBR 算法 Frequency-Based Replacement with Reversing Insert and Tail Decay (FBRRD)。FBRRD 算法针对上述的两个问题分别采取了将新的 Cache Line 插入到 New Section 的近似尾部区域，以及在每轮 Cache 替换中对 Old Section 的尾部数据做权值衰减的方法。在 6 个 trace 的测试中，FBRRD 算法相较于 FBR 算法分别获得了 0.54% ～ 1.66% 的 Hit Rate 提升，平均 Hit Rate 达到了 18.62%，高出第二名的 SRRIP_FP 算法 2.20%. 最后，本文分析了 FBRRD 的开销和可能的改进方向。

2 Method

2.1 FBR Policy

首先介绍 FBR 算法的基本原理并分析其设计的合理性，再给出对照实验的结果，并进一步分析 FRB 算法存在的不足之处。

2.1.1 Principle

Figure 1摘自 FBR 原论文[1]，其展示了 FBR 算法的工作流程和对缓存的结构划分。首先将缓存划分为如Figure b所示的 New Section, Middle Section 和 Old Section 三个区域，并为其中的每个 Cache Line 维护一个引用计数的值。Figure a的工作过程代表：

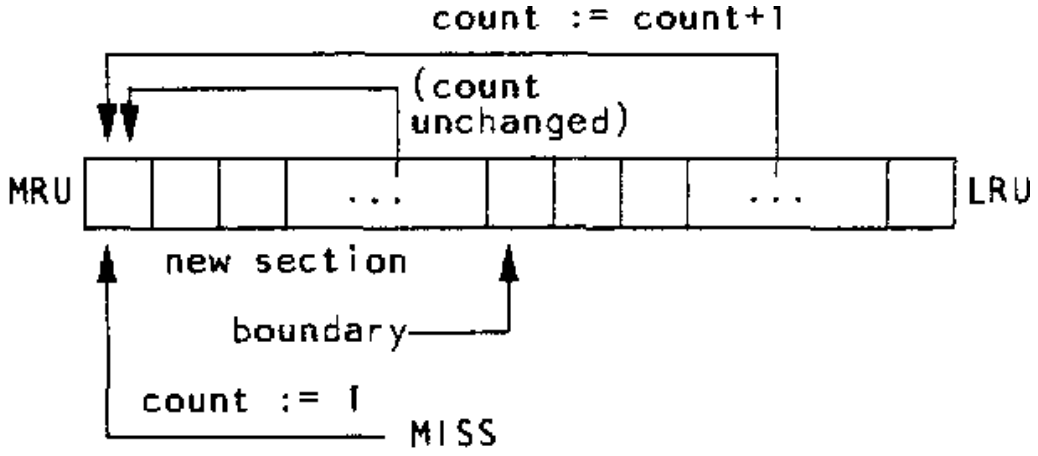
Hit 栈中任何位置的 Cache Line 被命中时都会被移动到 New Section 的头部，若命中位置在 New Section 内，则维持该 Cache Line 的引用计数值不变，否则引用计数值加 1。

Insert 新插入的 Cache Line 会被置于 New Section 的头部，并将其引用计数置为 1.

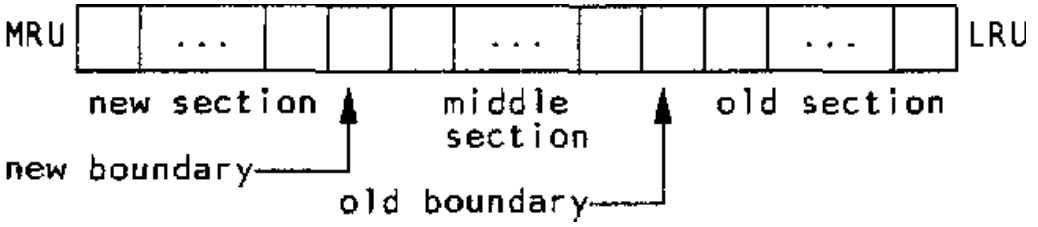
Find Victim 选择 Old Section 中引用计数值最小的 Cache Line 作为 Victim.

2.1.2 Analysis

总的来说，FBR 的优势在于巧妙地将 LRU 和 LFU 结合了起来，访问时序和访问次数信息的结合使得 FBR 算法在面对不同的访问模式时有更灵活，更鲁棒的表现。一方面，三个 Section 的划分是对访问时序信息的利用：被替换的块只在位于队尾的 Old Section 中产生，它们是较长时间未被访问的 Cache Line，而被命中的块和新读入的块则是按照 LRU 的模式插入到 New Section 的队首，它们是最近使用到的 Cache Line。另一方面，替换出引用计数值最小的块则是对访问次数信息的利用，这与 LFU 的思路是相近的。对于 FBR 命中率的分析可以根据以下五种访问模式进行讨论：



(a) Workflow



(b) Section division

Figure 1: FBR policy

Streaming Access 也即:

$$(a_1, a_2, a_3, ..., a_k) \quad k = \infty$$

在这种情况下所有的块按顺序被逐个访问，没有被重新访问的情况，因此 Hit Rate 为 0.

Thrashing Access and Recency-friendly Access 也即:

$$(a_1, a_2, ..., a_k)^N$$

$$(a_1, a_2, ..., a_k, a_k, ..., a_2, a_1)^N$$

此时若 $k <= CacheSize$ 显然 Hit Rate 为 100%。否则，此时所有 Cache Line 在缓存中按照进入的顺序排列，每个 Cache Line 的引用计数值都为 1，FBR 退化为了 LRU。每次队尾的 Cache Line 会被替换出去，Hit Rate 为 0.

Densly Access 也即密集地访问最近被访问的块，如:

$$(a_1, a_1, ..., a_1, a_x, a_y, ..., a_z, ..., a_1, a_1, ..., a_1)^N \quad x, y, z \neq 1$$

FBR 在这种情况中是有优势的。在一些单纯考虑访问次数(频率)信息的替换算法如 LFU，或者其他采用了访问频率信息的替换算法中，对于 a_1 这个 Cache Line 的长时间持续访问往往会使得其访问计数值变得很大，导致在这段密集访问结束后，这个 Cache Line 需要在缓存中停留很长时间才能被替换出去。然而在 FBR 中，由于对 New Section 区域的 Cache Line 进行重复访问并不会增加其引用计数值，因此 a_1 的引用计数值会始终保持它刚被插入时赋予的 1。当这一密集访问的序列结束后，a_1 掉入到 Old Section 中将很快会被置换出去。

Fixed Probability Access 也即访问每个 Cache Line 的概率都是相互独立且固定的，如：

$$(a_1, a_1, a_2, a_1, a_2, a_3, a_3, a_3, a_1, a_1, ...) \quad P(a_1) = 0.5 \quad P(a_2) = 0.2, \quad P(a_3) = 0.3$$

FRB 在这种情况中也是有优势的。因为访问序列在时间轴上的分布几乎没有规律而言，而是按照固定的概率值来确定当前访问哪一个 Cache Line, 所以一些单纯考虑访问时间序列信息的替换算法如 LRU 或者 FIFO 等就退化为了 RAND 的性能。然而 FRB 却可以根据访问计数值筛选出并留下被较大概率访问到的 Cache Line。

Mixed Access 混合型的访问模式. 也即可能出现：包括但不限于前面四种访问模式的组合访问。这是最通常的访问模式，直接用实验结果来说明该模式下 FBR 替换算法的性能。

2.1.3 Result

设置 New Section 部分占缓存比重为 $\frac{1}{4}$, Old Section 部分占缓存比重为 $\frac{1}{2}$, 可以得到：

Trace	RAND	LRU	FRU	SRRIP	SRRIP_FP	BRRIP	DRRIP	FBR
fort_1	9.84	9.76	11.14	10.54	10.08	**14.49**	10.66	11.64
fort_2	14.01	15.77	11.44	15.68	16.01	14.38	15.58	**16.39**
hpjy_1	14.84	16.06	11.23	16.10	16.21	14.69	16.01	**17.17**
hpjy_2	15.44	16.59	14.25	17.30	17.12	15.68	17.21	**18.39**
wzry_1	16.27	18.49	16.26	18.81	18.87	19.59	18.83	**20.89**
wzry_2	17.54	19.99	16.07	19.88	20.20	19.08	19.84	**21.92**

表 1: Hit Rate(%) on 6 traces

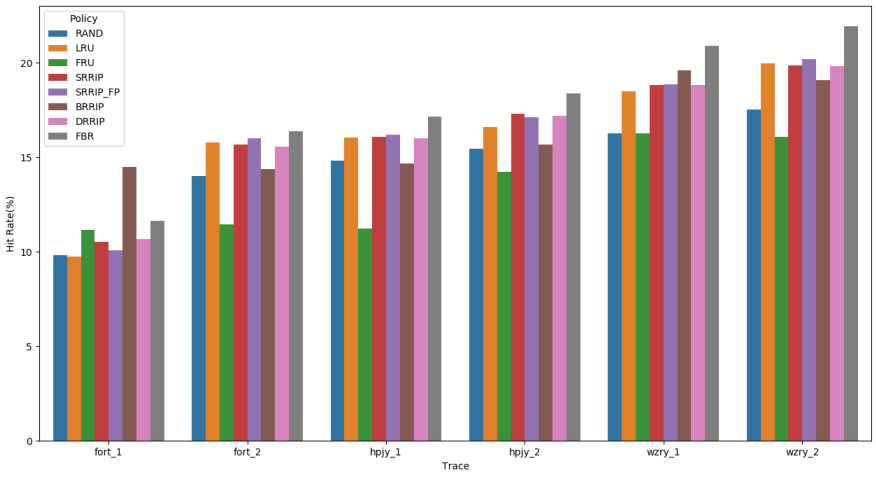


Figure 2: Hit Rate on 6 traces

Chart 1和Figure 2展示了已有的 7 种替换算法和 FRB 替换算法的 Hit Rate. 每个 trace 最右边灰色的柱体代表 FBR 替换算法，可见除了在 fort_1 上其 Hit Rate 位居第二名，在其他的 5 个 trace 上 FBR 都取得了最高的 Hit Rate.

为了进一步提高 Hit Rate, 可以先观察 fort_1。可以看出 BRRIP 算法[2]在 fort_1 上的 Hit Rate 是有显著优势的。对照 BRRIP 算法和 FBR 算法具体实现上的细节，可以发现它们之间一个重要的区别在于：BRRIP 算法每次插入新的 Cache Line 时，其 RRPV 的值有 $\frac{1}{16}$ 的概率为 2^M-2, 否则为 2^M-1，而 FBR 算法则是将新 Cache Line 的 LRU 值置为 0。虽然 FBR 的 LRU 值和 BRRIP 的 RRPV 值含义并不完全相同，但本质上它们都是一个赋予各个 Cache Line 在面对替换时优先级的序列。也就是说 BRRIP 以及 SRRIP[2], SRRIP_FP[2]和 DRRIP[2]在面对新插入的 Cache Line 时，序列赋予其的优先级都是较低的，而在 FBR 中赋予的优先级则是较高的。这种优先级之间的区别就引入了本文对于 FBR 算法第一部分的改进，即 Frequency-Based Replacement with Reversing Insert(FBRR)。

2.2 FBRR Policy

2.2.1 Analysis

如前文所述，FBR 赋予新插入的 Cache Line 高优先级的这一特点会使得其在一些缓存污染的情况下 Hit Rate 下降。

缓存污染是指因为不常用的数据被插入到了缓存中，使得常用的数据被挤出缓存，导致 Hit Rate 下降的情况。当然这里不常用的数据在替换算法看来，是被误认为常用的，因此才会被替换算法插入缓存。举例来说，试想有一个容量为 12 个 Cache Line 的缓存，其 New Section Size 为 4，Old Section Size 也为 4，现有如下的访问序列：

$$\{1,2,...,12\}^M,\{a,b,...,z\},\{1,2,...,12\}^N \quad M\to\infty$$

在持续对 $\{1,2,...,12\}^M$ 的数据访问的过程中 (不一定是按顺序循环访问，只要 12 个数字代表的 Cache Line 都被访问到了足够多的次数即可)，偶然出现了需要对其他数据 $\{a,b,...,z\}$ 访问一次的情况，按照 FBR 算法的流程，新来的 $\{a,b,...,z\}$ 会依次被赋予最高的优先级,也即插入到缓存的队首,$\{a,b,...,z\}$ 插入完成后,对于之前缓存中的 $\{1,2,...,12\}$, 其中访问计数值较小的 9 个都会被替换出去，这种情况下 Hit Rate 就会下降。详细的过程为：首先假设在即将访问 a 时，当前缓存中的情况为：

$$\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4}||\mathbf{5},\mathbf{6},\mathbf{7},\mathbf{8}||\mathbf{9},\mathbf{10},\mathbf{11},\mathbf{12}$$

其中 || 用来区分 Section，从左至右依次是 New Section, Middle Section 和 Old Section。其中加黑的 Cache Line 代表着有较大的访问计数值。接着插入 a, 那么 $9,10,11,12$ 中访问计数值小的那一个会被替换出去，不妨假设是 12：

$$a,\mathbf{1},\mathbf{2},\mathbf{3}||\mathbf{4},\mathbf{5},\mathbf{6},\mathbf{7}||\mathbf{8},\mathbf{9},\mathbf{10},\mathbf{11}$$

之后同理，直到插入 i 之前：

$$h,g,f,e||d,c,b,a||\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4}$$

再插入 i, 那么么 $1,2,3,4$ 中访问计数值小的那一个会被替换出去，不妨假设是 4：

$$i,h,g,f||e,d,c,b,||a,\mathbf{1},\mathbf{2},\mathbf{3}$$

再插入 j 时，a 访问计数值小于 $1,2,3$ 因此替换出 a：

$$j,i,h,g||f,e,d,c||b,\mathbf{1},\mathbf{2},\mathbf{3}$$

此后对于 $\{k, l, ..., z\}$ 的访问被替换的都是 Old Section 头部的 Cache Line，直到 z 访问完：

$$z, y, x, w||v, u, t, s||r, \mathbf{1}, \mathbf{2}, \mathbf{3}$$

此时再回到 $\{1, 2, ..., 12\}^M$ 的访问模式时，$\{4, 5, ..., 12\}$ 都会 Miss, Hit Rate 仅有 $\frac{3}{12} = \frac{1}{4}$.

而对于 BRRIP 以及 SRRIP,SRRIP_FP 和 DRRIP 算法而言，新来的 $\{a, b, ..., z\}$ 会被赋予较低的优先级，也即赋予较大的 RRPV 值 ($2^M - 1$ 或者 $2^M - 2$)。而由于它们都只被用到 1 次，因此与被用到多次的 $\{1, 2, ..., 12\}$ 相比它们的 RRPV 值都是较大的，所以它们紧接着又会被下一个插入的 Cache Line 依次替换出去，在这种情况下，缓存中损失的仅仅是 $\{1, 2, ..., 12\}$ 中被访问次数最少的那一个数据，例如：

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, z$$

此时再回到 $\{1, 2, ..., 12\}^M$ 的访问模式时，Hit Rate 为 $\frac{11}{12}$.

2.2.2 Principle

基于上述对比，可以将 FBR 算法在插入新的 Cache Line 时的位置按 BRRIP 等算法的方式适当调整：将新来的 Cache Line 插入到 New Section 的尾部区域，而不是队首。以上一小节中的同一个例子来进行分析，假设插入的位置为 New Section 的末尾，开始时缓存中为：

$$\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}||\mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}||\mathbf{9}, \mathbf{10}, \mathbf{11}, \mathbf{12}$$

插入 a 后：

$$\mathbf{1}, \mathbf{2}, \mathbf{3}, a||\mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}||\mathbf{8}, \mathbf{9}, \mathbf{10}, \mathbf{11}$$

一直到插入 e 后：

$$\mathbf{1}, \mathbf{2}, \mathbf{3}, e||d, c, b, a||\mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}$$

再插入 f：

$$\mathbf{1}, \mathbf{2}, \mathbf{3}, f||e, d, c, b||a, \mathbf{4}, \mathbf{5}, \mathbf{6}$$

此后对于 $\{g, h, ..., z\}$ 的访问被替换的都是 Old Section 头部的 Cache Line，直到 z 访问完：

$$\mathbf{1}, \mathbf{2}, \mathbf{3}, z||y, x, w, v||u, \mathbf{4}, \mathbf{5}, \mathbf{6}$$

此时再回到 $\{1, 2, ..., 12\}^M$ 的访问模式时，$\{7, 8, ..., 12\}$ 会 Miss, Hit Rate 提升到了 $\frac{6}{12} = \frac{1}{2}$，是插入到 New Section 首部 Hit Rate 的两倍。

有两点需要说明的是：首先这里的尾部区域不是严格意义上的最后一个 Cache Line，而是位于 New Section 后半部分的所有 Cache Line，也即位于

$$[\lceil \frac{NewSectionSize}{2} \rceil, NewSectionSize - 1]$$

之间的 Cache Line。这是因为如果都插入到最后一个 Cache Line, 那么在其他的访问模式下又会引发新的缓存缺失问题，实际上鲁棒性是不强的。改变插入位置的原意是适当降低新插入的 Cache Line 的优先级，因此只要是插入到相对靠后的位置就相当于降低了优先级。置于为什么不插入到前半部分，这是因为缓存的规模不大，New Section 又只占 25%, 因此插入到 New Section 前半部分的某个地方实际上对优先级的降低不够显著。而具体插入到后半部分的哪个位置，或者是按一定比例 (类似于 BRRIP) 插入到后半部分的多个位置，则是通过在实验中的具体表现来决定的。

第二,这里的 Reversing Insert 只到 New Section 的末尾部分为止,而不是 Middle Section 或者 Old Section 的某个位置，原因在于：若首次插入到 Middle Section 中，那么它的初始计数值为 1。当再次访问它时，它在被调入 New Section 的同时访问计数值会变成 2, 这破坏了原来多次访问同一 Cache Line 仍然保持计数值为 1 的性质，会使得后续该 Cache Line 进入 Old Section 中等待更长的时间才会被替换出去；若首次插入到 Old Section 中，那么刚被插入的 Cache Line 就有被替换出去的较大风险，这违背 FBR 适度保护新插入的 Cache Line 的本意。

2.2.3 Result

通过多次实验，可以得出一个较优的插入位置是 $\lceil \frac{NewSectionSize}{2} \rceil$, 与原始的 FBR 算法相比实验结果如下：

Trace	FBR	FBRR
fort_1	11.64	**12.35**
fort_2	16.39	**16.50**
hpjy_1	17.17	**17.21**
hpjy_2	18.39	**18.61**
wzry_1	20.89	**21.15**
wzry_2	21.92	**22.16**

表 2: Hit Rate(%) on 6 traces

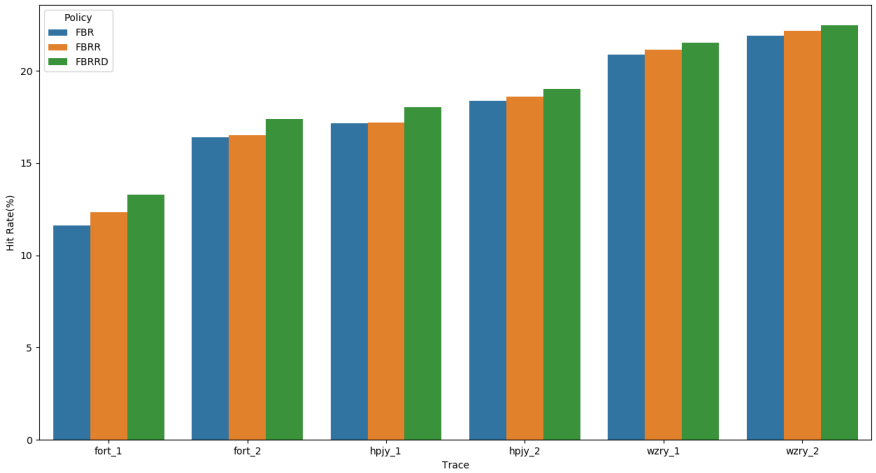


Figure 3: Hit Rate on 6 traces

由Chart 2和Figure 3的蓝色 (FBR) 与橙色 (FBRR) 柱体对比可知，FBRR 算法在 6 个 trace 上均获得了高于 FBR 算法的 Hit Rate。其中 FBRR 算法在 fort_1 上提升尤为明显，达到了 0.71%，这是符合原理上预期的。

2.3 FBRRD Policy

2.3.1 Analysis

FBR 算法在更新引用计数值时有一个特点：New Section 之外的 Hit 会增加引用计数值，而 New Section 之内的 Hit 则不会。前文已经提到这样做的目的是防止密集访问造成部分 Cache Line 引用计数值过大，从而难以被替换出去。但是，仅仅依靠这一特点还是不能完全避免 Cache Line 引用值过大的情况发生。

举例来说，试想有一个容量为 12 个 Cache Line 的缓存, 且其 New Section 部分大小为 4 个 Cache Line, Old Section 部分大小也为 4 个 Cache Line, 现有如下的访问序列：

$$\{1,2,3,4,5\}^K, \{a,b,c,d,e,f,g,h,i,j\}^M, \quad k \to \infty$$

假设开始时缓存为空，那么在每一轮对于 $\{1,2,3,4,5\}^K$ 的访问中，每个时刻总会有一个 Cache Line 是在 New Section 之外，也即 Middle Section 开头的，这意味着在下一轮访问中再次命中该 Cache Line 时，其访问计数值会加 1。因此每一轮访问结束后，这 5 个 Cache Line 的访问计数值都会被加 1。当访问轮数 k 很大时，这些计数值也会很大。当 k 轮结束后，访问序列中不会再出现对于 $\{1,2,3,4,5\}$ 的访问了。此时被淘汰到 Old Section 中的 $\{1,2,3,4,5\}$ 因为巨大的计数值将继续存在很长一段时间。而后续的 $\{a,b,c,d,e,f,g,h,i,j\}$ 的访问周期是 10, 当 h 即将被插入时，缓存状态为：

$$g,f,e,d||c,b,a,\mathbf{5}||\mathbf{4},\mathbf{3},\mathbf{2},\mathbf{1}$$

其中 || 将缓存从左到右分为了 New Section, Middle Section, Old Section 三个区域，黑体部分为访问计数值很大的 Cache Line.h 被插入时要从 $4,3,2,1$ 中选一个替换出去，它们的访问计数值是一样的，则队尾的 1 会被替换出去，此时缓存状态为：

$$h,g,f,e||d,c,b,a,||\mathbf{5},\mathbf{4},\mathbf{3},\mathbf{2}$$

插入 i 时同理：

$$i,h,g,f||e,d,c,b,||a,\mathbf{5},\mathbf{4},\mathbf{3}$$

插入 j 时，此时 a 的访问计数值远小于 Old Section 中的 $5,4,3$, 因此 a 被替换出去, 缓存状态变为：

$$j,i,h,g,||f,e,d,c||b,\mathbf{5},\mathbf{4},\mathbf{3}$$

接下来重新访问 a, 得到 Miss, 替换出 b, 缓存状态变为：

$$a,j,i,h||g,f,e,d||c,\mathbf{5},\mathbf{4},\mathbf{3}$$

重新访问 b, 得到 Miss, 替换出 c, 缓存状态变为：

$$b,a,j,i||h,g,f,e,||d,\mathbf{5},\mathbf{4},\mathbf{3}$$

再之后的重新访问都是同理的循环 Miss. 可见在这种情况下 FBR 算法的 Hit Rate 会大幅降低。而这种数据访问模式在实际中是经常会遇到的，例如循环访问 $a[5]$ 数组几万次，然后循环访问 $b[10]$ 数组，那么大量 Cache Miss 就会发生。

2.3.2 Principle

为了减轻 FRB 算法在上述访问模式中的问题，可以在每轮 Cache Line 替换时做 Tail Decay，也就是以一定概率将缓存尾部 Old Section 中 Cache Line 的访问计数值减 1(如果其大于 1)。这样即使 Old Section 中存在访问计数值较大的 Cache Line，也会逐步衰减到能被正常替换出去的程度。

2.3.3 Result

在多次实验后，发现以 0.002 的概率减小 Old Section 中所有 Cache Line 访问计数值，且特别地，以 0.01 的概率减小 Old Section 最后一个 Cache Line 的访问计数值可以取得 Hit Rate 较大的提升，具体实验结果如Chart 3和Figure 3的绿色柱体 (FBRRD) 所示。

Trace	FBR	FBRR	FBRRD
fort_1	11.64	12.35	**13.30**
fort_2	16.39	16.50	**17.39**
hpjy_1	17.17	17.21	**18.02**
hpjy_2	18.39	18.61	**19.04**
wzry_1	20.89	21.15	**21.51**
wzry_2	21.92	22.16	**22.46**
Average	17.73	18.00	**18.62**

表 3: Hit Rate(%) on 6 traces

可见相比较于 FBRR 算法，FBRRD 算法在 6 个 trace 上的 Hit Rate 又取得了更进一步的提升，其中在 fort_1 上的提升最为明显，达到了 0.95%, 这是符合原理上预期的.

最终 FBRRD 算法与缓存模拟器中已有的其他其中算法的对比如Chart 4 和Figure 4所示. 可见 FBRRD 算法除在 fort_1 上略低于 BRRIP 外，在其他的 trace 上均明显高于其他算法，且平均 Hit Rate 高出第二名的 SRRIP_FP 算法 2.20%.

Trace	RAND	LRU	FRU	SRRIP	SRRIP_FP	BRRIP	DRRIP	FBRRD
fort_1	9.84	9.76	11.14	10.54	10.08	**14.49**	10.66	13.30
fort_2	14.01	15.77	11.44	15.68	16.01	14.38	15.58	**17.39**
hpjy_1	14.84	16.06	11.23	16.10	16.21	14.69	16.01	**18.02**
hpjy_2	15.44	16.59	14.25	17.30	17.12	15.68	17.21	**19.04**
wzry_1	16.27	18.49	16.26	18.81	18.87	19.59	18.83	**21.51**
wzry_2	17.54	19.99	16.07	19.88	20.20	19.08	19.84	**22.46**
Average	14.66	16.11	13.40	16.39	16.42	16.32	16.36	**18.62**

表 4: Hit Rate(%) on 6 traces

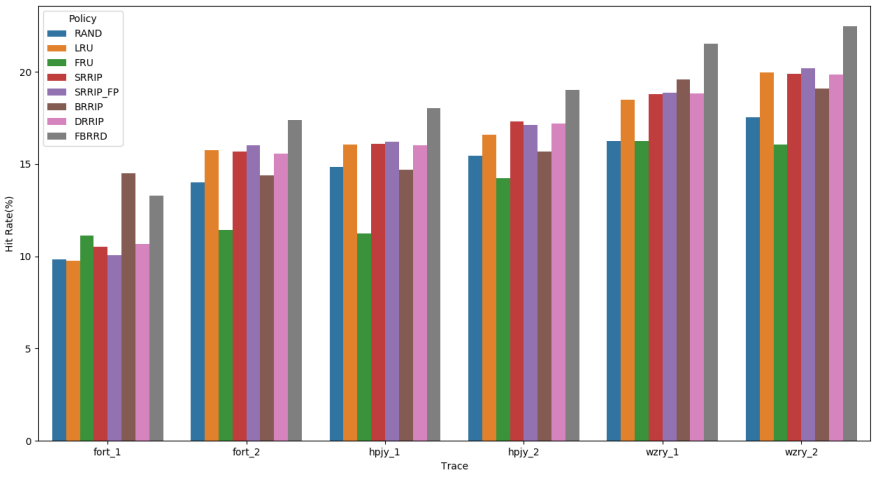
3 Discussion

3.1 Environment

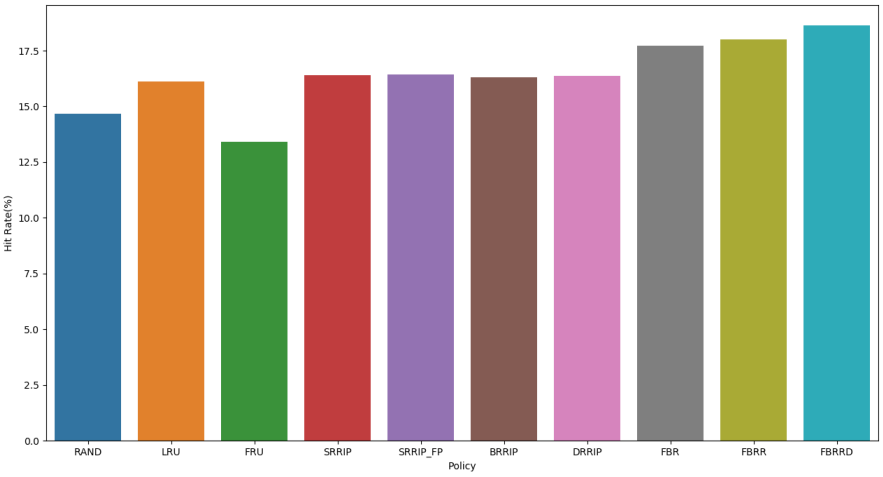
实验环境是 Ubuntu18.04 的虚拟机，分配到了 Intel i7-7700HK 的两个核以及 8G 内存。

3.2 Cost

Storage Cost 在空间开销上，FBR, FBRRF 和 FBRRD 相比已实现的 LRU 算法而言，对每个 Cache Line 除继承了 $\_u32$ 的 LRU 属性值，另外增加了一个 $\_u32$ 的 FBR 属性值作



(a) Detailed Hit Rate on 6 traces



(b) Average Hit Rate on 6 traces

Figure 4: Comparison among FBBRD policy and other policies

为访问计数值，除此之外在空间上没有其他的主要开销。

Time Cost 在时间开销上，FBR，FBRR 与 FBRRD 在每次 Hit/Insert 时需要遍历一次缓存中的 Cache Line，维护 LRU 和 FBR 属性值，Find Victim 时也需要遍历一次缓存中的 Cache Line，筛选出 Old Section 区域并找到最小值访问计数值进行替换。

3.3 Merits

FBRRD 算法相比于 FBR 的优点在于鲁棒性上的提升，在前文分析的多种访问模式下能取得更高的 Hit Rate，同时在时间与空间上的开销是与之持平的。

3.4 Drawbacks

FBRRD 算法的缺点在于除了原有的的 $NewSectionSize$ 和 $OldSectionSize$ 外还引入了 Reversing Insert 位置和衰减概率的超参数，这些超参数的引入增加了算法的复杂性，且在 6 个 trace 上通过实验找出的较优的超参数存在过拟合的可能性。

4 Conclusion

本文在实现了 FBR 算法的基础上讨论了其存在的两个问题，并提出了 FBRRD 算法分别解决或者减轻了这两个问题，使得 FBRRD 算法的 Hit Rate 在 6 个 Trace 上均得到了显著的提升。

FBRRD 算法也仍然存在着一些问题。例如 Tail Decay 的引入并不能完全解决部分 Cache Line 访问计数值过大的问题，由此引入的衰减概率超参数的值也依赖于具体的实验表现，这是 FBRRD 算法中不够优雅的地方。进一步改进 FBRRD 算法可以考虑引入自适应的衰减概率参数，New Section Size 参数和 Old Section Size 参数，使得这些参数能随着算法在运行过程中的表现得到实时的自动调整，进一步减少当前依赖于大量实验的经验性的超参数，并且提升算法在应对新的访问模式时的灵活性和鲁棒性。

5 References

[1] Robinson, John T., and Murthy V. Devarakonda. ”Data cache management using frequency-based replacement.” Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems. 1990.

[2] Jaleel, Aamer, et al. ”High performance cache replacement using re-reference interval prediction (RRIP).” ACM SIGARCH Computer Architecture News 38.3 (2010): 60-71.