arXiv:2304.04954v1 [cs.DS] 11 Apr 2023

一种组相联缓存中的阈值现象

Michael A. B ender* Rathish Das † Mart´ın Farach-Colton ‡ Guido Tagliavini §

摘要

在一种 α 路组相联缓存中，缓存被划分为大小为 α 的互不相交的组，每个项只能缓存在一个组中，通常由哈希函数选定该组。组相联缓存应用广泛，相较于全相联缓存具有诸多优势，例如在延迟或并发性方面，但往往会导致更多缓存未命中。随着集合大小 α 减小，这些优势会增加，但分页开销也会变得更高。

本文刻画了总大小为 k 的 α 路组相联LRU缓存的性能，将其表示为 $\alpha = \alpha(k)$ 的函数。假设使用完全随机的哈希函数选择组，我们证明如下：

对于 $\alpha = \omega(\log k)$， α路组相联LRU缓存的分页开销，与大小为$(1-o(1))k$ 的全相联LRU缓存的分页开销相差至多additive$O(1)$，其概率为 $1-1/\text{poly}(k)$；对于所有长度为 poly(k) 的请求序列均成立。

- 对于 $\alpha = o(\log k)$，以及所有 $c = O(1)$ 和 $r = O(1)$，对于某个长度为 $O(k^{1.01})$的请求序列，α路组相联 LRU缓存的分页开销与大小为k/r 的全相联 LRU缓存的分页开销不在 c 倍范围内。

- 对于 $\alpha = \omega(\log k)$，如果哈希函数可以偶尔更改，则一个α路组相联LRU缓存的分页开销，与大小为$(1-o(1))k$的全相联 LRU缓存相比，在因子 $1+o(1)$以内；对于任意长度（例如超多项式长度）的请求序列，该结论以 $1-1/\text{poly}(k)$ 的概率成立。

我们的一些结果可以推广到除 LRU 之外的其他分页算法，例如最不经常使用（LFU）算法。

1 引言

在一个 α 路组相联缓存中，缓存被划分为互不相交的组，每组大小为 α，并且存在一个索引函数 [57] ，用于选择每个项可放入的唯一组。组大小 α 也称为缓存的相联度。

组相联缓存广泛应用于硬件 [8, 35, 36, 13, 56, 49], ，因为与全相联缓存相比，它们速度更快、电路更简单且功耗更低 [28, 29, 44, 33] ；全相联缓存可以将任何项存储在缓存中的任意槽位。组相联缓存还被用作设计高度并发软件缓存的构件 [1, 48], ，因为组相联缓存中的各组彼此独立，可以并行运行。集合大小 α 越小，所带来的优势就越大。

然而，集合大小 α 越小，分页开销就越大，也就是说，缓存未命中的次数越多。这是因为一个项只能放入其所属组中的一个缓存槽位，因此

*石溪大学。 bender@cs.stonybrook.edu
†利物浦大学. rathish.das@liverpool.ac.uk
‡罗格斯大学。martin@farach-colton.com§罗格斯大学。guido.tag@rutgers.edu

当超过 α 个频繁访问的项落入同一组时，它们会争用该组中的槽位，导致分页增加。

因此，在设计组相联缓存时， α 必须选择得足够小，以提供所需的延迟、功耗、芯片面积或并发性；而 α 必须足够大，以将分页开销保持在较低水平。鉴于这一权衡，缓存设计者如何才能针对 α 进行优化？衡量较小 α 的优势取决于一些超出本文范围的细节。在此，我们关注权衡的另一面：相对于全相联缓存， α路组相联缓存的分页开销增加了多少，这一增加量是 α的函数？

回答这个问题是组相联缓存理论 [54, 55, 51, 59] 和实践 [29, 44, 2, 10] 中的一个众所周知的开放问题。此前唯一部分回答了该问题的理论研究假设项请求来自某个分布 [54, 55, 51, 59]；其界并不适用于对手。证明对抗性设置下的最坏情况界一直是一个开放问题。

现有理论文献也分析了其他类型的有限相联度缓存，即伴随缓存 [16, 39, 17] 和两级缓存 [43]。然而，由于这些分析是将分页开销与所研究的有限相联缓存的最优算法进行比较，因此并未提供相对于全相联缓存的分页开销信息，也就无法帮助缓存设计者选择合适的相联度，以权衡其优势与分页开销。

1.1 本文

本文解决了一个尚未解决的问题：分析组相联缓存相较于全相联缓存的分页开销，重点研究组相联最近最少使用（LRU）缓存。在这类缓存中，当某个项被取入但指定的组已满时，该组中最近访问时间最早的项会被逐出。我们假设索引函数 h 是一个哈希函数，它以均匀且相互独立的随机方式选择组。

我们的主要结果揭示了组相联缓存中一个显著的阈值现象。具体来说，如果大小为 k 的缓存具有 $\alpha = o(\log k)$ 的相联度，那么其分页开销会很大，即使与大小仅为 k 的一小部分的全相联缓存相比也是如此；而如果 $\alpha = \omega(\log k)$，那么其分页开销与大小大致相同的全相联缓存相近 k。因此，一方面，缓存设计者应确保相联度至少达到对数级别；另一方面，他们也应注意到，超过对数级别后继续提高相联度，收益会逐渐递减。

长请求序列的竞争分析 令 k 为缓存大小。我们证明，如果 $\alpha = \omega(\log k)$，那么 α 路组相联 LRU 以 $1 - 1/\mathrm{poly}(k)$ 的概率，在使用 $(1 + \Theta(\sqrt{\log(k)/\alpha}))$ 资源增强的情况下，对任意长度至多为 $\mathrm{poly}(k)$ 的请求序列，都能与全相联 LRU 达到 1-竞争。[1] （竞争性和资源增强的正式定义见第3节。）

事实上，这一分析源于一个更一般的定理，该定理揭示了所有相关参数之间的权衡，即组大小、资源增强因子、成功概率以及最长支持请求序列的长度。此外，该定理适用于一类我们称为稳定算法的分页算法。粗略地说，当稳定算法运行在小缓存上，且只获得部分请求序列时，其分页决策与同一算法运行在大缓存上、并获得完整请求序列时的分页决策近似。除了

[1]在本文中，只要适当地选择隐藏常数，$\mathrm{poly}(k)$ 就表示关于 k 的 *any* 多项式。

LRU 之外，这一类还包括最不经常使用（LFU），以及 LRU 的一种称为 LRU-K [41] 的推广。本文介绍并研究稳定算法这一类算法，它们本身也可能具有独立的研究价值。

组相联缓存的局限性 请注意，上述竞争分析适用于长度不超过某个特定值（该值由所涉及的参数决定）的所有请求序列，但不适用于任意长的请求序列。问题出在我们的分析上，还是组相联缓存本身存在局限？更一般地说，分析中的参数是否有任何改进空间？也就是说，我们能否在次对数级相联度下，或对超多项式长度的输入序列，或使用少于 $1+\Theta(\sqrt{\log(k)/\alpha})$ 的资源增强，实现 1-竞争（甚至 $O(1)$-竞争）？

我们通过证明一个更一般的权衡关系，表明这些问题的答案基本上都是否定的。我们证明，给定集合大小 α、资源增强因子 r 和竞争比 c，并且 r 不过于大时，存在一个长度不太长（由 α、r 和 c 决定）的对抗性请求序列，使得使用 r 资源增强的全相联 LRU 无法与 α 路组相联 LRU 形成 c 竞争。利用这一权衡关系，我们证明了我们的分析是紧的：

- 任意长度：组相联 LRU 在任意长的请求序列上不具备竞争力。

  相联度： α路组相联LRU 与全相联 LRU相比不具备 c-竞争性，使用 r资源增强时，在长度小至 $O(k^{1.01})$的请求序列上，对于任意$\alpha = o(\log k)$、 $r = O(1)$ 和 $c = O(1)$ 均成立。

- 资源增强： α路组相联 LRU使用 r资源增强时，对于长度小至 $O(k^{2.01})$的请求序列，不是相对于全相联LRU的 c-竞争算法，其中任意 $\alpha = \Omega(\log k)$、 $r = 1 + o(\sqrt{\log(k)/\alpha})$ 和 $c = O(1)$。

- 超多项式长度：对于长度为 ℓ 的请求序列，无论 $\alpha = \Omega(\log k)$,$r = 1 + \Theta(\sqrt{\log(k)/\alpha})$、 $c = \text{poly}(k)$ 还是超多项式 ℓ， α 路组相联 LRU 都无法在采用 r 资源增强时达到与全相联 LRU 的 c 竞争性。

事实上，这些否定结果不仅适用于 LRU，也适用于所有保守型分页算法，包括 FIFO、LFU 和时钟算法。

任意长度请求序列的竞争分析组相联缓存面对足够长的输入序列时会失效，因为对手（即使对手并不知道索引函数的随机选择）不需要花太长时间，就能构造出一个恶意项集合，使这些项不均匀地分布到各组中。这种集合会导致某些组超额订载，进而因相联度限制而造成不必要的缓存未命中；这类未命中称为冲突未命中 [29]。反复请求这些恶意集合中的项，会使组相联缓存中的未命中次数不断增加；但重要的是，全相联缓存不会出现这种情况（只要该集合小于缓存容量），因此两者之间的性能差距可能任意大。

为规避这一问题，我们采用重新散列：我们赋予组相联缓存刷新所有项并绘制新的哈希函数的能力。我们证明，在任意长度的请求序列上，每发生poly(k)次缓存未命中就进行一次重新散列，可以使组相联 LRU 几乎达到与全相联LRU 相竞争的 1 倍竞争比。具体而言，采用重新散列的 $\alpha = \omega(\log k)$路组相联 LRU 以 $1+1/\text{poly}(k)$ 的竞争比与全相联 LRU 相竞争，其概率为 $1-1/\text{poly}(k)$,

使用 $(1+\Theta(\sqrt{\log(k)/\alpha}))$-资源增强，在任意长度的请求序列上。此外，我们证明重新散列可以逐步完成，并达到与停止世界式重新散列相同的性能保证。

1.2 组织结构

本文的组织结构如下。第2节介绍相关工作。第3节回顾分页算法的基本事实，并介绍本文后续部分将用到的术语和概率工具。第4节对处理较长（但并非任意长）请求序列的组相联缓存进行竞争分析。第5节确定组相联缓存的性能极限。第6节为组相联 LRU 增加重新哈希的能力，并分析其在任意长请求序列上的表现。最后，第7节深入研究稳定算法。

2 相关工作

分析分页开销的首要算法技术是竞争分析 [12]。关于分页算法竞争分析的研究由来已久，其中大多聚焦于全相联算法 [53, 24, 15, 21, 22, 61, 62, 5, 4, 19, 20, 6]。

在低相联度场景（有时称为受限缓存）中，现有研究主要关注两类缓存，即两级缓存 [43] 和伴随缓存 [16, 39, 17]，它们是组相联缓存的推广。两级缓存本质上是组相联缓存，其中的项可以映射到多个组，并且可以在任意组之间移动，代价仅为一次缓存未命中成本的一部分。这意味着，有时将某组中的一项移到另一组，为该组腾出空间，可能比将其驱逐更合适。伴随缓存由组相联缓存（主缓存）和一个通常更小的全相联缓存（伴随缓存）组合而成。[2] 在所有情况下，目标都是设计在线算法，使其相对于针对特定缓存组织的最优离线算法具有竞争力。由此得到的算法未必实用——它们较为复杂，而且可能无法满足硬件缓存极为严格的延迟要求。

近期的工作研究了受组相联缓存放置规则约束、但不受其淘汰规则约束的分页算法——页可以随时从任意组中被驱逐。其理念是让组相联缓存镜像一个在旁边进行模拟的全相联分页算法所作出的分页决策。每当全相联算法驱逐一个页时，组相联缓存便驱逐同一个页，即使该受害页所在的组尚未满。关键在于，相对于该模拟过程，组相联缓存采用了资源增强，因此某个组变满的可能性很低。利用这一技术，Bender 等人 [11]证明了每个在线全相联算法都可以适用于组相联缓存的相联度限制。其缺点是，模拟全相联算法的计算成本很高。在一个截然不同的背景下，Frigo 等人 [26, 引理 16] 针对 LRU 证明了类似的模拟结果，不过使用了更大的资源增强。

随着相联度降低，组相联与全相联之间的性能差距会扩大。在极端情况下，对于 1 路组相联缓存（也称为直接映射缓存），Agrawal 等人 [7] 证明，其分页开销在期望意义下与最差离线全相联算法的分页开销相差一个线性因子，这一性质并不理想。

[2]在计算机体系结构文献中，伴随缓存被称为 *victim caches* [31].

在竞争分析成为分析在线算法的事实标准之前，概率模型是主要工具。对于存储系统的分析，最常用的是独立引用模型（IRM） [18, 34, 47], ，在该模型中，系统中的每个项都以某个固定概率独立地被访问。使用 IRM，Smith [54, 55] 在假设访问分布称为线性分页模型的情况下，比较了全相联LRU 和组相联 LRU中缓存未命中的概率。他得出结论：随着组的大小增长，这两个概率渐近相等。当然，由于 IRM是输入的分布模型，IRM分析并不适用于对抗性工作负载，其结论在很大程度上取决于访问分布的选择。

其他研究方向包括分析特定工作负载下组相联缓存的分页开销 [50, 25, 27], ，以及在组相联缓存中模拟 [3] 最初为全相联缓存设计的外部存储算法 [52]。

3 预备知识

分页算法 分页算法 $\mathcal{A}$ 由其所操作的缓存大小参数化。给定一个正整数 k，我们将使用大小为 k 的缓存的算法记为 $\mathcal{A}_k$ $\mathcal{A}$ 。令 $\mathcal{U}$ 为项的全集。令 $\sigma \in \mathcal{U}^*$ 为项请求序列。 $\mathcal{A}_k$ 在 σ上的成本，记为$C(\mathcal{A}_k, \sigma)$，定义为 $\mathcal{A}_k$ 在整个 σ过程中产生的缓存未命中次数。

我们将 σ_i 称为 i在 σ中的第...项。对于任意 $x \in \mathcal{U}$，我们将 σx 表示为将 x 追加到 σ后得到的请求序列。对于任意 $X \subseteq \mathcal{U}$，我们将 $\sigma[X]$ 写作从 σ 中移除每个并非针对 X中某个项的请求后得到的子序列。记号 $x \in \sigma$ 表示项 x 存在于请求序列 σ中。

我们用 $\mathcal{A}_k(\sigma)$ 表示 $\mathcal{A}_k$ 处理完请求序列 σ后缓存的项的集合。当输入序列 σ 的含义根据上下文明确时，我们用 $\mathcal{A}_k(i)$ 表示刚处理完 σ_i后由 $\mathcal{A}_k$ 缓存的项的集合。记 $\mathsf{Out}(\mathcal{A}_k, \sigma, x) := \mathcal{A}_k(\sigma) \setminus \mathcal{A}_k(\sigma x)$ 为在处理完序列 σ 后， $\mathcal{A}_k$ 响应对 x 的访问而驱逐的项的集合。

竞争分析 我们使用竞争分析 [53]来比较分页算法。如果对于每个满足 $|\sigma| \leq \ell$的请求序列 σ，分页算法 $\mathcal{A}$ 以概率 p 、使用 r资源增强时，相对于分页算法 $\mathcal{B}$ 是 c-竞争的，并且请求序列长度为 ℓ，则称其为.....

$$\Pr\left[C(\mathcal{A}_k, \sigma) \leq c \cdot C(\mathcal{B}_{k'}, \sigma) + O(1)\right] \geq p,$$

其中 $k = rk'$。这里的概率取自算法的随机性。在本文中，随机性通常来自索引哈希函数。如果 $\mathcal{A}$ 以概率 $1/2$ 、使用 r资源增强时，相对于 $\mathcal{B}$ 并非 c-竞争的，且请求序列长度为 ℓ，则我们称 $\mathcal{A}$ 在请求序列长度为 ℓ 时使用 r资源增强、相对于 $\mathcal{B}$ 并非c-竞争的。我们始终假设输入 σ 与算法所使用的随机比特无关，也就是说， σ 由一个无知对手生成。本文中的所有分页算法都是在线的，即每个请求只有在算法处理完前一个请求后才会到达。

分页算法的类别 在著名的确定性分页算法中，有最近最少使用（LRU）、最不经常使用（LFU）、先进先出（FIFO）和时钟算法。LRU 的一个广为人知的推广算法称为 LRU-K [42, 41, 14]，它总是淘汰最近使用次数排名第 K 的项。

访问是最久远的。特别地，常规 LRU 就是 LRU-1。[3] 分页算法是惰性的 [37]：只有在会导致缓存未命中时，才从慢速内存中取出一个项；在发生缓存未命中时至多驱逐一个项，并且只有在缓存已满时才驱逐项。LRU-K、FIFO、LFU 和时钟算法都属于这一类。相比之下，满时刷新 [32] 是一种非惰性算法：当取出一个项但每个槽位都已被占用时，它会刷新整个缓存。如果一个分页算法在输入的任意连续子序列上，且该子序列至多包含 k个不同项时，产生的缓存未命中次数不超过 k，则称其为保守型 [60]。LRU、FIFO、LFU 和时钟算法都是保守型；满时刷新并非保守型。

集中界 我们将使用以下标准的 Chernoff 界变体（参见例如 [23]）。

定理 1. 设 $X_1, \ldots, X_n$ 是负相关的 0/1 随机变量。设 $X := \sum_i X_i$。对于每个 $\varepsilon \in (0,1)$,

$$\Pr\left[X \geq (1+\varepsilon)\,\mathbb{E}\left[X\right]\right] \leq \exp(-\varepsilon^2\,\mathbb{E}\left[X\right]/3),$$

and

$$\Pr\left[X \leq (1-\varepsilon)\,\mathbb{E}\left[X\right]\right] \leq \exp(-\varepsilon^2\,\mathbb{E}\left[X\right]/2).$$

我们还将使用一个众所周知的事实：Chernoff 界是紧的。

定理 2（[40]）。设 $X_1, \ldots, X_n$ 是相互独立的 0/1 随机变量，且对于每个 i，$\Pr[X_i = 1] = p \leq 1/2$。设 $X := \sum_i X_i$。对于每个 $\varepsilon \in [0, 1/p - 2]$,

$$\Pr\left[X \geq (1+\varepsilon)\,\mathbb{E}\left[X\right]\right] \geq \frac{1}{4}\exp(-2\varepsilon^2\,\mathbb{E}\left[X\right]).$$

4 组相联缓存在线性序列上具有竞争性

我们将组相联缓存和全相联缓存建模为分页算法。对于任意分页算法 $\mathcal{A}$ 以及任意可整除 k 的 $\alpha = \alpha(k)$，我们将 α 路组相联 $\mathcal{A}$ 定义为如下分页算法。

> 算法：缓存大小为 k 的 α 路组相联 $\mathcal{A}$
>
> - 选择一个完全随机的哈希函数 h：$\mathcal{U} \to [k/\alpha]$。
> - 将缓存划分为 k/α 组，每组包含 α 个槽位，并在每组上分别运行一个 $\mathcal{A}_\alpha$ 实例。
> - 当请求某个项 x 时，在组 $h(x)$ 上执行该请求。

我们将 $\langle\mathcal{A}\rangle$ 称为 α 路组相联 $\mathcal{A}$ 算法。由于任何时候只考虑单个 α，因此我们在记号中省略 α。因为 $\alpha = k$ 的情况表示全相联缓存，所以我们始终假设 $\alpha < k$。为避免混淆缓存中的组与数学意义上的集合，我们将组相联缓存中的组称为桶。

本节旨在说明，对于任意属于我们称为稳定算法（稍后将在第 4.2 节中定义）这一类的惰性算法 $\mathcal{A}$，$\langle\mathcal{A}\rangle$ 造成的缓存未命中次数至多与 $\mathcal{A}$ 的大致相同。特别是，这一类算法包括

[3] LRU-2 通常优于 LRU [41]，因为它对孤立访问不那么敏感：只有当某个项最近至少被访问过两次时，它才会将该项视为重要项。

LFU 和 LRU-K 算法。本节结束时，我们将证明以下定理。

定理 3. 设 $\mathcal{A}$ 是一种惰性稳定分页算法。设 $\delta = \delta(k)$ 满足 $\delta \geq \sqrt{12\ln(k/\alpha)/\alpha}$，且对于所有足够大的 k，都有 $\delta \leq 1/2$。那么，在长度为 $O(\exp(\delta^2\alpha/24))$ 的请求序列上，使用 $(1-\delta)^{-1}$ 倍资源增强时，α 路组相联 $\mathcal{A}$ 以 $\mathcal{A}$ 的概率至少为 $1-\exp(-\delta^2\alpha/24)$ 达到 1-竞争。

粗略地说，定理 3 表明，只要 δ 不太小，且请求序列不过长，$\langle \mathcal{A} \rangle_k$ 的分页开销就大致与 $\mathcal{A}_{k'}$ 的相同，其中 $k' = (1-\delta)k$。可调参数 δ 是组相联缓存与其全相联对应缓存之间的容量差距，它控制着资源增强因子、成功概率和请求序列长度之间的权衡。一方面，为了最小化资源增强因子（$1-\delta$）$^{-1}$，我们希望 δ 尽可能小。另一方面，为了最大化成功概率和序列长度（它们都是 δ 的递增函数），我们希望 δ 尽可能大。在这条权衡曲线中，也许最有趣的区域是 δ 的低端；在该区域，资源增强被渐近地最小化，同时仍能提供多项式成功概率，并支持多项式长度的序列，正如下述结果所示。

命题 1. 设 $\mathcal{A}$ 是一种惰性稳定分页算法。假设 $\alpha = \omega(\log k)$。那么，在长度为 $\mathrm{poly}(k)$ 的请求序列上，α 路组相联 $\mathcal{A}$ 以高概率 [4]，使用 $(1+\Theta(\sqrt{\log(k)/\alpha}))$ 资源增强时，可达到相对于 $\mathcal{A}$ 的 1-竞争比。

证明。对于任意常数 $c > 0$，令 $\delta := \sqrt{24c\ln(k)/\alpha}$。由于 $\alpha = \omega(\log k)$，对于所有足够大的 k，δ 都处于适当范围内。此选择得到成功概率 $1-k^{-c}$ 和序列长度 $O(k^c)$。资源增强因子满足

$$1 + \delta \leq (1-\delta)^{-1} \leq 1 + 2\delta,$$

其中第二个 i 不等式是因为 $\delta \leq 1/2$。因此，$(1-\delta)^{-1} = 1+\Theta(\delta) = 1+\Theta(\sqrt{\log(k)/\alpha})$。 □

4.1 证明概要

在深入探讨定理 3的细节之前，我们先概述其在$\mathcal{A}$ =LRU这一特定情形下的证明。若某个项按照LRU顺序比全相联缓存中的每个项都更旧，则称该项为过时项。直观地说，驱逐非过时项是不良的，因为对它们的访问会在全相联缓存中导致命中，却会在组相联缓存中导致未命中。我们将驱逐非过时项称为不良驱逐。证明分为两步：首先，我们证明，如果不良驱逐很少，那么组相联缓存的成本至多约等于全相联缓存的成本。其次，我们证明一次驱逐为不良驱逐的概率很低；这意味着不良驱逐的总数很少。

我们通过以下一系列观察来给出发生糟糕驱逐的概率上界：如果项 x 被驱逐，且 x 不是陈旧的，那么 x 的箱子中的所有页也都不是陈旧的。这意味着，x 以及 x 的箱子中的所有其他项都存在于全相联缓存中。因此，全相联缓存中的项集具有一种不理想的性质：将它们哈希到组相联缓存中时，会导致某个箱子超载。关键在于，这个项集是确定性的，因此通过对箱子负载进行类似球与箱子的分析，可以证明这种情况不太可能发生。

[4] *With high probability*(以高概率)表示概率为 $1 - 1/\mathrm{poly}(k)$。

4.2 稳定的分页算法

请注意，在证明概要中，我们用到了 LRU 的以下性质：如果组相联缓存中的某个箱子驱逐了一个存在于全相联缓存中的项 x，那么 x 的箱子中的所有项也都存在于全相联缓存中。稳定算法对这一性质进行了形式化。

如果对于任意 $\tau \in \mathcal{U}^*$、 $X \subseteq \mathcal{U}$、 $z \in X$ 和正整数 $a > b$，分页算法 $\mathcal{A}$ 都满足以下条件，我们就称其为稳定的：

$$\mathsf{Out}(\mathcal{A}_b, \tau[X], z) \cap \mathcal{A}_a(\tau z) \neq \emptyset \text{ implies } \mathcal{A}_b(\tau[X]z) \subseteq \mathcal{A}_a(\tau z). \tag{1}$$

换句话说，如果一个实例 $\mathcal{A}$ 接收到请求序列 τz，而一个较小的实例接收到其子序列 $\tau[X]z$，并且在访问最后一项 z 后，较小的缓存淘汰了一项，而该项也存在于较大的缓存中，那么较小的缓存就完全包含在较大的缓存中。

我们也可以将稳定性视为一种无缝机制，用于纠正由分页算法 $\mathcal{A}$ 的实例管理的任意两个缓存之间的差异。粗略地说，公式 (1) 的逆否命题表明， $\mathcal{A}_b$ 总是优先淘汰 $\mathcal{A}_a$ 中不存在的项。因此，直观而言，稳定算法能够有效管理组相联缓存中的桶，因为它们试图近似一个较大的全相联缓存的状态。

我们将关于稳定算法的全面讨论推迟到第 7 节，在该节中我们将研究这类算法的适用范围，并深入分析区分稳定的分页算法与不稳定的分页算法的特征。以下事实是第 7 节的推论。

引理 1。LRU-K 和 LFU 是稳定的。FIFO 和时钟算法不是稳定的。

4.3 辅助结果

令 $\mathcal{X}$ 和 $\mathcal{Y}$ 为任意两个分页算法，其缓存大小任意。我们称项 x 是项 $\mathcal{X}$ 相对于 $\mathcal{Y}$ 在时间 i 发生的不良驱逐（当上下文中含义明确时，我们省略 $\mathcal{X}$、 $\mathcal{Y}$ 或 i），如果 x 因访问 σ_i 和 $x \in \mathcal{Y}(i)$ 而被 $\mathcal{X}$ 驱逐，即 $x \in \mathsf{Out}(\mathcal{X}, \sigma_1 \dots \sigma_{i-1}, \sigma_i) \cap \mathcal{Y}(i)$。

引理2。设 $\mathcal{X}$ 和 $\mathcal{Y}$ 是具有任意缓存大小的任意两个分页算法，且它们仅在请求某个页面时才获取该页面。那么，

$$C(\mathcal{X}, \sigma) \leq C(\mathcal{Y}, \sigma) + B,$$

其中， B 是 $\mathcal{X}$ 相对于 $\mathcal{Y}$ 在整个 σ期间的不良驱逐次数。

证明。注意到 $C(\mathcal{X}, \sigma) - C(\mathcal{Y}, \sigma) = M - H$，其中 H 是 $\mathcal{X}$ 中的缓存命中但在 $\mathcal{Y}$ 中未命中的数量，而 M 是 $\mathcal{X}$ 中的未命中但在 $\mathcal{Y}$ 中命中的数量。我们将后者称为坏未命中。因此，只需证明 $M \leq B$。

假设 σ_i 是一次不良未命中。注意， σ_i 必定在时间 i之前已被访问过，否则它在时间 i就不会出现在 $\mathcal{Y}$ 中，因为项只有在被请求时才会被取回。因此， σ_i 必定在过去被 $\mathcal{X}$ 驱逐过。令 $j < i$ 为 σ_i 最近一次被 $\mathcal{X}$驱逐的时间。只需证明 σ_i 是一次不良驱逐（在时间 j相对于 $\mathcal{Y}$ 驱逐 $\mathcal{X}$）；这便能建立从不良未命中到不良驱逐的单射，因为同一项不可能在未被驱逐的情况下发生两次未命中。因此，要完成证明，我们需要证明$\sigma_i \in \mathcal{Y}(j)$。

在时间区间 (j, i) 内，项 σ_i 未被访问，否则 j 就不会是 σ_i 在 $\mathcal{X}$ 中最近一次被驱逐的时间。由于 σ_i 在时间 i 于 $\mathcal{Y}$ 中命中，项 σ_i 在 (j, i) 期间未被访问，而项只有在收到请求时才会被调入，因此必有 $\sigma_i \in \mathcal{Y}(j)$。因此，在时间 j，相对于 $\mathcal{Y}$ 而言， σ_i 是对 $\mathcal{X}$ 的一次不良驱逐。□

特别地，引理2对惰性算法成立。（稍后在第6节中，我们将把这一结果应用于一般情形，即适用于每个时间步会执行多次驱逐的分页算法。）

接下来，我们证明一个适用于我们特定情形的标准球与箱子界。

引理 3. 设 $\delta = \delta(k)$ 满足 $\delta \geq \sqrt{12\ln(k/\alpha)/\alpha}$，且对所有足够大的 k 都有 $\delta \leq 1/2$。假设将 $k' = (1-\delta)k$ 个球随机且独立地投入 k/α 个箱子中。那么，对所有足够大的 k，所有箱子中的最大负载大于 α 的概率至多为 $\exp(-\delta^2\alpha/12)$。

证明. 令 $m := k' = (1-\delta)k$ 为球的数量，令 $n := k/\alpha$ 为箱子的数量。令 $h := m/n = (1-\delta)k/(k/\alpha) = (1-\delta)\alpha$ 为每个箱子的平均负载。令 L_i 为箱子 i 的负载。这是 m 个相互独立的二元随机变量之和，每个变量表示一个球是否落入箱子 i。根据定理 1 中的 Chernoff 界，

$$\begin{aligned}
\Pr\left[L_i \geq (1+\delta)\,\mathbb{E}\left[L_i\right]\right] &\leq \exp\left(-\frac{\delta^2\,\mathbb{E}\left[L_i\right]}{3}\right) \\
&= \exp\left(-\frac{\delta^2(1-\delta)\alpha}{3}\right) && \text{（因为 } \mathbb{E}\left[L_i\right] = h = (1-\delta)\alpha\text{）} \\
&\leq \exp\left(-\frac{\delta^2\alpha}{6}\right). && (\text{as } 1-\delta \geq 1/2)
\end{aligned}$$

注意到

$$(1+\delta)\,\mathbb{E}\left[L_i\right] = (1+\delta)(1-\delta)\alpha = (1-\delta^2)\alpha < \alpha.$$

因此，箱子 i 的负载大于 α 的概率至多为 $\exp(\delta^2\alpha/6)$。对所有 n 个箱子应用并集界，可得所有箱子中的最大负载大于 α 的概率为

$$\begin{aligned}
n\exp(-\delta^2\alpha/6) &= \frac{k}{\alpha}\exp(-\delta^2\alpha/6) && (\text{by definition of } n) \\
&= \exp\left(-\frac{\delta^2\alpha}{6} + \ln\left(\frac{k}{\alpha}\right)\right) \\
&\leq \exp\left(-\frac{\delta^2\alpha}{12}\right). && \text{（因为 } \delta^2\alpha/6 \geq 2\ln(k/\alpha)\text{，根据假设）}
\end{aligned}$$

至此，证明完成。 □

4.4 定理 3 的证明

对于每个时间步 i，令 B_i 为指示随机变量，当 σ_i 导致相对于 $\mathcal{A}_{k'}$ 在 $\langle\mathcal{A}\rangle_k$ 中发生不良驱逐时，其值为 1。

权利要求1. 我们有 $\Pr[B_i = 1] \leq \exp(-\delta^2\alpha/12)$。

证明。假设项 x 被 $\langle\mathcal{A}\rangle_k$ 驱逐，以响应 σ_i。那么，σ_i 必须与 x 哈希到同一个桶中。假设 x 是不良驱逐。令 $\tau := \sigma_1 \ldots \sigma_{i-1}$。那么，$x \in \mathsf{Out}(\langle\mathcal{A}\rangle_k, \tau, \sigma_i) \cap \mathcal{A}_{k'}(i)$。令 $X := \{y \in \mathcal{U} \mid h(y) = h(x)\}$ 为哈希到 x 的桶中的项组。那么，$x \in \mathsf{Out}(\mathcal{A}_\alpha, \tau[X], \sigma_i) \cap \mathcal{A}_{k'}(i)$，因为当且仅当 x 从其桶中被驱逐时，x 才会从组相联缓存中被驱逐。由于 $\mathcal{A}$ 是稳定的，这意味着 $\mathcal{A}_\alpha(\tau[X]\sigma_i) \subseteq \mathcal{A}_{k'}(\tau\sigma_i)$，也就是说，$x$ 桶中的所有项也都在 $\mathcal{A}_{k'}(i)$ 中。因此，如果 $S_i := \mathcal{A}_{k'}(i)$，我们得出结论

$$\Pr\left[B_i = 1\right] \leq \Pr[\text{There are } \alpha+1 \text{ items in } S_i \text{ that hash to the same bucket}].$$

为了对此概率给出上界，我们将该事件建模为一个球与箱子的游戏，其中每个球代表一个项，每个箱子代表一个桶。关键的是，组 S_i 与 $\langle \mathcal{A} \rangle_k$ 的随机性无关，因此球的放置相互独立且均匀随机。球的数量至多为 $|S_i| \le k'$，箱子的数量至多为 k/α，因此根据引理 3，此概率至多为 $\exp(-\delta^2\alpha/12)$。 □

令 $B := \sum_{i=1}^{|\sigma|} B_i$ 为不良驱逐的数量；由于 $\mathcal{A}$ 是惰性的，因此每一步至多发生 1 次不良驱逐。根据权利要求1和期望的线性性，$\mathbb{E}[B] \le |\sigma| \exp(-\delta^2\alpha/12)$。根据马尔可夫不等式，$\Pr\left[B \ge |\sigma| \exp(-\delta^2\alpha/24)\right] \le \exp(-\delta^2\alpha/24)$。

由于 $\mathcal{A}$ 是惰性的，我们可以应用引理2，得到

$$C(\langle \mathcal{A} \rangle_k, \sigma) \le C(\mathcal{A}_{k'}, \sigma) + |\sigma| \exp(-\delta^2\alpha/24),$$

其概率为 $1 - \exp(-\delta^2\alpha/24)$。当 $|\sigma| = O(\exp(\delta^2\alpha/24))$时，这意味着 $\langle \mathcal{A} \rangle_k$ 相对于 $\mathcal{A}_{k'}$是 1-竞争的。证明完毕。

5 组相联缓存对于任意长度的请求序列都不具备竞争性

定理 3 的主要局限在于，它不适用于任意长度的请求序列。也就是说，对于 α 和 δ的任意有效选择，组相联缓存的竞争性只能保证到某个特定数量的请求为止。一个自然的问题是，我们能否将这一结果扩展到任意长的输入序列，也许可以允许更大的资源增强因子，或更大的竞争比。

本节的主要结果是：对于所有保守型算法（包括 LRU），这个问题的答案是否定的。

定理 4. 设 $\mathcal{A}$是一个保守型分页算法。假设对于所有足够大的 k，都有 $|\mathcal{U}| \ge 16k \exp(8(1-\delta)^{-1}\delta^2\alpha)$ 且 $(1-\delta)^{-1} \le k/(2\alpha)$。那么，在长度为.....的请求序列上，$\alpha$路组相联 $\mathcal{A}$ 使用$(1-\delta)^{-1}$资源增强时，无法与 c竞争。

$$O(c\alpha k \exp(16(1-\delta)^{-1}\delta^2\alpha)).$$

粗略地说，定理 4 表明，只要资源增强 $r = (1-\delta)^{-1}$ 不太大，且全集 $\mathcal{U}$ 足够大，那么组相联缓存能够处理的请求序列长度就存在一个硬性上限；超过这个上限，其性能就会下降。

推论 1. 设 $\mathcal{A}$ 是一个保守型分页算法。假设 $\mathcal{U}$、α 和 $r = (1-\delta)^{-1}$ 满足定理 4 的条件。那么，对于任意 $c \ge 1$，在任意长的请求序列上，α路组相联 $\mathcal{A}$ 使用 r资源增强时，无法与 c竞争。

例如，只要全集足够大，该推论就适用于所有 $\alpha \le k/8$ 和 $r \le 4$。我们还可以进一步利用定理 4证明，在 α、r 和 c的一些常见范围内，不可能实现竞争性。作为比较，请回想一下，根据命题 1，对于 $\alpha = \omega(\log k)$ 和 $r = 1 + \Theta(\sqrt{\log(k)/\alpha})$，在多项式长度的请求序列上可以实现 1-竞争性。

命题 2. 设 $\mathcal{A}$ 是一个保守型分页算法。为简单起见，假设 $\mathcal{U}$ 是无限的。在以下情况下，α路组相联 $\mathcal{A}$ 对于使用 r资源增强的 $\mathcal{A}$，在长度为 $O(\alpha k^{1+o(1)}) = O(k^{2.01})$ 的请求序列上不是 c-竞争的：

1. 对于 $\alpha = \Omega(\log k)$，使得 $\alpha \le k/3$, $r = 1 + o(\sqrt{\log(k)/\alpha})$ 和 $c = O(1)$；

2. 对于 $\alpha = o(\log k)$、 $r = O(1)$ 和 $c = O(1)$;

3. 对于 $\alpha = 1$、 $r = o(\log k)$ 和 $c = O(1)$。

证明。三种情况的证明类似，因此我们只证明第一种情况。令 $\delta := 1 - 1/r$。注意到$(1-\delta)^{-1} = r$。由于 $\alpha = \Omega(\log k)$，我们有 $r = 1 + o(1)$，因此， $r = 1 + o(1) \le 3/2 \le k/(2\alpha)$，所以我们满足定理4 的条件。因此， α路组相联的 $\mathcal{A}$在使用 r资源增强时，对于长度为的请求序列，不具有相对于 $\mathcal{A}$ 的 c竞争性

$$\begin{aligned} O(c\alpha k \exp(16r\delta^2\alpha)) &= O(\alpha k \exp(16r\delta^2\alpha)) && (\text{as } c = O(1)) \\ &= O(\alpha k \exp(o(\log k))) && (\text{as } \delta = o(\sqrt{\log(k)/\alpha}) \text{ and } r = 1 + o(1)) \\ &= O(\alpha k^{1+o(1)}). \end{aligned}$$

□

有趣的是，命题2 的第3部分意味着，直接映射缓存（即具有 $\alpha = 1$的组相联缓存）即使在规模小至 $O(k^{1+o(1)})$的请求序列上，采用次对数级资源增强，也无法与使用资源增强的全相联缓存保持常数竞争比。这印证了直接映射缓存性能欠佳这一普遍看法。

命题 2 的第 1 部分和第 2 部分意味着命题 1 是最优的：即使允许竞争比为 $O(1)$ 且成功概率为次多项式（但至少为 $1/2$），集合大小和资源增强因子也无法进一步缩小。下面的命题表明，事实上，即使竞争比是多项式，命题 1 中请求序列的多项式长度也无法改进。

命题 3. 假设 $\mathcal{U}$ 是无限的。对于满足 $\alpha \le k/3$、 $c = \text{poly}(k)$ 且 $r = 1 + \Theta(\sqrt{\log(k)/\alpha})$ 的 $\alpha = \Omega(\log k)$，在超多项式长度的请求序列上，使用 r-资源增强的 α 路组相联 $\mathcal{A}$，对于 $\mathcal{A}$ 而言不是 c-竞争的。

证明与命题 2 的证明类似，因此略去。

5.1 一个辅助结果

考虑一个球与箱子的游戏，其中 m 个球被随机且独立地投入 n 个箱子。令 $h := m/n$。如果游戏结束时某个箱子中至少有 $h + a$ 个球，则称该箱子为 a-饱和的。令 $f(n, m, \varepsilon) := n\exp(-2\varepsilon^2 h)$。

引理 4. 假设将 m 个球扔进 $n \ge 2$ 个箱子。令 $h := m/n$。对于每个 $\varepsilon \in [0, n-2]$， εh-饱和箱子的数量大于 $f(n, m, \varepsilon)/8$，其概率至少为 $1 - \exp(-f(n, m, \varepsilon)/32$。

证明。令 X_i 为指示随机变量，当且仅当箱子 i 是 εh-饱和时取值为 1。令 L_i 为箱子 i 的负载。那么，$\Pr[X_i = 1] = \Pr[L_i \ge h + \varepsilon h]$。随机变量 L_i 是 m 个独立同分布的 0/1 随机变量之和，每个变量表示一个球是否落入箱子 i。特别地，$\mathbb{E}[L_i] = h$。由于每个球落入箱子 i 的概率为 $1/n \le 1/2$，且 $\varepsilon \in [0, n-2]$，根据定理 2 中的反向 Chernoff 界，我们有

$$\Pr[L_i \ge h + \varepsilon h] \ge \frac{1}{4}\exp(-2\varepsilon^2 h).$$

令 $X := \sum_i X_i$ 表示已被 εh 填满的箱子数量。根据期望的线性性， $\mathbb{E}[X] = \sum_i \mathbb{E}[X_i] \ge f(n, m, \varepsilon)/4$。

利用关于负相关随机变量的标准结果 [58], , 可以证明随机变量 $X_1, \ldots, X_n$ 负相关。因此，根据 Chernoff 界，

$$\begin{aligned}\Pr\left[X \leq f(n,m,\varepsilon)/8\right] &\leq \Pr\left[X \leq \mathbb{E}\left[X\right]/2\right] \\ &\leq \exp(-\mathbb{E}\left[X\right]/8) \leq \exp(-f(n,m,\varepsilon)/32).\end{aligned}$$

□

5.2 定理 4 的证明

考虑以下对手生成的请求序列。

对手：定理 4 的请求序列

1. 令 $s := 16\exp(8(1-\delta)^{-1}\delta^2\alpha)$，并令 $t := c\alpha s^2$。 2. 选取 $S_1, \ldots, S_s \subseteq \mathcal{U}$ 个互不相交的组，每组包含 $k' = (1-\delta)k$ 个项。 3. 对于 $i = 1, \ldots, s$： (a) 重复 t 次： i. 依次访问 S_i 中的每个项。

请注意，根据假设，由于全集足够大，对手可以执行步骤 2。设 σ 为所生成的请求序列。其长度为

$$|\sigma| = k't < kt = kc\alpha s^2 = O(c\alpha k \exp(8(1-\delta)^{-1}\delta^2\alpha)^2),$$

如要求所示。我们将证明， $C(\langle\mathcal{A}\rangle_k, \sigma) \geq cC(\mathcal{A}_{k'}, \sigma) + \omega(1)$ 的概率严格大于$1/2$，这意味着该结论成立。

由于 $\mathcal{A}_{k'}$ 是保守型，因此仅在首次访问 S_i 中的每个页面时未命中。因此，$C(\mathcal{A}_{k'}, \sigma) = k's$。现在我们分析 $\langle\mathcal{A}\rangle_k$ 在 σ上的开销。

断言 2。在步骤 3 的循环的每次迭代中， $\langle\mathcal{A}\rangle_k$ 中至少会发生 $2csk$ 次驱逐，概率至少为 $1 - \exp(-k/(2\alpha s))$。

证明。固定一次第 3-a-i步的执行。我们按通常方式将对 S_i 中各项的访问建模为一个球与箱子游戏：每个球代表一个项，每个箱子代表一个桶。共有 $m = (1-\delta)k$ 个球被投入 $n = k/\alpha$ 个箱子；这些至少是2 个箱子，因为 $\alpha \leq k/2$。令$h := m/n = (1-\delta)\alpha$。令 $\varepsilon := 2\delta/(1-\delta)$。注意，$\varepsilon \leq n - 2$，因为这等价于假设 $(1-\delta)^{-1} \leq k/(2\alpha)$。我们有

$$h + \varepsilon h = (1+\varepsilon)h = (1+\varepsilon)(1-\delta)\alpha > \alpha.$$

在最后一个不等式中，我们使用了对 ε 的选择。因此，如果一个箱子是 εh 饱和的，那么无论步骤开始时 $\langle\mathcal{A}\rangle_k$ 的状态如何，与之关联的桶都至少发生了一次驱逐。设 X 为 εh 饱和箱子的数量。因此，此步骤中 $\langle\mathcal{A}\rangle_k$ 的驱逐次数至少为 X。

关键的是，对于步骤 3 的循环（外层循环）的某次固定迭代，步骤 3-a 的循环（内层循环）每次迭代中，球箱游戏的随机箱子选择都相同。因此，外层循环某次固定迭代期间的驱逐次数至少为 tX。

请注意，

$$\begin{aligned} f(n,m,\varepsilon) &= f\left(\frac{k}{\alpha},(1-\delta)k,\frac{2\delta}{1-\delta}\right) \\ &= \frac{k}{\alpha}\exp\left(-2\left(\frac{2\delta}{1-\delta}\right)^2(1-\delta)\alpha\right) \\ &= \frac{k}{\alpha}\exp\left(-\frac{8\delta^2\alpha}{1-\delta}\right) = \frac{16k}{\alpha s}. \end{aligned}$$

根据引理 4，$X \geq f(n,m,\varepsilon)/8 = 2k/(\alpha s)$ 的概率至少为 $1-\exp(-f(n,m,\varepsilon)/32) = 1-\exp(-k/(2\alpha s))$。证明最后指出，$t(2k/(\alpha s)) = 2csk$。 □

因此，第 3 步循环每次迭代的开销至少为

$$2csk \geq csk' + \Omega(k) = c\,C(\mathcal{A}_{k'},\sigma) + \Omega(k),$$

其概率为 $1-\exp(-k/(2\alpha s))$。由于各个 S_i 互不相交，且哈希函数完全独立，因此该事件在第 3 步循环的每次迭代中独立发生。因此，第 3 步中至少有一次迭代的开销不小于 $c\,C(\mathcal{A}_{k'},\sigma) + \Omega(k)$ 的概率至少为

$$1-\exp(-k/(2\alpha s))^s = 1-\exp(-k/(2\alpha)) \geq 1-e^{-1} \approx 0.63.$$

因此，$C(\langle\mathcal{A}\rangle_k,\sigma) \geq c\,C(\mathcal{A}_{k'},\sigma) + \omega(1)$ 的概率至少为 0.63。

6 从长序列到所有序列

定理 4 的证明表明，如果我们希望 $\langle\mathcal{A}\rangle$ 在所有请求序列上都保持竞争力，就必须使其不受利用重复模式的对手序列影响。一种自然的尝试方法是让算法能够获取新的哈希函数，也就是进行重新散列。这样，每当算法认为当前哈希函数已遭到利用时，就切换到一个新的哈希函数。[5]

我们每发生 $\text{poly}(k)$ 次缓存未命中就重新散列一次。请注意，在固定数量的访问（与缓存未命中次数无关）后定期重新散列会产生不利影响，因为这会给对手提供无穷多次机会，利用分配中的负载不均衡。事实上，对手现在只需固定一组包含 $k' = (1-\delta)k$ 个项的集合，然后依次访问集合中的每个元素，如此无限重复即可。

重新散列的一个缺点是，它可能会增加分页开销，因为这会让桶“冷却下来”。重新散列时，桶必须突然改为缓存另一组项，而它们对这些项过去的访问情况所知有限（甚至一无所知）。当$\mathcal{A}$ 的分页决策很大程度上依赖于完整的访问历史记录（例如 LFU）时，桶实例会在重新散列后受到影响，并且可能永远无法恢复。

不会受到这个问题影响的一种分页算法是 LRU。这是因为 LRU 只依赖于各项最近一次访问的信息，因此在重新散列后，项 x 一旦首次被访问，分配给 x 的新桶就拥有关于 x的最新访问信息。本节余下部分将聚焦于 $\mathcal{A}$ =LRU。

每当抽取一个新的哈希函数时，当前使用旧函数进行哈希的项都必须重新映射。考虑以下简单的方法，我们称之为完全刷新：当重新哈希时，

[5]有趣的是，近期的缓存设计已采用重新散列来防御利用冲突未命中发起的时序攻击 [45, 46]。

触发后，刷新缓存并生成一个新的哈希函数。如果再次访问这些项，它们将被放入新的桶中。

算法：缓存大小为 k 的 α 路组相联 LRU，并采用完全清空策略

- 将缓存大小为 k 的 α 路组相联 LRU 增强如下：

  – 维护缓存未命中次数的计数器。– 令 $d \geq 2$ 为任意常数。每发生 k^d 次未命中，就驱逐缓存中的所有项，抽取一个新的哈希函数，并将计数器重置。

我们将该算法记为 $\langle \text{LRU} \rangle^{\text{FF}}$。

定理 5. 假设 $\alpha = \omega(\log k)$。那么，采用完全清空策略的 α 路组相联 LRU，以 $(1 + 1/\text{poly}(k))$ 的竞争比，在所有请求序列上以高概率与 LRU 相比，使用 $(1 + \Theta(\sqrt{\log(k)/\alpha}))$ 的资源增强。

证明。对于任意常数 $c > d$，令 $\delta := \sqrt{36c \ln(k)/\alpha}$。令 $k' := (1-\delta)k$。我们将比较 $\langle \text{LRU} \rangle_k^{\text{FF}}$ 的分页开销与 $\text{LRU}_{k'}$ 的分页开销。

将 σ 划分为阶段如下。每个阶段是一个时间步的极大序列，在此期间$\text{LRU}_{k'}$ 恰好发生一次缓存未命中。（这唯一的一次未命中发生在该阶段的第一次访问中。）第一个阶段从 σ的开头开始，所有其他阶段都紧接在前一个阶段之后开始。令 π_r 为第 r个阶段。我们进一步将每个 π_r 划分为若干子阶段。每个子阶段要么从 π_r 的开头开始，要么在 $\langle \text{LRU} \rangle_k^{\text{FF}}$中触发重新哈希时开始，并持续到下一次重新哈希之前或 π_r结束之前，以先发生者为准。我们将 π_r中的第 s 个子阶段记为 π_r^s。

令 i_r 为 π_r 开始的时间步，并令 $S_r := \text{LRU}_{k'}(i_r)$。注意，由于 $\text{LRU}_{k'}$ 仅在首次访问 π_r 时发生未命中，因此在该阶段，$\text{LRU}_{k'}$ 缓存的项集合始终为 S_r。

将 $\langle \text{LRU} \rangle_k^{\text{FF}}$ 的不良驱逐分为两类：由于刷新导致的驱逐，以及其余驱逐。我们将后者称为常规不良驱逐。令 F 表示由于刷新导致的不良驱逐次数，令 R 表示常规不良驱逐次数。根据引理2，

$$C(\langle \text{LRU} \rangle_k^{\text{FF}}, \sigma) \leq C(\text{LRU}_{k'}, \sigma) + R + F. \tag{2}$$

由于重新哈希的次数至多为 $1/k^d$所占比例的$\langle LRU \rangle_k$缓存未命中总数，且每次重新哈希都会导致 k 驱逐，

$$F \leq k^{-d+1} C(\langle \text{LRU} \rangle_k^{\text{FF}}, \sigma). \tag{3}$$

结合公式（2）和公式（3），

$$\begin{aligned} C(\langle \text{LRU} \rangle_k^{\text{FF}}, \sigma) &\leq \frac{1}{1 - k^{-d+1}} (C(\text{LRU}_{k'}, \sigma) + R) \\ &\leq (1 + 2k^{-d+1})(C(\text{LRU}_{k'}, \sigma) + R). \end{aligned} \tag{4}$$

断言 3. 我们有 $\Pr[\text{在 } \pi_r^s] \leq k^{-3c}$ 期间存在常规不良驱逐。[6]

[6]此权利要求与权利要求1类似，但有两个关键区别：它涉及重新哈希，并且针对的是整个子阶段（其长度可以任意长），而非某个单一时刻。

证明。假设某个项 x 在 $\langle \text{LRU} \rangle_k$ 中于 π_r^s期间发生了一次常规不良驱逐，设其发生在时间步j。那么， $x \in \text{LRU}_{k'}(j)$。根据 LRU 的定义， x桶中的所有其他项，其访问时间都比 x更近。这意味着这些项也都在 $\text{LRU}_{k'}(j)$ 中。[7] 因此，$\text{LRU}_{k'}(j)$中有 $\alpha+1$ 个项哈希到同一个桶。此外，由于一个阶段内 $\text{LRU}_{k'}$ 中缓存项的集合不会改变，因此 $\text{LRU}_{k'}(j) = S_r$；事实上，x桶中的所有项都在 S_r中。因此，

$$\Pr[\text{期间存在一次常规不良驱逐 } \pi_r^s] \leq \Pr[\text{在 } S_r \text{ 中有 } \alpha+1 \text{ 项哈希到同一存储桶}]. \tag{5}$$

我们使用球和箱子的分析来给出该概率的上界。关键在于， S_r 是一个确定性的组。 δ 的选择满足引理 3 的假设，因此 S_r 导致桶溢出的概率至多为 $\exp(-\delta^2\alpha/12) = k^{-3c}$，正如所需。 □

命题 4。如果在 π_r^s 期间没有发生常规不良驱逐，那么它要么是 π_r 中的第一个子阶段，要么是最后一个子阶段。

证明。我们证明逆否命题。假设 π_r^s 既不是第一个子阶段，也不是最后一个子阶段。那么， π_r^s以一次重新哈希开始，并在下一次重新哈希之前结束。因此，在π_r^s期间， $\langle \text{LRU} \rangle_k^{\text{FF}}$中有 k^d 次缓存未命中。由于在 π_r期间只请求来自 S_r 的项，并且 $\langle \text{LRU} \rangle_k^{\text{FF}}$在$\pi_r^s$期间未命中超过 $k > |S_r|$ 次，因此存在某个项 $x \in S_r$ ，在π_r^s期间至少有一次被 $\langle \text{LRU} \rangle_k^{\text{FF}}$驱逐。 □

令 R_r 表示 π_r 期间的常规不良驱逐次数。

结论 5。我们有 $\mathbb{E}[R_r] \leq O(k^{-2c})$。

证明。由于每个子阶段最多可以有 k^d 次未命中，因此也最多可以有 k^d 次驱逐。特别是，最多可以有 k^d 次常规不良驱逐。因此，

$$\Pr[R_r \geq \ell] \leq \Pr[\text{At least } \lceil \ell/k^d \rceil \text{ subphases in } \pi_r \text{ have regular bad evictions}].$$

如果 $\lceil \ell/k^d \rceil$ 中的子阶段发生常规不良驱逐，那么根据断言 4， π_r^1、 π_r^2、.....、 $\pi_r^{\lceil \ell/k^d \rceil}$要么都发生常规不良驱逐，要么 π_r^2、 π_r^3、.....、 $\pi_r^{\lceil \ell/k^d \rceil+1}$ 都发生常规不良驱逐，因为所有中间子阶段都发生常规不良驱逐。因此，根据并集界，

$$\begin{aligned}
\Pr[R_r \geq \ell] &\leq \Pr\left[\pi_r^1, \pi_r^2, \ldots, \pi_r^{\lceil \ell/k^d \rceil} \text{ all have regular bad evictions}\right] \\
&\quad + \Pr\left[\pi_r^2, \pi_r^3, \ldots, \pi_r^{\lceil \ell/k^d \rceil+1} \text{ all have regular bad evictions}\right] \\
&\leq 2(k^{-3c})^{\lceil \ell/k^d \rceil}. \\
&\qquad \text{(because subphases use independent hash functions, and by Cl}^{\text{目标 3)}}
\end{aligned} \tag{6}$$

最后，

$$\begin{aligned}
\mathbb{E}[R_r] &= \sum_{\ell \geq 1} \Pr[R_r \geq \ell] = \sum_{\ell \geq 1} 2(k^{-3c})^{\lceil \ell/k^d \rceil} \\
&= 2k^d \sum_{a \geq 1} (k^{-3c})^a \leq k^d O(k^{-3c}) \\
&= O(k^{-2c}). \qquad \text{(as } c > d\text{)}
\end{aligned}$$

[7]注意，证明的这一部分对于 LRU-K （带有 $K \geq 2$）和 LFU 并不成立，因为重新哈希会清除组相联缓存中的访问历史。

□

注意，阶段数恰好为 $C(\mathrm{LRU}_{k'}, \sigma)$。根据断言 5 和期望的线性性，$\mathbb{E}[R] \le C(\mathrm{LRU}_{k'}, \sigma)\, O(k^{-2c})$。根据马尔可夫不等式，$R \ge C(\mathrm{LRU}_{k'}, \sigma)\, O(k^{-c})$ 的概率为 $O(k^{-c})$。将其代入公式 (4)，得到

$$\begin{aligned} C(\langle \mathrm{LRU} \rangle_k^{\mathrm{FF}}, \sigma) &\le (1 + 2k^{-d+1})(1 + O(k^{-c}))\, C(\mathrm{LRU}_{k'}, \sigma) \\ &\le (1 + O(k^{-d+1}))\, C(\mathrm{LRU}_{k'}, \sigma) \qquad (\text{as } c > d) \end{aligned}$$

其概率为 $1 - O(k^{-c})$。证明至此结束。 □

6.1 增量式刷新

完全刷新的一项缺点是，它会使执行停止，并在慢速内存中执行最多 k 次连续写入。另一种做法是将刷新去摊销，把驱逐分散到一段时间内。我们称这种方法为增量式刷新，其工作方式如下：触发重新哈希时，我们生成一个新的哈希函数，并立即恢复执行。在下一次重新哈希之前的任意时刻，选取一个尚未重新映射的任意缓存项（即重新哈希开始时已缓存、且仍位于旧桶中的项），并将其驱逐。如果执行过程中命中一个尚未重新映射的项，我们就将它插入新桶中（如果新桶已满，则可能导致一次驱逐），并命令旧桶的 LRU 实例将其删除。当不再有任何项由旧函数映射时，重新哈希即告完成；此时我们丢弃旧哈希函数。重要的是，由于每次重新哈希都会在下一次开始之前完成，因此任何时候最多只有两个哈希函数在使用。每次访问时，我们都会用这两个函数查找请求的项。

然而，采用增量刷新时，定理 5 中的竞争分析会出现两个问题，原因是同时使用了两个哈希函数。第一个问题是，由于索引函数在整个子阶段内不断变化，公式 (5) 上界中的哈希函数取决于不良驱逐发生的时间。幸运的是，这无需修正，因为无论具体发生在何时，变化中的哈希函数始终是完全随机的，而这正是我们通过球箱过程进一步对概率取上界所需要的。

第二个问题是，重新哈希开始之前的随机放置与开始之后的放置并不独立。这意味着各子阶段中的随机选择不再相互独立，因此公式 (6) 后面的推导不再成立。对此有一个简单的修正方法：由于每个哈希函数最多只在连续两个子阶段内起作用，我们可以对每隔一个子阶段发生不良驱逐的概率进行上界估计。因此，上界变为

$$\Pr[R_r \ge \ell] \le 2(k^{-3c})^{\lceil \ell / k^d \rceil / 2},$$

这只会改变最终结果中的常数。其余证明无需修改即可成立。

我们将带有增量刷新机制的 α 路组相联 LRU 记为 $\langle \mathrm{LRU} \rangle^{\mathrm{IF}}$.

命题4。定理5对 $\langle \mathrm{LRU} \rangle^{IF}$ 成立。

6.2 针对最优离线算法的竞争分析

众所周知，使用 r-资源增强 [53]时，LRU 相对于离线最优全相联分页算法 OPT 是 $(1 + 1/(r-1))$-竞争的。结合定理5和命题4，我们可以将组相联 LRU 缓存与 OPT 进行比较。

命题5。假设 $\alpha = \omega(\log k)$。对于所有 $r > 1$, $\langle \text{LRU} \rangle^{FF}$ 和 $\langle \text{LRU} \rangle^{IF}$，在所有请求序列上，使用 $(1+o(1))r$-资源增强时，它们以高概率是相对于 OPT 的 $(1 + 1/(r-1) + o(1))$-竞争算法。

证明。令 $k' = k/(1 + \Theta(\sqrt{\log(k)/\alpha}))$ 和 $k'' = k'/r$。我们将比较 $\langle \text{LRU} \rangle_k^{\text{FF}}$ 与 $\text{OPT}_{k''}$。因此，资源增强为 $k/k'' = (1 + \Theta(\sqrt{\log(k)/\alpha}))r$。我们以高概率有，

$$\begin{aligned} C(\langle \text{LRU} \rangle_k^{\text{FF}}, \sigma) &\le (1 + 1/\operatorname{poly}(k))\, C(\text{LRU}_{k'}, \sigma) && \text{(by Theorem 5)} \\ &\le (1 + 1/\operatorname{poly}(k))(1 + 1/(r-1))\, C(\text{OPT}_{k''}, \sigma) \\ &\qquad && \text{(by the competitive analysis of LRU)} \\ &= (1 + 1/(r-1) + 1/\operatorname{poly}(k))\, C(\text{OPT}_{k''}, \sigma). \end{aligned}$$

对于 $\langle \text{LRU} \rangle^{\text{IF}}$，证明当然是类似的。 □

特别地，令 $r = 2$，我们得出，组相联LRU缓存借助 $(2 + o(1))$-资源增强，在所有请求序列上以高概率具有相对于 OPT 的 $(2 + o(1))$-竞争比。

7 稳定分页算法的类别

本节旨在理解稳定算法这一类。我们希望回答以下问题：哪些已知的分页算法是稳定的？哪些不是？更一般地说，一个算法稳定的必要条件和/或充分条件有哪些？

在本节中，我们假设项来自一个全序空间。和通常一样，我们考虑的所有分页算法都是在线算法。

7.1 栈算法

稳定算法与栈算法密切相关 [38]。如果对于每个 σ 和 k，分页算法 $\mathcal{A}$ 满足 $\mathcal{A}_k(\sigma) \subseteq \mathcal{A}_{k+1}(\sigma)$，那么它就是栈算法。栈算法的主要特性是不会出现贝拉迪异常，[8] 因为除非大小为 $k+1$ 的缓存也发生缓存未命中，否则大小为 k 的缓存不可能发生缓存未命中。因此，像 FIFO 和时钟算法 [9]，这样会出现贝拉迪异常的算法，都不是栈算法。

什么样的分页算法属于栈算法？粗略地说，栈算法会根据不依赖于 k 的排名来选择要驱逐的对象。具体定义如下。设 $\{\preceq_\sigma\}_{\sigma \in \mathcal{U}^*}$ 是 $\mathcal{U}$ 上全序关系的一个族。如果对于每个 $\tau \in \mathcal{U}^*$、$z \in \mathcal{U}$ 和 k，当 $x \in \mathcal{A}_k(\tau) \setminus \text{Out}(\mathcal{A}_k, \tau, z)$ 且 $y \in \text{Out}(\mathcal{A}_k, \tau, z)$ 时，$x \preceq_{\tau z} y$，那么我们称分页算法 $\mathcal{A}$ 符合 $\{\preceq_\sigma\}$。当 $\mathcal{A}$ 是惰性的，这意味着唯一被驱逐的项 y，恰好是 $\mathcal{A}_k(\tau)$ 中按 $\preceq_{\tau z}$ 排名最大的项。

以下定理展示了堆栈算法与序族之间的密切关系。尽管这种联系已为人所知 [38]，但据我们所知，先前没有任何工作给出完整的证明，因此我们在此将其列出。

定理 6。令 $\mathcal{A}$ 为一种惰性分页算法。则，当且仅当存在一个族 $\{\preceq_\sigma\}$ 使得 $\mathcal{A}$ 与其相符时，$\mathcal{A}$ 是一种栈算法。

为了证明这个定理，我们需要以下技术引理。

[8] 如果对于某些 σ 和整数 $a > b$, $C(\mathcal{A}_a, \sigma) > C(\mathcal{A}_b, \sigma)$，某个分页算法 $\mathcal{A}$ 会出现 Belady 异常 [9]。也就是说，出乎意料的是，在某些请求序列上，较小的缓存表现优于较大的缓存。

引理7.令 $\mathcal{A}$ 为一种惰性分页算法。则，对于每个 τ、 k 和 z，i f

$$\mathcal{A}_k(\tau) \subseteq \mathcal{A}_{k+1}(\tau) \text{ and } \mathit{Out}(\mathcal{A}_{k+1}, \tau, z) \cap \mathcal{A}_k(\tau z) \neq \emptyset,$$

then

$$\mathit{Out}(\mathcal{A}_k, \tau, z) \cap \mathcal{A}_{k+1}(\tau z) \neq \emptyset.$$

这个引理的证明推迟到本节末尾。

定理 6 的证明。(⇒) 令 $\Sigma = \{\sigma_1, \ldots, \sigma_{|\sigma|}\}$。我们将定义相对于 $\preceq_\sigma$ 的第 i 小元素。令 $s = |\Sigma|$。如果 $i > s$，答案就是未访问项集合 $\mathcal{U} \setminus \Sigma$ 中第 $(i - s)$ 小的项。否则， $i \leq s$。如果 $i = 1$，答案就是 $\sigma_{|\sigma|}$。否则， $1 < i \leq s$。注意，对于所有 $j \leq s$，$|\mathcal{A}_j(\sigma)| = j$，因为 $\mathcal{A}$ 是一种惰性算法。结合栈算法的包含性质，我们可以推得 $|\mathcal{A}_i(\sigma) \setminus \mathcal{A}_{i-1}(\sigma)| = 1$。我们将相对于 $\preceq_\sigma$ 的第 i 小元素定义为 $\mathcal{A}_i(\sigma) \setminus \mathcal{A}_{i-1}(\sigma)$ 中唯一的项。

这个序关系族满足： $\mathcal{A}_k(\sigma)$恰好是关于$\preceq_\sigma$的最小$\{k, s\}$ 个项的组。这意味着 $\mathcal{A}$ 符合 $\{\preceq_\sigma\}$。

(⇐) 我们要证明，对于任意 σ 和 k，都有 $\mathcal{A}_k(\sigma) \subseteq \mathcal{A}_{k+1}(\sigma)$。固定 k，并对$|\sigma|$使用归纳法。当 $|\sigma| = 0$时，结论显然成立。设 $\sigma = \tau z$，并假设结论对 τ成立，即

$$\mathcal{A}_k(\tau) \subseteq \mathcal{A}_{k+1}(\tau). \tag{7}$$

如果 z 没有在 $\mathcal{A}_{k+1}$中导致驱逐，那么由公式 (7) 以及 $\mathcal{A}$ 是惰性的这一事实可知，$\mathcal{A}_k(\tau z) \subseteq \mathcal{A}_{k+1}(\tau z)$，因此得证。

否则，令 $x \in \mathsf{Out}(\mathcal{A}_{k+1}, \tau, z)$。由于 $\mathcal{A}$ 是惰性的， $\mathcal{A}_{k+1}$没有驱逐其他项。因此，根据公式 (7)，只需证明 $x \notin \mathcal{A}_k(\tau z)$。反设 $x \in \mathcal{A}_k(\tau z)$。那么，

$$x \in \mathsf{Out}(\mathcal{A}_{k+1}, \tau, z) \cap \mathcal{A}_k(\tau z).$$

根据引理 7，存在一个

$$y \in \mathsf{Out}(\mathcal{A}_k, \tau, z) \cap \mathcal{A}_{k+1}(\tau z).$$

因此，对于 z， $\mathcal{A}_k$ 被驱逐而非 x，以及 $\mathcal{A}_{k+1}$ 被驱逐而非 y。由于 $\mathcal{A}$ 符合 $\preceq_\sigma$，这意味着 $x \preceq_\sigma y$ 和 $y \preceq_\sigma x$，矛盾。归纳法至此完成。 □

7.2 一个必要条件

接下来，我们建立稳定算法与栈算法之间的关系。

定理 7. 令 $\mathcal{A}$ 为惰性分页算法。如果 $\mathcal{A}$ 是稳定的，那么 $\mathcal{A}$ 是栈式的。

证明。我们想要证明，对于每个 k 和 σ， $\mathcal{A}_k(\sigma) \subseteq \mathcal{A}_{k+1}(\sigma)$。证明的结构类似于定理6。再次，我们固定 k，并对 $|\sigma|$使用归纳法。 $|\sigma| = 0$ 的情形是平凡的。令 $\sigma = \tau z$，并假设命题对 τ成立，即 $\mathcal{A}_k(\tau) \subseteq \mathcal{A}_{k+1}(\tau)$。由于 $\mathcal{A}$是惰性的，只需证明：如果 z 导致 $\mathcal{A}_{k+1}$ 驱逐一个项 x，那么 $x \notin \mathcal{A}_k(\tau z)$。

假设 $x \in \mathsf{Out}(\mathcal{A}_{k+1}, \tau, z)$，并为导出矛盾而假设 $x \in \mathcal{A}_k(\tau z)$。那么，根据引理 7，存在一个 $y \in \mathsf{Out}(\mathcal{A}_k, \tau, z) \cap \mathcal{A}_{k+1}(\tau z)$。

直观地说，这与 $\mathcal{A}$是稳定的这一事实相矛盾，因为较小的实例 $\mathcal{A}_k$ 应该驱逐 x，而它已不再位于较大的实例 $\mathcal{A}_{k+1}$中；相反， y仍在 $\mathcal{A}_{k+1}$中。从形式上看，根据关于 $X = \mathcal{U}$、 $a = k + 1$ 和 $b = k$的稳定算法定义，我们有that$y \in \mathsf{Out}(\mathcal{A}_k, \tau, z) \cap \mathcal{A}_{k+1}(\tau) \neq \emptyset$蕴含 $\mathcal{A}_k(\tau z) \subseteq \mathcal{A}_{k+1}(\tau z)$，这就产生了矛盾，因为$x \in \mathcal{A}_k(\tau z) \setminus \mathcal{A}_{k+1}(\tau z)$。

□

该定理为惰性稳定算法提供了一个必要条件，我们可以用它排除一些著名的、并非栈算法的惰性分页算法。

推论 2. FIFO 和时钟算法不是稳定的。

人们可能会想知道，稳定的算法和栈算法实际上是否相同，至少在将范围限制于惰性算法这一类时是如此。我们证明事实并非如此。

命题 6。存在一种非稳定的惰性栈算法。

证明。设 $\mathcal{R}$ 为分页算法，它始终淘汰在最近两次访问之间请求次数最多的项 x；此度量称为重用距离 [30]。设 $\Phi(\sigma, x)$ 为在 σ 中对项 x 最近两次访问之间的请求次数。若 x 的访问次数不足两次，则我们定义$\Phi(\sigma, x) = \infty$。当且仅当 $\Phi(\sigma, x) < \Phi(\sigma, y)$，或 $(\Phi(\sigma, x) = \Phi(\sigma, y)$ 且 $x \leq y)$ 时，考虑全序 $x \preceq_\sigma y$。这定义了一个序关系族，并且根据定义，$\mathcal{R}$ 符合该序关系族。因此，根据定理 6，$\mathcal{R}$ 是一种栈算法。

我们断言 $\mathcal{R}$ 不稳定。令 $\mathcal{U} = \{A, B, C, Y, Z\}$ 为项的全集。我们将比较 $\mathcal{R}_3$ 和 $\mathcal{R}_4$。令 $\sigma = AYZZZZABYYBC$。令 $X = \{A, B, C, Y\}$ 为 $\mathcal{R}_3$读取的项集。我们将证明，稳定算法的定义在 σ的最后一次访问中遭到违反。

一方面，$\mathcal{R}_3$ 读取序列 $\sigma[X] = AYABYYBC$，并且 $Y \preceq_{\sigma[X]} A \preceq_{\sigma[X]} B$。因此，在对 C的最终访问期间，$\mathcal{R}_3$ 驱逐了 B。另一方面，$\mathcal{R}_4$ 读取整个序列 σ，并且$Y \preceq_\sigma Z \preceq_\sigma B \preceq_\sigma A$。因此，当访问 C 时，$\mathcal{R}_4$ 驱逐了 A。由于 $\mathcal{R}_3$ 驱逐了 B（它仍在 $\mathcal{R}_4$中），且它仍包含 A（它不在 $\mathcal{R}_4$中），$\mathcal{R}$ 不是稳定的。

□

7.3 一个充分条件

假设某个分页算法遵循一个族 $\{\preceq_\sigma\}$。注意，在极端情况下，次序可能会从一个步骤到下一个步骤突然改变，也就是说，$\preceq_\sigma$ 可能与$\preceq_{\sigma x}$完全不同。这是因为次序只不过是该算法用来决定驱逐哪个项的一条规则，而这条规则可以随时自由改变。

我们将把注意力限制在不表现出此类不稳定性的序列族上。如果对于任意 σ，以及满足 $y \neq z$ 的 x、y、$z \in \sigma$，都有.....，则称 $\{\preceq_\sigma\}$ 是单调的。

$$\text{if } x \preceq_\sigma y, \text{ then } x \preceq_{\sigma z} y.$$

换言之，当一个访问只能改变被访问项的相对顺序，且被访问项在顺序中的位置不会变大时，该族是单调的。

引理 8. 令 $\mathcal{A}$ 是符合单调族 $\{\preceq_\sigma\}$的惰性分页算法。固定任意请求序列 σ。令 $n = |\sigma|$、$\Sigma_n = \{\sigma_1, \ldots, \sigma_n\}$和 $s = |\Sigma_n|$。令 x_1、...、x_s为 Σ_n 中按 $\preceq_\sigma$ 排序的元素。然后，对于每个 $k \leq s$,

$$\{x_1, \ldots, x_{k-1}\} \subseteq \mathcal{A}_k(\sigma).$$

引理 8 指出，缓存内容基本上由序关系族决定，除了缓存中最大的项可能例外。其证明见本节末尾。下面的推论概括了一项关键推论。

推论 3. 设 $\mathcal{A}$ 是一个符合单调族 $\{\preceq_\sigma\}$ 的惰性分页算法。那么，对于每个 σ、k 和 $y \in \sigma$，如果 $y \notin \mathcal{A}_k(\sigma)$，则至多存在一个 $x \in \mathcal{A}_k(\sigma)$ 使得 $y \preceq_\sigma x$。

回顾一下，稳定算法具有这样的性质：分别服务于请求序列 σ 和 $\sigma[X]$ 的两个实例的分页决策是“纠缠”的。我们希望通过序族来捕捉这一特征。我们称 $\{\preceq_\sigma\}$ 是自相似的，如果对于每个 σ、 $X \subseteq \mathcal{U}$ 和 x、$y \in \sigma[X]$,

$$\text{if } x \preceq_{\sigma[X]} y, \text{ then } x \preceq_\sigma y.$$

现在我们可以陈述本节的主要定理了。

定理 8. 设 $\mathcal{A}$ 是一个惰性分页算法。如果 $\mathcal{A}$ 符合一个单调且自相似的族 $\{\preceq_\sigma\}$，那么 $\mathcal{A}$是稳定的。

证明。设 $\tau \in \mathcal{U}^*$、 $X \subseteq \mathcal{U}$、 $z \in X$ 和 $a > b$。根据稳定算法的定义，我们要证明 $\mathsf{Out}(\mathcal{A}_b, \tau[X], z) \cap \mathcal{A}_a(\tau z) \neq \emptyset$蕴含 $\mathcal{A}_b(\tau[X]z) \subseteq \mathcal{A}_a(\tau z)$。假设前件成立；设 $x \in \mathsf{Out}(\mathcal{A}_b, \tau[X], z) \cap \mathcal{A}_a(\tau z)$。设 $y \in \mathcal{A}_b(\tau[X]z)$，接下来证明 $y \in \mathcal{A}_a(\tau z)$。

如果 $y = z$，则无须证明；假设 $y \neq z$。由于 $x \in \mathsf{Out}(\mathcal{A}_b, \tau[X], z)$，我们也有$x \neq z$。

由于 $y \in \mathcal{A}_b(\tau[X]z)$、 $y \neq z$和 $\mathcal{A}$是惰性的，我们有 $y \in \mathcal{A}_b(\tau[X])$。因此，在响应 z 时， x 被 $\mathcal{A}_b$ 逐出，而 y 仍保留在缓存中，所以 $y \preceq_{\tau[X]z} x$。由于 $y \neq z$，单调性性质得出

$$y \preceq_{\tau[X]} x. \tag{8}$$

那么，自相似性属性意味着

$$y \preceq_\tau x. \tag{9}$$

由于 $x \neq z$，根据单调性，我们有，

$$y \preceq_{\tau z} x. \tag{10}$$

我们声称 $y \preceq_\tau z$。由于 $x \in \mathsf{Out}(\mathcal{A}_b, \tau[X], z)$ 且 $\mathcal{A}$是惰性的，我们有 $z \notin \mathcal{A}_b(\tau[X])$。注意，它不可能是 $z \preceq_{\tau[X]} y$，因为公式 (8) 将意味着 $z \preceq_{\tau[X]} x$，但根据推论 3，关于 $\preceq_{\tau[X]}$， $z \notin \mathcal{A}_b(\tau[X])$ 不可能小于 $\mathcal{A}_b(\tau[X])$ 中的两个项。因此， $y \preceq_{\tau[X]} z$。根据自相似性，

$$y \preceq_\tau z. \tag{11}$$

回顾一下，我们的目标是证明 $y \in \mathcal{A}_a(\tau z)$。为反证，假设 $y \notin \mathcal{A}_a(\tau z)$。由于 $y \neq z$ 且 $\mathcal{A}$ 是惰性的，我们有

$$y \notin \mathcal{A}_a(\tau). \tag{12}$$

类似地，由于 $x \in \mathcal{A}_a(\tau z)$， $x \neq z$ 且 $\mathcal{A}$ 是惰性的，我们有

$$x \in \mathcal{A}_a(\tau). \tag{13}$$

为避免与推论 3 矛盾，公式 (9)、公式 (11)、公式 (12) 和公式 (13) 表明

$$z \notin \mathcal{A}_a(\tau), \tag{14}$$

并且

$$w \preceq_\tau y \text{ for every } w \in \mathcal{A}_a(\tau) \setminus \{x\}. \tag{15}$$

单调性表明

$$w \preceq_{\tau z} y \text{ for every } w \in \mathcal{A}_a(\tau) \setminus \{x\}. \tag{16}$$

因此，根据公式 (10) 和公式 (16)， x 是 $\mathcal{A}_a(\tau)$ 中相对于 $\preceq_{\tau z}$ 的最大项。根据公式 (14)，请求z 在 $\mathcal{A}_a$ 中未命中（且缓存已满，因为 $\mathcal{A}$ 是惰性的，并且此前已经发生过驱逐，即驱逐了 y）。因此， $x \in \mathsf{Out}(\mathcal{A}_a, \tau, z)$，这与假设矛盾。 □

7.4 稳定算法示例

引理 5。LRU-K 是稳定的。

证明。我们将对 $\mathcal{A}$ = LRU-K使用定理8。令 $\Phi(\sigma, x)$ 表示自K对项 x 的最近一次访问以来，在 σ 中的请求数。如果 x 被访问的次数少于 K，则我们定义 $\Phi(\sigma, x) = \infty$ 。当且仅当 $\Phi(\sigma, x) < \Phi(\sigma, y)$ 或 $(\Phi(\sigma, x) = \Phi(\sigma, y)$ 且 $x \leq y)$时，令 $x \preceq_\sigma y$。这定义了LRU-K 所遵循的序关系族。该族是单调的，因为访问一个项不会增加其第 K 近的访问时间，而所有其他项的相对顺序保持不变，因为在新的访问之后，它们第 K近的访问时间都增加 1。为完成证明，我们说明 $\{\preceq_\sigma\}$ 也是自相似的。

假设 $x \preceq_{\sigma[X]} y$，其中 x、$y \in \sigma[X]$，并令我们证明 $x \preceq_\sigma y$。如果 $x = y$，则该命题显然成立，因此假设 $x \neq y$。如果 $\Phi(\sigma[X], x) = \Phi(\sigma[X], y)$，那么它们必定都等于 ∞，因为两个不同项的第 K 近访问时间不可能相同。因此， x 和 y 在 $\sigma[X]$ 中被访问的次数都少于 K 次，因此在 σ 中也是如此，所以 $\Phi(\sigma, x) = \Phi(\sigma, y) = \infty$。因此，$x \preceq_\sigma y$。

否则， $\Phi(\sigma[X], x) \neq \Phi(\sigma[X], y)$。不失一般性，假设 $\Phi(\sigma[X], x) < \Phi(\sigma[X], y)$。从 $\sigma[X]$ 开始，我们逐一重新插入从 σ中移除的每次访问。关键的是，这些访问都不是对 x 或 y的访问。由于对 y 的倒数第 K 次访问比对 x的访问更早，因此每次我们在序列中插入一次项访问时，要么二者都变旧一个单位时间，要么只有 y变旧。于是，在这一过程结束时，对 y 的倒数第 K 次访问仍然比对x的访问更早，即 $\Phi(\sigma, x) < \Phi(\sigma, y)$。因此， $x \preceq_\sigma y$，证明完毕。 □

引理 6：LFU 是稳定的。

证明。我们只略述证明，因为它与前一个证明类似。令 $\Phi(\sigma, x)$为对 x 在 σ中的访问次数。令 $x \preceq_\sigma y$ 当且仅当 $\Phi(\sigma, x) > \Phi(\sigma, y)$ 或（$\Phi(\sigma, x) = \Phi(\sigma, y)$且$x \leq y$）。LFU遵循此序关系族，可以证明该序关系族是单调的且具有自相似性。根据定理 8，LFU 是稳定的。 □

7.5 缺失的证明

本节余下部分将证明引理 7 和引理 8。为方便起见，我们在此重述它们。

引理 7. 设 $\mathcal{A}$ 是一种惰性分页算法。那么，对于任意的 τ、 k 和 z，如果

$$\mathcal{A}_k(\tau) \subseteq \mathcal{A}_{k+1}(\tau) \text{ and } \mathit{Out}(\mathcal{A}_{k+1}, \tau, z) \cap \mathcal{A}_k(\tau z) \neq \emptyset,$$

then

$$\mathit{Out}(\mathcal{A}_k, \tau, z) \cap \mathcal{A}_{k+1}(\tau z) \neq \emptyset.$$

证明. 为使论证简洁，每次使用 $\mathcal{A}$ 时，我们都不再明确提及其惰性。假设

$$\mathcal{A}_k(\tau) \subseteq \mathcal{A}_{k+1}(\tau), \tag{17}$$

and

$$\mathsf{Out}(\mathcal{A}_{k+1}, \tau, z) \cap \mathcal{A}_k(\tau z) \neq \emptyset, \tag{18}$$

并证明 $\mathsf{Out}(\mathcal{A}_k, \tau, z) \cap \mathcal{A}_{k+1}(\tau z) \neq \emptyset$。

令 $x \in \mathsf{Out}(\mathcal{A}_{k+1}, \tau, z) \cap \mathcal{A}_k(\tau z)$，根据公式（18）。由于 $\mathcal{A}_{k+1}$驱逐了 x 以获取 z，当访问 z 时，它拥有一个已满的缓存。然后， $\mathcal{A}_k$ 当时也已满。注意， z 在 $\mathcal{A}_k$中未命中；

否则，如果 $z \in \mathcal{A}_k(\tau)$，那么根据公式 (17)，$z \in \mathcal{A}_{k+1}(\tau)$，这与 z在 $\mathcal{A}_{k+1}$中导致了一次驱逐这一事实相矛盾。因此，$\mathcal{A}_k$ 在未命中 z时已满，并且它必定驱逐了一个项，即

$$y \in \mathsf{Out}(\mathcal{A}_k, \tau, z). \tag{19}$$

为完成证明，我们证明 $y \in \mathcal{A}_{k+1}(\tau z)$。公式 (19) 蕴含 $y \in \mathcal{A}_k(\tau)$。然后，根据公式 (17)，

$$y \in \mathcal{A}_{k+1}(\tau). \tag{20}$$

注意到 $y \neq x$，因为 $x \in \mathcal{A}_k(\tau z)$ 且 $y \notin \mathcal{A}_k(\tau z)$。特别是，$y$ 未被 $\mathcal{A}_{k+1}$ 驱逐。结合公式 (20)，我们得出 $y \in \mathcal{A}_{k+1}(\tau z)$，正如所期望的。 □

引理 8。设 $\mathcal{A}$ 是一个符合单调族 $\{\preceq_\sigma\}$ 的惰性分页算法。固定任意请求序列 σ。设 $n = |\sigma|$、$\Sigma_n = \{\sigma_1, \ldots, \sigma_n\}$ 和 $s = |\Sigma_n|$。设 x_1、……、x_s 是 Σ_n 中按 $\preceq_\sigma$ 排序的元素。那么，对于每个 $k \leq s$,

$$\{x_1, \ldots, x_{k-1}\} \subseteq \mathcal{A}_k(\sigma).$$

证明。再次，我们将使用 $\mathcal{A}$ 是惰性的这一事实，而不对此作明确说明。证明对 n进行归纳。$n = 0$ 的情形是平凡的。设 $n \geq 1$。设 $\tau = \sigma_1 \ldots \sigma_{n-1}$。令 y_1、……、$y_{|\Sigma_{n-1}|}$ 为按 $\preceq_\tau$ 排序的 Σ_{n-1} 的项。

固定 $k \leq s$。由归纳假设,

$$\{y_1, \ldots, y_{k-1}\} \subseteq \mathcal{A}_k(\tau). \tag{21}$$

由于序关系族是单调的,

$$\{x_1, \ldots, x_{k-1}\} \subseteq \{y_1, \ldots, y_{k-1}\} \cup \{\sigma_n\}. \tag{22}$$

（直观地说，当访问 σ_n 时，它会从项 y_1、y_2、……组成的有序列表中的当前位置被取出并向前移动；如果之前没有访问过 σ_n，则会将其插入列表中的某个位置。)

如果 $\sigma_n \in \mathcal{A}_k(\tau)$，那么公式 (21) 表明

$$\{y_1, \ldots, y_{k-1}\} \cup \{\sigma_n\} \subseteq \mathcal{A}_k(\sigma).$$

因此，由公式 (22)，$\{x_1, \ldots, x_{k-1}\} \subseteq \mathcal{A}_k(\sigma)$。

否则，$\sigma_n \notin \mathcal{A}_k(\tau)$。根据公式 (21)，我们有 $\mathcal{A}_k(\tau) = \{y_1, \ldots, y_{k-1}, z\}$，其中某个 z 在关于 $\preceq_\tau$ 的意义下，比 $\mathcal{A}_k(\tau)$ 中其他所有项都大。根据单调性，z 在关于$\preceq_\sigma$的意义下，也是 $\mathcal{A}_k(\tau)$ 中最大的。因此，当 σ_n 被访问时，z 会被逐出，并且

$$\mathcal{A}_k(\sigma) = \{y_1, \ldots, y_{k-1}, \sigma_n\}.$$

由公式 (22)，我们得出 $\{x_1, \ldots, x_{k-1}\} \subseteq \mathcal{A}_k(\sigma)$，正如所需。归纳法至此完成。

□

8 结论

缓存设计者可采用的最小相联度 α a 是多少，才能使组相联缓存的命中率接近全相联缓存？这个数十年来悬而未决的问题的答案是：将 α 设置为略高于缓存大小的对数。然而，这项性能保证仅在有限时间内有效。如果缓存设计者还希望组相联缓存能在任意长的执行过程中保持相近的性能，那么每经过多项式次缓存未命中后进行一次重新散列即可。

致谢

我们感谢 SPAA ’23 的匿名评审者提出的意见，这些意见帮助我们改进了论文的表达。我们谨此感谢美国国家科学基金会通过 CCF-2118830、CCF-2106827、CSR-1763680、CNS-1938709、CNS-2118620 和 CCF-2106999 等项目提供的资助。本研究还部分得到了加拿大自然科学与工程研究委员会的支持。

参考文献

[1] Dolev Adas、GilEinziger和Roy Friedman。有限相联度使并发软件缓存变得轻而易举。载于第23届国际分布式计算与网络会议（ICDCN）论文集，第87–96页，纽约，纽约州，美国，2022年。计算机协会。[2] Anant Agarwal、JohnHennessy 和 Mark Horowitz。操作系统和多道程序工作负载的缓存性能。ACM计算系统汇刊，6(4):393–431，1988年11月。[3] Alok Aggarwal和 S.Vitter, Jeffrey。排序及相关问题的输入/输出复杂度。ACM通讯，31(9):1116–1127，1988年9月。[4] Kunal Agrawal、Michael A. Bender、Rathish Das、William Kuszmaul、Enoch Peserico和Michele Scquizzato。绿色分页与并行分页。载于第32届ACM算法与体系结构并行性研讨会（SPAA）论文集，第493–495页，2020年7月。[5] Kunal Agrawal、Michael A. Bender、Rathish Das、William Kuszmaul、EnochPeserico和Michele Scquizzato。并行分页和绿色分页的紧确界。载于第32届年度 ACM-SIAM离散算法研讨会（SODA）论文集，第3022–3041页，2021年1月。[6] Kunal Agrawal、Michael A Bender、Rathish Das、William Kuszmaul、Enoch Peserico 和Michele Scquizzato。具有最优总完成时间的在线并行分页。载于第34届 ACM算法与体系结构并行性研讨会论文集，第205–216页，2022年。[7] Kunal Agrawal、MichaelA. Bender 和 Jeremy T.Fineman。最差的页面置换策略。计算系统理论，44:175–185，2 2009年。[8] Kathirgamar Aingaran、Sumti Jairath、Georgios Konstadinidis、Serena Leung、PaulLoewen- stein、Curtis McAllister、Stephen Phillips、Zoran Radovic、Ram Sivaramakrishnan、David Smentek 和Thomas Wicki。M7：甲骨文的下一代SPARC处理器。IEEE Micro，35(2):36–45，2015年。[9] L. A. Belady、R.A.Nelson和G.S.Shedler。在分页机器上运行的某些程序的时空特征异常。ACM通讯，12(6):349–353，1969年6月。[10] James Bell、David Casasent 和 C.Gordon Bell。对替代缓存组织形式的研究。IEEE计算机汇刊，C-23(4):346–351，1974年。[11] Michael A. Bender、AbhishekBhattacharjee、Alex Conway、Mart´ın Farach-Colton、Rob John-son、Sudarsun Kannan、William Kuszmaul、Nirjhar Mukherjee、Don Porter、Guido Tagliavini、JanetVorobyeva 和 Evan West。分页与地址转换问题。载于第33届ACM算法与体系结构并行性研讨会（SPAA）论文集，第105–117页，纽约，纽约州，美国，2021年。计算机协会。

[12] Allan Borodin 和 Ran El-Yaniv。《在线计算与竞争分析》。剑桥大学出版社，美国，1998年。[13] Bill Bowhill、Blaine Stackhouse、Nevine Nassif、Zibing Yang、Arvind Raghavan、Charles Mor- ganti、Chris Houghton、Dan Krueger、Olivier Franza、Jayen Desai、Jason Crop、Dave Bradley、Chris Bostak、Sal Bhimji 和 Matt Becker。Xeon® 处理器 E5-2600 v3：22纳米 18核产品系列。载于《2015年 IEEE 国际固态电路会议（ISSCC）技术论文摘要》，第1–3页。IEEE计算机学会，2015年。[14] Joan Boyar、Martin R. Ehmsen、Jens S. Kohrt 和 Kim S. Larsen。LRU 与 LRU-K 的理论比较。《信息学学报》，47：359–374，2010年。[15] Joan Boyar、Lene M. Favrholdt 和 Kim S. Larsen。相对最坏顺序比在分页问题中的应用。《计算机与系统科学杂志》，73(5)：818–843，2007年8月。[16] Mark Brehob、Richard Enbody、Eric Torng 和 Stephen Wagner。在线受限缓存。载于《第十二届 ACM-SIAM 离散算法年度研讨会（SODA）论文集》，第374–383页，美国，2001年。工业与应用数学学会。[17] Niv Buchbinder、Shahar Chen 和 Joseph (Seffi) Naor。受限缓存与拟阵缓存的竞争算法。载于 Andreas S. Schulz 和 Dorothea Wagner 编，《第22届欧洲算法年度研讨会（ESA）论文集》，第209–221页，柏林、海德堡，2014年9月。施普林格出版社。[18] Edward G. Coffman 和 Peter J. Denning。《操作系统理论》。普伦蒂斯·霍尔专业技术参考丛书，1973年。[19] Rathish Das、Kunal Agrawal、Michael A Bender、Jonathan Berry、Benjamin Moseley 和 Cynthia A Phillips。如何自动管理高带宽内存。载于《第32届 ACM 算法与体系结构并行性研讨会（SPAA）论文集》，第187–199页，2020年。[20] Daniel DeLayo、Kenny Zhang、Kunal Agrawal、Michael A Bender、Jonathan Berry、Rathish Das、Benjamin Moseley 和 Cynthia A Phillips。自动 HBM 管理：模型与算法。载于《第34届 ACM 算法与体系结构并行性研讨会（SPAA）论文集》，2022年。[21] Reza Dorrigiv 和 Alejandro L´opez-Ortiz。缩小理论与实践之间的差距：在线算法分析的新度量。载于《国际算法与计算研讨会（WALCOM）》，第13–24页。施普林格出版社，柏林-海德堡，2008年。[22] Reza Dorrigiv、Alejandro L´opez-Ortiz 和 J. Ian Munro。分页算法的相对支配性。《理论计算机科学》，410(38–40)：3694–3701，2009年9月。[23] Devdatt Dubhashi 和 Alessandro Panconesi。《随机化算法分析中的测度集中》。剑桥大学出版社，美国，第一版，2009年。[24] Amos Fiat、Richard M. Karp、Michael Luby、Lyle A. McGeoch、Daniel D. Sleator 和 Neal E. Young。竞争性分页算法。《算法杂志》，12(4)：685–699，1991年12月。[25] James D. Fix。搜索树的组相联缓存性能。载于《第十四届 ACM-SIAM 离散算法年度研讨会（SODA）论文集》，第565–572页，美国，2003年。工业与应用数学学会。

[26] M. Frigo、Charles E. Leiserson、Harald Prokop 和 Sridhar Ramachandran。缓存无关算法。载于《1999年IEEE第40届计算机科学基础年度研讨会（FOCS）论文集》，第285页，美国加利福尼亚州洛斯阿拉米托斯，1999年10月。IEEE计算机学会。[27] John S. Harper、Darren J. Kerbyson 和 Graham R. Nudd。组相联缓存行为的分析建模。《IEEE计算机汇刊》，48(10):1009–1024，1999年10月。[28] Mark D. Hill。直接映射缓存的优势。《Computer》，21(12):25–40，1988年12月。[29] Mark D. Hill 和 Alan J. Smith。评估CPU缓存中的相联度。《IEEE计算机汇刊》，38(12):1612–1630，1989年12月。[30] Song Jiang 和 Xiaodong Zhang。Lirs：一种高效的低跨引用间隔最近使用组替换策略，用于提升缓冲缓存性能。载于《2002年ACM SIGMETRICS计算机系统测量与建模国际会议论文集》，SIGMETRICS '02，第31–42页，美国纽约州纽约，2002年。计算机协会。[31] Norman P. Jouppi。通过增加小型全相联缓存和预取缓冲区来提升直接映射缓存性能。《ACM SIGARCH计算机体系结构新闻》，18，1990年。[32] Anna R. Karlin、Mark S. Manasse、Larry Rudolph 和 Daniel D. Sleator。竞争式窥探缓存。载于《第27届计算机科学基础年度研讨会（FOCS）论文集》，第244–254页，1986年。[33] Richard E. Kessler、R. Jooss、Alvin R. Lebeck 和 Mark D. Hill。组相联的低成本实现。载于《第16届计算机体系结构国际年度研讨会（ISCA）》，第131–139页。IEEE计算机学会，1989年。[34] William F. King。需求分页算法分析。载于IFIP大会，1971年。[35] Georgios K. Konstadinidis、Hongping Penny Li、Francis Schumacher、Venkat Krishnaswamy、Hoyeol Cho、Sudesna Dash、Robert P. Masleid、Chaoyang Zheng、Yuanjung David Lin、Paul Loewenstein、Heechoul Park、Vijay Srinivasan、Dawei Huang、Changku Hwang、Wenjay Hsu、Curtis McAllister、Jeff Brooks、Ha Pham、Sebastian Turullols、Yifan Yanggong、Robert Golla、Alan P. Smith 和 Ali Vahidsafa。SPARC M7：一款采用20纳米制程、拥有32核和64 MB L3缓存的处理器。《IEEE固态电路期刊》，51(1):79–91，2016年。[36] Nasser Kurd、Muntaquim Chowdhury、Edward Burton、Thomas P. Thomas、Christopher Mozak、Brent Boswell、Manoj Lal、Anant Deval、Jonathan Douglas、Mahmoud Elassal、Ankireddy Nalamalpu、Timothy M. Wilson、Matthew Merten、Srinivas Chennupaty、Wilfred Gomes 和 Rajesh Kumar。Haswell：一系列IA 22纳米处理器。载于《2014年IEEE国际固态电路会议技术论文集（ISSCC）》，第112–113页。IEEE计算机学会，2014年。[37] Mark S. Manasse、Lyle A. McGeoch 和 Daniel D. Sleator。服务器问题的竞争算法。《算法杂志》，11(2):208–230，1990年5月。[38] Richard L. Mattson、Jan Gecsei、D. R. Slutz 和 I. L. Traiger。存储层次结构的评估技术。《IBM系统期刊》，9(2):78–117，1970年6月。

[39] M. Mendel 和 Steven S. Seiden。在线伴随缓存。《理论计算机科学》，324(2–3):183–200，2004年9月。[40] Nima Mousavi。Chernoff 界有多紧？笔记，2012年。[41] Elizabeth J. O’Neil、Patrick E. O’Neil 和 Gerhard Weikum。用于数据库磁盘缓冲的LRU-K页面置换算法。载于《1993年ACM SIGMOD数据管理国际会议论文集》，第297–306页，美国纽约州纽约，1993年。计算机协会。[42] Elizabeth J. O’Neil、Patrick E. O’Neil 和 Gerhard Weikum。LRU-K页面置换算法的最优性证明。《ACM期刊》，46(1):92–112，1999年1月。[43] Enoch Peserico。具有任意相联度的在线分页。载于《第十四届 ACM-SIAM 离散算法年度研讨会（SODA）论文集》，第555–564页，美国，2003年。工业与应用数学学会。[44] S. Prybylski、M. Horowitz 和 J. Hennessy。缓存设计中的性能权衡。载于《第15届计算机体系结构国际年度研讨会（ISCA）论文集》，第290–298页，美国华盛顿特区，1988年。IEEE计算机学会。[45] Moinuddin K. Qureshi。Ceaser：通过加密地址和重新映射缓解基于冲突的缓存攻击。载于《第51届IEEE/ACM微体系结构国际年度研讨会（MICRO）论文集》，第775–787页。IEEE出版社，2018年。[46] Moinuddin K. Qureshi。加密地址缓存的新型攻击与防御。载于《第46届计算机体系结构国际研讨会（ISCA）论文集》，第360–371页，美国纽约州纽约，2019年。计算机协会。[47] Gururaj S. Rao。高速缓存存储器的性能分析。《ACM期刊》，25(3):378–395，1978年7月。[48] RocksDB。块缓存。RocksDB维基，2022年。最后访问日期：2023-01-09。[49] Efraim Rotem、Adi Yoaz、Lihu Rappoport、Stephen J. Robinson、Julius Yuli Mandelblat、Arik Gihon、Eliezer Weissmann、Rajshree Chabukswar、Vadim Basin、Russell Fenger、Monica Gupta 和 Ahmad Yasin。英特尔Alder Lake CPU架构。《IEEE Micro》，42(3):13–19，2022年。[50] Peter Sanders。通过组相联缓存访问多个序列。载于《第26届自动机、语言与编程国际研讨会（ICALP）论文集》，第655–664页，柏林、海德堡，1999年。施普林格出版社。[51] Rathijit Sen 和 David A. Wood。缓存的基于重用的在线模型。载于《计算机系统测量与建模国际会议（SIGMETRICS）论文集》，第279–292页，美国纽约州纽约，2013年。计算机协会。[52] Sandeep Sen、Siddhartha Chatterjee 和 Neeraj Dumir。迈向缓存高效算法理论。《ACM期刊》，49(6):828–858，2002年11月。[53] Daniel D. Sleator 和 Robert E. Tarjan。链表更新与分页规则的摊销效率。《ACM通讯》，28(2):202–208，1985年2月。

[54] Alan J. Smith. 论组相联页映射的有效性及其在主存管理中的应用。载于《第二届国际软件工程会议（ICSE）论文集》，第286–292页，美国华盛顿特区，1976年。IEEE计算机学会出版社。[55] Alan J. Smith. 组相联存储器映射算法及其在缓存和主存中的应用比较研究。《IEEE软件工程汇刊》，4(2):121–130，1978年3月。[56] Simon M. Tam、Harry Muljono、Min Huang、Sitaraman Iyer、Kalapi Royneogi、Nagmohan Satti、Rizwan Qureshi、Wei Chen、Tom Wang、Hubert Hsieh、Sujal Vora 和 Eddie Wang。Skylake-sp：一款14纳米28核Xeon®处理器。载于《2018年IEEE国际固态电路会议（ISSCC）》，第34–36页。IEEE计算机学会，2018年。[57] Nigel Topham 和 Antonio Gonz´alez。通过随机缓存放置消除冲突。《IEEE计算机汇刊》，48(2):185–192，1999年2月。[58] David Wajc。负关联——定义、性质与应用，2017年。[59] Xiaoya Xiang、Chen Ding、Hao Luo 和 Bin Bao。HOTL：高阶局部性理论。载于《第十八届体系结构支持编程语言与操作系统国际会议（ASPLOS）论文集》，第343–356页，美国纽约，2013年。计算机协会。[60] Neal Young。《竞争分页与双重引导的在线加权缓存与监视算法》。博士论文，普林斯顿大学，美国，1992年。[61] Neil Young。k服务器对偶与分页的宽松竞争性。《Algorithmica》，11(6):525–541，1994年。[62] Neil Young。在线文件缓存。《Algorithmica》，33(3):371–383，2002年。