arXiv:2305.02508v1 [cs.DS] 4 May 2023

通过标记实现带预留的高效缓存*

Sharat Ibrahimpur ✉数学系，伦敦政治经济学院，英国

ManishPurohit ！
Google Research，美国
Zoya Svitkina Google Research，美国

ErikVee ！
GoogleResearch，美国
JoshuaR.Wang ！
Google Research，美国

摘要

在线缓存是在线算法领域中最基础、研究最为广泛的问题之一。创新的算法思想与分析方法——包括势函数和原始-对偶技术——为这一仍在不断发展的领域提供了深刻见解。在本文中，我们提出了一种新的分析技术：首先利用势函数对在线算法的代价进行上界估计，然后结合一种新的对偶拟合策略，对离线最优算法的代价进行下界估计。我们将这些技术应用于 Ibrahimpur 等人近期提出的带预留的缓存问题，并通过标记策略给出一种具有 $O(\log k)$ 竞争比的分数型在线算法，其中 k 表示缓存的大小。我们还设计了一种新的在线舍入算法，该算法可在多项式时间内运行，从而获得一种具有 $O(\log k)$ 竞争比的随机整数算法。此外，我们还为经典无权分页问题中的随机标记策略提供了一种全新且简洁的证明。

<escape>**2012年 ACM 学科分类** 计算理论 → 缓存与分页算法</escape>

关键词和短语 近似算法、在线算法、缓存

资助 *Sharat Ibrahimpur*：获得了以下来源的资助：加拿大自然科学与工程研究理事会（NSERC）资助项目 327620- 09 和一项 NSERC DAS 奖、欧洲研究理事会（ERC）通过欧盟地平线2020研究与创新计划提供的资助（资助协议编号：ScaleOpt–757481），以及荷兰研究理事会 NWO Vidi 资助 016.Vidi.189.087。

1 引言

缓存是许多计算机系统中的关键组成部分，包括计算机网络、分布式系统和网络应用。缓存背后的理念很简单：将频繁使用的数据项存储在缓存中，以便后续请求可以直接从缓存中得到服务，从而减少数据检索所需的资源。在经典的无权缓存问题中，一系列页面请求逐个到达，算法需要维护一个较小的页面集合并将其存放在缓存中，使未能由缓存服务的请求数量最少。

传统的缓存算法，无论在理论上还是实践中，都是为了优化系统的全局效率而设计的，旨在最大化命中率，即从缓存中满足的请求所占的比例。然而，这种观点并不特别适用于多用户或多处理器环境中的缓存管理。许多云计算服务允许多个用户共享同一台物理工作站，从而共享

* 扩展摘要将刊载于 2023 年第50届 ICALP 会议论文集。

缓存系统。在此类多用户环境中，传统缓存策略可能导致不理想的结果，因为某些用户可能完全无法从缓存中获益。近期，Ibrahimpur 等人 [10] 提出了带预留的缓存 模型，既能确保一定的用户级公平性保证，又能尽力提升系统的全局效率。在这一模型中，大小为 k 的缓存由 m 个代理共享，并保证每个代理 i 都拥有大小为 k_i的预留缓存。随后，算法尝试尽量减少未从缓存中服务的请求总数，同时保证在任意时间步，每个代理 i 在缓存中至少持有 k_i 个页面。与经典分页问题不同，即使在算法提前知晓整个页面请求序列的离线情形下，带预留的缓存 仍是 NP 完全问题；Ibrahimpur 等人 [10]给出了一个 2-近似算法。他们还通过原始-对偶技术，为带预留的缓存 给出了一个 $O(\log k)$-竞争的在线分数算法，随后设计了一种舍入方案，从而得到一个 $O(\log k)$-竞争的在线随机化算法。遗憾的是，[10] 中提出的舍入方案无法在多项式时间内运行；而分数原始-对偶算法虽然表述简单，实现起来却并不容易。

缓存及其众多变体一直是理论计算机科学中研究最为广泛的问题之一。长期以来，它一直是检验新型算法与分析技术的试验场，人们也通过势函数分析、原始对偶算法，甚至学习增强型算法等通用技术对其进行了研究。对于经典的无权缓存问题，一个尤为简单的算法——随机标记 [9], ——已知能够达到最优竞争比（相差一个常数因子）。在任意时刻，随机标记算法都会将缓存中的页面划分为已标记和未标记 页面；发生缓存未命中时，它会从未标记页面中均匀随机地淘汰一个页面。缓存命中以及被载入缓存的页面都会被标记。当发生缓存未命中但已没有未标记页面时，就会开始一个新的阶段 ，并将缓存中的所有页面都设为未标记。在本文中，我们以此算法为基础，对其进行调整，使之适用于带预留的缓存。

我们的贡献

我们研究 Ibrahimpur 等人 [10] 提出的在线情境下的带预留的缓存模型，并改进了他们的结果。我们的第一个主要结果是一种更简单的分数算法，它是经典缓存中随机标记算法的推广。

▶ **定理 1.** 存在一种用于在线带预留的缓存的标记算法，其竞争比为 $O(\log k)$。即使允许最优离线算法在缓存中存放分数页，该竞争保证仍然成立。

我们指出，定理 1 中的算法及其分析比经典随机标记算法更为复杂。其中一个难点是，由于存在预留约束，针对带预留的缓存问题的标记式算法不能随意淘汰任意未标记页面。另一个关键难点在于，在我们的设定中，甚至连阶段的概念都不容易定义。具体而言，与经典缓存不同，可能出现这样的情况：缓存中仍有未标记页面，但由于预留约束，没有任何页面可以被淘汰以腾出空间来存放新页面。因此，我们需要制定一条规则，将那些因预留约束而导致算法无法干净地结束一个阶段的代理孤立出来，同时确保此类孤立代理已标记的页面不会过早被清除。为此，我们引入全局阶段和局部阶段的概念，以有效地刻画每个代理的状态。我们将在第 3.1 节对此作进一步阐述。

我们对分数标记算法的分析引入了两个新颖的组成部分，它们本身也可能具有独立的研究价值。首先，我们利用一个新的势函数对分数标记算法产生的总代价给出上界。该势函数在第 3.3 节中介绍，只取决于算法的决策，而与最优解无关。据我们所知，此前所有基于势函数、针对缓存 [4, 5, 6, 10] （及其变体）的分析，所定义的势函数都依赖于最优解。其次，我们通过对偶拟合方法，为最优解的代价引入了一个新的下界。我们的方法还给出了一种新的简洁证明，证明经典随机标记随机标记 [9] 对于无权分页是 $O(\log k)$-竞争的（见附录 A）。

我们还设计了一种新的在线舍入算法，可将确定性分数算法转换为随机化整数算法，同时仅使竞争比损失一个常数因子。通过一种经过仔细设计的离散化技术（受 Adamaszek 等人[2]的启发），这种新的舍入算法可在多项式时间内运行，并且仅使用有限的随机化。我们的分数标记算法（算法 1）保持如下性质：在任意时刻，特定页面 p 要么完全位于缓存中，要么该页面至少有 $1/k$ 的部分已被驱逐。我们利用这一关键性质证明，在任意时刻t ，都可以对分数解进行离散化，使得任意页面被驱逐的部分都是 $1/k^3$的整数倍。这种离散化使我们能够维护一个关于可行整数缓存状态的分布，且该分布的支撑大小有界。

▶ **定理 2.** 对于带预留的在线缓存，存在一个多项式时间的 $O(\log k)$竞争比随机化整数算法。

其他相关工作

无权缓存（也称为分页）问题已得到广泛研究，其最优竞争比即使精确到常数因子也已广为人知。已知存在紧的算法[1, 13] ，其竞争比恰为 H_k，其中 H_k 是第 k个调和数。近期，Agrawal 等人 [3] 考虑了并行分页 模型，其中多个代理 p 共享一个缓存——每个代理都会收到一个页面请求序列，而算法必须随时决定如何在代理之间划分缓存。该模型允许p 处理器同时推进，即并发地产生缓存命中和未命中。多代理分页也已在系统领域 [7, 15, 16]得到广泛研究，通常是在多核系统缓存的背景下。与带预留的缓存 设置密切相关的是，受多代理环境中公平性约束的驱动，近期有若干系统 [11, 12, 14, 17] 旨在为每位用户提供隔离保证 ，也就是说，保证每位用户的缓存命中率至少不低于为其分配独立缓存时的命中率。同样受公平性约束的驱动，Chiplunkar 等人 [8]考虑了最小最大分页问题，其目标是最小化任一代理产生的最大页面错误数。

2 预备知识与符号说明

形式化地说，带预留的缓存问题的一个实例由以下内容组成。给定代理数量 m 和总（整数）缓存容量 k。令 $[m]$ 表示集合 $\{1, \ldots, m\}$。每个代理 $i \in [m]$ 拥有一组页面 $\mathcal{P}(i)$（称为 i-页面），并具有一个预留缓存大小 $k_i \geq 0$。每个页面都有唯一的所有者，即对所有 $i \neq j$, $\mathcal{P}(i) \cap \mathcal{P}(j) = \varnothing$；我们用 $\mathcal{P} \triangleq \cup_{i \in [m]} \mathcal{P}(i)$表示所有页面的全集。对于任意页面 $p \in \mathcal{P}$，令 $ag(p)$表示拥有页面 p的唯一代理。我们不失一般性地假设至少

有一个单位的缓存未被预留：$\sum_{i\in[m]} k_i < k$.[1] 在每个时间步 t，都会请求一个页面 $p_t \in \mathcal{P}$。我们可以将以上所有内容整合为一个实例元组：$\sigma = (m, k, \{\mathcal{P}(i)\}, \{k_i\}, \{p_t\})$。

带预留的整数算法会维护缓存中的一组 k 页面，使得对于每个代理 i，缓存始终至少包含来自$\mathcal{P}(i)$的 k_i 个页面。在时间 t，页面请求 p_t 会被揭示给算法。如果该页面当前不在缓存中，则称算法发生了一次缓存未命中，并且必须通过可能驱逐另一个页面 q_t，将 p_t 加载到缓存中。对于任意（整数）算法$\mathcal{A}$，我们将其在实例 σ 上的缓存未命中总次数记为代价$_{\mathcal{A}}(\sigma)$。

带预留的分数算法会维护一个分数 $x_p \in [0,1]$，表示每个页面 $p \in$ 在$\mathcal{P}$ 中的缓存占比，使得缓存中页面的总大小至多为 k，即$\sum_{p\in\mathcal{P}} x_p \le k$，并且 i页面的总大小至少为 k_i，即$\sum_{p\in\mathcal{P}(i)} x_p \ge k_i$。在时间 t，页面请求 p_t 会被揭示给算法，算法因此产生大小为 $1 - x_{p_t}$的分数缓存未命中。随后，算法必须将 p_t 完全加载到缓存中（$x_{p_t} \leftarrow 1$），并可能驱逐其他页面。对于任意（分数）算法$\mathcal{A}$，我们再次将其在实例 σ 上的缓存未命中总大小记为代价$_{\mathcal{A}}(\sigma)$。

令 cost$_{OPT}(\sigma)$ 表示在实例 σ 上最优离线算法的代价。

▶ **定义3** (竞争比)。对于带预留的缓存，若在线算法 A 满足：对任意实例 σ，带预留的缓存 c-竞争，即 $+b$，其中 b 是一个与 σ 中页面请求数量无关的常数。期望值是对算法所做的所有随机选择（如果有）取的。

3 带预留缓存的分数型 O(log k)-竞争算法

对于任意时刻 t 和页面 $p \in \mathcal{P}$，算法维护变量 $y_p^t \in [0,1]$，表示页面 p 位于缓存之外的部分。于是，$x_p^t \triangleq 1 - y_p^t$ 表示 p 位于缓存中的部分。算法 1 确保任意时刻 t 的可行性：所有 y 值的总和恰好等于互补缓存大小 $|\mathcal{P}| - k$，即 $\sum_{p\in\mathcal{P}} y_p^t = |\mathcal{P}| - k$；并且，任一代理 i 的页面对应的 y 值总和不超过其相应的互补保留大小，$\sum_{p\in\mathcal{P}(i)} y_p^t \le |\mathcal{P}(i)| - k_i$。在时刻 t 收到页面 p_t 的请求时，算法通过支付 $y_{p_t}^t$ 的提取成本，将 p_t 完整提取到缓存中，同时驱逐总量为 $y_{p_t}^t$ 的其他适当选定页面。

3.1 分数算法

完整算法（在证明中称为算法 $\mathcal{A}$）见算法 1。这里我们先作高层次的讨论。在任意时刻 t，如果 i 代理紧的，则称其为紧的；当 $\sum_{p\in\mathcal{P}(i)} x_p^t = k_i$ 时，算法不得进一步驱逐该代理 i的任何页面（即使是分数式地驱逐也不允许）。相反，如果 $\sum_{p\in\mathcal{P}(i)} x_p^t > k_i$，则称代理 i 为非紧的。

该算法是一种分数标记算法，按阶段运行：每个阶段对应于一段最长的页面请求序列，在保持可行性且确保不会驱逐任何“已标记”页面的情况下，可以处理该序列。在每个阶段内，通过持续驱逐具有最小 y_q 值的“可用”（定义如下）未标记页面的无限小部分，将当前请求的页面 p_t 完全载入缓存；如果 q有多个选择，则以相同速率同时驱逐所有这些页面。页面

[1] 如果缓存全部被预留，问题就会按代理分解为标准缓存任务。

p_t 在处理完毕后会被标记，并且该标记只能在阶段结束时擦除。

在一个阶段结束时，如果此时缓存中标记的页面严格少于 k_i 个，则代理 i 被指定为孤立。一旦未来某个时刻有 k_i 个页面被标记，该指定就会立即变为非孤立。孤立代理本质上是在其自身拥有的页面和自己的预留空间中运行一个独立的缓存实例。在一个阶段结束时，非孤立代理所拥有页面的标记（即至少有 k_i 个已标记i页面的代理）将被清除。

下面说明页面 q 何时可被驱逐。显然，必须满足 $y_q^t < 1$，否则页面 q 已完全位于缓存之外。此外，$ag(q)$ 必须是非紧的，也就是说，驱逐页面 q 不得违反拥有该页面的代理的预留约束。将 q 视为可驱逐页面的最后一个条件，取决于代理 $i_t := ag(p_t)$ 是否孤立：(i) 如果代理 i_t 是孤立的，那么应满足 $ag(q) = i_t$，也就是说，只有未标记的 i_t 页面才可被驱逐；(ii) 如果代理 i_t 不是孤立的，那么 $ag(q)$ 也应是非孤立的。我们再次说明，在所有可驱逐页面中，首先驱逐 y_q^t 值最小的页面。

记号 令 $\mathbf{I}(t) \subsetneq [m]$ 表示时刻 t处孤立代理的集合。对于全局阶段 r_0，我们用 $\mathbf{I}(r_0)$ 表示阶段 r_0结束时孤立代理的集合。令 $\mathbf{T}(t)$ 表示时刻 t处紧约束代理的集合。在任意时刻 t，令 r_i^t 表示代理 i的局部阶段计数器 r_i 的值，并令 $R_i = r_i^T$表示代理 i的局部阶段总数。根据定义，对于任意代理 $i \in [m]$，$P(i, r_i - 1)$ 和 $P(i, r_i)$ 分别表示代理 i第 r_i个局部阶段开始时和第 r_i个局部阶段结束时缓存中（以整数形式）的 i-页面集合。对于任意代理 $i \in [m]$，令 $M(i,t) \subseteq \mathbf{P}(i)$ 表示缓存中带标记的i-页面集合，并令 $U(i,t) = P(i, r_i^t - 1) \setminus M(i,t)$ 表示未标记的 i-页面集合。

我们强调，只有在针对某些 i, t 下 $P(i, r_i^t - 1)$ 中的页面时，未标记页面这一概念才有意义；特别是，每个 i-页面 $q \in \mathbf{P}(i) \setminus (P(i, r_i^t - 1) \cup M(i,t))$ 都未被标记，但我们不会称其为未标记页面。类似于随机标记算法 [9], 所使用的干净页和过时页面概念，我们定义干净页、伪干净页面和过时页面如下。固定一个代理 i，设 r_i 为时刻 t 该代理的局部阶段计数器。任何 i-页面 $q \in P(i, r_i - 1)$ 都被视为过时页面。当前请求的页面 p_t 若满足 $p_t \notin P(i, r_i - 1)$，则称其为干净页。接下来，若当前请求的页面 p_t 满足 $p_t \in P(i, r_i - 1)$，且在算法 A 开始将 p_t 取入缓存之前 $y_{p_t}^t = 1$ 成立，则称其为伪干净页面。引理 7 和 8 表明，伪干净页面必然属于这样一个代理：它在（全局）阶段 r_0 开始时是孤立的，但在时刻 t 是非孤立的。为简化记号，只要时间索引可由上下文明确，我们就省略所有记号中的上标 t。

下面的引理汇总了算法执行过程中始终维持的一系列关键不变量；这些不变量可直接通过考察算法 1 得出。

▶**引理 4.** 算法 *1* 维持以下不变量。

(i) 新阶段开始时，所有已标记页面都属于孤立代理。**(ii)** 在任何时刻 t，所有孤立代理都是紧的。**(iii)** 在任何时刻 t 且对于任意代理 i，代理 i 的所有未标记页面都具有相同的 y值。

[2] 代理 i_t 在此处被视为非紧的，因为在逐出其他 $q \in \mathcal{P}(i_t) \setminus p_t$ 的同时获取 p_t 不会违反储备可行性。

算法 1 用于带预留的缓存的分数标记算法

```
1  /* Initialization */
2  r_0 ← 1                                                    /* global phase counter */
3  r_i ← 1, ∀i ∈ [m]                                          /* local phase counters */
4  令 P(i, 0) ⊂ P(i) 为初始缓存中的 i-页面集合（假设 |P(i, 0)| ≥ k_i） ∀i ∈ [m]
5  所有代理 i ∈ [m] 均为非孤立，且缓存中的所有页面 p 均为未标记
6  对于代理 i_t 的每个页面请求 p_t 执行
7      如果 y_{p_t} = 0，即 x_{p_t} = 1，那么
8          标记页面 p_t 并处理请求。
9      否则如果 代理 i_t 是孤立的，则
10         /* Continuously fetch page p_t while uniformly evicting all unmarked
               i_t-pages.  */
11         设置 y_{p_t} ← 0，标记 页面 p_t 并处理请求。
12         以相同速率增加缓存中所有未标记的 i_t-页面的 y_q，直到缓存
               变得可行，即 ∑_{p∈P} y_p ≥ |P| − k 成立。
13         如果代理 i_t 现在拥有 k_i 个已标记页面则
14             将 i_t 指定为非孤立
15     否则，如果 ∃ 页面 q 由某个非紧代理[2]拥有且满足 y_q < 1，则
16         /* Continuously fetch page p_t while uniformly evicting all unmarked
               pages (belonging to any non-tight agent) with the least y-value.  */
17         设置 y_{p_t} ← 0，标记页面 p_t 并处理请求。
18         以相同速率增加非紧代理的所有未标记页面的 y_q，其
               最小的 y 值，直到缓存变为可行，即 ∑_{p∈P} y_p ≥ |P| − k 成立。
19     else
20         /* End of phase */
21         对于每个代理 i ∈ [m] 执行
22             如果 i 的已标记页面严格少于 k_i 个，那么
23                 将 i 指定为孤立。
24             else
25                 /* i is non-isolated and undergoes a phase reset */
26                 将 P(i, r_i) ← 设为缓存中所有（整数且已标记的）i-页面的集合。
27                 集合 r_i ← r_i + 1
28                 所有已标记的 i 页面现在都已取消标记
29         集合 r_0 ← r_0 + 1
30         在新的阶段重新处理当前页面请求 p_t
```

（iv）属于孤立代理的任何页面仅在同一代理的另一个页面被请求的时间步中才会被（分数式地）驱逐。

以下引理表明，该算法定义良好，并且算法 1 第 12 行和第 18 行中的操作始终可行。

▶ **引理5**。如果请求的页面 p_t 具有 $x^t_{p_t} \in [0,1)$，且代理 i_t 是孤立的，那么通过淘汰代理 i_t的未标记页面，可以完整获取 p_t。

证明。由于请求页面 p_t 时代理 i_t 处于孤立状态，$\sum_{p\in\mathcal{P}(i)} x^t_p = k_i$（根据引理 4 中的不变量 (ii)），且其缓存中已标记页面少于 k_i 个。因此，$\sum_{p\in U(i,\ t)} x^t_p \geq 1$ 和 $\sum_{p\in U(i,\ t)\setminus\{p_t\}} x^t_p \geq 1 - x^t_{p_t}$。◀

▶ **引理 6.** 如果请求的页面 p_t 满足 $0 < x^t_{p_t} < 1$，且其所有者 i_t 是非孤立的，那么始终存在足够的属于非紧代理的页面分数质量，可以被

被驱逐以完整获取页面 p_{t}。特别地，算法 *1* 的第 *18* 行定义良好。

证明。假设页面 p_t 以连续方式被获取。要证明页面 p_t 可以被完整获取，只需证明在任意瞬时 t，当 $x^t_{p_t} < 1$时，始终存在一个属于非紧代理 i 的未标记页面 q，使得 $x^t_q > 0$，即页面q可以被驱逐。注意到$k = \sum_{q\in \mathrm{P}} x^t_q = \sum_{i\in \mathrm{T}\,(t)} k_i + \sum_{i\notin \mathrm{T}\,(t)} \sum_{q\in \mathrm{P}(i)} x^t_q$。由于 k 和 $\{k_i\}^{i\in[m]}$的整数性，我们必有 $\sum_{i\notin \mathrm{T}\,(t)} \sum_{q\in \mathrm{P}(i)} x^t_q$为整数。由于任何已标记页面 q始终满足 $x^t_q = 1$，因此 $\mu := \sum_{i\notin \mathrm{T}\,(t)} \sum_{q\in U(i,t)} x^t_q$也为整数。此外，由于 $i_t \notin \mathrm{T}\,(t)$ 且 $x^t_{p_t} > 0$，我们必有 $\mu \geq 1$，因此存在一个属于某个非紧代理 i 的页面 q，满足 $x^t_q > 0$，正如所需。◀

由于我们总是驱逐 y 值最小的可用页面，因此在任意时间步 t，所有可用页面（即属于非紧代理且满足 $y_q < 1$ 的未标记页面 q）始终具有相同的 y 值。我们将这个共同的 y 值记为 h^*，并将相应的可驱逐页面集合（其 y值为 h^*）称为前沿。接下来的两个关键结构引理将这一性质形式化。

▶**引理7**。在任意时间 t，设 i 是一个在当前阶段开始时处于孤立状态的代理，且 q 是它的一个未标记页面。那么，$y^t_q < 1$ 当且仅当 i 在时间 t 仍处于孤立状态。

证明。对于若 i 仍然是孤立的情况。根据不变量 (ii)，它也是紧的。根据不变量 (iii)，它所有未标记页面的 x值都相同，且总共占据$k_i - |M(i,\ t)| > 0$ 单位的缓存空间。因此，$x^t_q > 0$ 且 $y^t_q < 1$。

对于 仅当 方向，假设 i 不再是孤立的。在第 k_i个 i-页面被请求并标记之前，已经标记了 $(k_i - 1)$ i-页面。由于 i 是紧的，其所有未标记页面的 x值之和必定为 1。随后，算法用第 k_i个已标记页面替换了所有这些页面，而所有剩余未标记页面的 x值变为 0。因此，对于一个新变为非孤立的代理 i，该性质仍然成立。在该阶段的剩余时间内，这一性质持续成立，因为对于未标记页面，y_q 永远不会减小。◀

▶ **引理 8.** 在任意时间 t，都存在一个值 $h^*_t \in [0, 1]$，使得：对于当前阶段开始时处于非孤立状态的任意代理 i，以及任意未标记的 i-页面 q，$y_q \leq h^*_t$ 成立；并且当 i 不是紧的时，$y_q = h^*_t$ 成立。

证明。我们通过归纳法证明该引理。显然，在阶段开始时引理成立：属于非孤立代理的所有未标记页面的 y值均为 0。现在考虑阶段 r 中的某个时间 t，并假设该引理对本阶段中 t 之前的所有时间点均成立。我们还可以假设 $y_{p_t} > 0$，否则此时间步中不会修改任何变量。

根据归纳假设，任意未标记的 i-页面 q 都满足 $y_q \leq h^*_{t-1}$，并且当且仅当 i 是非紧约束时，该不等式取等号。如果代理 i_t 是非紧约束的，那么它的未标记页面已经属于前沿。否则，算法 $\mathcal{A}$ 通过提高其他未标记 i_t-页面的y值，将 p_t 完整载入缓存，直到发生以下情况之一：(a) p_t 被完整载入。此时，i_t 仍保持紧约束；或者 (b) 未标记 i_t-页面的 y值变得等于前沿的 y值，即 h^*_{t-1}。在后一种情况下，未标记的 i_t-页面成为前沿的一部分，并且前沿的 y值统一增加，直到 p_t 被完整载入缓存。如果某个代理 i' 在载入操作完成之前变为紧约束，那么它的未标记页面将从前沿中排除，且相应的 y值在此时间步剩余期间保持不变。在所有情况下，引理仍然成立，因为前沿的 y值从未降低，并且只有紧约束代理才会从前沿中移除。

◀

▶ 注释 9. 在任意阶段内，h_t^* 随时间单调不减，并且在端点处取值为 0 和 1。这是因为对于未标记页面，$\mathcal{A}$ 始终不会减少 y_q $q \neq p_t$。

以下引理表明，任何未完全包含在算法 $\mathcal{A}$ 的缓存中的页面，都必须至少有 $1/k$ 部分被驱逐。该性质将在第 3.3 节和第 4 节中发挥作用。

▶**引理10**。 在任意时间步 t结束时，对于任意页面 $p \in P$，我们有 $y_p^t = 0$ 或$y_p^t \geq 1/k$。

证明。首先，注意对于所有已标记页面，我们有 $y_p^t = 1 - x_p^t = 0$。令 $i \in \mathrm{T}(t)$ 为任意一个紧约束代理。则我们有 $k_i = \sum_{p \in \mathrm{P}(i)} x_p^t = |M(i,t)| + \sum_{p \in U(i,t)} x_p^t$。根据引理 4（第 iii 部分），代理 i的所有未标记页面都具有相同的 y值： $y_p^t = 1 - x_p^t = h_i$（记为）。由于 k_i是整数，我们有 $h_i = 0$或 h_i。重新整理后，我们得到 $|U(i,t)|h_i = |M(i,t)| + |U(i,t)| - k_i$。由于右侧的所有项都是整数，因此我们有 $h_i = 0$或$h_i \geq 1/|U(i,t)| \geq 1/k$。

根据引理 8，属于非紧代理的所有未标记页面 p 都满足 $y_p^t = h_t^*$。令$U(t)$表示属于所有非紧代理、且在阶段开始时同样未处于孤立状态的所有未标记页面的集合。回顾定义可知，我们有 $|U(t)| \leq k$，因为 $U(t)$中的所有页面在当前阶段开始时都必定已完全位于缓存中。由于我们有 $k = \sum_{i \in \mathrm{T}(t)} k_i + \sum_{i \notin \mathrm{T}(t)} \sum_{p \in \mathrm{P}(i)} x_p^t$，再次根据 k和 $\{k_i\}$的整数性，我们必有 $\sum_{p \in U(t)} y_p^t = \sum_{p \in U(t)} h_t^*$为整数。因此，或者 $h_t^* = 0$，或者$h_t^* \geq 1/|U(t)| \geq 1/k$。◀

3.2 分析概述

在任意时刻 t，我们将 $\bigcup_{i \in [m]} P(i, r_i - 1)$ 中各页面的 y 值集合视为系统的 状态 。我们定义一个非负势函数 Ψ，它完全是该状态的函数。对于每个页面请求 p_t，我们尝试用势函数的增量来界定算法的代价，从而将算法产生的总代价界定为势函数的最终值。此方法有两个难点：(i) 阶段结束时，势函数会骤降，因为所有被完全驱逐的未标记页面不再计入状态；(ii) 当代理 i_t 在阶段开始时处于孤立状态、但现在变为非孤立状态时，势能变化不足以弥补获取代价。在这两种情况下，我们都将在线算法产生的代价计入一个新量，该量是集合 $\{P(i, r_i)\}$ 的函数。为完成分析，我们证明该量不超过最优解的代价。

3.3 势函数分析

考虑函数 ϕ： $[0,1] \to \mathbb{R}_{\geq 0}$，其定义如下：

$$\phi(h) \triangleq 2h \cdot \ln(1 + kh) \tag{1}$$

当 h 从 0 增至 1 时， $\phi(h)$从 0 增至 $2\ln(1+k)$。

任意时刻 t 的势能定义如下：

$$\Psi(t) \triangleq \sum_{i=1}^{m} \sum_{p \in U(i,t)} \phi(y_p^t) \tag{2}$$

请注意，任何时刻只有未标记页面 t 会对势能产生贡献。因此，当页面p_t 在时间 t 被调取并标记后，它便不再对势能产生贡献。但由于 ϕ 是

单调的，新被驱逐的页面会增加其对势能的贡献。我们指出，势能完全是系统状态的函数，其状态由缓存中未标记页面的 y 值定义，因此始终受一个与页面请求序列长度无关的量所界定。

▶ **引理 11**。 对于任意 $h \geq 1/k$，有 $\phi(h) \geq h$ 以及 $\phi'(h) \geq 1 + 2\ln(1+kh)$。

证明。第一个结论由对数不等式 $\ln(1+x) \geq x/(1+x)$ 得出，该不等式对任意非负的 x 均成立：当 $kh \geq 1$ 时，我们有 $\phi(h) = 2h\ln(1+kh) \geq h \cdot 2kh/(1+kh) \geq h$。接下来，$\phi'(h) = \frac{d\phi}{dh} = 2(1 - 1/(1+kh) + \ln(1+kh))$。因此，对于任意 $h \geq 1/k$，有 $1/2 \geq 1/(1+kh)$，从而得到另一个结论。◀

本节其余部分将用于证明以下定理，其中我们根据集合 $\{P(i, r_i)\}$ 以及对伪干净页面的请求次数来界定算法产生的总代价。

▶**定理 12**。 A 处理前 T 个页面请求所产生的代价满足以下界：

$$\text{cost}_{\mathcal{A}}(\sigma) \leq 2\ln(1+k) \cdot \left(mk + \sum_{t=1}^{T} \mathbb{1}_{p_t \text{ is pseudo-clean}} + \sum_{i \in [m]} \sum_{r_i=1}^{R_i} |P(i, r_i - 1) \setminus P(i, r_i)| \right).$$

回想一下，算法在时间 t 获取页面 p_t 会产生代价 $y^t_{p_t}$。因此，算法产生的总代价就是 $\text{cost}_{\mathcal{A}}(\sigma) = \sum_t y^t_{p_t}$。我们首先对请求的页面 p_t 至少部分位于缓存中的时间步（即 $y^t_{p_t} < 1$）所产生的代价进行界定。根据引理 5 和 6，算法在此时间步不会经历阶段转换。

▶**引理 13.** 考虑任意一个时间步 t，使得 $y^t_{p_t} < 1$ 对于当前请求的（未标记）页面 p_t成立。令 $\Delta\Psi(t)$ 表示时间步t期间势函数的变化。那么 $y^t_{p_t} \leq \Delta\Psi(t)$。

证明。 我们假设 $y^t_{p_t} \geq \frac{1}{k}$，因为否则根据引理10，必有 $y^t_{p_t} = 0$，于是引理显然成立。由于 $y^t_{p_t} < 1$，根据引理8，代理 i_t 要么是紧约束代理，要么有$y^t_q = y^t_{p_t}$ 对每个未标记页面 q，该页面由本阶段开始时非孤立的任一非紧代理 i 持有。无论哪种情况，为腾出空间而被驱逐的页面，其初始 p_t值都 y 至少为 $y^t_{p_t} \geq 1/k$。势函数 Ψ 在这一步中因两个因素而变化：(i) Ψ 随着页面 p_t 一经标记便不再对势能有贡献而下降；(ii) Ψ 这一步中，被（分数式地）驱逐页面的 y值上升，因而使其增加。

设 $h \triangleq y^t_{p_t}$。在时间 t 开始时，页面 p_t 对势能的贡献恰为 $\phi(h) = 2h\ln(1+kh)$；一旦 p_t 被标记，这部分贡献就会消失。为了证明该引理，只需证明，在驱逐属于非紧代理的未标记页面的 h 数量期间，势函数（不计入 p_t 的贡献）的增长速率始终至少为 $1+2\ln(1+kh)$：总计的 1 项支付 h 的提取成本，而总计的 $2\ln(1+kh)$ 项支付势能损失 $2h\ln(1+kh)$。由引理 11 可知，这一点直接源于本时间步中被分数驱逐的页面的 y 值此前已至少为 $h \geq 1/k$。这里还用到了函数 $h' \mapsto \ln(1+kh')$ 的单调性。◀

我们仍需对算法在到达请求所对应的页面完全不在缓存中时产生的代价进行上界估计。注意，此时算法产生的代价恰好为单位代价。

所有这些时间步的代价。下面的引理表明，算法产生的总代价可以由阶段结束时势函数的下降量，以及仅取决于处理对完全位于缓存之外的页面的请求时势函数变化量的一项来界定。

▶**引理 14**。 对于任意全局阶段 r_0，令 $\Delta\Psi(r_0)$ 表示阶段 r_0 结束时势函数的变化量（算法 *1* 中的第 *30* 行）。令 R_0 表示全局阶段的总数， T 表示阶段 R_0结束时的时间。那么，对于 *A* 处理前 T 个页面请求所产生的代价，我们有如下上界：

$$\text{cost}_{\mathcal{A}}(\sigma) \leq 2mk\ln(1+k) + \sum_{t\in[T]:y^t_{p_t}=1}(1-\Delta\Psi(t)) - \sum_{r_0=1}^{R_0}\Delta\Psi(r_0)$$

证明。 我们有：

$$\begin{aligned}\text{cost}_{\mathcal{A}}(\sigma) &= \sum_{t\in[T]} y^t_{p_t} = \sum_{t\in[T]:y^t_{p_t}<1} y^t_{p_t} + |\{t\in[T] : y^t_{p_t}=1\}| \\ &\leq \sum_{t:y^t_{p_t}<1}\Delta\Psi(t) + |\{t : y^t_{p_t}=1\}| \qquad \text{(Using Lemma 13)} \\ &= \Psi(T) - \Psi(0) - \sum_{t:y^t_{p_t}=1}\Delta\Psi(t) - \sum_{r_0=1}^{R_0}\Delta\Psi(r_0) + |\{t : y^t_{p_t}=1\}|.\end{aligned}$$

引理由 $\Psi(T) \leq 2mk\ln(1+k)$ 且 $\Psi(0) = 0$ 得出。 $\Psi(T)$ 的界是因为我们有 m 个代理，每个代理有 $|P(i, r_i - 1)| \leq k$，且 $\phi(1) = 2\ln(1+k)$。◀

因此，只需对请求的页面完全位于缓存之外的时间步，界定总代价和势能下降量，并对阶段发生变化时的势能下降量进行界定即可。

定理 12 的证明。 考虑任意一个时间步 t，此时当前请求的页面 p_t 完全位于缓存之外，即 $y^t_{p_t} = 1$。根据页面 p_t 在当时是否属于集合 $P(i_t, r_{i_t} - 1)$，我们将此类请求分为两种情况。换言之，我们对 p_t 是干净还是伪干净进行分类讨论。（回想一下，只有$P(i_t, r_{i_t} - 1)$ 中的未标记页面才会计入势能）。

情况 *1*： $p_t \notin P(i_t, r_{i_t} - 1)$，即 p_t 是干净的。 由于页面 $p_t \notin U(i, t)$，因此在请求得到处理之前（或之后），它都不会计入势能。考虑此时间步中被算法（分数式地）驱逐的任意页面 q。根据引理 4，驱逐之前，我们有 $y_q = 0$ 或 $y_q \geq 1/k$。在任一情况下，根据引理 11，我们有 $\Delta\phi(y_q) \geq \Delta y_q$，其中 Δy_q表示此时间步中页面 q 的 y值的变化。由于我们有 $\sum_q \Delta y_q = y^t_{p_t} = 1$，因此有 $\Delta\Psi(t) = \sum_q \Delta\phi(y_q) \geq 1$。

情况 *2*： $p_t \in P(i_t, r_{i_t} - 1)$，即 p_t 是伪干净的。 根据上述相同的推理，我们有 $\sum_{q\neq p_t}\Delta\phi(y_q) \geq 1$。不过，在这种情况下，页面 p_t 在该时间步开始时也恰好为势能贡献了 $2\ln(1+k)$。因此，我们有 $\Delta\Psi(t) = \sum_{q\neq p_t}\Delta\phi(y_q) - 2\ln(1+k) \geq 1 - 2\ln(1+k)$。

合并这两种情况可得：

$$\sum_{t:y^t_{p_t}=1}(1-\Delta\Psi(t)) \leq 2\ln(1+k)\cdot|\{t : y_{p_t}=1 \text{ and } p_t \in P(i_t, r_{i_t}-1)\}| \tag{3}$$

考虑某个阶段 r_0 结束时的情形，并设 i 是一个非隔离代理。令 r_i 表示代理 i 当前的局部阶段，该阶段也必须与全局阶段 r_0同时结束。考虑 $U(i,t)$ 中任意一个未标记页面 q。由于阶段 r_0 即将结束，页面 q 必须被完全驱逐，因此会使势能增加 $\phi(1)$。阶段 r_0 结束且阶段 r_0+1 开始后，页面 q不再对势能有贡献。注意，这些未标记页面的集合恰好是 $P(i,r_i-1)\setminus P(i,r_i)$。因此，在（全局）阶段 r_0 结束时，势能的变化为：

$$\Delta\Psi(r_0) = -2\ln(1+k)\cdot \sum_{i\notin\mathcal{I}(r_0)} |P(i,r_i-1)\setminus P(i,r_i)|$$

由于代理只有在全局阶段结束时处于非孤立状态，才会更改其局部阶段，因此我们有：

$$\sum_{r_0=1}^{R_0}\Delta\Psi(r_0) = -2\ln(1+k)\cdot \sum_{i\in[m]}\sum_{r_i=1}^{R_i} |P(i,r_i-1)\setminus P(i,r_i)|. \tag{4}$$

定理现在由引理 14 得出。 ◀

3.4 通过对偶拟合得到最优值的下界

在本节中，我们通过对偶拟合，给出一个新颖的基于线性规划的下界，用于衡量任何带预留缓存离线算法的代价。即使在经典的无权分页场景中，这种下界分析也是全新的。关键的是，这里推导出的下界与定理 12 中用于界定分数算法$\mathcal{A}$ 代价的两项完全吻合，从而完成了主要结果（定理 1）的证明。

现在我们描述带预留缓存问题的线性松弛及其对偶规划。以下记号将会用到。对于任意页面 $q\in$P，令 $t_{q,1}<t_{q,2}<\cdots$表示在线序列中请求 q 的时间步。对于整数 $a\geq 0$，定义 $I(q,a)=\{t_{q,a}+1,\ldots,t_{q,a+1}-1\}$ 为对 q 的第 a 次和第 $(a+1)$ 次请求之间的时间区间。我们对所有页面定义 $t_{q,0}\triangleq 0$。令 $a(q,t)$ 表示截至时间 t （含该时刻）已出现的对页面 q 的请求次数。因此，根据定义，对于任意时间 t 和页面 $q\in$P$\setminus\{p_t\}$，有 $t\in I(q,a(q,t))$。原始线性规划的变量为 $y(q,a)\in[0,1]$，表示在页面 q 的第 a 次和第 $(a+1)$ 次请求之间被驱逐的部分；也就是说，在时间区间 $I(q,a)$ 内，页面 q 的 $1-y(q,a)$ 部分留在缓存中。为方便起见，对于任意 $i\in[m]$，我们定义 $n\triangleq|$P$|$ 和 $n_i\triangleq|$P$(i)|$。原始约束的第一组和第二组分别编码了所有时刻的缓存大小约束和代理级预留约束。对偶线性规划的变量为 $\alpha(t)$ 和 $\beta(t,i)$，分别对应这些原始约束。此外，我们还有对偶变量 $\gamma(q,a)$，对应编码 $y(q,a)\leq 1$ 的原始约束。除非负性外，对偶规划对每个区间 $I(q,a)$ 只有一个约束。原始线性规划和对偶线性规划如下所示。我们强调，这些线性规划仅用于分析，算法本身无需求解任何线性规划。

Primal LP

$$\min \quad \sum_{q\in\mathcal{P}}\sum_{a\geq 1} y(q,a)$$

subject to:

$$\sum_{q\in\mathcal{P},q\neq p_t} y(q,a(q,t)) \geq n-k \qquad \forall t \quad (5)$$

$$\sum_{q\in\mathcal{P}(i),q\neq p_t} y(q,a(q,t)) \leq n_i-k_i \qquad \forall t,\forall i \quad (6)$$

$$y(q,a) \leq 1 \qquad \forall q,\forall a \quad (7)$$

$$y \geq 0 \quad (8)$$

Dual LP

$$\max \sum_t (n-k)\alpha(t) - \sum_{t,i}(n_i-k_i)\beta(t,i) - \sum_{q,a}\gamma(q,a)$$

subject to:

$$\sum_{t\in I(q,a)} \big(\alpha(t)-\beta(t,ag(q))\big) - \gamma(q,a) \leq 1 \ \forall q,\forall a \quad (9)$$

$$\alpha,\beta,\gamma \geq 0 \quad (10)$$

考虑标记全局阶段 R_0 结束的时间 T，其中 R_0 为某个整数。令 OPT $=\text{cost}_{\text{OPT}}(\sigma)$ 表示最优离线算法产生的总代价。根据弱线性规划对偶性，对于任意可行的对偶解，对偶线性规划的目标函数给出了 OPT 的下界。现在我们构造一个显式对偶解 $(\alpha,\ \beta,\ \gamma)$，其目标值大致等于算法所见干净页和伪干净页面的总数。相关记号和术语可参见第 3.1 节。我们在每个（全局）阶段结束时分两个阶段更新对偶解。第一阶段的更新记为 update$(r_0,\ 1)$，较为简单，用于计入属于非孤立代理的过时页面；这些页面在该代理最近的局部阶段中被驱逐。第二阶段的更新记为 update$(r_0,\ 2)$，较为复杂，用于计入在当前阶段失去孤立状态的代理所拥有的伪干净页面。我们维护的对偶解始终近似满足可行性约束，误差因子为 $O(1)$，因此，该对偶解的目标值在常数因子范围内构成 OPT(T) 的下界。需要指出的是，假设 T 标记阶段结束并不失一般性，因为这至多会使下界产生加性损失 $O(k)$。形式化地，我们证明如下结论。

▶**定理 15.** 令 T 表示全局阶段 R_0 结束的时间点，并令 $(R_i)^{i\in[m]}$表示相应的局部阶段计数器。令 (α,β,γ) 表示截至时间 T构造出的对偶解，即按以下顺序依次应用对偶更新所得的解：更新$(1,1)$、更新$(1,2)$、更新$(2,1)$、更新$(2,2)$.....、更新$(R_0,1)$以及更新$(R_0,2)$。我们有：

(a) 对偶解近似可行：对于任意 i 页 q 和整数 $a \geq 0$,$\sum_{t\in I(q,a)}(\alpha(t)-\beta(t,i))-\gamma(q,a) \leq 5$ 成立。*(b)* (α,β,γ) 的对偶目标值为：

$$\begin{aligned}\text{dual}(R_0) &\triangleq \sum_{t=1}^{T}(n-k)\alpha(t) - \sum_{t=1}^{T}\sum_{i\in[m]}(n_i-k_i)\beta(t,i) - \sum_{q\in\mathcal{P}}\sum_{a=1}^{a(q,T)}\gamma(q,a)\\ &= \sum_{i\in[m]}\sum_{r_i=1}^{R_i}|P(i,r_i-1)\setminus P(i,r_i)| + \sum_{r_0=1}^{R_0}\sum_{i\in\mathcal{I}(r_0-1)\setminus\mathcal{I}(r_0)}(|P(i,r_i-1)\cup P(i,r_i)|-k_i).\end{aligned}$$

我们首先展示定理 15 如何推出我们的分数算法 $\mathcal{A}$ 具有 $O(\log k)$-竞争性。

定理 1 的证明。在定理 12 中，我们证明了以下关于代价的上界：

由$\mathcal{A}$ 处理前 T 个页面请求所产生的：

$$\text{cost}_{\mathcal{A}}(\sigma) \leq 2\ln(1+k) \cdot \Big(mk + \sum_{t=1}^{T} \mathbb{1}_{p_t \text{ is pseudo-clean}} + \sum_{i \in [m]} \sum_{r_i=1}^{R_i} |P(i, r_i - 1) \setminus P(i, r_i)| \Big).$$

显然，上述成本表达式中的第二个非平凡项与 dual(R_0) 表达式中的第一项相匹配。现在考虑任意一个全局阶段 $r_0 \in \{1, \ldots, R_0\}$ 以及该阶段中的一个时间步 t 。根据定义，伪干净页面 p_t 必然是陈旧的，即 $p_t \in P(i_t, r_{i_t} - 1)$ 成立，并且代理 i_t 必须在阶段 r_0 开始时处于孤立状态，但到时间 t 时已变为非孤立状态。因此， $i_t \in \mathcal{I}(r_0 - 1) \setminus \mathcal{I}(r_0)$，且有以下结论：

$$|P(i, r_i - 1) \cup P(i, r_i)| - k_i \geq |P(i, r_i)| - k_i \geq |\{t \in \text{phase } r_0 : p_t \text{ is pseudo-clean}\}|.$$

在上述内容中，最后一个不等式成立，是因为在 $P(i, r_i)$ 中的所有页面中（按照它们被 $\mathcal{A}$ 标记的顺序），前 k_i 个页面不是伪干净页面。因此，$\mathcal{A}$ 的成本表达式中的第一个非平凡项可以用 dual(R_0) 中的第二项进行界定。

总的来说，我们已经证明 $\text{cost}_{\mathcal{A}}(\sigma) \leq 2\ln(1+k) \cdot \big(mk + \text{dual}(R_0)\big)$ 成立。由于对偶解是 $O(1)$-可行的，我们得到 $\mathcal{A}$ 是 $O(\log k)$-competitive。◀

现在，我们详细说明对偶更新的具体过程。初始时，所有对偶变量 $\{\alpha(t)\}$、$\{\beta(i,t)\}$、$\{\gamma(q,a)\}$（其中 $t \in [T], i \in [m], q \in \mathrm{P}$ $a \in [a(q,T)]$ ）均设为零。我们假设对偶更新按照定理 15 中给出的顺序执行。也就是说，$\{\text{update}(r_0, s)\}_{r_0 \in [R_0], s \in \{1,2\}}$ 中的更新按照 r_0 递增的顺序执行，并且在每个阶段内，首先执行第一阶段更新。稍微滥用一下符号，令 dual(r_0, s) 表示在截至更新 update(r_0, s) （含该更新）的所有更新均已执行后，对偶解的目标值，其中 $r_0 \in [R_0], s \in \{1, 2\}$。注意，$\text{dual}(R_0) = \text{dual}(R_0, 2)$。我们还定义 $\text{dual}(0, 1) = \text{dual}(0, 2) := 0$。在整个更新过程中，我们确保对偶目标值始终不会下降，即满足 $0 \leq \text{dual}(1, 1) \leq \text{dual}(1, 2) \leq \cdots \leq \text{dual}(R_0, 1) \leq \text{dual}(R_0, 2)$。我们指出，$\beta$ 变量可能会减少，而这种情况只会发生在第二阶段；然而，α 和 γ 变量绝不会减少。

在第 3.4 节中，我们描述更新的第一阶段，并证明对偶目标的增益对应定理 15(b) 中的第一项。在第 3.4 节中，我们描述更新的第二阶段，并证明对偶目标的增益对应定理 15(b) 中的第二项。最后，在第 3.4 节中，我们证明所维护的对偶解始终在常数因子范围内可行，从而完成定理 15 的证明。

对偶更新的第一阶段

固定一个阶段 $r_0 \in [R_0]$ ，并考虑在阶段 r_0结束时被指定为孤立的代理集合 $\mathcal{I}(r_0) \subsetneq [m]$ 。令 $C(r_0)$ 表示满足以下两个条件的时间步集合 t （在此阶段内）：(a) 在分数算法 $\mathcal{A}$ 中，紧接在请求 p_t 之前；以及 (b) 在时间 t时， i_t 并非孤立。定义 $\ell^{(r_0)} \triangleq |C(r_0)|$。不难看出，以下是 $\ell^{(r_0)}$ 的等价表达式。

$$\ell^{(r_0)} := \sum_{i \notin \mathcal{I}(r_0 - 1) \cup \mathcal{I}(r_0)} |P(i, r_i) \setminus P(i, r_i - 1)| + \sum_{i \in \mathcal{I}(r_0 - 1) \setminus \mathcal{I}(r_0)} (|P(i, r_i)| - k_i). \tag{11}$$

请注意，对于在阶段 r_0, $\ell^{(r_0)}$开始和结束时均为非孤立状态的代理， $\ell^{(r_0)}$会统计其所有干净页面。然而，对于在此阶段开始时处于孤立状态、但在结束时已不再孤立的代理， $\ell^{(r_0)}$ 仅统计干净页面和伪干净页面， nρnὺp

是在代理变为非孤立之后 才被请求。粗略地说，定义 $\ell^{(r_0)}$ 的动机源于这样一种直觉：离线算法在阶段 r_0 中处理页面请求时，平均而言应当承担 $\Omega(\ell^{(r_0)})$ 的代价。

update(r_0, 1) 的说明：对于每个时刻 $t \in C(r_0)$，我们分别执行以下更新。首先，将 $\alpha(t)$ 增加 $1/\ell^{(r_0)}$。接着，对于每个代理 $i \in \mathrm{I}(r_0)$，将 $\beta(t,i)$ 增加 $1/\ell^{(r_0)}$。最后，对于每个代理 $i \notin \mathrm{I}(r_0)$，对于每个 i-页面 $q \in \mathrm{P}(i) \setminus (P(i, r_i - 1) \cup P(i, r_i))$，将 $\gamma(q, a(q,t))$ 增加 $1/\ell^{(r_0)}$。

从更新的说明中可以清楚看出，在 update(r_0,1) 中被修改的 α 和 β 变量此前均为 0。不过，对于受影响的 γ 变量，并没有这样的保证。我们还要指出，在 update(r_0,1) 期间，同一个 $\gamma(q,a)$ 变量可能会增加多次；当存在多个具有相同 $a(q,t)$ 值的时刻 $t \in C(r_0)$ 时，就会发生这种情况。事实上，由于 $\gamma(q,a)$ 变量来源于区间 $I(q,a)$，而这些区间可能跨越多个阶段，因此同一个 γ 变量可能在不同的 update(r_0,1) 步骤中增加不同的 $1/\ell^{(r_0)}$ 数值。

为方便起见，我们用 $t \in$ phase r_0 简写阶段 r_0 中的所有时间点。以下结果对我们很有用。

▶ **引理 16**。 令 (α, β, γ) 表示应用 更新$(r_0, 1)$后立即得到的对偶解。我们有：***(a)*** $\sum_{t\in\text{阶段 } r_0} \alpha(t) = 1$；以及 ***(b)*** 对任意代理 $i \in I(r_0)$，$\sum_{t\in\text{阶段 } r_0} \beta(t,i) = 1$。

证明。 这直接来自我们对 $\ell^{(r_0)} = |C(r_0)|$ 的选择。 ◀

本节的关键技术结果是，由 update(r_0,1) 带来的对偶目标值增益，等于非孤立代理所拥有的、在其最近的局部阶段中未被请求的过时页面数量。为方便起见，我们用前缀 Δ 指代 update(r_0,1) 期间发生的变化。

▶ **引理 17.** 我们有 $\Delta\mathrm{dual}(r_0, 1) = \sum_{i\notin\mathcal{I}(r_0)} |P(i, r_i - 1) \setminus P(i, r_i)|$，其中 $\Delta\mathrm{dual}(r_0, 1) \triangleq \mathrm{dual}(r_0, 1) - \mathrm{dual}(r_0 - 1, 2)$ 表示执行更新$(r_0, 1)$后对偶目标的变化。

证明。 由于受影响的 $\alpha(t)$ 和 $\beta(t,i)$ 变量仅为满足 $t \in C(r_0)$ 的变量，且它们都恰好增加了 $1/|C(r_0)|$，因此我们得到：

$$\begin{aligned}
\Delta\mathrm{dual}(r_0, 1) &= \sum_t (n-k)\Delta\alpha(t) - \sum_{t,i} (n_i - k_i)\Delta\beta(t,i) - \sum_{q,a} \Delta\gamma(q,a) \\
&= (n-k) - \sum_{i\in\mathcal{I}(r_0)} (n_i - k_i) - \sum_{q,a} \Delta\gamma(q,a) \\
&= \Big(\sum_{i\notin\mathcal{I}(r_0)} n_i\Big) - k + \Big(\sum_{i\in\mathcal{I}(r_0)} k_i\Big) - \sum_{q,a} \Delta\gamma(q,a)
\end{aligned}$$

现在注意，对于每个 $t \in C(r_0)$ 和 $i \notin \mathrm{I}(r_0)$，update(r_0, 1) 步骤都会增加恰好对应 $n_i - |P(i, r_i - 1)$ 个唯一 i-页面的 $\gamma(q,a)$ 变量，每个变量增加 $1/\ell^{(r_0)}$。因此，我们有 $\sum_{q,a} \Delta\gamma(q,a) = (1/\ell^{(r_0)}) \cdot \sum_{t\in C(r_0)} \sum_{i\notin\mathrm{I}(r_0)} (n_i - |P(i, r_i - 1) \cup P(i, r_i)|) = \sum_{i\notin\mathrm{I}(r_0)} (n_i - |P(i, r_i - 1) \cup P(i, r_i)|)$。代回上面的方程，我们得到：

$$\begin{aligned}
\Delta\mathrm{dual}(r_0, 1) &= \Big(\sum_{i\notin\mathcal{I}(r_0)} n_i\Big) - k + \Big(\sum_{i\in\mathcal{I}(r_0)} k_i\Big) - \sum_{i\notin\mathcal{I}(r_0)} (n_i - |P(i, r_i - 1) \cup P(i, r_i)|) \\
&= \Big(-k + \sum_{i\in\mathcal{I}(r_0)} k_i + \sum_{i\notin\mathcal{I}(r_0)} |P(i, r_i)|\Big) + \sum_{i\notin\mathcal{I}(r_0)} |P(i, r_i - 1) \setminus P(i, r_i)|
\end{aligned}$$

该引理由于注意到括号中的上述项之和为 0 而得出：这是因为在阶段 r_0 结束时，大小为 k 的缓存恰好包含孤立代理 $i \in \mathcal{I}(r_0)$ 的 k_i 个（分数）页面，以及非孤立代理 $i \notin \mathcal{I}(r_0)$ 的 $|P(i,\ r_i)|$ 个（整数）页面。◀

对偶更新的第二阶段

现在，我们将介绍在每个阶段 $r_0 \in [R_0]$ 结束时执行的对偶更新的第二阶段。与第一阶段不同，在第一阶段中，我们只增加阶段 r_0中各时间步的 α 和 β 变量，而在第二阶段中，我们会减少前一阶段 $r_0 - 1$中各时间步的 β 变量。

更新$(r_0, 2)$的描述：这些对偶更新对应于在阶段 r_0-1 结束时处于孤立状态、但在阶段 r_0结束时不再处于孤立状态的代理。考虑一个代理$i \in \mathrm{I}(r_0-1)\backslash\mathrm{I}(r_0)$。对于阶段 r_0-1 中满足 $\beta(t,i) > 0$ （即 $t \in C(r_0-1)$）的每个时刻 t，我们执行以下操作：对于所有 i-页面 $q \in \mathrm{P}(i)\backslash\big(P(i,r_i-1)\cup P(i,r_i)\big)$，将 $\gamma(q,a(q,t))$增加 $\beta(t,i)$，随后将 $\beta(t,i)$重置为0。

为清楚起见，我们说明如下：(i) 将 $\beta(t,i)$ 重置为零，是变量被减小时唯一的对偶更新；(ii) 对阶段 r_0-1（即前一阶段）的时间点应用 $\beta(t,i)$ 更新；(iii) 我们在上文更新的 $\gamma(q,a(q,t))$ 变量，在阶段 r_0-1 末尾应用 update$(r_0-1,1)$ 时保持不变，因为当时其所有者 i 被指定为孤立。减小 $\beta(t,i)$ 变量的原因是，这会增加对偶目标值，而这对于支付与伪干净页面相关的代价是必要的。我们在下述引理中对此进行形式化说明。

▶ 引理 *18*。我们有 $\Delta dual(r_0, 2) = \sum_{i\in I(r_0-1)\backslash I(r_0)} (|P(i, r_i - 1) \cup P(i, r_i)| - k_i)$，其中 $\Delta dual(r_0, 2) \triangleq dual(r_0, 2) - dual(r_0, 1)$ 是应用 $update(r_0, 2)$ 后对偶目标值的变化。

证明。固定一个代理 $i \in \mathrm{I}(r_0-1)\backslash\mathrm{I}(r_0)$。在引理 16 中，我们证明了在更新$(r_0-1, 1)$之后，$\sum_{t\in}$阶段 $_{r_0-1}\beta(t,i) = 1$和 $\beta(t,i) \in \{0, 1/\ell^{(r_0-1)}\}$。考虑阶段 r_0-1 中满足$\beta(t,i) > 0$的任意时刻 t。根据更新$(r_0, 2)$的定义，对于所有页面 $q \in \mathrm{P}(i) \setminus \big(P(i,r_i-1) \cup P(i,r_i)\big)$，我们将 $\beta(t,i)$ 减少 $1/\ell^{(r_0-1)}$，同时将$\gamma(q,a(q,t))$ 增加相同的量。回顾对偶目标函数中的系数，我们可以看到，与代理 i对应的更新使对偶目标函数恰好增加：

$$(n_i - k_i) - |\mathcal{P}(i) \setminus (P(i,r_i) \cup P(i,r_i-1))| = |(P(i,r_i) \cup P(i,r_i-1))| - k_i. \qquad \blacktriangleleft$$

近似对偶可行性

We 通过证明对偶解始终近似可行，完成本节。

▶ 引理 *19.* 设 (α,β,γ) 表示在对某个 $r_0 \in [R_0]$ 和 $s \in \{0, 1\}$ 应用更新(r_0, s) 后立即得到的对偶解。对于任意 i-页面 q 及满足 $a \le a(q,T)$ 的整数 $a \ge 1$，我们有 $\sum_{t\in I(q,a)}(\alpha(t) - \beta(t,i)) - \gamma(q,a) \le 5$。

证明。首先，就本证明而言，r_0 和 s 的具体取值无关紧要，因此我们忽略它们。固定任意一个 i-页面 q 及满足 $a \le a(q,T)$ 的整数 a。回顾一下，$I(q,a) = \{t_{q,a}+1, \ldots, t_{q,a+1}-1\}$，其中 $t_{q,a'}$ 表示第 a' 次请求 q 的时间；我们重新定义 $t_{q,a+1}$，令其在 $t_{q,a+1} > T$ 成立时取值为 $T+1$。

如果 $I(q,\ a)$ 为空，则该引理显然成立，因此我们假设其不为空。令 $r_0^b,\ r_0^e \in [R_0]$分别表示包含时间步 $t_{q,\ a}+1$ 和 $t_{q,\ a+1}-1$ 的全局阶段。显然，

$r_0^b \le r_0^e$。该引理的另一个简单情形是 $r_0^e \le r_0^b + 1$ 成立。由于所有对偶变量均非负，且任一阶段中所有 $\alpha(t)$ 变量之和至多为 1（由引理 16 可知），因此很容易得到所需结论。形式化地，

$$\sum_{t \in I(q,a)} \big(\alpha(t) - \beta(t,i)\big) - \gamma(q,a) \le \sum_{t \in \text{phase } r_0^b} \alpha(t) + \sum_{t \in \text{phase } r_0^e} \alpha(t) \le 2.$$

现在假设 $r_0^b + 2 \le r_0^e$ 成立。定义 $Z := \{r_0^b + 1, \ldots, r_0^e - 1\}$。重复上述计算，我们得到：

$$\sum_{t \in I(q,a)} (\alpha(t) - \beta(t,i)) - \gamma(q,a) \le 2 + \left\{ \sum_{r_0 \in Z} \sum_{t \in \text{phase } r_0} \big(\alpha(t) - \beta(t,i)\big) \right\} - \gamma(q,a),$$

因此，引理的关键在于证明：跨越 Z 中多个阶段的时间点上，$\delta(t) \triangleq \alpha(t) - \beta(t,i)$ 的总和不会比 $\gamma(q,a)$ 大太多。对于一个阶段 $r_0 \in Z$，我们滥用记号 $\delta(r_0)$ 来表示 $\sum_{t \in \text{阶段 } r_0} \delta(t)$。注意，根据 β 变量的非负性和引理 17，$\delta(r_0) \le 1$ 对每个 $r_0 \in Z$ 都成立。我们对阶段 $r_0 \in Z$ 进行分类讨论，以便更好地掌握对偶更新过程中发生的变化。

(a) 假设 $i \in \mathrm{I}(r_0) \cap \mathrm{I}(r_0 + 1)$ 成立。由于 i 在阶段 r_0 结束时是孤立的，我们知道，对于某个 $t \in C(r_0)$，$\alpha(t)$ 中的任何增加（作为 update$(r_0, 1)$ 的一部分）都会伴随着 $\beta(t,i)$ 中相同的增加。由于 $i \in \mathrm{I}(r_0+1)$ 成立，update$(r_0+1, 2)$ 不会将任何 $\{\beta(t,i)\}_{t \in \text{phase } r_0}$ 变量减少/重置为 0。因此，$\delta(r_0) = 0$ 成立。注意，可能有任意多个阶段 r_0 属于这种情况，但这对我们来说不是问题，因为 $\delta(r_0) = 0$。

(b) 假设 $i \in \mathrm{I}(r_0) \setminus \mathrm{I}(r_0 + 1)$ 和 $q \in P(i, r_i - 1) \cup P(i, r_i)$ 成立。对于此情形，我们采用平凡界 $\delta(r_0) \le 1$。由于 i 在阶段 r_0 结束时是孤立的，但在阶段 r_0+1 结束时是非孤立的，局部阶段计数器 r_i 在阶段 $r_0 + 1$ 结束时增加 1，随后 $P(i, \cdot)$ 集合会更新，纳入一些新的一组已标记 i-页面。根据 $I(q,a)$ 的定义，在 Z 中的任何阶段都不会对 q 发出页面请求。因此，在 q 从 $P(i, \cdot)$ 集合中被移除之前，代理 i 的局部阶段至多增加 2 次；按照 A 的设计，在下一次请求页面 q 之前，它不会进入任何后续的 $P(i, r_i)$，而这在 Z 中的任何阶段都不会发生。

(c) 假设 $i \in \mathrm{I}(r_0) \setminus \mathrm{I}(r_0 + 1)$ 和 $q \notin P(i, r_i - 1) \cup P(i, r_i)$ 成立。与上面的情况 (a) 类似，我们知道，对于某个 $t \in C(r_0)$，$\alpha(t)$ 的任何增加（作为 update$(r_0, 1)$ 的一部分）都会伴随着 $\beta(t,i)$ 的相同幅度增加。现在，虽然 update$(r_0 + 1, 2)$ 会将所有 $\{\beta(t,i)\}_{t \in C(r_0)}$ 变量减小/重置为 0，但它也会使 $\gamma(q,a)$ 增加相同的幅度，因为 $q \notin P(i, r_i - 1) \cup P(i, r_i)$。因此，$\sum_{t \in \text{阶段 } r_0} \alpha(t)$ 等于 update$(r_0 + 1, 2)$ 导致的 $\gamma(q,a)$ 增量。所以，差值实质上为 0。

(d) 假设 $i \notin \mathrm{I}(r_0)$ 和 $q \in P(i, r_i - 1) \cup P(i, r_i)$ 成立。对于这种情况，我们使用平凡界 $\delta(r_0) \le 1$。由于 i 在阶段 r_0 结束时是非孤立的，局部阶段计数器 r_i 会在阶段 r_0 结束时加 1。重复情形 (c) 中的论证，在 Z 的其余阶段里，在 q 从 $P(i, \cdot)$ 集合中被移除之前，代理 i 的局部阶段最多还会增加 1 次。

(e) 假设 $i \notin \mathrm{I}(r_0)$ 和 $q \notin P(i, r_i - 1) \cup P(i, r_i)$ 成立。由于在阶段 r_0 结束时 i 是非孤立的，且 q 不属于 $P(i, r_i - 1) \cup P(i, r_i)$，我们知道，对于某个 $t \in C(r_0)$，$\alpha(t)$ 的任何增加（作为更新$(r_0, 1)$ 的一部分）都会伴随着 $\gamma(q,a)$ 的相同幅度增加。因此，阶段 r_0 $\alpha(t)$ 等于由于更新$(r_0, 1)$ 导致的 $\gamma(q,a)$ 的增加。因此，差值实际上为 0。

根据上述分类讨论可知，$\{\sum_{r_0 \in Z} \sum_{t \in \text{阶段 } r_0} (\alpha(t) - \beta(t,i))\} - \gamma(q,a)$ 不超过满足情形 (b) 或 (d) 的阶段数。由于我们已经论证此类情况最多出现 3 次，因此 $\sum_{t \in I(q,a)} (\alpha(t) - \beta(t,i)) - \gamma(q,a) \leq 2 + 3 = 5$ 成立。◀

我们现在通过结合上述引理来证明本节中的主定理。

定理 15 的证明。 定理的第一部分由引理 19 得出。第二部分直接由引理17和引理18得出，因为我们有 对偶$(R_0) = \sum_{r_0=1}^{R_0} (\Delta$对偶$(r_0,1) + \Delta$对偶$(r_0,2)$
)

◀

4 舍入

在本节中，我们将展示如何将分数算法 1 转换为一种用于带预留的缓存的随机化整数在线算法，并且该算法的每一步都能在多项式时间内运行。

该算法在 $N = k^3$ 个有效缓存状态上维持均匀分布。在每一步中，这些状态都会根据分数算法的操作进行更新。随机化算法在开始时从这些状态中均匀随机地选择一个，然后在整个运行过程中始终遵循该状态。

初始时，分布中的所有 N 缓存状态都与算法 1 中的初始缓存状态相同。每次页面请求后，根据分数算法的值 x_p^t，分布分两步进行更新。首先，我们将这些分数离散化为$\tilde{x}_p^t$，得到离散化版本；该版本同样是一个可行的分数解。这一步基于 [2] 中的技术。其次，我们更新 N 缓存状态，使它们全部保持有效，并且对于每个页面 p，恰好有$N \cdot \tilde{x}_p^t$ 个状态包含 p。这一步基于 [10] 中的技术。值得注意的是，舍入过程可以在线完成，因为它不需要知道任何未来页面请求。

4.1 离散化过程

在本小节中，我们将说明如何执行第一步：将分数算法的值 x_p^t 离散化为 $\tilde{x}_p^t$，使其成为$\frac{1}{N}$ 的倍数。我们的方法相当简单：按照任意顺序遍历页面 π，但要确保属于同一代理的所有页面连续排列（即任意排列代理，并任意排列每个代理的页面，但不同代理的页面不能交错排列）。然后，对于 $i \in [|\mathcal{P}|]$，设：

$$\tilde{x}_{\pi(i)}^t \triangleq \left\lfloor \sum_{j=1}^{i} x_{\pi(j)}^t \right\rfloor_{1/N} - \left\lfloor \sum_{j=1}^{i-1} x_{\pi(j)}^t \right\rfloor_{1/N}$$

其中$\lfloor a \rfloor_b$ 表示将 a 向下舍入至最接近的 b 的整数倍；形式化地：$\lfloor a \rfloor_b \triangleq b \lfloor a/b \rfloor$。

▶**引理20。** 离散化满足以下保证：
1. $\tilde{x}_p^t$ 是 $1/N$2. $\left|\tilde{x}_p^t - x_p^t\right| < 1/N$3. 对于每个代理 i, $\left|\sum_{p \in \mathcal{P}(i)} \tilde{x}_p^t - \sum_{p \in \mathcal{P}(i)} x_p^t\right| < 1/N$4. 如果 $x_p^t \in \{0, 1\}$，则 $\tilde{x}_p^t = x_p^t$

证明。 第一项保证 $\tilde{x}_p^t$ 是 $1/N$的倍数，这可由 $\lfloor \cdot \rfloor_{1/N}$ 按照定义是$1/N$ 的倍数得出。

现在我们证明第二项保证 $\left|\tilde{x}_p^t - x_p^t\right| < 1/N$。令 i 为 $\pi^{-1}(p)$，于是得到：

$$
\begin{aligned}
a - \lfloor a \rfloor_b &= a - b\lfloor a/b \rfloor \\
&\in \big[a - b(a/b), a - b(a/b - 1)\big) \\
&= [0, b) \\
\tilde{x}_p^t - x_p^t &= \tilde{x}_{\pi(i)}^t - x_{\pi(i)}^t \\
&= \underbrace{\left[\left\lfloor \sum_{j=1}^{i} x_{\pi(j)}^t \right\rfloor_{1/N} - \left\lfloor \sum_{j=1}^{i-1} x_{\pi(j)}^t \right\rfloor_{1/N}\right]}_{\tilde{x}_{\pi(i)}^t} - \underbrace{\left[\sum_{j=1}^{i} x_{\pi(j)}^t - \sum_{j=1}^{i-1} x_{\pi(j)}^t\right]}_{x_{\pi(i)}^t} \\
&= \left[\sum_{j=1}^{i-1} x_{\pi(j)}^t - \left\lfloor \sum_{j=1}^{i-1} x_{\pi(j)}^t \right\rfloor_{1/N}\right] - \left[\sum_{j=1}^{i} x_{\pi(j)}^t - \left\lfloor \sum_{j=1}^{i} x_{\pi(j)}^t \right\rfloor_{1/N}\right] \\
&\in \left(-\frac{1}{N}, \frac{1}{N}\right) \\
\left|\tilde{x}_p^t - x_p^t\right| &< \frac{1}{N}
\end{aligned}
$$

第三个保证的证明基本相同。这里我们用到了这样一个事实：每个代理的页面在 π 中是连续的。假设我们的代理的页面为 $\{\pi(\ell), \ldots, \pi(h)\}$：

$$
\begin{aligned}
\sum_{p\in\mathcal{P}(i)} \tilde{x}_p^t - \sum_{p\in\mathcal{P}(i)} x_p^t &= \sum_{i=\ell}^{h} \tilde{x}_{\pi(i)}^t - \sum_{i=\ell}^{h} x_{\pi(i)}^t \\
&= \left[\left\lfloor \sum_{j=1}^{h} x_{\pi(j)}^t \right\rfloor_{1/N} - \left\lfloor \sum_{j=1}^{\ell-1} x_{\pi(j)}^t \right\rfloor_{1/N}\right] - \left[\sum_{j=1}^{h} x_{\pi(j)}^t - \sum_{j=1}^{\ell-1} x_{\pi(j)}^t\right] \\
&= \left[\sum_{j=1}^{\ell-1} x_{\pi(j)}^t - \left\lfloor \sum_{j=1}^{\ell-1} x_{\pi(j)}^t \right\rfloor_{1/N}\right] - \left[\sum_{j=1}^{h} x_{\pi(j)}^t - \left\lfloor \sum_{j=1}^{h} x_{\pi(j)}^t \right\rfloor_{1/N}\right] \\
&\in \left(-\frac{1}{N}, \frac{1}{N}\right) \\
\left|\sum_{p\in\mathcal{P}(i)} \tilde{x}_p^t - \sum_{p\in\mathcal{P}(i)} x_p^t\right| &< \frac{1}{N}
\end{aligned}
$$

第四个保证可由以下观察得出：若 $x_{\pi(i)}^t$ 是整数，则 $\lfloor\cdot\rfloor^{1/N}$ 运算符使 $\sum_{j=1}^{i} x_{\pi(j)}^t$ 的减少量与使 $\sum_{j=1}^{i-1} x_{\pi(j)}^t$ 的减少量相同。◀

►**推论 21**。如果 $\{x_p^t\}$ 满足总缓存容量 $\left(\sum_{p\in\mathcal{P}} x_p^t \le k\right)$ 和预留要求 $\left(\sum_{p\in\mathcal{P}(i)} x_p^t \ge k_i\right)$，那么 $\{\tilde{x}_p^t\}$ 也满足。

证明。由于望远镜求和论证，总缓存容量约束仍然成立：

$$
\sum_{p\in\mathcal{P}} \tilde{x}_p^t = \sum_{i\in[|\mathcal{P}|]} \tilde{x}_{\pi(i)}^t = \sum_{i\in[|\mathcal{P}|]} \left[\left\lfloor \sum_{j=1}^{i} x_{\pi(j)}^t \right\rfloor_{1/N} - \left\lfloor \sum_{j=1}^{i-1} x_{\pi(j)}^t \right\rfloor_{1/N}\right]
$$

$$= \left\lfloor \sum_{j=1}^{|\mathcal{P}|} x^t_{\pi(j)} \right\rfloor_{1/N} \leq \sum_{j=1}^{|\mathcal{P}|} x^t_{\pi(j)} \leq k$$

接下来，我们将证明预留缓存大小均得到满足。反设，对于某个代理 i, $\sum_{p\in\mathcal{P}(i)} \tilde{x}^t_p < k_i$。由于该不等式右侧是整数，因而是 $1/N$ 的倍数；而左侧也是$1/N$ 的倍数，因为它是若干个 $1/N$ 的倍数之和（根据引理20的第一个保证），所以左侧必然至少比右侧小一个完整的 $1/N$ 倍数：$\sum_{p\in\mathcal{P}(i)} \tilde{x}^t_p \leq k_i - 1/N$。但这与引理20的第三个保证 $\left|\sum_{p\in\mathcal{P}(i)} \tilde{x}^t_p - \sum_{p\in\mathcal{P}(i)} x^t_p\right| < 1/N$ 矛盾。因此，对于所有代理 i, $\sum_{p\in\mathcal{P}(i)} \tilde{x}^t_p \geq k_i$，结论成立，证明完毕。◀

▶ **推论 22.** $\sum_p \left|\tilde{x}^t_p - x^t_p\right| \leq k^2/N$

证明。 不失一般性，代理的数量至多为 k，因为我们可以将所有没有任何储备的代理合并。每个代理 i 通过该算法至多拥有 k 个分数页（即在 i的局部阶段开始时位于缓存中的 i 页）。根据引理20的第二项保证，每个分数页的 x 值的失真至多为 $1/N$；而根据第四项保证，非分数页的值不会失真。证毕。◀

▶**引理23**。 设 x^t_p 和 x^{t+1}_p分别为分数算法中连续两个时间步的缓存中每个页面 p 的数量，$\tilde{x}^t_p$ 和 $\tilde{x}^{t+1}_p$为相应的离散化值。那么，从 $\tilde{x}^t$ 到 $\tilde{x}^{t+1}$ 的缓存更新代价（记为 $\tilde{c}$）至多是从 x^t 到 x^{t+1} 的缓存更新代价（记为 c）的两倍。

证明。 引理10意味着以下两种情况之一成立： $c = 0$ （即请求的页面已经位于缓存中，因此缓存状态没有发生变化），或者 $c \geq 1/k$。在第一种情况下，离散化后的缓存状态也没有发生变化，因此 $\tilde{c} = 0$。因此，我们重点考虑第二种情况。根据三角不等式，对于任意页面 p,

$$|\tilde{x}^t_p - \tilde{x}^{t+1}_p| \leq |\tilde{x}^t_p - x^t_p| + |x^t_p - x^{t+1}_p| + |x^{t+1}_p - \tilde{x}^{t+1}_p|.$$

我们注意到，由于将页面添加到缓存会产生代价，且原始分数解和离散化后的解都向缓存中添加与其逐出的页面量相等的页面质量，因此 $x^{t+1}_p|$ $\tilde{c}$ p ，类似地，对于 也是如此。对 上述不等式求和，可得 $2c \geq \sum_p|$

$$2\tilde{c} = \sum_p |\tilde{x}^t_p - \tilde{x}^{t+1}_p| \leq \sum_p |x^t_p - x^{t+1}_p| + 2k^2/N = 2c + 2/k \leq 4c,$$

其中，我们根据推论 22 得到 $\sum_p(|\tilde{x}^t_p - x^t_p| + |x^{t+1}_p - \tilde{x}^{t+1}_p|) \leq 2k^2/N$，然后 $N = k^3$，接着$c \geq 1/k$。◀

4.2 更新缓存状态的分布

在本小节中，我们将说明如何执行第二步：更新 N 缓存状态。我们希望 (i) 所有缓存状态都有效，(ii) 恰好有 $N \cdot \tilde{x}^t_p$ 个状态包含页面 p（由于离散化步骤，该数值为整数），并且 (iii) 不要使用过多驱逐操作。

形式化地，设$\mathcal{X}$ 是一个由 N 个缓存状态构成的集合，每个状态包含 k 个页面，对应于时间步 t 的离散化值 $\tilde{x}^t_p$。给定时间步 $t+1$ 的离散化值 $\tilde{x}^{t+1}_p$，我们说明如何将$\mathcal{X}$转换为$\mathcal{X}'$,，使每个页面 p 恰好出现在 $N \cdot \tilde{x}^{t+1}_p$ 个缓存状态中，并且每个缓存状态都满足所有预留要求。

设 P 是一个页面多重集，其中页面在缓存中的占比从时间 t 到时间 $t+1$有所增加，且每个页面 p出现 $\max(0,(\tilde{x}_p^{t+1}-\tilde{x}_p^t)N)$ 次。设 Q 是一个类似的、对应占比减少的多重集，其中每个页面出现 $\max(0,(\tilde{x}_p^t-\tilde{x}_p^{t+1})N)$ 次。由于缓存中的页面总量不变，因此 $|P|=|Q|$。我们在 P 和 Q 中的页面之间建立匹配，并据此逐对转换$\mathcal{X}$为$\mathcal{X}'$。匹配按如下方式构造：首先，将 P 和 Q 中属于同一代理的页面相互匹配。然后，任意匹配 P 和 Q 中剩余的页面。

▶ **引理 24.** 设 $(p_1,q_1),(p_2,q_2)$ ... 是上文所述的 P与 Q 之间的匹配。那么对于任意 j，将页面 p_1, ..., p_j 的 $1/N$ 份额加入 $\tilde{x}^t$，并从中移除页面 q_1, ..., q_j 的 $1/N$ 份额后得到的分数解，满足所有预留要求。

证明。随着各对依次应用，分数解从 $\tilde{x}^t$ 变为 $\tilde{x}^{t+1}$，二者都满足预留要求（根据推论 21）。归纳地，假设应用到 (p_{j-1},q_{j-1}) 为止的各对后，预留要求仍然得到满足。如果 p_j和 q_j 属于同一个代理 i，那么缓存中的 i-页面数量不会改变，因此仍满足要求。如果 p_j 和 q_j 属于不同的代理，那么我们必定处于匹配构造的第二阶段。这意味着 P 中不再有属于 $ag(q_j)$ 的页面，因此在应用剩余匹配对的过程中，缓存中的 $ag(q_j)$- 页面数量不会增加。这意味着应用 (p_j,q_j) 后，缓存中的 $ag(q_j)$-页面数量介于其在 $\tilde{x}^t$ 和 $\tilde{x}^{t+1}$中的数量之间。由于这两种情况都满足预留要求，中间解也满足要求。◀

现在，我们展示如何使用匹配中的下一对（p,q）来修改 $\mathcal{X}$。这一过程遵循 [10], 中的步骤，不同之处在于，我们仅处理数量有限的 N 集合，并且 p 的增加量和 q 的减少量均固定为 $1/N$。

令 $\mathcal{X}$ 为当前的缓存状态集合（可能已由前面的页面对修改）。如果存在一个缓存状态 $S\in\mathcal{X}$，使得 $p\notin S$ 和 $q\in S$，则将 p 添加到 S，并从S中移除 q。否则，找到缓存状态 $S\in\mathcal{X}$ 和 $T\in\mathcal{X}$，使得 $p\notin S$ 和 $q\in T$，将 p 添加到 S，并从 T中移除 q。接下来，将某个页面 $r\in S\setminus T$从 S移动到 T，以将集合大小调整回 k。

此时，每个页面都位于正确数量的缓存状态中。然而，在移除相应页面的缓存状态中，代理 i 可能有一个页面违反预留要求，适用于 $ag(q)$ 或 $ag(r)$。在这种情况下，假设代理 i 在缓存状态 $V\in\mathcal{X}$中违反了预留要求。由于根据引理 24，每项预留要求在平均意义上都得到满足，因此必定存在另一个集合 $W\in\mathcal{X}$，其中属于代理 i的页面严格多于 k_i 个。我们将这样一个页面从 W 移至 V。现在 V 有 $k+1$ 个页面，因此必定存在某个代理 j，其在 V 中拥有超过 k_j 个页面。我们将 j的一个页面从 V 移至 W，以恢复集合大小。至此，更新完成，得到与分数 $\tilde{x}^{t+1}$ 对应的新有效集合。

最后，我们通过将更新到$\mathcal{X}$的代价与分数算法的代价进行比较，给出其上界，从而得到随机化算法的期望代价。

▶**引理 25.** 更新到 X 的代价至多为从 $\tilde{x}^t$到 $\tilde{x}^{t+1}$ 的分数缓存更新代价的 6 倍。

证明。匹配中的每一对 (p,q) 对应离散化分数解所产生的代价 $1/N$。在对 $\mathcal{X}$中的集合进行更新时，每次从某个缓存状态中移除一个页面，都会给随机化算法带来代价 $1/N$。最多会进行以下六次移除：从 q 中移除 T；从 r 中移除 S；为满足 $ag(q)$ 和 $ag(r)$ 的预留要求，各交换两个页面。◀

定理 2 的证明可通过结合引理 23 和 25 得出。

参考文献

1 Dimitris Achlioptas、Marek Chrobak 和 John Noga。《随机化分页算法的竞争性分析》。《理论计算机科学》，234(1):203–218，2000年。doi:10.1016/S0304-3975(98)00116-9。2 Anna Adamaszek、Artur Czumaj、Matthias Englert 和 Harald Räcke。一种 $O(\log k)$-竞争算法，用于广义缓存。《ACM 算法汇刊》，15(1):6:1–6:18，2018年。 doi:10.1145/3280826。3 Kunal Agrawal、Michael A. Bender、Rathish Das、William Kuszmaul、Enoch Peserico 和 Michele Scquizzato。并行分页与绿色分页的紧界。载于第 32 届离散算法研讨会论文集，第 3022–3041 页，2021年。doi:10.1137/1.9781611976465.180。4 Nikhil Bansal、Niv Buchbinder 和 Joseph Seffi Naor。随机化在线加权分页的简单分析。未发表手稿，2010年。5 Nikhil Bansal、Niv Buchbinder 和 Joseph (Seffi) Naor。迈向随机化 k-服务器猜想：一种原始-对偶方法（扩展摘要）。载于第 21 届离散算法研讨会论文集，第 40–55 页，2010年。 doi:10.1137/1.9781611973075.5。6 Nikhil Bansal、Christian Coester、Ravi Kumar、Manish Purohit 和 Erik Vee。学习增强型加权分页。载于第 33 届离散算法研讨会论文集，第 67–89 页，2022年。 doi:10.1137/1.9781611977073.4。7 Jichuan Chang 和 Gurindar S Sohi。芯片多处理器的协作式缓存分区。载于第 21 届 ACM 超级计算国际会议论文集，第 242–252 页，2007年。 doi:10.1145/1274971.1275005。8 Ashish Chiplunkar、Monika Henzinger、Sagar Sudhir Kale 和 Maximilian Vötsch。在线极小极大分页。载于第 34 届离散算法研讨会论文集，第 1545–1565 页，2023。 doi:10.1137/1.9781611977554.ch57。9 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。doi:10.1016/0196-6774(91)90041-V。10 Sharat Ibrahimpur、Manish Purohit、Zoya Svitkina、Erik Vee 和 Joshua R. Wang。带预留的缓存。载于第 25 届组合优化近似算法国际会议（APPROX）论文集，第 245 卷，第 52:1–52:16 页，2022年。 doi:10.4230/LIPIcs.APPROX/RANDOM.2022.52。11 Wu Kan、Tu Kaiwei、Patel Yuvraj、Sen Rathijit、Park Kwanghyun、Arpaci-Dusseau Andrea 和 Remzi Arpaci-Dusseau。NyxCache：灵活高效的多租户持久内存缓存。载于第 20 届 USENIX 文件与存储技术大会（FAST）论文集，第 1–16 页，2022年。网址：https://www.usenix.org/conference/fast22/presentation/wu。12 Mayuresh Kunjir、Brandon Fain、Kamesh Munagala 和 Shivnath Babu。ROBUS：数据并行工作负载的公平缓存分配。载于 2017 年 ACM 数据管理国际会议（SIGMOD）论文集，第 219–234 页，2017年。doi:10.1145/3035918.3064018。13 Lyle A. McGeoch 和 Daniel D. Sleator。一种强竞争性随机化分页算法。《Algorithmica》，6:816–825，1991年。 doi:10.1007/BF01759073。14 Qifan Pu、Haoyuan Li、Matei Zaharia、Ali Ghodsi 和 Ion Stoica。FairRide：近乎最优的公平缓存共享。载于第 13 届 USENIX 网络系统设计与实现研讨会（NSDI）论文集，第 393–406 页，2016年。网址：https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/pu。15 Harold S. Stone、John Turek 和 Joel L. Wolf。缓存存储器的最优分区。《IEEE 计算机汇刊》，41(9):1054–1068，1992年。 doi:10.1109/12.165388。

16 G Edward Suh、Larry Rudolph 和 Srinivas Devadas。共享缓存内存的动态分区。《超级计算杂志》，28(1):7–26，2004年。doi:10.1023/B:SUPE.0000014800.27383.8f。17 Yinghao Yu、Wei Wang、Jun Zhang 和 Khaled Ben Letaief。LACS：具有隔离保证的负载感知缓存共享。载于第 *39* 届 *IEEE* 国际分布式计算系统会议（*ICDCS*）论文集，第 207–217 页。IEEE，2019年。doi:10.1109/ICDCS.2019.00029。

A 无权分页的随机标记分析

本节讨论经典无权分页问题，并给出一个新的证明，表明随机标记 [9] 算法是 $O(\log k)$-竞争的。

回顾一下，随机标记算法按如下方式分阶段运行。初始时，缓存中的所有页面都处于未标记状态。在时间 t请求页面 p_t 时，如果 p_t 已在缓存中，则将其标记，不进行其他更改。如果 p_t 不在缓存中，则从当前缓存中的未标记页面集合中均匀随机地选取一个页面 q 并将其驱逐。随后将页面 p_t 放入缓存，并将其标记。当没有更多未标记页面可供驱逐时，当前阶段结束——缓存中的所有页面都变为未标记状态，并开始新阶段。

令 RM 表示上述随机标记算法，并令 $\mathrm{cost}_{\mathrm{RM}}(\sigma)$ 表示在实例 σ上产生的总代价。为进行分析，我们考虑标记算法的如下分数版本：该算法像之前一样维护已标记和未标记页面集合。在任意时刻 t，令 x_p^t 表示页面 p 在缓存中的比例，令 $y_p^t \triangleq 1 - x_p^t$表示页面 p 在缓存外的比例。当请求页面 p_t 时，分数标记算法令 $x_{p_t} = 1$，并对缓存中所有未标记页面 q 均匀增加 y_q。与随机标记算法一样，当所有未标记页面都已从缓存中完全驱逐时，一个阶段便结束。注意，对任意页面 p 和时刻 t，分数标记算法中的值y_p^t 恰好等于随机标记算法截至该时刻已将页面 p从缓存中驱逐的边缘概率。令 FM 表示分数标记算法，则有：$\mathrm{E}[\mathrm{cost}_{\mathrm{RM}}(\sigma)] = \mathrm{cost}_{\mathrm{FM}}(\sigma)$。本节余下部分将重点界定分数标记算法产生的总代价。

势函数

考虑分数标记算法的任意阶段 r，令 P^{r-1} 表示该阶段开始时缓存中恰好有 k 个页面的集合，令 P^r 表示该阶段结束时缓存中的 k 个页面的集合。根据定义，P^r 恰好是阶段 r 期间所请求的 k 个不同页面的集合。按照标准术语，如果页面 $p \in P^r$ 满足 $p \in P^{r-1}$，我们称其为过时页面，否则称 p 为干净页面。令 $\ell^{(r)} = |P^r \setminus P^{r-1}|$ 表示阶段 r 中干净页面的总数。在任意时间 t，令 $U(t)$ 表示时间 t 的未标记页面集合。考虑以下势函数：

$$\Psi(t) = \sum_{p \in U(t)} \phi(y_p^t)$$

其中 $\phi(h) = 2h\ln(1 + kh)$，与前文相同。在任意时间步 t，令 $\Delta\Psi(t)$ 表示处理请求页面时势能的总变化量 p_t[3]。令 $\Delta\Psi(r)$ 表示势能变化量

[3] 特别是，$\Delta\Psi(t)$不包括因阶段变化而产生的任何势能变化。

这是由于阶段 r 结束所致。

▶ **引理 26.** 在任意时刻 t，如果 $y^t_{p_t} < 1$，则 $\Delta\Psi(t) \geq y^t_{p_t}$。

证明。 在任意时刻 t，缓存中的所有未标记页面都以相同的分数式量被驱逐，即 $y^t_p = y^t_q = h^*, \ \forall p \neq q \in U(t)$。处理请求页面 p_t时，随着页面 p_t 不再对势能产生贡献，势能随之下降；与此同时，驱逐所有其他未标记页面会使势能增加。页面 p_t 在此步骤开始时对势能的贡献恰为$\phi(h^*) = 2h^*\ln(1+kh^*)$。另一方面，要获取页面 p_t，其他未标记页面总共被驱逐了 h^*。根据引理 11，在整个驱逐过程中，势能至少以 $(1+2\ln(1+kh^*))$ 的速率增加。将两部分贡献相加，得到：

$$\Delta\Psi(t) \geq -2h^*\log(1+kh^*) + h^*(1+2\ln(1+kh^*)) = y^t_{p_t}.$$ ◀

▶ **引理 27.** 在任意阶段 r 结束时，势能下降 $\Delta\Psi(r) = -2\ell^{(r)}\ln(1+k)$.

证明。 在阶段 r结束时， $P^{r-1}\backslash P^r$ 中的所有页面都未标记，且已从缓存中完全逐出。每个此类页面在阶段结束前的势能中恰好贡献 $\phi(1) = 2\ln(1+k)$。阶段 r 结束后（阶段 $r+1$ 开始时），所有这些页面都不再对势能有贡献。注意到 $|P^{r-1} \setminus P^r| = |P^r \setminus P^{r-1}| = \ell^{(r)}$，即可得证。 ◀

前两个引理使我们能够仅根据干净页面的总数来界定分数标记算法所产生的总代价。

▶ **引理 28.** 我们有 $\text{代价}_{FM}(\sigma) \leq 2k\ln(1+k) + (1+2\ln(1+k))\sum_{r=1}^{R}\ell^{(r)}$.

证明。 我们有如下等式与不等式链：

$$\begin{aligned}
\text{cost}_{\text{FM}}(\sigma) &= \sum_{t\in[T]} y^t_{p_t} = \sum_{t\in[T]:y^t_{p_t}<1} y^t_{p_t} + |\{t\in[T] : y^1_{p_t} = 1\}| \\
&= \sum_{t\in[T]:y^t_{p_t}<1} y^t_{p_t} + \sum_{r=1}^{R}\ell^{(r)} \\
&\leq \sum_{t\in[T]:y^t_{p_t}<1} \Delta\Psi(t) + \sum_{r=1}^{R}\ell^{(r)} \\
&= \Psi(T) - \Psi(0) - \sum_{r=1}^{R}\Delta\Psi(r) + \sum_{r=1}^{R}\ell^{(r)} \\
&\leq 2k\ln(1+k) + (1+2\ln(1+k))\sum_{r=1}^{R}\ell^{(r)}.
\end{aligned}$$

◀

▶ **定理 29。** 随机标记算法是 $O(\log k)$-竞争的。

证明。 对于任意实例 σ，我们有 $\mathbb{E}[\text{代价}_{\text{RM}}(\sigma)] = \text{代价}_{\text{FM}}(\sigma)$。注意到 $\text{代价}_{OPT}(\sigma) \geq (1/2)\cdot\sum_{r=1}^{R}\ell^{(r)}$，如 [9]所示，即可得出所需的竞争比界限。我们也可以使用第 3.4 节中给出的对偶拟合论证，证明即使是分数最优解也至少为 $(1/2)\cdot\sum_{r=1}^{R}\ell^{(r)}$，但为简洁起见，我们省略细节。 ◀