2DIO：用于存储基准测试的可配置且缓存精确的轨迹生成

Yirong Wang 东北大学 美国马萨诸塞州波士顿
wang.yiron@northeastern.edu

Isaac Khor 东北大学 美国马萨诸塞州波士顿
khor.i@northeastern.edu

Peter Desnoyers 东北大学 美国马萨诸塞州波士顿
P.Desnoyers@northeastern.edu

arXiv:2603.19971v1 [cs.OS] 20 Mar 2026

摘要

我们介绍 2DIO，这是一种微基准测试工具，可生成缓存精确且高负载的 I/O 跟踪。现有工具只能生成命中率曲线表现良好的凹形跟踪，而 2DIO 能生成复杂缓存行为可调的跟踪，尤其是性能陡变区和平台期。

我们的框架将工作负载编码为紧凑的参数三元组，同时捕捉短期近期性和长期频率。这种简约的参数化让研究人员能够轻松地将各项调整转化为在不同驱逐策略下可预测的缓存效果，还能扫描参数空间，全面探索所需的缓存行为，或通过校准参数以匹配观测到的行为来模拟真实跟踪。

经过调优的参数具有可移植性：评估系统的规模发生变化时，跟踪记录的占用空间和长度也会随之变化，同时保留相对缓存行为。

评估结果表明，2DIO 能够生成涵盖连续一系列“假设情境”缓存行为的跟踪记录，并以很高的准确度重现真实世界的缓存行为。

CCS 概念：• **综合与参考** → **性能**；
• **信息系统** → **分层存储管理**。

关键词：缓存、轨迹生成、性能评估、基准测试

ACM 引用格式：
Yirong Wang、Isaac Khor 和 Peter Desnoyers。2026。2DIO：面向存储基准测试的可配置且缓存精准的轨迹生成。载于欧洲计算机系统会议（*EUROSYS '26*），*2026*年*4*月*27*日至*30*日，英国苏格兰爱丁堡。ACM，纽约，纽约州，美国，16页。
https://doi.org/10.1145/3767295.3769391

1 引言

存储系统研究需要测量存储系统的性能，并将测得的结果与其他系统进行比较。这些系统通常规模庞大、结构复杂，供复杂的应用程序使用。性能不仅受系统硬件和配置影响，也受应用程序或工作负载特征影响，因此测量结果必然与测量时所用的工作负载密切相关。存储系统表现出复杂且依赖于工作负载的行为，因此，选择合适的工作负载对于测量至关重要，只有这样，测量结果才能用于比较和预测这些系统的性能。

理想情况下，这些测量应当具备以下特点：(1) 具有预测性，能够准确估计系统（或不同系统的相对性能）在未来执行中的表现；(2) 具有泛化能力，所提供的预测（尤其是比较性预测）应尽可能适用于广泛的应用程序和配置；(3) 可复现，即不同研究人员应能够进行相同的测量并比较结果；(4) 易于实施，所需投入和成本相对于被测系统而言不过分。

1.1 基准测试类型

我们首先回顾评估存储的三种方式：（a）真实工作负载，即目标应用程序本身；（b）轨迹回放，即记录真实工作负载下执行的操作，并以与原始操作或多或少的保真度重现这些操作；（c）合成工作负载，即通过算法生成行为，旨在模拟真实工作负载或其行为的某些方面。

真实应用程序。 在某些情况下，例如 HPC 采购，真实应用程序基准测试是“黄金标准”，能够给出所需的确切答案；然而，对于存储研究而言，它们存在诸多不足。尤其是，结果不具备普适性：如果用户 A 使用工作负载 A 测试一个系统，而用户 B 使用工作负载 B 测试另一个系统，那么我们几乎无法了解系统 A 与系统 B 之间的差异。
轨迹回放。 轨迹回放提供了一种方法，可使用相同的真实世界工作负载测试不同的系统。在实际运行的系统上记录各个 I/O 操作，CloudPhysics[35] 和阿里云 [3] 等轨迹语料库涵盖了多种工作负载。使用标准（例如 fio [4]）或自定义重放工具，将这些跟踪记录重放到被测系统上，从而按需重现原始工作负载。然而，现代轨迹语料库规模庞大且难以处理（CloudPhysics 和阿里云的语料库分别为 70 GB 和 700GB），这使得

本作品采用知识共享署名 4.0 国际许可协议授权。*EUROSYS '26*，英国苏格兰爱丁堡© 2026 版权所有归所有者/作者所有。ACM ISBN 979-8-4007-2212-7/26/04
https://doi.org/10.1145/3767295.3769391

穷尽式回放不切实际；由于发布可能包含敏感数据的语料库在流程上存在很高门槛，新语料库也十分少见，导致新的应用程序（例如大语言模型）需要很长时间才能体现在可用的跟踪记录中。

合成轨迹。 能够模仿真实工作负载的合成轨迹生成器解决了轨迹回放中的许多问题。合成生成器无需庞大的轨迹库，便可轻松集成到各种工具中。

从概念上讲，创建此类生成器的任务很简单：测量工作负载，将其编码为概率模型，并根据该模型生成新的轨迹。通过能够捕捉工作负载关键特征的参数，人们可以系统地探索整个空间，而不必从一组不透明的真实跟踪记录中随意挑选点。

但究竟应该测量并复现哪些特征？答案似乎很简单：那些会影响被测系统的特征。如果性能取决于 I/O 大小，就记录并复现其分布；如果读操作和写操作之间存在差异，就捕捉二者的比例；如果性能随寻道距离而变化，也要对此建模。

如果被评估的系统是缓存，那么有人会认为命中率曲线（HRC），即命中率随缓存大小变化的曲线，最为关键。尽管 fio 等生成器支持会影响 HRC 的频率偏斜模型（齐夫分布、帕累托分布、分区），我们将证明，仅凭这一点远不足以复现真实工作负载中观察到的 HRC。

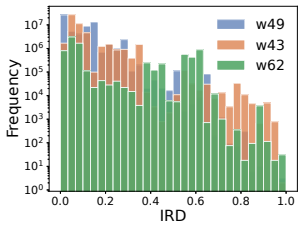
1.2 缓存准确的合成轨迹

工作负载生成器通常并非用于预测系统的行为，而是用于比较系统。例如，本会议发表的论文中的评估部分就常常如此：数据存储领域的论文经常使用 fio、Filebench [10],和轨迹回放，将自己的系统与先前的工作进行比较。这些评估采用：(a) 可复现的工作负载，以便比较实验；(b) 多种工作负载，涵盖各种可能的应用行为；以及 (c) 至少一些逼真的工作负载，用来模拟真实的应用行为。如果系统 A 在设计完善的评估中大多数或全部测试上都优于系统 B，业界通常会认为，这表明 A 在实际应用中很可能优于 B。

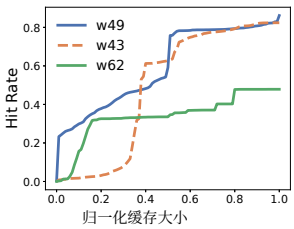
对于性能深受缓存命中率影响的系统而言，该工作负载的可缓存性十分重要——
否则，我们的基准测试可能会在实际场景中本应命中的情况下测出未命中性能，反之亦然。[1]

频率分布。 目前，重现工作负载可缓存性的先进方法似乎是采用项目频率模型，使用参数化分布（通常为齐夫分布）或从现有工作负载中实测得到的分布 [16, 24, 29, 31, 36, 37, 42]。这种方法源自 Denning 的独立引用模型 （IRM） [1]：每次到达都

[1]几十年前，IRCache 网页缓存的“对比测试”中就出现过这一问题： [15], 使用均匀随机分布的工作负载，使一些采用了更先进缓存算法的供应商处于不利地位。

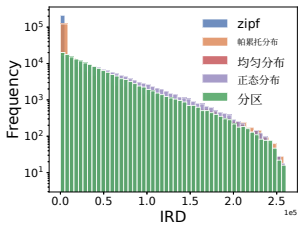


(a) IRD直方图

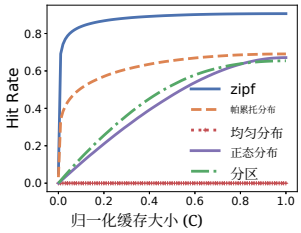


(b) 最近最少使用命中率

图 1. CloudPhysics 的几条轨迹展示了多种不同的命中率行为。缓存大小根据轨迹占用空间进行归一化，即轨迹中访问的唯一数据块总数。



(a) IRD直方图



(b) 最近最少使用命中率

图 2. fio 生成的合成轨迹的 IRD 分布和命中率行为。

被假定为从可能的项目集合中进行一次独立加权选择。

独立引用模型（IRM）在某些情况下能够准确建模，例如内容分发网络（CDN），其中许多独立来源被聚合到一个数据流中；但对于块存储系统，它的效果则大相径庭，因为这类系统的典型工作负载由少数高度相关的数据流组成。例如，图 1 展示了 CloudPhysics 语料库 [34, 35] 中三个真实世界块跟踪记录的行为。它们的引用间距离（IRD）直方图（图 1(a)）呈现出明显的尖峰和空洞。将这些轨迹输入最近最少使用（LRU）缓存后，会产生复杂的非凹型命中率曲线（HRC）（图 1(b)），其中包括性能悬崖——缓存略微增大便带来显著收益；以及平台期——继续增加缓存几乎没有额外收益。相比之下，图 2 展示了 fio 生成的工作负载，其 IRD 分布严格递减，因此产生凹型HRC，反映出增加缓存所带来的收益递减。

新近性分布。从数学上来说，不可能通过操纵频率分布来生成非凹*LRU* 命中率曲线 *[33]*。LRU 缓存中的一次访问是否命中，完全取决于上一次访问同一项以来的 IRD，尤其取决于该项在这段时间内是否被淘汰。由于独立引用模型在每次引用时访问特定项的概率相同，这些 IRD 始终服从指数分布；而命中率曲线之所以会呈现非凹性，恰恰是因为 IRD（例如扫描型工作负载下的 IRD ）不具有无记忆性。

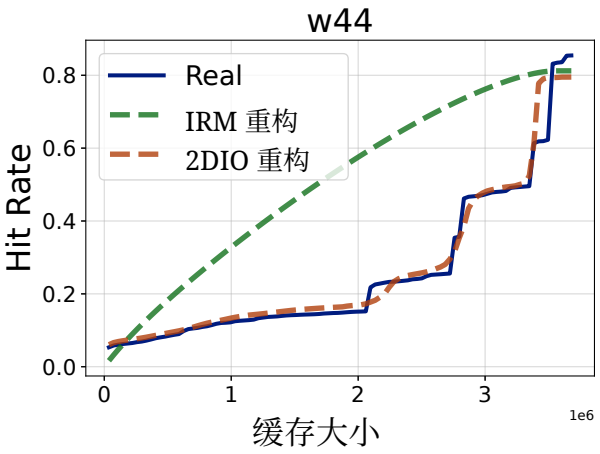


图 3. CloudPhysics w44 的 LRU 命中率曲线：原始轨迹（蓝色）、2DIO 生成的轨迹（橙色），以及使用经验测得的项目频率分布重构的轨迹（绿色）。缓存大小以块数计。

缓存建模社区 [9, 11–13, 21, 28, 34,35, 39,41] 长期以来一直使用近期性模型（例如栈距离 (SD) [20] 和 IRD [7, 8] ）来刻画非无记忆流量。尽管这些模型在预测给定轨迹的命中率曲线方面十分有效，但据我们所知，尚无先前工作反向使用这些模型，即生成与目标命中率曲线相匹配的轨迹。

为弥补这一空缺，我们提出了 Gen-from-IRD，这是一种根据给定 IRD 分布生成访问的算法。通过 Gen-from-IRD 生成的轨迹能够精确呈现目标 LRU HRC；对于使用近期性的缓存算法，其准确度远高于 IRM 生成的轨迹。

频率 +新近性分布。我们提出了 2DIO，这是一种能够忠实生成（或再现）访问频率和近期性的合成轨迹生成器。

基于 Gen-from-IRD，2DIO 将这些特征编码为紧凑的参数三元组，即轨迹配置，以创建或"仿造"呈现所需性能陡变区/平台期的工作负载。因此，研究人员可以通过精确指定目标命中率曲线来自定义轨迹，也可以在不同规模下模拟真实工作负载，以满足各种基准测试任务的需求。

2DIO 通过将用于刻画短期行为的近期性（IRD）模型与用于刻画长期行为的频率（IRM）模型相结合来实现这一点。图3展示了这种方法的有效性，其中显示了 CloudPhysics w44 的 LRU HRC，以及根据实测特征生成的 IRM 和 2DIO 重构结果。尽管 IRM 重构忠实地再现了轨迹中不同块的访问频率，但得到的命中率曲线简单且呈凹形，未能再现原始轨迹中的任何性能异常。相比之下，2DIO 生成的轨迹所得结果与原始轨迹非常相似，再现了性能陡变区和平台期，仅有轻微偏差。

这种方法适用于更广泛的领域，例如内容分发网络和 Web 缓存；在这些场景中，无论对象大小如何，性能都以对象或请求命中率来衡量。

简洁的参数化对于确保可处理性至关重要。为此，2DIO (a) 将任意非无记忆IRD分布近似为粗粒度的分段概率密度函数 (PDF)，并且 (b) 将数值分布限制在相对较短的 IRD 范围内，同时使用通用IRM频率分布（例如齐夫分布）近似长持续时间 IRD 的尾部。这种精简的参数化创建了标准化、可复现的轨迹配置文件，从而实现一致的跨系统评估；同时，也能将工作负载行为的专有测量结果提炼为一种更便于公开或在组织间共享的形式。

1.3 贡献

2DIO 的贡献包括：

1. 使用简洁的参数三元组来编码复杂的工作负载特征，只需少量数值；2. 允许在实验中对参数空间进行"扫描"，从精确复现真实世界模式到构造假设性的"如果……会怎样"场景，生成一系列复杂的 LRU 缓存行为；3. 在任意缩放占用空间和长度的同时复现这些行为，从而支持在不同规模下进行真实且具有压力的系统基准测试。

本文其余部分安排如下：第 2 节介绍工作负载模型和真实世界轨迹行为的背景。第 3 节概述整体方法，并介绍两个主要算法。第 4 节说明设计与实现细节。第 5 节评估 2DIO 的保真度、可配置性和可扩展性，并讨论其局限性。第 6 节回顾相关研究，第 7 节总结全文。

2 背景

2.1 工作负载模型

长期以来，缓存算法的特点一直体现在它们如何利用缓存项的近期性和频率，也就是说，分别依据某个项上次出现距今的时间，以及该项过去出现的频率来进行判断。最近最少使用仅依据近期性做出驱逐决策；而最不经常使用（LFU）则仅依据频率。更广泛的缓存算法（例如先进先出（FIFO）、CLOCK）主要基于近期性，但也会对频率做出响应。

在描述这些模型时，我们假设占用空间为 M个不同项（例如，一个大小为 4096 字节的块）；缓存最多可容纳其中 C 个，绝不会更多；并有一个引用流r_1，r_2，$\cdots$，表示对集合 $\{0, 1, \cdots M-1\}$ 中各项的访问。遵循先前的研究 [34, 35],，我们将引用流模拟为一个离散时间过程，并以距离（访问次数）为单位测量时间 t。例如，引用 r_4 与 r_1 之间的距离为 3。

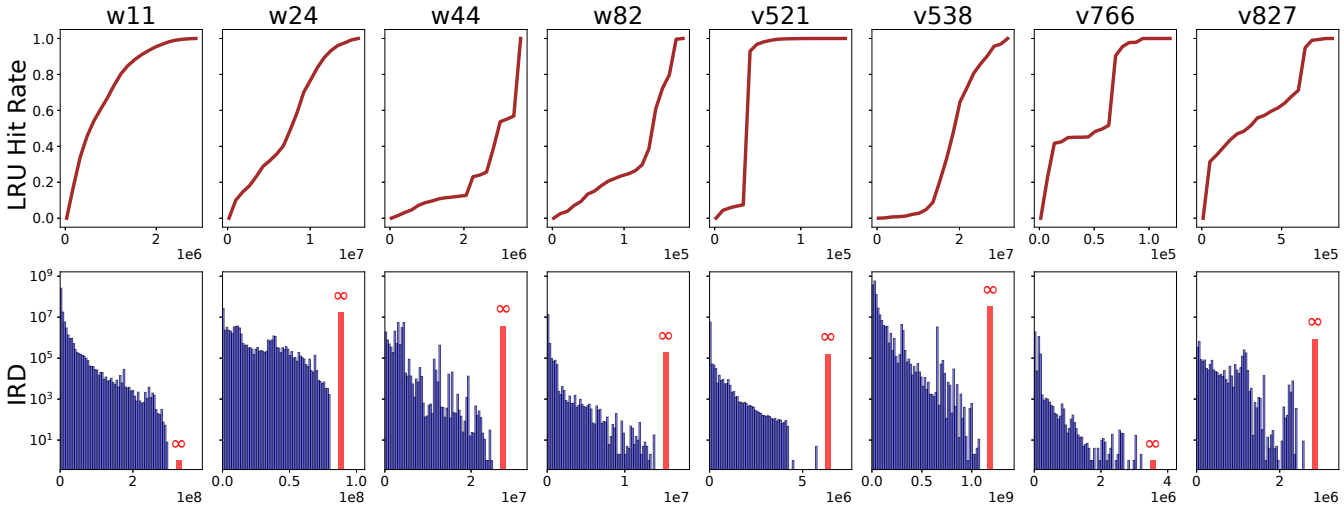


图 4. 阿里云和 CloudPhysics 真实跟踪记录的 LRU 命中率曲线和 IRD 直方图。

独立引用模型。IRM [1, 5] 仅通过项目频率来刻画访问流；每个项 i 都被赋予权重 w_i，$i \in \{0, 1, \cdots M-1\}$，并且 $\sum_i w_i = 1$。因此，对 i 的访问以概率w_i独立选取。

独立引用模型（IRM）工作负载广泛用于微基准测试 [4, 6, 10, 18], , 通常采用其简化形式。在热/冷模型中 [25]，$r_H M$ 热点项以 λ_H的速率被访问，其余（$1 - r_H$）M冷项则以 $1 - \lambda_H$ 的速率被访问；在齐夫模型中，λ_i 对应于 Zipf分布。

依赖引用模型。或者，也可以忽略频率[2]，并通过单个项的多次访问之间的距离分布来描述访问流。

这种方法已广泛用于缓存建模，即根据工作负载特征预测缓存性能 [9, 13, 20, 21, 28, 40, 41]。

对于这种距离，人们采用了两种度量：栈距离 [20] (SD) 和引用间距 [7, 8] (IRD)。给定对同一项的两个连续引用 r_i 和 r_j，IRD 指的是它们在引用流中的间隔（即 $j - i$），而 SD 是分隔这两次访问的访问所引用的唯一项数量（即$|\{r_{i+1}, r_{i+2}, \cdots r_{j-1}\}|$）。

AET 近似。SD 与 LRU 命中率之间存在直接对应关系：如果两次引用之间的 SD 为 C，那么第二次访问将在大小为C 或更大的任何 LRU 缓存中命中。

这种对应关系可以近似表示为 IRD 与最近最少使用未命中率 [11, 14] 之间的相关性：如果已知项目在被驱逐前留在缓存中的平均时间（平均驻留时间），以及请求引用间距离的累积分布函数 $F(t)$，便可近似得出缓存大小 C（即栈距离）。由此可得 that

$$C = \int_0^{AET(C)} P(t)dt,$$

其中，$P(t) = 1 - F(t)$ 表示某个项目在 t 之前未被再次使用的概率，而 $AET(C)$ 表示该缓存中项目的平均驱逐时间。

缓存大小为 C 时的未命中率，是重用时间大于 $AET(C)$ 的概率：

$$P_{miss}(C) = P(AET(C)).$$

这称为 Che 近似 [11],或 AET 近似 [14]。

2.2 真实块跟踪特征

独立引用模型（IRM）很可能适用于描述内容分发网络（CDN）等场景中的工作负载：大量独立来源汇聚为单一请求流，从而"淹没"单个流中引用之间的相关性。块存储工作负载则有所不同。它们的引用通常由一个或少数几个应用程序驱动，因此保留了应用程序行为所造成的引用间相关性。

表 1 列出了阿里云（v521、v538、v766、v827）和 CloudPhysics（w11、w24、w44、w82）的若干跟踪记录。之所以选取这一子集进行后续分析（第 5.1、5.3 节），是因为它们展现出多样的缓存行为。长度和占用空间是在将每条跟踪记录转换为 PARDA [21] 格式[3] 后测得的。

图 4 展示了它们的 IRD 直方图及相应的 LRUHRC。可以看出，HRC 高度非凹，具有陡峭的区段（陡变区）和平坦的区域（平台期）。这些分别对应底层 IRD 分布中的尖峰 和空洞 。从零开始的空洞可能是由操作系统

[2]频率不会影响 LRU 性能，但会影响先进先出和 CLOCK，并且可能对较新的替换算法产生更大的影响。

[3]一系列不含额外元数据的 64 位引用，用于缓存模拟。

表 1. 后续分析所选的轨迹子集。

轨迹标识符	长度（N）	占用空间（M）
w11	296893045	2992519
w24	81762918	16487648
w44	25257814	3679382
w82	14198758	189785
v521	5974956	158018
v538	1204044775	33006370
v766	3335779	124146
v827	3198158	851527

缓冲缓存 [38], 吸收低 IRD 的访问，而尖峰 可能是由类似扫描的行为造成的。

其中少数表现出"行为良好"、类似独立引用模型的特征，例如，w11 的 IRD直方图呈下降趋势，对应于一条凹形命中率曲线。

所有轨迹中都显示出相当比例的"单次命中对象"，它们在轨迹[4]的整个持续期间内仅被引用一次，之后再也不会被访问。在 IRD 测量中，这些访问会被记录为 IRD ∞。

3 2DIO 框架

2DIO 核心中的轨迹生成算法可以视为一种简单的离散事件模拟：每个对象反复执行以下操作：(a) 访问自身一次；(b) 从 IRD 分布中选择休眠时间 t；(c) 休眠 t。由于我们关注的是访问之间的距离，而非实际时间，因此对于所生成轨迹中的每个位置，这些算法始终会恰好生成一次访问。

3.1 从 IRD 生成 2DIO

算法 1 展示了一种根据给定 IRD 分布生成轨迹的有效方法。

输入。该算法接受以下输入参数：
(1) f – IRD 分布；(2) M – 占用空间；(3) N – 轨迹长度

输出： 合成轨迹 π_s。

初始化： 从 f 中抽取 M 个休眠时间 t。对于每个非无穷的 t，为其分配一个唯一地址 a，并将该对添加到优先队列（堆）中。

轨迹生成。 为了生成单个引用，从 IRD 分布中抽取一个 t。如果 $t = \infty$ 从单例池中抽取一个项目（地址超出 M），否则从优先队列底部取出配对 t_0， a_0，生成一次访问 a_0 到轨迹中，并将 a_0的休眠时间更新为 $t_0 + t$，然后放回优先队列。算法持续执行，直到轨迹长度达到 N。

[4]CloudPhysics 和阿里云的跟踪记录分别持续 7 天和 30 天；超过此时间段的任何访问都不太可能命中缓存。

算法 1：Gen-from-IRD：从.....生成轨迹
给定的引用间距（IRD）分布 f。

输入： f， M， N
输出： π_s
堆 $\leftarrow \emptyset$
$a \leftarrow 0$
当堆.*size* $< M$ **时**
  $t \overset{\text{独立同分布}}{\sim} f$
  如果 $t \neq \infty$，**则**
    堆.insert($< t,\ a >$)
    $a \leftarrow a + 1$
$\pi_s \leftarrow \emptyset$
对于 $j = 0 \ldots N-1$，**执行**
  $t \overset{\text{独立同分布}}{\sim} f$
  如果 $t = \infty$，**则**
    π_s.追加(a)
    $a \leftarrow a + 1$
  else
    $< t_0,\ a_0 >\leftarrow$ 堆.pop()
    π_s.append(a_0)
    堆.replace($< t_0 + t,\ a_0 >$)
返回 π_s

如果只关注调整最近最少使用或其他完全基于近期性的命中率曲线，那么该算法就足够了，因为仅凭 IRD 便可决定这些缓存策略的性能 [8, 11]。然而，先进先出和 CLOCK 等策略还会响应频率，而不仅仅是近期性。因此，我们描述了 Gen-from-IRD 的一种扩展，该扩展能够引入可调的流行度偏斜。

3.2 基于 IRD+IRM 的 2DIO 生成

Gen-from-IRD 高度依赖输入 IRD 分布的准确性，通常需要其足够长，以呈现长尾。超过驱逐时间的 IRD 不会影响 非凹形 命中率曲线特征，因此可以改用简洁的独立引用模型输入（例如，齐夫分布（1.2））进行近似，同时显式强制执行频率分布。

我们介绍算法2：Gen-from-2D。它本质上源自 Gen-from-IRD，但融合了独立引用模型到达过程。在此过程中，项目频率遵循分布g，此类项目的每次生成都以概率 P_{IRM} 触发。该过程如图 5 所示。将短间隔独立引用距离分布 f 与独立引用模型长尾 g 融合后，生成器便能同时调节这两个维度，从而塑造各种缓存策略的行为。

输入： 该算法需要 5 个输入：
(1) P_{IRM} – 独立引用的比例 (2) g – 项目频率分布
(3) f – 以分段量化表示的 IRD (4) M – 占用空间
(5) N – 轨迹长度

算法2：Gen-from-2D：从以下分布生成轨迹
IRD 分布 f的概率为 $1-P_{IRM}$，并
项目频率分布 g 的概率为
概率 P_{IRM}。

输入： P_{IRM}， g， f， M， N

输出： π_s
堆 $\leftarrow \emptyset$
$a \leftarrow 0$
当堆.$size$ $< M$ **时**
  $t \overset{独立同分布}{\sim} f$
  如果 $t \neq \infty$**那么**
    堆.insert($< t, a >$)
    $a \leftarrow a + 1$
$\pi_s \leftarrow \emptyset$
对于 $j = 0 \ldots N-1$ **执行**
  $Random \overset{独立同分布}{\sim}$ 均匀分布(0,1)
  如果 随机数 $< P_{IRM}$ **则**
    addr $\overset{独立同分布}{\sim} g$
    π_s.追加(地址)
  else
    $t \overset{独立同分布}{\sim} f$
    如果 $t = \infty$ **那么**
      π_s.追加(a)
      $a \leftarrow a + 1$
    else
      $< t_0, \ a_0 >\leftarrow$ 堆.pop()
      π_s.追加(a_0)
      Heap.replace($< t_0 + t, \ a_0 >$)
返回 π_s

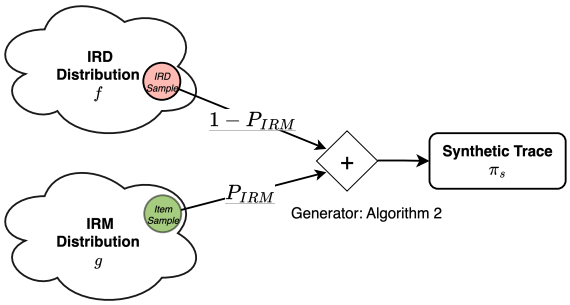


图 5. IRD + IRM 轨迹生成使用算法2：以概率 P_{IRM} 从独立引用模型过程中抽取引用，并以 $1-P_{IRM}$ 从 IRD 更新过程中抽取引用。

输出： 合成轨迹 π_s。
初始化：参见 Gen-from-IRD。
轨迹生成：该算法与 Gen-from-IRD 类似。不过，每次迭代时，它会以 P_{IRM} 的概率从 g中选择一项。

3.3 参数定制

按照算法2，2DIO 接收 5 个输入： P_{IRM}、 g、 f、M 和 N。我们将前三个定义为轨迹配置，表示为三元组 $\theta = P_{IRM}$、 g、 f。我们将由 f生成的项称为相关到达，将由 g 生成的项称为独立到达。 M 和 N 是尺度参数，不会影响（归一化后的）缓存行为。

2DIO 的目标是使用紧凑的 θ 从头创建合成轨迹；它也可以通过校准 θ 以匹配相似行为来重现真实跟踪。对于这两种情况，2DIO 都旨在近似目标缓存行为，而非精确重现它；并且，同一个精度目标可能通过多种 θ选择来实现。

本质上，准确性与简洁性之间存在权衡：复现真实轨迹可能需要很长的轨迹或经验分布 f（如图3所示），而生成新的行为只需一个简洁的分布 f（例如，少于 10 个值，并在目标百分位处设置尖峰），因此具有很高的可操作性。

3.3.1 配置 IRD 分布。 f指定一种 IRD 分布，算法据此为每个项目抽取休眠时间。一个典型场景是构建一条轨迹，使缓存大小约为数据集的 5% 时能够获得缓存收益。van den Berg 和 Towsley（1993）已证明 [33], ，这无法通过调整频率分布实现；必须对底层 IRD 分布进行塑形，例如在特定值处形成尖峰/空洞 。

我们注意到，这可以通过第 2.1 节中提出的平均驻留时间近似公式 [11, 14] 来实现：

$$C = SD(\tau_c) = \int_0^{\tau_c} P(t)dt, \tag{1}$$

由于公式（1）是双射的，IRD 域上的每个平均驱逐时间 τ_c 都唯一映射到一个缓存大小 C，反之亦然。
相应的命中率为：

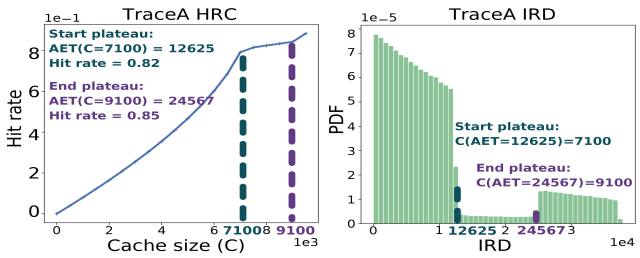
$$P_{hit}(C) = 1 - P(\tau_c), \tag{2}$$

因此，每个 τ_c 也会为其所近似的栈距离 C 产生唯一的命中率。

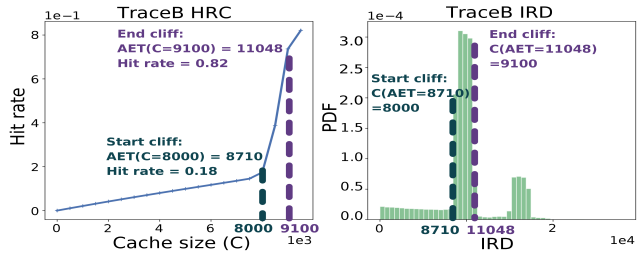
图 6 展示了 TraceA 的 IRD 中的一个空洞与其命中率曲线中的一个平台之间的对应关系，以及 TraceB 的 IRD 中的一个尖峰与其命中率曲线中的一个悬崖之间的对应关系。为利用这种对应关系，2DIO 提供了模型精确接口**fgen**。设 IRD 分布 $f(X = i)$ 是一个具有有限支集 $i \in \{1, 2, \cdots \ k\}$ 的概率质量函数。设**fgen(**k,I ϵ**)** 是一个生成 f 的函数，为集合 I 中的元素分配更高的概率（尖峰），并为其补集 I 中的元素分配更低的概率（空洞）。这些空洞的总概率质量为 ϵ，从而得到如下分布：

$$f(i) = \begin{cases} \frac{1-\epsilon}{|\mathcal{I}|}, & i \in \mathcal{I} \\ \frac{\epsilon}{k-|\mathcal{I}|}, & i \in \overline{\mathcal{I}} \end{cases} \tag{3}$$

2DIO 随后在自动调优的 IRD 样本空间 S := $\{1, 2, \cdots \ T_{max}\}$ 上拟合 f （参见第 4.1 节），得到的分布具有



（a）HRC 中从 C_i 到 C_j ↔ 的平台期对应于 f 中从 $AET(C_i)$ 到 $AET(C_j)$ 的空洞



（b）HRC 中从 C_i 到 C_j ↔ 的陡变区对应于 f 中从 $AET(C_i)$ 到 $AET(C_j)$ 的尖峰

图6。 HRC 中的平台期和陡变区与 IRD 中的空洞和尖峰之间的对应关系。合成轨迹示例——轨迹A和轨迹B：最近最少使用 HRC（左），以及对应的 IRD 分布（右）。

在独立参考距离区间处出现尖峰：

$$\{[i \times \frac{T_{max}}{k},\ (i+1) \times \frac{T_{max}}{k}], \quad \forall i \in \mathcal{I}\}$$

以及其他位置的空洞，它们表现为缓存大小区间中的 HRC陡变区：

$$\{[SD(i \times \frac{T_{max}}{k}),\ SD((i+1) \times \frac{T_{max}}{k})], \quad \forall i \in \mathcal{I}\}$$

并且在 C 域的其他位置也会出现平台期，遵循公式（1）。

3.3.2 选择独立引用模型分布。 g 指定一个项目频率分布，算法从中直接选择项目并将其添加到轨迹中。g 的选择决定了项目频率分布（即流行度）。P_{IRM}则决定了这些到达在生成的轨迹中所占的比例。

请注意，f 是一个有限分布，生成的 IRD 介于 0 和最大值 T_{max}之间。拟合 P_{IRM} 和 g需要检查大于T_{max} 的值对应的 IRD 分布。最终的 IRD 分布由相关到达和独立到达的 IRD 合并而成，后者形成尾部。

3.3.3 探索所需轨迹行为的参数空间。在实际使用中，用户无需了解底层拟合模型。2DIO 提供交互式可视化工具，帮助调整参数。用户可以：(1) 从内置的 cachesim 库中选择一种缓存算法；(2) 从直观（或默认）的输入值开始；然后 (3) 拖动一个滑块或输入数字来更改任意单个参数，并立即观察实时重新计算的命中率曲线。

虽然即时模拟命中率曲线可能会消耗大量资源，但在此过程中使用较小的跟踪占用量 M 和长度 N（例如分别为 100 和 10000）可最大限度地减少开销。完成调优后，这些参数可在不同规模下重复使用，同时不影响保真度（参见第5.3节）。

4 实现

2DIO 软件包[5] 包含一个用 C++编写的命令行工具 trace-gen，用于生成轨迹；还包含一个用于缓存模拟与分析的 Python 库。trace-gen 的输入包括占用空间 M、轨迹长度 N和轨迹配置 $\theta = P_{\text{IRM}},\ g,\ f$。

由于我们关注块级HRC，因此假定所有访问单元大小均匀，这在存储缓存评估中很常见。

本节详细介绍 IRD 和独立引用模型采样器的实现，介绍若干默认轨迹配置文件，并提供用于生成具有目标缓存行为的轨迹的示例命令。

4.1 相关到达采样

trace-gen 通过**fgen** 接口接受 f输入。我们为该工具包预先配置了六种默认轨迹配置文件。ˆ 这些配置概括了若干现实世界IRD模式，标记为 $f_{\hat{a}}$ 至 f_g（见附录A表6）。它们对应的HRC和IRD 分布如同一附录中的图11 所示。

自动调优的 IRD 样本空间。 f拟合到一个自动调优的k分箱样本空间 S。算法2中的 IRD采样器以概率 $f(i)$ 选择分箱 i，并从该分箱中均匀分布地采样一个 IRD。

为确保 S 与从目标轨迹 π_s中经验测得的 IRD 分布相匹配，T_{max}基于M 进行自动调优，使得抽取的 IRD 样本均值等于 M。因此有

$$M = \sum_{i=1}^{k} b_i \cdot f(i),$$

其中，b_i 是分箱 i的中点：

$$b_i = \frac{(2i-1)}{2} \times \frac{T_{max}}{k}.$$

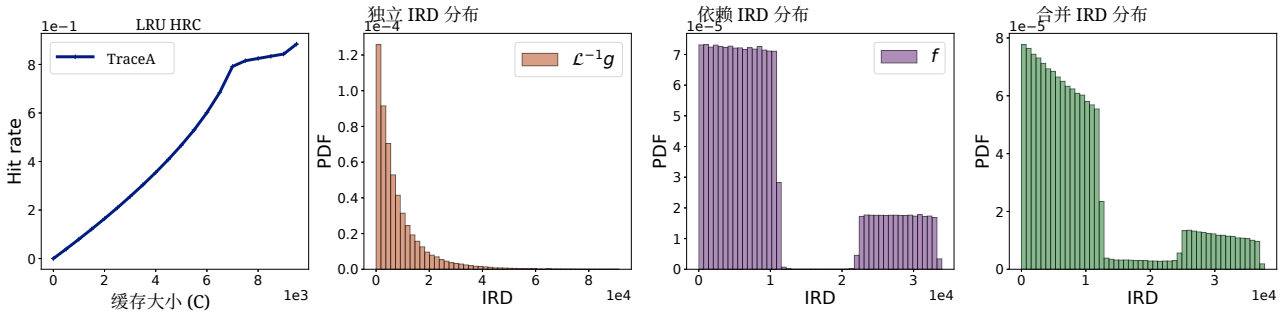
因此，T_{max} 可通过以下方式求解：

$$T_{max} = \frac{2Mk}{\sum_{i=1}^{k}(2i-1) \cdot f(i)}.$$

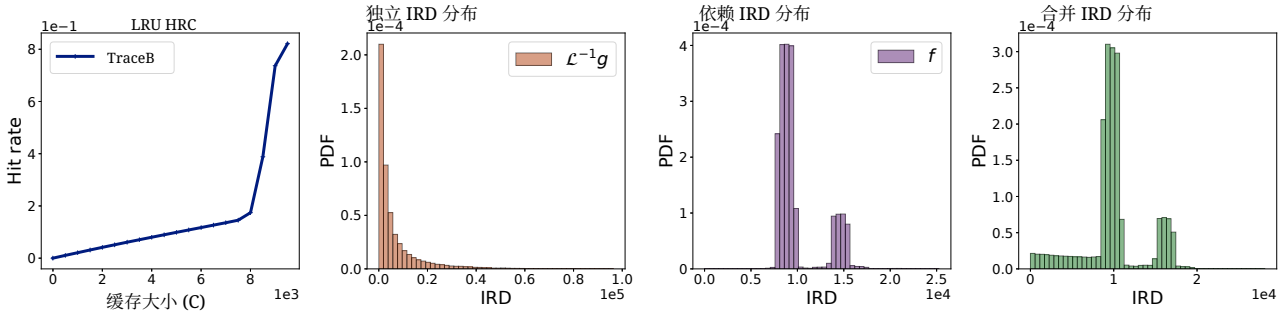
4.2 对独立到达进行采样

独立到达的项目频率分布 g 可在样本空间 U 上遵循齐夫分布、帕累托分布、正态分布或均匀分布。trace-gen 接受字符串形式的 IRM 类型作为输入；若未指定，则默认使用 "zipf" $\alpha = 1.2$。

[5]可访问 https://github.com/Effygal/trace-gen；工件也可在 https://zenodo.org/records/17202588 获取。



(a) TraceA： $P_{IRM} = 0.1$, g = 齐夫分布(1.2)， f通过经验拟合得到一个简单的三类 IRD 分布



(b) TraceB： $P_{IRM} = 0.1$, g = 帕累托分布(2.5,1)， f通过经验拟合得到一个 15 类 IRD 分布

图 7. TraceA 和 TraceB 的命中率曲线，并分别展示依赖到达、独立到达和合并到达的 IRD 直方图。$\mathrm{L}^{-1}g$ 表示 g的拉普拉斯逆变换，将其频域表示映射到 IRD 域。

对于指定的每种 IRM 类型，都可以单独配置相应的参数，从而生成如表 2 所示的概率质量函数（PMF） g 。IRM 采样器根据相应的概率质量函数，以概率 $g(i)$从 U 中选择一个项目 i 。

表 2. trace-gen 支持的独立引用模型类型及相应的概率质量函数表达式。

IRM 类型	可配置	PMF
齐夫型 帕累托分布	α $\alpha,\ x_m$	$g(i) = \left(\frac{1}{i}\right)^{\alpha}$ $g(i) = \left(\frac{x_m}{i}\right)^{\alpha}$
正态分布	$\mu,\ \sigma$	$g(i) = \frac{1}{\sigma\sqrt{2\pi}}$ 指数 $\left(-\frac{(i-\mu)^2}{2\sigma^2}\right)$
均匀分布	$a = 0,\ b = M-1$	$g(i) = \frac{1}{M}$
经验分布	计数 n_i	$g(i) = \frac{n_i}{\sum_j n_j}$

4.3 二维生成命令

ˆ 要使用 $f = f_b$ 和默认的 g = 齐夫分布(1.2)生成 TraceA，请使用：

```
trace -gen -m 10000 -n 1000000 -f b -p 0.1
```

独立到达、相关到达和合并到达的命中率曲线及 IRD 如图 7(a) 所示。若要使用明确指定的 f = 帕累托分布(2.5,1) 生成 TraceB，请使用：

```
trace -gen -m 10000 -n 1000000 -f fgen
      :15:0.01:1    ,3 ,5 ,9   -g  pareto :2.5 ,1    -p
      0.1
```

相同的可视化效果见图 7(b)。在这两条轨迹中，90% 的到达请求从 f采样，只有 10% 从 g采样，因此独立影响较小。结果是，两条轨迹都呈现出高度非凹 的命中率曲线。

如果指定了 I/O 大小分布，可以将其载入以改变请求大小。例如，–sizedist 1,1,1:1,3,4 表示 1、3 或 4 个块的请求出现概率相等。不过，这样做可能会影响精心构造的 IRD 分布。有关这一限制，请参见第 5.4 节。

5 评估

本节从保真度、可配置性和可扩展性等方面评估 2 DIO。涉及真实跟踪的评估均在第 2.2 节所述的同一子集上进行。所有缓存模拟均使用我们内置的 c achesim 库完成。

由于 LRU 命中率完全由近期性决定， [7, 20], 我们使用 LRU HRC 来评估我们的方法

生成/重现近期性的准确度，而现有生成器尚未实现这一点。

所有模拟均在 Ubuntu 24.04 LTS 上运行，使用 AMD Ryzen 5 7600 六核处理器和 62 GB 内存。GAN 超参数搜索和训练任务在一个 HPC 节点上运行，该节点配备 1 个 NVIDIA Tesla V100-SXM2 GPU、2 个 CPU 核心和 8 GB 分配内存。

5.1 使用简洁参数复现真实轨迹

保真度通过块粒度的 HRC 准确度进行评估。为此，我们为每条轨迹校准配置文件 θ，旨在重新生成能够产生相似 HRC 的合成轨迹——见表 3；所有配置文件均使用少于 10 个数值表示。随后，我们在较小的足迹 $M = 100$ 和长度 $N = 10$k 上重新生成每条轨迹，只有一条除外 [6]，，从而有效降低缓存模拟成本。

表 3. 用于"仿造"每条真实世界轨迹的简约轨迹配置。**fgen**遵循公式 (3)。

轨迹标识符	P_{IRM}	g	f
w11	1.0	齐夫分布 (1.3)	None
w24	0.45	齐夫分布 (1.2)	**fgen**(30, [1, 2], $5e-3$)
w44	0.0	None	**fgen**(30, [9, 13, 17, 19], $2.5e-2$)
w82	0.2	齐夫分布 (1.2)	**fgen**(100, [12, 13, 19], $1e-3$)
v521	0.0	None	**fgen**(100, [2], $2e-3$)
v538	0.1	齐夫分布 (1.2)	**fgen**(40, [3, 4], $5e-3$)
v766	0.0	None	**fgen**(40, [0, 5], $5.7e-3$)
v827	0.2	齐夫分布 (1.2)	**fgen**(60, [0, 13], $5e-3$)

图 8 比较了每条 2DIO 生成的轨迹（橙色虚线）与其对应原始轨迹（蓝色实线）的 HRC。相对于占用空间 M归一化的缓存大小下，所有相对悬崖 和平台 位置均得到保留，仅存在轻微偏差。

与现有工作的比较。 TraceRaR [19] 和生成对抗网络（GAN） [43] 代表了针对 I/O 回放准确性进行优化的先进轨迹合成方法。我们在此评估此类优化是否也能保持 HRC 保真度。

TraceRaR。 TraceRaR [19] 扩展给定轨迹，同时保留请求速率、读写比和偏移特征，以实现有效的扩展回放。

我们使用 TraceRaR 将每条轨迹扩展至原始长度的两倍。在合成输出中，前半部分与原始轨迹完全相同，因此产生相同的命中率曲线。然而，后半部分看起来符合独立引用模型；将这部分追加到真实轨迹会扰乱其近期性。因此，图 8 中的绿色虚线 HRC 与原始曲线明显偏离。

LLGAN。 LLGAN [43] 对生成器（G）和判别器（D）均采用单层长短期记忆网络（LSTM）架构，并且都针对交叉熵损失进行了优化。

由于原始实现未公开，且超参数调优具有工作负载特异性，我们复现了论文提出的设计[7]。每条轨迹均在两个维度上进行训练：⟦逻辑块地址、长度⟧[8]。我们使用 Optuna[2],进行了多轮每轮 50 次试验的超参数搜索，寻找使 D 损失稳定在 0。5 左右且生成器损失最小的组合，这通常表明生成对抗网络训练良好。表4总结了每条轨迹的优化超参数，包括用于确保可复现性的随机种子。

合理性检查。 在对 HRC 准确度得出任何结论之前，我们首先根据原论文使用的评估指标——最大均值差异[9]（MMD^2）——验证 LLGAN合成轨迹。该指标计算逻辑块地址和长度上的联合分布差异。表4的最后两列报告了训练 D 损失和所得的 MMD^2；$MMD^2 \to 0$ 表明，合成逻辑块地址和长度的联合分布接近原始轨迹——具体的可视化比较见附录 B 图 12。

图 8 中的红色虚线曲线表示 LLGAN 生成的命中率曲线。值得注意的是，准确复现 LBA 和长度分布并不意味着 HRC 保真度高。例如，尽管合成 w44 的 MMD^2,较高，但其生成的 HRC 与原始 HRC 最为接近；而合成 w11 的 MMD^2最低，生成的 HRC 却明显不匹配。

5.2 创建涵盖各种"假设情景"的跟踪记录

典型的微基准测试 [4, 6, 10, 17, 18, 22] 主要通过少数几个粗粒度参数来模拟工作负载，例如读写比例、对象大小，或数据库、文件、Web 服务器等更宽泛的"工作负载特征"。缓存所观察到的时空局部性由这些聚合参数自然决定，因此这些工作负载在块级实际上与缓存无关。很少有微基准测试 [4, 17, 18] 支持指定 IRM 块访问频率，例如齐夫分布、帕累托分布或自定义分区分布。表 5 汇总了各工具支持的随机类型，这些类型大多仅限于独立访问模式。[10]

2DIO 的一项关键优势是能够调整全方位的参数，从而生成连续变化的工作负载

[6]w44 可能需要 $M \geq 10$k 和 $N \geq 1$m，才能重现高分辨率 HRC。

[7]可通过 https://github.com/Effygal/gan-io 获取[8]只有逻辑块地址（逻辑块地址）和 I/O 大小（长度）与缓存性能相关。[9]使用 RBF 核和中位带宽计算 σ。[10]fio 和 2DIO 支持异构请求大小（例如，多块请求），会引入顺序访问并触发类似扫描的缓存行为，但扫描位置仍难以确定

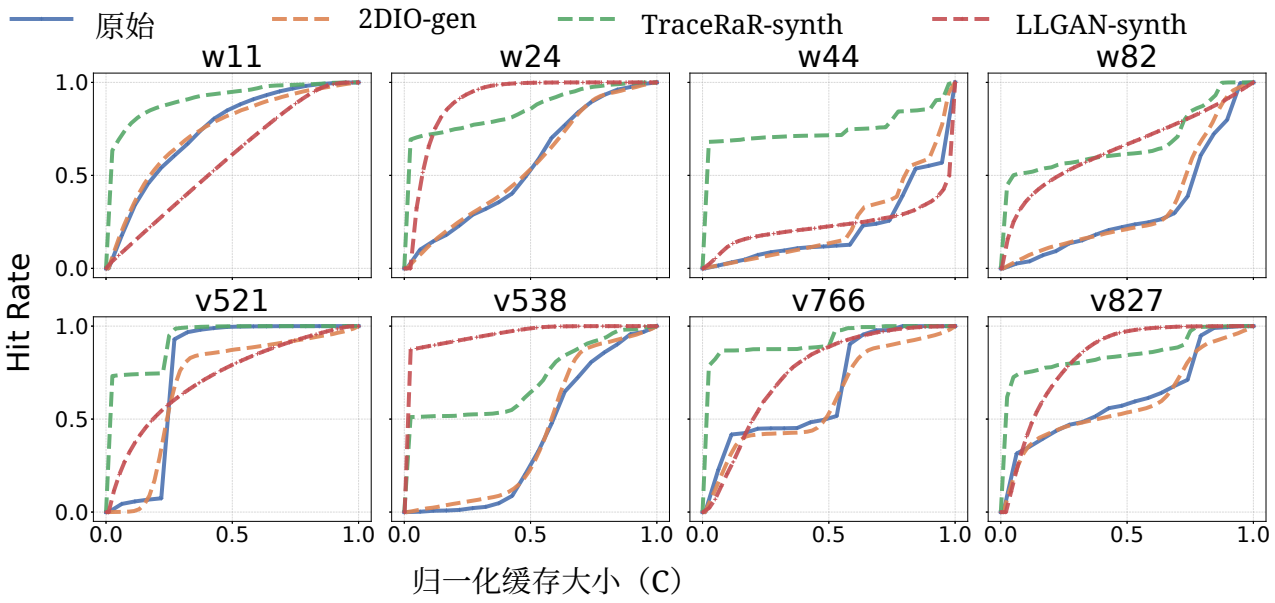


图8. 2DIO 与 TraceRaR 和 LLGAN 合成轨迹相比，复现真实世界轨迹HRC。

表4. 用于 LLGAN 合成的 Optuna 优化超参数及其所得 D 损失和 MMD^2。

轨迹	随机种子	隐藏维度	潜在维度	批量大小	序列长度	训练轮数	G 学习率	D 学习率	G 更新次数	D 更新次数	D 损失	MMD^2
w11	77	126	10	64	12	30	0.000132	0.000454	1	2	0.5411	0.005128
w24	77	108	10	128	12	30	0.000090	0.000397	2	3	0.5308	0.071148
w82	42	121	10	128	12	30	0.000075	0.000108	1	2	0.4856	0.047067
w44	42	100	16	128	12	30	0.000248	0.000139	3	2	0.5600	0.060406
v521	77	101	10	64	12	30	0.000099	0.000157	1	3	0.5079	0.118840
v538	42	102	10	64	12	30	0.000415	0.000281	1	3	0.5510	0.016633
v827	77	108	10	128	12	30	0.000090	0.000397	2	3	0.5099	0.080228
v766	77	111	10	64	12	30	0.000254	0.000141	1	3	0.4818	0.013860

表 5. 可用工具支持的随机类型。此处仅展示合成 I/O 生成，这是其完整功能集中的一小部分。

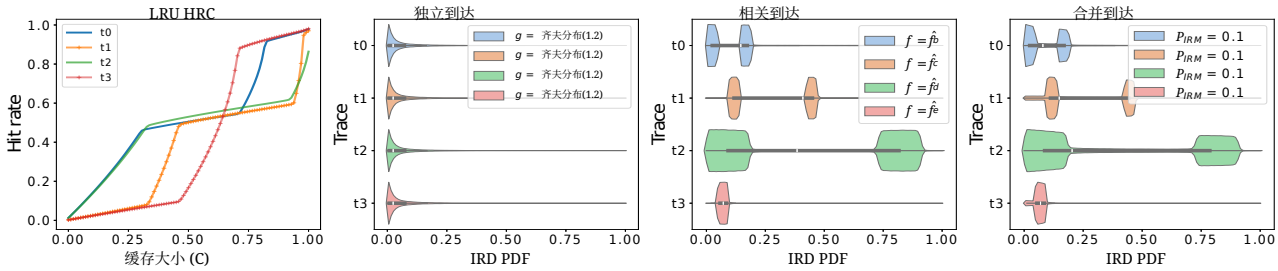
Tool	独立						依赖		混合
	齐夫分布	帕累托分布	分区	正态分布	均匀分布	经验分布	顺序	IRD	
IOzone [22]	✗	✗	✗	✗	✗	✗	✗	✗	✗
Bonnie++ [6]	✗	✗	✗	✗	✗	✗	✗	✗	✗
IOmeter [18]	✗	✗	✓	✗	✗	✗	✗	✗	✗
Sysbench [17]	✓	✓	✓	✗	✗	✗	✗	✗	✗
FIO [4]	✓	✓	✓	✓	✓	✗	✓	✗	✗
2DIO	✓	✓	✓	✓	✓	✓	✓	✓	✓

行为。为评估这一点，我们使用 $\hat{}\,f_b$ 到 $\hat{}\,f_g$[11]（见第 4.1 节）生成 12 条示例轨迹 t0–t11，每条轨迹的占用空间为 $M = 10$ k，长度为 $N = 1$m。图 9 分别展示了独立到达、依赖到达和合并到达的 IRD，以及由此产生的 HRC。每个子图都说明了单独调节 f、g 和 P_{IRM} 会影响模拟的命中率曲线。图 9(a) 展示了移动 f 中尖峰位置如何导致悬崖和平台位置发生相应变化。在 $P_{IRM} = 0.1$ 固定时，IRD尖峰主要决定命中率曲线的悬崖，而 IRM 流量的影响较小。图 9(b) 展示了切换 IRM 类型 g 对命中率曲线形状的影响。由于 90% 的 $\hat{}$ 到达是独立的，f_f 对合并后的 IRD 影响很小，因此所有轨迹中合并后的 IRD 均呈指数状，命中率曲线也均为凹形。图 9(c) 展示了调整 P_{IRM} 的影响，其中 t7–t11 是在固定 $\hat{}g =$ 齐夫分布 (1.2) 和 $f = fg$ 的条件下生成的；将 P_{IRM} 从 0.1 逐渐增加到 0.9，会使命中率曲线的凹度逐渐增大。

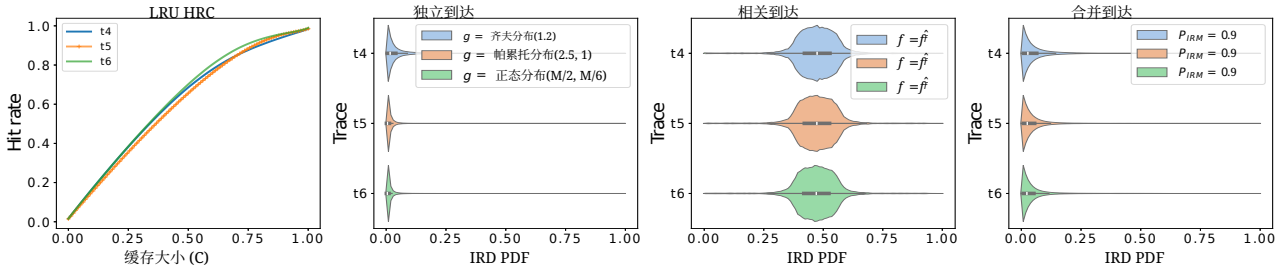
[11] $\hat{}fg =$ **fgen(** (54, [5, 11, 12, 13, 14, 17, 30, 50], $1e-2$) **)**; 尖峰值由人工重新分配。

5.3 保真度持久升降采样

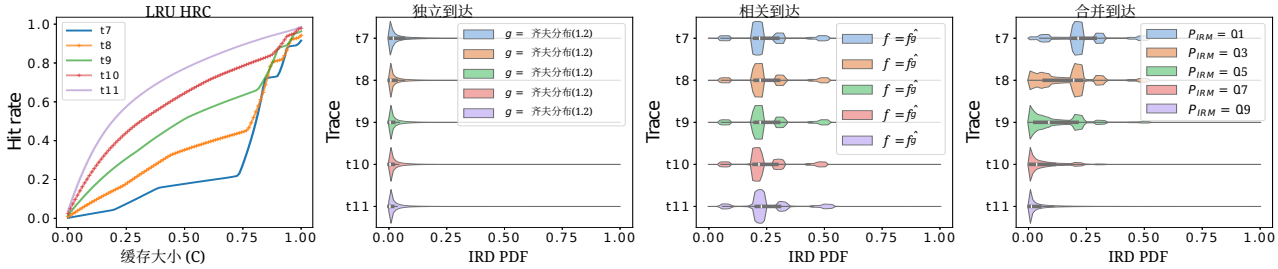
在本次评估中，我们在固定 θ 的条件下，以不同规模生成每条轨迹（见第 5.1 节表 3），并以各规模下相对于原始轨迹的平均绝对误差（MAE）评估命中率曲线的准确性。



(a) 调整 f 以配置 HRC 悬崖和平台的位置



(b) 调整 g 以配置凹形 HRC 曲线



(c) 调整 P_{IRM} 以配置 HRC 凹陷程度

图9。 调整各参数对最终 HRC 的影响，示例采用针对 $M = 10k$ 和 $N = 1m$ 的 2DIO 生成的轨迹。从左到右的三个小提琴图分别展示：(1) 在 IRM 的独立到达条件下的 IRD 概率密度函数；(2) 从输入 f 中采样的相关到达所产生的 IRD 概率密度函数；(3) 合并这两个过程得到的合并后的IRD概率密度函数。

缩放 N 并 M 同时进行。我们将 M 从 10k 缩减至 100，并将 N 从 1m 缩减至 10k，同时保持 N/M的比例固定为 100。图 10(a) 展示了结果。我们将 2DIO 在三种规模下生成的命中率曲线与原始曲线进行了比较。平均绝对误差始终约为 0。03 至 0。05。尽管在最小规模下（$M = 100, N = 10k$），陡变区的分辨率有所"平滑"，但整体缓存行为仍然保持不变。

扩展占用空间 M。 在将 N固定为 100k 的情况下，将 M从 10k 调整至 100，得到图 10(b) 所示的结果，其中平均绝对误差介于 0。02 和 0。05 之间。

扩展轨迹长度 N。 这里将 M 固定为 1k。图 10(c) 展示了当 N 从 10k 变化至 1m 时的命中率曲线和平均绝对误差；平均绝对误差约为 0.04。

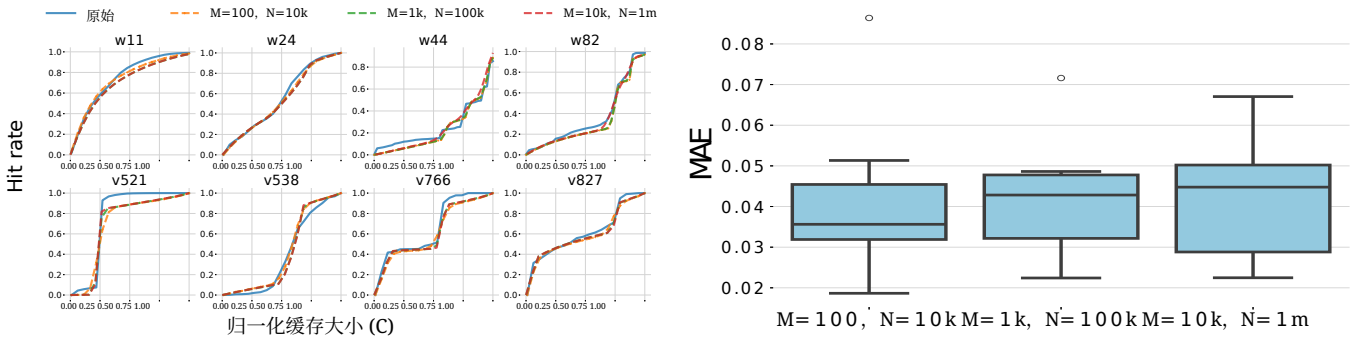
请注意，平均绝对误差不一定会随着尺度增大而改善，也不一定会随着尺度缩小而恶化。准确性主要取决于在初始尺度下对 θ 参数的调优效果。

5.4 局限性

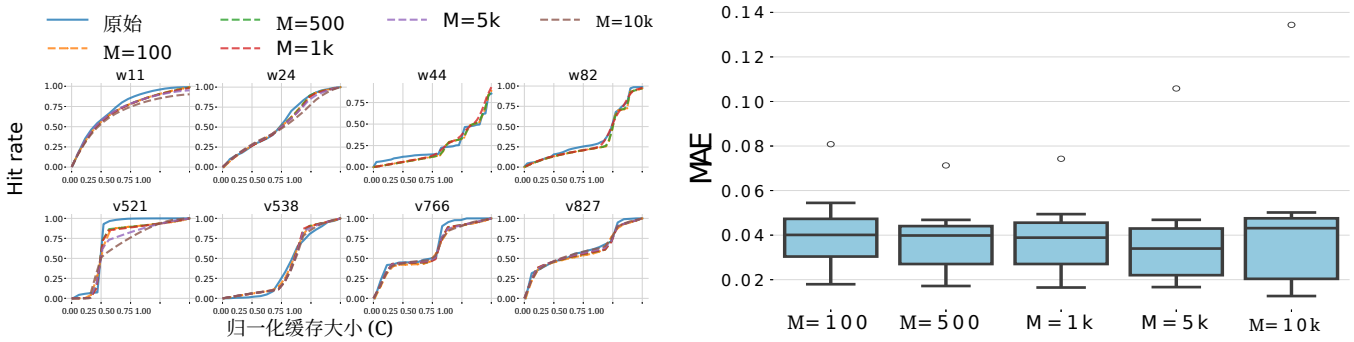
2DIO 生成的跟踪记录遵循 SPC 格式 [32]，并可在任何存储系统上重放。用户可以指定读写比和请求大小分布。然而，指定混合单块引用和多块引用的请求大小分布，实际上会在任意位置引入顺序访问，这可能会扭曲 IRD 样本空间，并使精心构造的尖峰难以处理。

6 相关工作

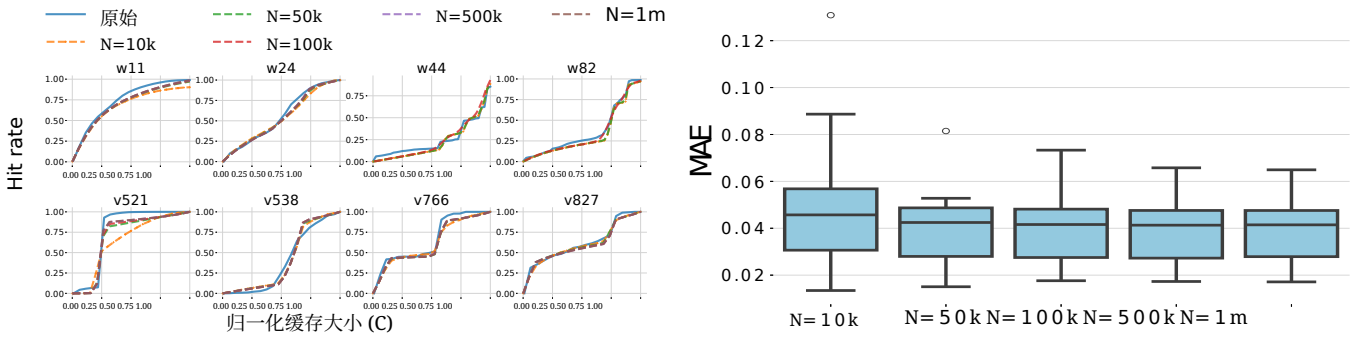
重新缩放模拟。一个密切相关的领域是 Waldspurger 等人提出的缩小规模模拟 [34, 35]，它通过从给定轨迹中采样请求，同时保留其底层栈距离分布，来近似 LRUHRC。



(a) 在固定 N/M 的情况下扩展 M 和 N



（b）固定 N 时的扩展 M



（c）固定 M 时的扩展 N

Figure 10. 可扩展性测试结果。在不同规模的 M 和 N 下，命中率曲线的平均绝对误差（MAE）准确度。

这些工作与 2DIO 的根本区别在于，它们将轨迹转换为命中率曲线；而 2DIO 则反其道而行之，将命中率曲线转换为轨迹。两者之所以相似，是因为它们都依赖于数十年前开创的缓存建模研究（Mattson 等人，1970 [20]；Denning，1968 [7]）。

扩展合成轨迹可以在保留关键分布的同时延长轨迹长度，从而改进调试和性能测试，如 TraceRaR [19] 所示。近来，通过合并来自多个存储单元 [23], 的降采样轨迹，扩展规模已使得在单个节点或小型集群上进行分布式存储评估成为可能，从而生成具有代表性的或基于磁盘容量和放置策略的反事实跟踪记录。

CDN 工作负载合成。 生产环境中的内容分发网络使用了一系列工作负载生成工具 [26, 27, 30] 。最新的 JEDI [27] 能够生成合成轨迹，同时匹配原始轨迹的对象级属性（对象大小分布、流行度分布、请求大小分布）和缓存级属性（对象级和字节级命中率曲线）。其评估结果显示，在各种策略下重新生成工作负载时，命中率曲线模拟误差较小；然而，所有报告的命中率曲线似乎都表明这些工作负载类似于独立引用模型。

7 结论

2DIO 提供了一种实用方法，可生成既真实又可预测的合成轨迹。它将短期近期性和长期频率特征浓缩为简洁的参数三元组，从而生成具有复杂、非凹形命中率曲线的轨迹，其中包括真实工作负载中常见的性能陡变区和平台期。这种参数化足够简洁，可在实验中进行穷举探索，同时又具有很强的可扩展性，能够适配各种待评估系统，而不改变缓存行为。该工具轻量易用，可轻松集成到现有基准测试工具中，从而为跨系统基准测试提供标准化、可复现且可共享的平台。

致谢

我们感谢匿名评审和 shepherd Bhu- van Urgaonkar 提供的详尽且建设性的反馈。本研究的完成部分得益于红帽、VMware/Broadcom 以及美国国家科学基金会 CNS-1910327 项目的支持。

参考文献

[1] Alfred V Aho、Peter J Denning 和 Jeffrey D Ullman。1971年。最优页面置换原理。《*ACM*学报》（*JACM*）18卷1期（1971年），80–93页。[2] Takuya Akiba、Shotaro Sano、Toshihiko Yanase、Takeru Ohta 和 Masanori Koyama。2019年。Optuna：新一代超参数优化框架。载于第*25*届 *ACM SIGKDD* 国际知识发现与数据挖掘会议论文集。2623–2631页。[3] 阿里巴巴。 [无日期]。Alibaba/block-traces。https://github.com/alibaba/ block-traces[4] J. Axboe。2003年。GitHub - axboe/fio：灵活 I/O 测试工具 — github.com。https://github.com/axboe/fio。 [访问日期：08-07-2024]。[5] E. G. Coffman 和 Peter J. Denning。1973年。操作系统理论。Prentice-Hall，新泽西州恩格尔伍德克利夫斯。[6] R. Coker。2003年。Bonnie++ 现为 1.03e（2.0 之前的最后一个版本）！ — coker.com.au。https://www.coker.com.au/bonnie++/。 [访问日期：08-07-2024]。[7] Peter J. Denning。1968年。程序行为的工作集模型。《*ACM*通讯》 11, 5 1968 323 333 10 1145 363095 363141（5[8] 1972）， –。doi：。/。Peter J Denning 和 Stuart C Schwartz。工作集模型的性质。《*ACM*通讯》15卷3期（1972年），191–198页。[9] David Eklov 和 Erik Hagersten。2010年。StatStack：LRU 缓存的高效建模。载于*2010 IEEE* 系统与软件性能分析国际研讨会（*ISPASS*）。IEEE，55–65页。[10] Filebench。 [无日期]。Filebench/filebench：一种文件系统和存储基准工具，使用自定义语言生成多种工作负载。https://github.com/filebench/filebench [11] Hao Che、Ye Tung 和 Zhijun Wang。2002年。分层 Web 缓存系统：建模、设计与实验结果。*IEEE*通信选定领域期刊20卷7期（2002年9月），1305–1314页。doi:10.1109/JSAC.2002.801752[12] Gerhard Hasslinger、Konstantinos Ntougias、Frank Hasslinger 和 Oliver Hohlfeld。2023年。先进先出、基于时钟和 LRU 缓存性能的解析结果与近似结果的范围和准确性。《未来互联网》15卷3期（2023年2月），91。doi:10.3390/fi15030091[13] Xiameng Hu、Xiaolin Wang、Yechen Li、Lan Zhou、Yingwei Luo、Chen Ding、Song Jiang 和 Zhenlin Wang。2015年。{LAMA}：针对键值缓存优化的局部性感知内存分配。载于*2015 USENIX* 年度技术会议（*USENIX ATC 15*）. 57–69.[14] Xiameng Hu、Xiaolin Wang、Lan Zhou、Yingwei Luo、Chen Ding 和 Zhenlin Wang。2016。缓存数据淘汰的动态建模。载于*2016* 年 *USENIX* 年度技术会议（*USENIX ATC 16*）。USENIX 协会，丹佛，CO，351–364。[15] IRCache。2000。首届半年一次的 Web 缓存竞赛。http: //www.ircache.net/n01（存档于 https://web.archive.org/web/*/http: //www.ircache.net/n01）。[16] Minji Kang、Soyee Choi、Gihwan Oh 和 Sang-Won Lee。2020。2r：在闪存存储中高效隔离冷页面。《*VLDB* 会刊》13, 12（2020），2004–2017。[17] Alexey Kopytov。2004。Sysbench：一种系统性能基准测试。*http://sysbench. sourceforge. net/*（2004）。[18] David D Levine。1998。Iometer 用户指南。英特尔服务器架构实验室40（1998）。[19] Bingzhe Li、Farnaz Toussi、Clark Anderson、David J Lilja 和 David HC Du。2017。TraceRaR：用于重放、分析和重新生成跟踪记录的 I/O 性能评估工具。载于*2017* 年国际网络、架构与存储会议（*NAS*）。IEEE，1–10。[20] R.L. Mattson、J. Gecsei、D. R. Slutz 和 I. L. Traiger。1970。存储层次结构的评估技术。《*IBM* 系统期刊》9, 2（1970），78 117 10 1147 92 0078 –。doi: 。/sj. 。Qingpeng Niu、[21] James Dinan、Qingda Lu 和 Ponnuswamy Sadayap-pan。2012。PARDA：一种快速的并行重用距离分析算法。载于*2012* 年 *IEEE* 第 *26* 届国际并行与分布式处理研讨会。IEEE，1284–1294。[22] William D Norcott。2003。Iozone 文件系统基准测试。*http://www. iozone. org/*（2003）。[23] Phitchaya Mangpo Phothilimthana、Saurabh Kadekodi、Soroush Gho-drati、Selene Moon 和 Martin Maas。2024。Thesios：根据 I/O 样本合成准确的反事实 I/O 跟踪记录。载于第 *29* 届 *ACM* 编程语言与操作系统架构支持国际会议论文集，第 *3* 卷。1016–1032。[24] Roman Pletka、Ioannis Koltsidas、Nikolas Ioannou、Saša Tomić、Niko-laos Papandreou、Thomas Parnell、Haralampos Pozidis、Aaron Fry 和 Tim Fisher。2018。管理下一代 NAND 闪存，以实现企业级耐久性和延迟目标。*ACM*《存储汇刊》（*TOS*）14, 4（2018），1–25。[25] Mendel Rosenblum 和 John K. Ousterhout。1991。日志结构文件系统的设计与实现。载于第 *13* 届 *ACM* 操作系统原理研讨会。ACM，太平洋格罗夫，加利福尼亚州，美国，1–15。[26] Anirudh Sabnis 和 Ramesh K Sitaraman。2021。TRAGEN：用于真实缓存模拟的合成跟踪生成器。载于第 *21* 届 *ACM* 互联网测量会议论文集。366–379。[27] Anirudh Sabnis 和 Ramesh K Sitara man。2022。JEDI：用于缓存模拟的模型驱动跟踪生成。载于第 *22* 届 *ACM* 互联网测量会议论文集。679–693。[28] Rathijit Sen 和 David A Wood。2013。基于重用的缓存在线模型。载于*ACM SIGMETRICS*/计算机系统测量与建模国际会议论文集。279–292。[29] Radu Stoica、Roman Pletka、Nikolas Ioannou、Nikolaos Papandreou、Sasa Tomic 和 Haris Pozidis。2019。了解混合闪存控制器的设计权衡。载于*2019* 年 *IEEE* 第 *27* 届计算机与电信系统建模、分析与模拟国际研讨会（*MASCOTS*）。IEEE，152–164。[30] Aditya Sundarrajan、Mingdong Feng、Mangesh Kasbekar 和 Ramesh K Sitaraman。2017。足迹描述符：全球 CDN 中缓存配置的理论与实践。载于第 *13* 届新兴网络实验与技术国际会议论文集。55–67。[31] Jian Tan、Tieying Zhang、Feifei Li、Jie Chen、Qixing Zheng、Ping Zhang、Honglin Qiao、Yue Shi、Wei Cao 和 Rui Zhang。2019。ibtune：大规模云数据库的个性化缓冲区调优。《VLDB 会刊》

《*VLDB* 会刊》12, 10（2019），1221–1234。[32] 存储性能委员会、马萨诸塞大学阿默斯特分校。无日期。SPC跟踪数据：存储性能委员会跟踪数据。https://skulddata.cs.umass.edu/traces/storage/SPC-Traces.pdf。访问日期：2025-01-02。[33] Jacob van den Berg 和 D Toswley。1993。采用最近最少使用或先进先出替换策略及独立引用的两级存储模型的未命中率特性。*IEEE* 计算机汇刊42, 4（1993），508–512。[34] Carl Waldspurger、Trausti Saemundsson、Irfan Ahmad 和 Nohhyun Park。2017。利用微型模拟进行缓存建模与优化。载于 *2017* 年 *USENIX* 年度技术会议（*USENIX ATC 17*）。USENIX 协会，圣克拉拉，CA，487–498。[35] Carl A. Waldspurger、Nohhyun Park、Alexander Garthwaite 和 Irfan Ahmad。2015。使用 SHARDS 高效构建 MRC。载于第 *13* 届 *USENIX* 文件与存储技术会议（*FAST 15*）。USENIX 协会，圣克拉拉，CA，95–110。[36] Hua Wang、Jiawei Zhang、Ping Huang、Xinbo Yi、Bin Cheng 和 Ke Zhou。2020。缓存所需的数据：通过“一次性访问排除”策略减少云缓存中的写入流量。*ACM*《存储汇刊》（*TOS*）16, 3（2020），1–24。[37] Qiuping Wang、Jinhong Li、Patrick PC Lee、Tao Ouyang、Chao Shi 和 Lilong Huang。2022。通过推断块失效时间来分离数据，以减少日志结构存储中的写放大。载于第 *20* 届 *USENIX* 文件与存储技术会议（*FAST 22*）。429–444。[38] D.L. Willick、D.L. Eager 和 R.B. Bunt。1993。网络文件服务器的磁盘缓存替换策略。载于 *[1993]* 第 *13* 届分布式计算系统国际会议论文集。IEEE 计算机学会出版社，美国宾夕法尼亚州匹兹堡，2–11。doi:10.1109/ICDCS.1993.287729[39] Jake Wires、Stephen Ingram、Zachary Drudi、Nicholas JA Harvey 和 Andrew Warfield。2014。利用计数器栈刻画存储工作负载。载于第 *11* 届 {*USENIX*} 操作系统设计与实现研讨会（{*OSDI*} *14*）。335–349。[40] Jake Wires、Stephen Ingram、Zachary Drudi、Nicholas J. A. Harvey 和 Andrew Warfield。2014。利用计数器栈刻画存储工作负载。载于第 *11* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 14*）。USENIX 协会，布鲁姆菲尔德，CO，335–349。[41] Xiaoya Xiang、Bin Bao、Chen Ding 和 Yaoqing Gao。2011。共享缓存中程序工作集的线性时间建模。载于*2011* 年国际并行架构与编译技术会议。IEEE，350–360。[42] Zhu Yuan、Xueqiang Lv、Ping Xie、Haojie Ge 和 Xindong You。2024。CSEA：云存储系统中基于气候季节的细粒度能量感知框架。计算机学报67, 2（2024），423–436。[43] Heyu Zhang、Zhen Yang、Yulai Xie、Yafeng Wu、Jiakun Li、Dan Feng、Avani Wildani 和 Darrell Long。2024。使用生成对抗网络准确生成 I/O 工作负载。载于*2024* 年国际网络、架构与存储会议（*NAS*）。IEEE，1–9。

2DIO 内置轨迹配置

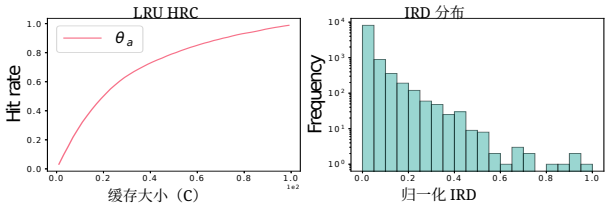
表 6 列出了第 4.1 节介绍的默认轨迹配置，其中 **fgen** 由公式 (3) 定义。图 11 展示了每种配置的命中率曲线和 IRD 分布，呈现了一组典型的工作负载行为，反映了真实世界中常见的模式。

表 6. 2DIO默认轨迹配置。

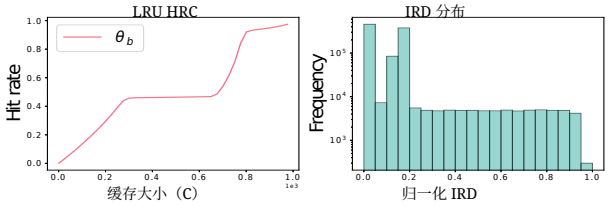
	P_{IRM}	g	f
θ_a	1.0	齐夫分布(3.0)	$\hat{f}_a$ =无
θ_b	0.0	None	$\hat{f}_b$ = **fgen**（20， [0, 3]， $5e-3$）
θ_c	0.0	None	$\hat{f}_c$ = **fgen**(20, [2, 9], $5e-3$)
θ_d	0.0	None	$\hat{f}_d$ = **fgen**(5, [0, 4], $1e-2$)
θ_e	0.0	None	$\hat{f}_e$ = **fgen**(20, [1], $5e-3$)
θ_f	0.0	None	$\hat{f}_f$ = **fgen**(5, [2], $5e-3$)

B LLGAN 合成轨迹验证

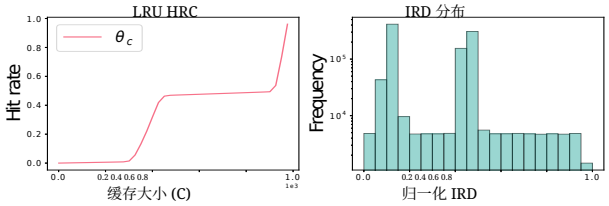
图 12 绘制了 LLGAN合成轨迹中逻辑块地址（LBA）和长度保真度的核密度估计（KDE），并与原始轨迹进行比较，以此作为第 5.1 节的合理性检查。合成 w11 与原始轨迹最为接近，这与其在表 4 中最低的 MMD² 相符；而合成 w44则明显偏离（MMD 中位数为² ≈ 0.06），但 HRC 拟合度最高。



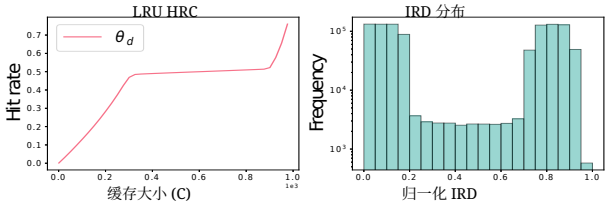
(a) $\theta_a = P_{IRM} = 1.0, g =$ 齐夫分布(3.0) $f =$ 无



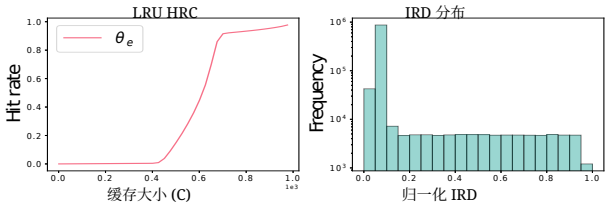
(b) $\theta_b = P_{IRM} = 0.0, g =$ 无 $f = \hat{f}_b$



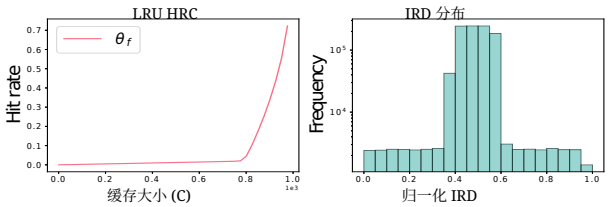
(c) $\theta_c = P_{IRM} = 0.0, g =$ 无 $f = \hat{f}_c$



(d) $\theta_d = P_{IRM} = 0.0, g =$ 无 $f = \hat{f}_d$



(e) $\theta_e = P_{IRM} = 0.0, g =$ 无 $f = \hat{f}_e$



(f) $\theta_f = P_{IRM} = 0.0, g =$ 无 $f = \hat{f}_f$

图11。 2DIO默认轨迹配置的 HRC 和 IRD 分布 θ_a–θ_f。

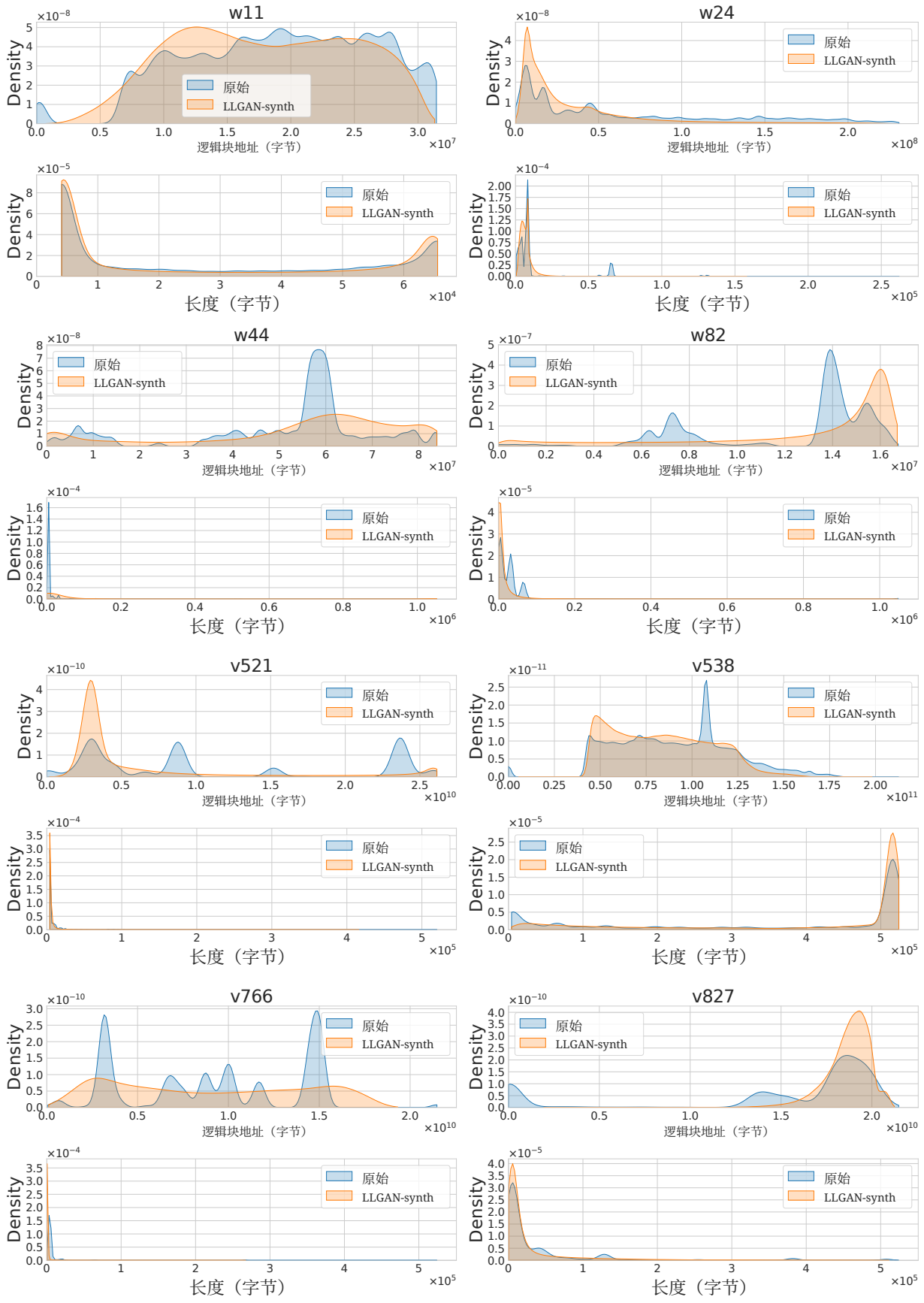


图 12。逻辑块地址的核密度估计 a 和长度，展示 LLGAN合成轨迹与原始轨迹之间的分布相似性 .