[论文]原版-double

> 原文：《Improving the Performance of Deduplication-based Backup Systems via Container Utilization based Hot Fingerprint Entry Distilling》

摘要

重复数据消除技术构建一个由指纹条目组成的索引，以识别和消除重复数据的重复副本。基于磁盘的索引查找的瓶颈和消除重复数据块导致的数据碎片是重复数据消除中的两个具有挑战性的问题。基于重复数据消除的备份系统通常使用将连续块与其指纹一起存储的容器来保留数据位置，以缓解这两个问题，但这仍然不够。为了解决这两个问题，我们提出了一种基于容器利用率的热指纹条目提取策略，以提高基于重复数据消除的备份系统的性能。我们将索引分为三个部分，即热门指纹条目、碎片指纹条目和无用指纹条目。利用率小于给定阈值的容器称为稀疏容器。指向非稀疏容器的指纹条目是热指纹条目。对于剩余的指纹条目，如果指纹条目与即将到来的备份块的任何指纹匹配，则将其分类为碎片指纹条目。否则，它将被归类为无用的指纹条目。我们观察到，热指纹条目只占索引的一小部分，而其余指纹条目占索引的大部分。这一有趣的观察结果启发我们开发一种名为HID的热门指纹条目提取方法。HID从索引中分离无用的指纹条目，以提高内存利用率并绕过磁盘访问。

此外，HID将碎片指纹条目分离，使基于重复数据消除的备份系统直接重写碎片块，从而减轻不利的碎片化。此外，HID还引入了一个功能，将碎片块视为唯一块。这一功能弥补了Bloom过滤器无法直接识别某些重复块（即碎片块）的缺点。为了充分利用上述特性，我们提出了一种称为EHID的进化HID策略。

EHID包含Bloom过滤器，只有热指纹才会映射到该过滤器。在这样做的过程中，EHID表现出两个显著的特征：（i）EHID避免了磁盘访问以识别唯一块和碎片块；（ii）EHID大幅降低集成Bloom滤波器的假阳性率。这些显著特征将EHID推向高效模式。我们的实验结果表明，当使用Linux数据集和FSL数据集时，我们的方法分别将索引的平均内存开销减少了34.11%和25.13%。此外，与最先进的方法HAR相比，使用Linux数据集，EHID将平均备份吞吐量提高了2.25倍；当涉及到FSL数据集时，EHID将平均磁盘I/O流量减少了高达66.21%。EHID还略微提高了系统的恢复性能。

ACM模式引用

Datong Zhang, Yuhui Deng, Yi Zhou, Yifeng Zhu, and Xiao Qin. 2020. Improving the Performance of Deduplication-based Backup Systems via Container Utilization based Hot Fingerprint Entry Distilling. J. ACM 37, 4, Article 111 (August 2020), 25 pages.

https://doi.org/10.1145/1122445.1122456

介绍

数据的爆炸性增长成为数据存储系统面临的日益严峻的挑战[13，27]。重复数据消除是一种现代技术，可以识别和存储唯一的块，从而显著减少对存储空间的需求[4，12]。此外，重复数据消除可以最大限度地减少网络存储系统中冗余数据的网络传输[23]。由于其高效性，重复数据消除已被广泛研究。新兴的重复数据消除应用，如I/O重复数据消除[18]、文件系统重复数据消除[33]和内存系统重复数据删除[15，16，32]，都在宣扬重复数据消除的日益普及和重要性。基于重复数据消除的备份系统首先将备份流划分为固定或可变大小的块[26，29]，然后为每个块计算SHA–1摘要[28]（即指纹）。为了保留备份流的位置，容器[19，34]用于存储连续的数据块及其指纹。当备份块时，系统向（指纹）索引发出查找请求，以将当前块的指纹与存储块的指纹进行比较。如果发生匹配，则块是重复副本，将从备份流中删除。否则，块是唯一的，并且将被存储。

由于每个区块备份都会导致索引查找来定位匹配的指纹，因此当索引的大小太大而无法存储在内存中时，由于基于磁盘的索引访问性能较差，重复数据消除的性能会大幅下降[14]。先前的研究表明，访问磁盘驻留索引的速度至少是访问内存驻留索引的1000倍[9]。频繁的基于磁盘的索引查找是基于重复数据消除的备份系统中最重要的性能瓶颈之一[31]。例如，假设数据块的平均大小为8KB，备份800TB的数据集将产生至少2TB的指纹，这将太大而无法存储在存储器中[31]。有一种重复数据消除系统将25GB的内存用于索引[14]，如此巨大的索引将对内存造成巨大压力，并导致大量磁盘I/O。基于重复数据消除的备份系统中的另一个瓶颈是数据碎片[17，24]。碎片化表示一种现象，即重复数据消除后，备份流中逻辑上连续的数据块在物理上分散在不同的容器中。严重分散的数据块被视为碎片块。数据碎片会降低恢复备份流的性能，因为碎片会破坏空间局部性[21]。

为了避免上述对索引的昂贵查找请求，将容器元数据（即存储在容器中的指纹）从磁盘预取到存储器，以快速识别备份流中的连续重复块。因此，形成了由多个指纹条目组成的索引，以便于容器元数据预取；并且每个指纹条目将指纹映射到一个或多个容器的地址[11]。图1简要显示了基于重复数据消除的备份系统中的几个关键概念（例如，指纹条目、索引表和容器）及其关系。请注意，索引表由两部分组成：基于内存的索引和基于磁盘的索引。在虚线框中，第一行中的数字表示存储的数据块的指纹，而第二行中的号码表示容器的ID，每个容器存储特定的数据块。

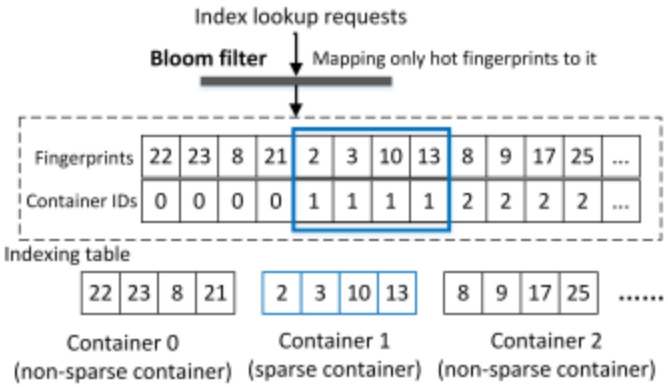


Fig. 1. The relationship of fingerprint entries and containers.

动机1：分析重复数据消除过程表明，索引表中的指纹条目可分为三类：热指纹条目、碎片指纹条目和无用指纹条目。让我们用图1中的例子来详细说明这些术语。在图1的底部是三个容器，其中容器0和2是非稀疏容器，容器1是稀疏容器。在本研究中，利用率（详见第3节）小于给定阈值的容器称为稀疏容器；指向非稀疏容器（即容器0和2）的指纹条目是热指纹条目，而冷指纹条目指向稀疏容器，因此在本例中我们有四个冷指纹条目2、3、10和13。如果冷指纹条目的指纹与未来备份流中包含的任何数据块的指纹相同，则该冷指纹条目被分类为碎片指纹输入。否则，这个冷指纹条目是无用的指纹条目，因为在未来的备份流块中没有匹配的指纹。将指纹条目分为三类的基本原理有三个方面。首先，内存中的热指纹条目有助于避免基于磁盘的索引访问，这会在重复数据消除过程中加快备份流中重复的块标识。其次，在内存中保留无用的指纹条目浪费了稀缺的内存空间，因为无用的指纹条目的重复块识别没有贡献。第三，碎片化指纹条目反映了碎片化的数据块（请参阅第3节中的详细讨论），应该将其重写为新的容器，以缓解数据碎片化。

动机2：从索引表中删除碎片指纹条目使基于重复数据消除的备份系统将碎片块视为唯一的块，并直接将其重写到新的容器中，从而缓解数据碎片化。现在，我们用图1中的上述示例演示了删除碎片指纹条目如何使基于重复数据消除的备份系统直接重写碎片块，以减轻数据碎片。让我们首先回顾一些术语如下。**碎片化是指在重复数据消除后，备份流中逻辑上连续的数据块在物理上分散在不同的容器中的现象，这种块被称为碎片化块。**稀疏容器是指利用率小于给定阈值的容器。因为图1中的容器1是一个稀疏容器，所以指纹2、3、10和13表示的相应数据块几乎不会出现在后续的备份流中。为了缓解数据碎片化，即使稀疏容器的一些块出现在随后的备份流中，它们也会被视为碎片化的块，并与其他唯一的块一起重写到新的容器中。例如，我们假设后续备份流包含四个块2、A、B和C，每个块向索引表（全局指纹条目）发出索引查找请求；那么，只有指纹2匹配。因此，后续备份流中的新块2是重复的块，因为在稀疏容器1中存在新块2的重复副本。我们不是对新块2进行重复数据消除，而是将新块2视为分段块，并将其与备份流中的其他唯一块一起重写到新容器中，以减轻数据分段。注意，传统的基于去重的系统首先将新的块2识别为重复的块，因为分段的指纹条目（例如指纹2的条目）没有被移除，然后使用额外的重写算法来确定重复的块是否是分段的块，这增加了计算开销。如果我们从索引表中删除碎片化的指纹条目，则新的块2被标识为唯一的块，并直接重写到新的容器中，以减轻数据碎片化。请注意，在本文中，

稀疏容器1中存储的块2和备份流中的新块2都被称为碎片块。分段块的指纹条目（例如指纹2的条目）被称为分段指纹条目。

受上述动机的启发，在本研究中，我们首先提出了一种基于容器利用率（详见第3节）的热指纹条目提取方法HID，以缓解磁盘I/O瓶颈并减少数据碎片。HID利用容器利用率来识别热指纹条目和冷指纹条目。HID在备份过程结束时从索引表中分离冷指纹条目。然后在下一个备份过程中，只使用热指纹条目来构建索引表（包含基于内存的索引和基于磁盘的索引）。通过这样做，HID提高了基于内存的索引查找的命中率，因为索引表中的所有指纹条目都是热指纹条目，从而避免了频繁的基于磁盘的索引访问。此外，HID减少了数据碎片，因为碎片指纹条目的分离使基于重复数据消除的备份系统能够将碎片块视为唯一块，直接将其重写到容器中。

动机3：值得注意的是，我们提出了一个显著的特征，称为FTU——碎片块被视为唯一块。FTU的机制是HID只在索引表中保留热指纹条目，在索引表上不留下碎片指纹条目。当对分段块执行索引查找时，索引表中没有匹配的条目，因此分段块被视为唯一块。FTU的引入弥补了Bloom滤波器的不足。Bloom过滤器是一种重复数据消除优化技术，用于识别备份流中的唯一块。然而，由于误报问题（即哈希冲突），Bloom过滤器无法识别备份流中的重复块[5，22，34]。如果没有FTU，碎片块可能会降低基于重复数据消除的备份系统的性能。更具体地说，分段块是重复块的子集。Bloom过滤器不会识别备份流中的每个碎片块，并且会向索引表发出索引查找请求（见图1），这可能会导致基于磁盘的索引访问。幸运的是，与FTU匹配的Bloom过滤器（通过仅将热指纹映射到该Bloom过滤器）将碎片块视为唯一的块，并且可以有效地识别这些碎片块，从而避免索引表查找。

为了充分利用FTU和Bloom滤波器的优势，我们最终提出了一种基于容器利用率的进化热指纹条目提取策略——EHID——来改进我们的初始HID。EHID嵌入Bloom过滤器，并仅将热指纹映射到Bloom过滤器中。在这样做的过程中，EHID表现出三个优点。首先，EHID在识别唯一块和碎片块时有效地避免了基于磁盘的索引访问。其次，通过仅将热指纹映射到集成Bloom过滤器中，降低了Bloom过滤器的误报率（请参见第5.3节）。第三，EHID在基于重复数据消除的备份系统中实现了相当大的性能改进。综上所述，本文的贡献如下：

- 我们获得了一些令人信服且有价值的发现。首先，为了减少基于磁盘的索引查找并缓解数据碎片化，我们将全局指纹条目分为三类：热指纹条目、无用指纹条目和碎片指纹条目。无用的指纹条目对于识别备份流中的重复块是无用的，所以将它们缓存在内存中是不明智的。分离无用的指纹条目可以缓解基于磁盘的索引查找问题。碎片指纹条目的分离使基于重复数据消除的备份系统能够将碎片块视为唯一的块，直接将其重写到新的容器中，从而减轻数据碎片。其次，我们揭示了容器利用率是促进指纹条目的3类分类的有效指标。指向非稀疏容器的指纹条目是热指纹条目，而指向稀疏容器的指印条目是无用的或碎片化的指纹条目。
- 我们提出HID来处理基于磁盘的索引查找瓶颈，并缓解基于重复数据消除的备份系统中的数据碎片。
- 我们提出了一种称为EHID的进化HID。EHID嵌入Bloom过滤器，并且仅将热指纹映射到嵌入的Bloom过滤器。通过这样做，EHID显著避免了昂贵的基于磁盘的索引访问。大量实验结果表明，EHID实现了相当高的性能。

本文的其余部分组织如下。第2节总结了相关工作。第三部分介绍了相关的背景知识。第4节介绍了我们对指纹录入类别转换的动机和分析。第5节演示了我们的设计和实现。进行了综合实验，以评估第6节中提出的方法。第7节是本文的结语。

相关工作

已经进行了许多研究来处理基于磁盘的索引查找的瓶颈。例如，Bloom滤波器[5]被精确的重复数据消除方法[22，34]用于识别唯一的块。然而，Bloom过滤器的效率随着备份数据量的增加而下降。这是因为Bloom滤波器的假阳性率[5，22，34]随着其空间使用量的增长而变大。接近精确的重复数据消除方法通过对所有指纹条目的一部分进行粗略采样（例如，随机采样、均匀采样）来执行重复数据消除[3，14，20，31]。由于采样指纹项的数量远小于指纹项的总数，因此近乎精确的重复数据消除方法可以减少索引查找导致的磁盘访问次数。尽管精确和近乎精确的重复数据消除方法减少了磁盘I/O的数量，但这两种方法都涉及大量无用的指纹条目和碎片化的指纹条目，这可能会导致频繁的数据丢失。为了减轻数据碎片，HAR（即历史感知重写算法）利用容器利用率来识别和重写碎片块[10]。请注意，重写碎片块是指允许在磁盘上存储少量重复的块。不幸的是，HAR忽略了这样一个事实，即容器利用率可以促进对指纹条目进行分类。在这项研究中，我们利用容器利用率对指纹条目进行分类，旨在加快索引查找并缓解数据碎片。为了使相关工作更加清晰和全面，以下两段将更详细地介绍每项工作。

为了减少对基于磁盘的索引的访问，已经提出了许多重复数据消除方法。朱等人[34]使用Bloom过滤器来识别唯一的块，并使用容器来保留备份流的位置，从而克服了磁盘瓶颈。MFDedup[35]通过使用数据分类方法来生成最佳数据布局，从而维护备份工作负载的位置。然而，随着Bloom滤波器的空间使用量的增长，Bloom滤波器[5]的假阳性率急剧增加。稀疏索引方法[20]从存储块的所有指纹条目（密集索引）中随机抽取一定百分比的指纹条目，以构建稀疏索引。稀疏索引方法减少了索引的内存开销，提高了备份吞吐量。但是，其重复数据消除率取决于备份流的位置和采样率。郭等[14]将采样方法从逻辑局部性应用到物理局部性。Extreme Binning[3]和ChunkStash[8]利用了文件的相似性理论，而不是备份流的局部性。但是，它对文件的相似性很敏感。Xia等人利用局部性和相似性，它适用于相似性较弱或较低的工作负载[31]。然而，由上述构造的索引表包含大量的冷指纹条目，并且冷指纹条目导致在执行索引查找时频繁地访问基于磁盘的索引。

由于碎片化大大降低了恢复性能，因此已经提出了许多用于减轻碎片化的重写算法。大多数重写算法分配小尺寸存储器来缓冲备份流中的连续数据块。缓冲区中的数据块序列称为逻辑序列。相反，位于磁盘上的数据块序列称为物理序列。如果逻辑序列与物理序列明显不同，则重写算法确定缓冲区具有许多分段块并对其进行重写。然而，由于分配内存只缓冲备份流的一小部分，因此它们无法准确地识别碎片块。为了解决这个问题，HAR[10]使用先前备份流的全局信息来预测后一备份流的碎片。HAR克服了其他重写算法的不足，提高了恢复性能和去重率。不幸的是，HAR没有考虑到容器利用率这一重要指标反映了指纹条目的类别。此外，在HAR中更新碎片指纹条目的开销降低了备份性能。

背景

分段：图2显示，通过处理第一个备份流，将唯一的块按顺序聚合到容器中。第一备份流的数据块存储在容器1、2、3和4中。当第二备份流到达时，块A被标识为重复的。因此，A不被存储，而是被指向位于容器1中的共享块A的指针所代替。下一个组块L被标识为唯一组块，因此在该示例中它被存储在容器5中。在此过程之后，将对备份流的其余数据块逐一进行重复数据消除。最后，第二个备份流的所有连续数据块都分散在不同的容器中，这称为数据碎片。

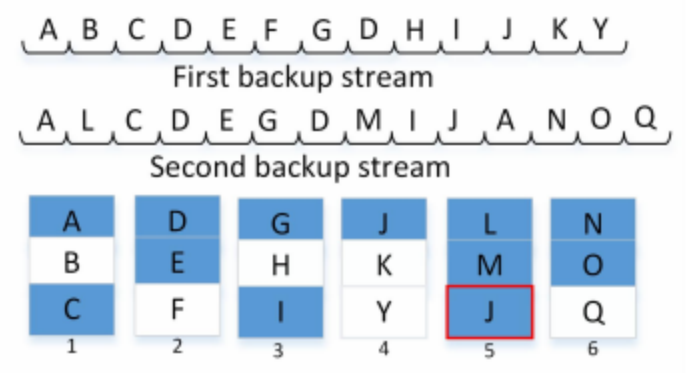


Fig. 2. Example of deduplication for two backup streams.

容器利用率：图2表明，对于第一个备份流，容器1有三个共享块A、B和C。然而，对于第二个备份流而言，容器1只有两个共享块C和A。对于所有后一个备份流来说，容器1很有可能只有不到两个的共享块，因为备份数据被连续地修改（删除/添加/修改数据块）。换句话说，随着备份和归档系统中备份版本的增长，容器中共享块的数量会逐渐减少。这种效果通过容器利用率来反映，并且可以表示为以下等式1。容器利用率定义为：

$$CU_i^k = R_i^k / S, \tag{1}$$

其中k表示第k个备份流，i表示第i个容器，S表示容器大小（数据块的数量），R_i^k 表示第k备份流的第i个容器中的共享块的数量，CU_i^k 表示用于第k个备用流的第i个容器的容器利用率。

如图2所示，由于CU^1_1=3/3=100%；CU^2_1 = 2/3 = 66.66%，我们可以得出结论，容器利用率越高，在数据备份和恢复过程中访问该容器的频率就越高。容器利用具有历史传承特征[10]。对于容器i，其容器利用率随着备份版本（流）的增加而降低。即$CU^1_i \geqslant CU^2_i \geqslant \cdots \geqslant CU^n_i$。因此，对于典型的备份场景，除非发生罕见的回滚，否则容器利用率不会增加[10]。尽管在极少数情况下，容器利用率的增长对整体系统性能的影响微乎其微。

此外，备份内重复对容器利用率没有影响。例如，如图2所示，容器2包含三个数据块D、E和F。尽管第一个备份流包含数据块E、F和D的两个重复副本，但R^1_2仍计算为3。这表明，无论E、F或D在第一备份流中重复出现多少次，R^1_2都保持不变。由于S是固定值，根据等式1，容器利用率CU^1_2也保持不变。

稀疏容器：稀疏容器表示利用率低于给定阈值的容器。例如，如果阈值P设置为70%，对于第二个备份流，容器1是稀疏容器，因为CU^2_1＝66:66% < P。根据其历史继承特性，如果特定容器在前一个备份流中被标识为稀疏容器，则该容器在随后的备份流中仍将是稀疏容器。这是我们方法的基本前提。

重写碎片块：为了提高恢复性能，HAR[10]重写碎片块并减少稀疏容器的数量。如图2所示。例如，假设稀疏阈值P＝50%，由于CU^2_4＝1 / 3＝33.33% < P，所以容器4是稀疏容器。当处理第二备份流时，由于共享块J的指纹条目指向稀疏容器4，所以复制的块J（第二备份串流中的块J）被识别为分段块。因此，分段的块J与唯一的块L和M一起被写入到新的容器5。当恢复第二备份流时，块J直接从缓存在存储器中的容器5获得，因为在恢复前一个块L时容器5已经从磁盘读取到存储器。相反，如果不重写分段的块J，读取容器4不可避免地会获得块J，这会导致磁盘I/O并损害恢复性能。

详细的重复数据消除过程：如图3所示，索引表（包含基于内存的索引和基于磁盘的索引）由存储块的所有指纹条目组成。每个指纹条目都将指纹映射到相应的容器ID（逻辑地址）。容器位于磁盘存储器中，每个容器都保存相应的指纹。配方用于记录用于恢复数据的逻辑块序列。活动容器用于存储唯一的块和重写的块。字节流（备份流）已经被划分为块，并且指纹已经被计算出来。当备份新的区块时，系统首先转到容器元数据缓存以定位相同的指纹。如果指纹没有在缓存中命中，那么系统将生成索引表的查找请求，这可能会导致访问基于磁盘的索引。在搜索索引结束时，确定新块的属性。如果新的区块是唯一的区块，系统会将其写入活动容器，并生成一个新的指纹条目并将其插入索引表。如果新区块是重复的区块，则不会对其进行存储。如果新区块是分段区块，系统会将其重写到活动容器，然后更新分段指纹条目（使条目指向活动容器而不是稀疏容器）。请注意，更新碎片指纹条目可能需要访问基于磁盘的索引，因为它必须再次查找索引表以定位碎片指纹条目，然后更新条目的值（容器ID）。

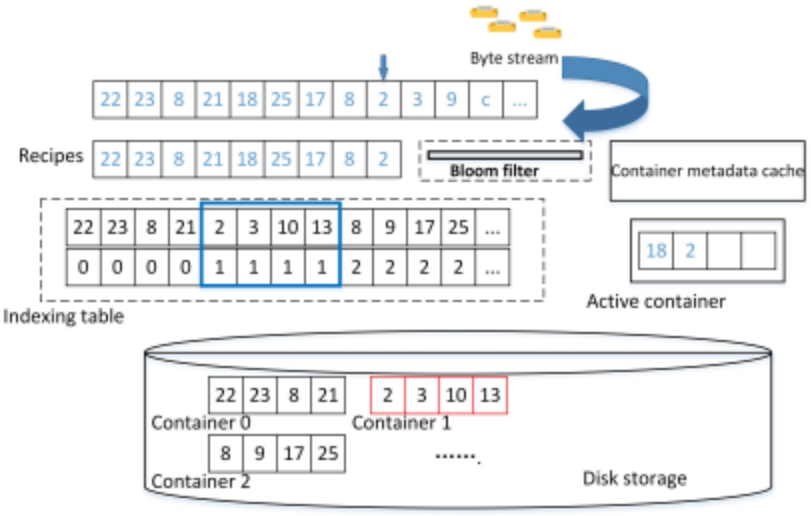


Fig. 3. The detailed process of deduplication.

恢复过程：在重复数据消除过程之后，文件的数据块可能会完全混乱，分散在所有磁盘的各个容器中，索引表中的指纹条目也会混乱。因此，由于缺乏数据块的序列信息，无法利用索引表来恢复数据。使用存储在磁盘中的文件配方来恢复数据，而不是使用索引表。文件配方由指纹和相应数据块的物理地

址组成。文件配方中的指纹序列与备份流中的数据块序列一致。每个备份版本都会生成一个特定的文件配方。如果需要第k个备份版本进行恢复，系统将首先分析文件配方的元数据，获得第k个文件配方的存储路径，将文件配方加载到内存中，并从磁盘读取数据块，利用文件配方信息恢复数据。由于文件配方独立于索引表，因此从索引表中删除任何指纹条目都不会对恢复过程产生任何影响。

动机探讨与指纹录入类别转换分析

动机讨论

查找索引表和不断增加的碎片块分别会导致频繁访问基于磁盘的索引和数据碎片，从而导致基于重复数据消除的备份系统出现主要性能瓶颈。我们的关键思想是，分离无用的指纹条目对识别和消除重复的块没有影响，而分离碎片化的指纹条目使系统直接重写碎片化的块并减少数据碎片。我们的主要观察结果是，稀疏容器反映了无用和碎片化的指纹条目。例如，图3显示了索引表中指纹为2、3、10和13的指纹条目是冷指纹条目，因为它们都指向容器1，假设容器1是稀疏容器。

这些事实使我们能够将指纹条目分为热门、碎片和无用，而碎片和无用的指纹条目构成了冷指纹条目。例如，在图3中，索引表中指向稀疏容器的指纹条目被分类为冷指纹条目，其余的被分类为热指纹条目。更具体地说，指纹为10和13的指纹条目是无用的指纹条目，因为由于数据备份场景的特点，它们在后续备份过程中不会被访问（详见第3.2节）。指纹条目2和3是碎片指纹条目，因为在随后的字节流中存在重复的指纹（块）2和3，但是它们的共享块位于稀疏容器1中。在本文中，我们将重复块2和3以及共享块2和共享块3等块称为碎片块。

分离无用的指纹条目可以有效地节省内存空间，因此它总是被执行的。但是，分离碎片指纹条目是可选的。在HID的设计中，我们选择分离碎片指纹条目，原因有以下三个：

- 分离碎片指纹条目引入了FTU功能。FTU功能表示，分离碎片指纹条目使基于重复数据消除的备份系统将碎片块视为唯一块。例如，在图3中，指纹2和3的条目被分离后，字节流中的分段块2和3通过遍历索引表被识别为唯一块。
- 分离碎片指纹条目使得基于重复数据消除的备份系统直接重写碎片块，这种直接重写机制减轻了数据碎片，避免了更新碎片指纹条目的开销。例如，由于FTU的特性，字节流中的分段块2和3（见图2）被转换为唯一块（FTU的唯一块）。基于重复数据消除的备份系统直接将这些FTU的唯一块写入活动容器，并为这些块生成新的指纹条目，而不是更新分离的碎片指纹条目。
- 将指向稀疏容器的指纹条目（即，冷指纹条目）划分为无用指纹条目和碎片指纹条目的操作是昂贵的。为了避免这种开销，指向稀疏容器的指纹条目直接与所有指纹条目分离。

集成Bloom过滤器和FTU：FTU是对Bloom过滤器的补充。基于重复数据消除的备份系统中的Bloom过滤器可以通过将所有指纹映射到其自身中来有效地识别唯一的块。然而，由于其假阳性，它缺乏识别重复块的能力[5，22]。具体来说，字节流中的重复块必须查找索引，这可能导致基于磁盘的索引访问。

幸运的是，通过集成FTU，Bloom过滤器能够识别一些重复的块，因为FTU将所有碎片块视为唯一的块。请注意，分段块属于重复块，而分段块只是重复块的一小部分。总之，Bloom过滤器和FTU的组合进一步防止了基于重复数据消除的备份系统进行基于磁盘的索引访问。在代码实现层面，只要我们将热指纹而不是所有指纹映射到集成的Bloom过滤器，Bloom过滤器和FTU就可以很好地集成。

基于上述分析，我们提出了EHID，通过集成Bloom过滤器和FTU来进一步提高基于重复数据消除的备份系统的备份性能。EHID将Bloom过滤器放在容器元数据缓存和索引表之间，它只将热指纹映射到这个Bloom过滤器中（见图3）。在这样做的过程中，大多数唯一块（正常和FTU的唯一块）由Bloom过滤器而不是索引表来标识。例如，由于冷指纹条目（即指纹2、3、10和13的条目）没有映射到Bloom过滤器，所以当处理字节流中的新块2和3时，Bloom过滤器将这些块识别为唯一的块。因此，这避免了查找索引表来确定这两个块的属性（唯一、重复或分段）。注意，传统方法将继续搜索索引表，因为它们将冷指纹（即，2、3、10和13）映射到Bloom过滤器，结果是Bloom过滤器不能将新块2和3作为唯一块来处理。除了FTU和Bloom滤波器的相互增益之外，仅将热指纹而不是所有指纹映射到Bloom滤波器中可以降低误报率。

指纹录入类别转换分析

HID和EHID仍然需要解决两个关键问题。也就是说，随着工作负载的发展，无用或碎片化的指纹条目是否可以重新分类为热门指纹条目？当这种情况发生时，HID和EHID如何处理？为了回答这两个问题，我们进行以下分析。请注意，创建索引表（全局指纹条目）是为了快速识别备份流中的重复块。考虑以下典型的数据备份场景：

工程师创建一个文件，该文件由{A1；A2；A3；A4}$_0$表示，下标0表示该文件是初始版本，“{}”中的每个字符表示一个数据块，{A1；A3；A4}$_0$表示文件包含四个数据块A1、A2、A3和A4。为了保护文件，工程师使用基于重复数据消除的备份系统进行第一次备份。由于此时备份存储中没有存储块，因此索引表为空。索引表由（∅）0表示，其中下标0表示索引表尚未更新，“（）”中的每个字符表示条目的指纹。当数据块A1–A4被备份时，系统搜索索引表（∅）0。由于索引表为空，这些数据块被标识为唯一块，并存储到磁盘中。然后，系统为存储的四个块创建指纹a1–a4的条目。由于索引表为空，这些数据块被标识为唯一块，并存储到磁盘中。然后，系统为存储的四个块创建指纹a1–a4的条目。此时，索引表从（∅）0更新为（a1—a4）$_1$。

稍后，工程师修改文件，文件变为{A1；A2；B1；A4}$_1$。此时，他进行第二次备份。系统遍历（a1—a4）$_1$，并分别找到指纹a1、a2和a4。因此，数据块A1、A2和A4被识别为重复块。数据块B1被识别为唯一块，因为在（a1—a4）1中没有对应的指纹。然后，系统为数据块b1创建指纹条目b1，并将b1插入索引表。现在，索引表从（a1—a4）1更新为（a1—a4；b1）$_2$。我们可以发现，在第二次备份期间，只有（a1—a4）1中的指纹条目a1、a2和a4有助于识别{a1；a2；B1；a4}$_1$中的重复块a1、a2和a4。换句话说，从索引表（a1—a4）1中删除指纹a3的条目对重复数据消除过程没有影响。

之后，工程师再次修改文件，文件变为{A1；A2；B1；C1；C2}$_2$。此时，她进行第三次备份。同样，只有（a1—a4；b1）$_2$中的指纹a1、a2和b1的条目有助于识别{a1；a2；b1；C1；C2}$_2$中的重复块a1、a2和b1。因此，从（a1—a4；b1）$_2$中删除指纹条目a3和a4不会产生副作用。

在上面的备份示例中，索引表按以下顺序更新：

$$(\varnothing)_0 \rightarrow (a1, \underline{a2}, a3, a4)_1 \rightarrow (a1, \underline{a2}, \underline{a3}, a4, b1)_2 \rightarrow (\underline{a1}, \underline{a2}, \underline{a3}, a4, b1, c1, c2)_3 \rightarrow \ldots \rightarrow (\underline{a1}, \underline{a2}, \underline{a3}, \underline{a4}, \underline{b1}, c1, c2, \ldots, xm)_n, \quad (2)$$

其中n是备份的数量，带下划线的字母表示无用的指纹条目。此示例表明，尽管指纹条目的总数随着备份版本的增加而增加，但无用指纹条目的数量也相应增加。先前的研究[6，7，11]表明，随着备份的增加，碎片块的数量会增加，碎片指纹条目的数量也会增加。因此，我们可以推断，热指纹条目的数量保持在稳定水平，或者随着备份版本的增加而缓慢增长。

在典型的数据备份场景中，随着备份版本的增加，三个指纹条目的变化趋势如下：

- 热指纹条目：如果指纹条目在当前备份中被归类为热指纹条目，则在后续备份中，它可能是热指纹条目或成为无用或碎片指纹条目。
- 无用指纹条目：如果指纹条目在当前备份中被归类为无用，那么在后续备份中它仍然是无用指纹条目。
- 碎片指纹条目：如果指纹条目在当前备份中被归类为碎片指纹条目，那么在后续备份中，它仍然是碎片指纹条目。

由于典型数据备份场景的特点，冷指纹条目（即无用指纹条目和碎片指纹条目）不再被重新归类为热指纹条目。因此，HID和EHID在任何备份版本中都能有效工作。

设计和实现

体系结构概述

图4描述了我们的EHID策略的总体架构和工作流程。整个系统可分为三个大模块，包括指纹查找模块（FLM）、容器利用率监测模块（CUMM）和索引分离模块（ISM）。这些模块通过重复以下三个步骤协同工作。

- 在步骤1，当备份每个新的数据块时，FLM负责识别唯一的块和重复的块。同时，CUMM通过记录备份过程中哪个数据块属于哪个容器来计算每个容器的利用率，然后获得稀疏容器的列表。该列表用于区分热指纹条目和冷指纹条目。

- 在步骤2，ISM通过检查指纹条目是否指向列表上记录的稀疏容器，从所有指纹条目中识别冷指纹条目和热指纹条目。如果指纹条目指向稀疏容器，则它是冷指纹条目。否则，天气会很热。通过这种方式，ISM分离冷指纹条目和热指纹条目。
- 在步骤3，热指纹条目（热索引文件）成为由基于内存和基于磁盘的索引组成的索引表。热指纹用于在下一次备份开始时初始化Bloom过滤器。最后，系统返回步骤1并重复上述过程。

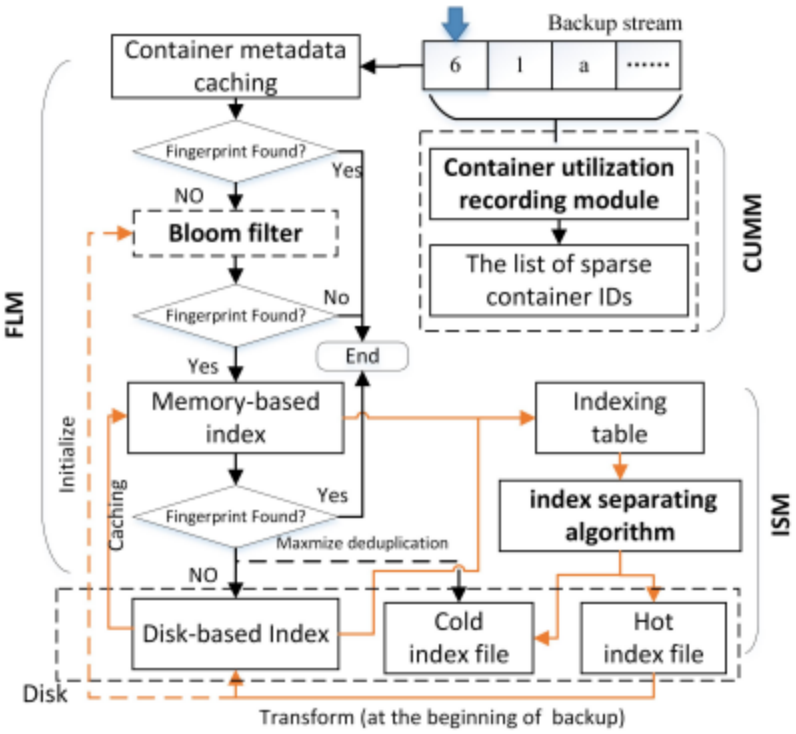


Fig. 4. Architecture overview of EHID based deduplication-based backup system.

关键过程：如图4中的橙色箭头所示，基于内存和基于磁盘的索引构成了索引表。随着备份的数据越来越多，索引表中所有指纹条目的数量也会越来越多，即使一部分指纹条目逐渐变为冷指纹条目。索引分离算法（见图4）在每个备份过程结束时将索引表中的指纹条目划分为冷索引文件和热索引文件。以这种方式，越来越多的冷指纹条目被保存到冷索引文件，并且热索引文件中的热指纹条目的数量保持较小。在每次备份开始时，只有热索引文件用于构建索引表（基于磁盘的索引和基于内存的索引），该文件还用于初始化Bloom过滤器。Bloom过滤器在每次备份开始时都会重新初始化。除非我们的设计要求最大化重复数据消除率，否则FLM不会查询冷索引文件（请注意，传统的基于重复数据消除的备份系统仅提供FLM模块）。

冷索引文件和热索引文件存储在磁盘中。在特定备份的初始阶段，系统将热索引文件作为基于磁盘的索引，并将基于磁盘的一部分索引读取到内存中。此时，索引表等于基于磁盘的索引，并且索引表和基于磁盘的索引来包含基于内存的索引中的指纹条目。在备份过程中，会不断生成新的指纹条目，并将其插入到基于内存的索引中。如果分配的内存大小是固定的，则插入的指纹条目将触发内存中指纹条目的替换。在这个阶段，基于磁盘的索引可能不包括所有基于内存的索引。因此，用于重复数据消除的索引表应同时包含基于磁盘的索引和基于内存的索引，如图4所示。当前备份完成后，ISM将索引表分离为

冷索引文件和热索引文件，并将它们再次存储在磁盘中。因此，每个备份版本都会更新热索引文件和冷索引文件。此外，在备份过程中，索引表会为后续备份流不断更新。

CUMM和ISM的开销较低。CUMM使用64位整数数组来记录稀疏容器。在我们的实验中，一个备份有大约3500个容器。假设所有的容器都是稀疏容器，CUMM只需要27.34KB的内存空间和一点计算开销即可操作。ISM通过只遍历索引表一次来分离冷指纹条目和热指纹条目。假设所有指纹条目的数量为N，则该分离过程的时间复杂度为O（N）。值得注意的是，对于访问基于磁盘的索引的查找请求，时间复杂度也是O（N）。此外，在开始下一次备份之前，可以离线进行索引分离，以最大限度地减少开销。

热指纹条目的提取算法

Algorithm 1 Hot Fingerprint Entries Distilling Algorithm

Input: $list_n$; //list of sparse container IDs in nth backup
$glIndex_n$; // all fingerprint entries in nth backup stream
Output: $hIndex_n$; // hot fingerprint entries in nth backup stream
$coIndex_n$; // cold fingerprint entries in nth backup stream

```
1: function INIT()
2:     glIndex_n ← hIndex_{n-1};
3: end function
4: function LOOKUP( )
5:     while (new_chunk ≠ end_backupStream) do
6:         if (new_chunk = uniq_chunk) then
7:             insert(glIndex_n, entry_uniq_chunk);
8:         end if
9:         new_chunk ← new_chunk−>next;
10:    end while
11: end function
12: function SEPARATES()
13:    get first in glIndex_n;
14:    while (first ≠ end_glIndex_n) do
15:        if first−>data.value in set(list_n) then
16:            write(first−>data, coIndex_n);
17:        else
18:            write(first−>data, hIndex_n);
19:        end if
20:        first ← first−>next;
21:    end while
22: end function
```

算法1演示了HID的基本工作过程。它概括为以下三个重要步骤：

- 当第n次备份开始时，由第（n−1）次备份中的separate函数提取的热指纹条目（例如 $hIndex_{n-1}$）会在当前备份（例如glIndexn）中构建索引表，而不考虑$coIndex_{n-1}$。

- 在备份过程中，查找功能将唯一块的指纹条目（例如$entry_{uniq_chunk}$）插入索引表（例如$gIIndex_n$）。同时，索引表中所有指纹条目的一小部分被转换为冷指纹条目。
- 当备份过程完成时，separate函数提取热指纹条目（例如$hIndeX_n$），并将它们存储到磁盘中以备下次备份。当下一次备份开始时，算法1再次返回到第一步。

EHID

图4概述了EHID的体系结构和工作流程。EHID有两个优点：

- EHID充分利用FTU和Bloom过滤器来有效地识别正常的唯一块和FTU的唯一块（即碎片块）。此设计可防止基于重复数据消除的备份系统在识别备份流中的唯一块时访问基于磁盘的索引。
- EHID采用的Bloom过滤器工作效率高。Bloom滤波器的假阳性率随着其空间使用的增长而增加。幸运的是，EHID及时地分离了冷指纹条目，同时最大限度地减少了热指纹条目的数量。这样的效果减少了Bloom滤波器的热指纹占用的空间。

访问基于磁盘的索引的分析

下面正式分析EHID的效率。假设一个备份流中的数据块数量是固定的，并且每个唯一的数据块必须触发查询基于磁盘的索引的请求。相反，每个重复的块可以以相似的概率触发对基于磁盘的索引的搜索，因为重复的块的索引查找请求可以在容器元数据高速缓存或基于存储器的索引中提供。每个重复块触发基于磁盘的索引查找请求的平均概率应该相等，并用P_{find}表示。假设uniqs和dups分别表示唯一块和重复块的数量。然后，对于传统的基于重复数据消除的备份系统，在一个备份流中访问基于磁盘的索引的查找请求数定义为：

$$disk\_lookups_{tradis} = uniqs + dups * p_{find}. \tag{3}$$

我们假设Bloom滤波器的假阳性率为0；对于配置了Bloom筛选器的传统基于重复数据消除的备份系统，基于磁盘的索引查找请求数定义为：

$$disk\_lookups_{tradis+Bloomfilter} = dups * p_{find}. \tag{4}$$

对于EHID，基于磁盘的索引查找请求的数量定义为：

$$disk\_lookups_{EHID} = dups^{'} * p_{find}. \tag{5}$$

请注意，dups'＜dups，因为EHID将一些重复的块（即碎片块）视为唯一的块（如FTU功能）。换言之，EHID识别的唯一块的数量大于传统的基于重复数据消除的备份系统（即$uniqs' \geqslant uniqs$）识别的块的数量。相应地，EHID识别的重复块的数量小于传统的基于重复数据消除的备份系统（即

dups′≤dups）。因此，我们可以推导出 $disk\_lookups_{tradis} \geq disk\_lookup_{tradis+Bloomfilter} \geq disk\_lookups_{EHID}$，这个表达式表明EHID访问基于磁盘的索引所需的查找请求数量最小。

我们将举一个例子来说明为什么EHID策略可以利用Bloom过滤器来识别碎片块，避免磁盘中的索引访问，从而提高备份吞吐量。请注意，传统方法使用的Bloom过滤器无法识别碎片块。假设磁盘中存储有一个分段块J1，则新的备份块J2是J1的重复块。为了减轻数据碎片，尽管J2是一个重复的块，但它必须被写入磁盘。由于EHID不将碎片指纹映射到Bloom过滤器（以集成Bloom过滤器和FTU），因此系统无法找到块J1的指纹的映射位。因此，新的组块J2被识别为唯一的组块，并且将被写入磁盘。如果碎片指纹被映射到Bloom过滤器，则块J1的映射比特将被定位。由于Bloom过滤器的误报，系统必须进一步按顺序检查基于内存的索引和基于磁盘的索引，以识别块J2的属性，包括唯一块、重复块和碎片块。查找指纹（条目）可能会在以下步骤中触发磁盘I/O。

EHID Bloom滤波器的假阳性率

EHID的另一个迷人特征是它降低了集成Bloom滤波器的假阳性率。Bloom滤波器的假阳性率（FPR）的计算公式可以表示为：

$$FPR = (1 - e^{(-kn/m)})^k, \tag{6}$$

其中k表示散列函数的数量，m是Bloom滤波器的大小（比特），n是映射指纹的数量。假设具有k个散列函数和250000个指纹的128KB Bloom过滤器，FPR为14.28%，这意味着大约有14个查找请求用于访问索引表以处理每100个块。EHID只将热指纹（占Linux跟踪上所有指纹的37.8%）映射到Bloom过滤器，因此FPR降低到0.78%。这个例子表明，对索引表的访问显著减少。图5显示了FPR和热指纹条目的百分比之间的关系。结果表明，当百分比从100%降低到37.8%时，Bloom filter的FPR从14.2%降低到0.8%。请注意，x轴上的热指纹条目的百分比是由Linux数据集的1到213的不同备份版本生成的。

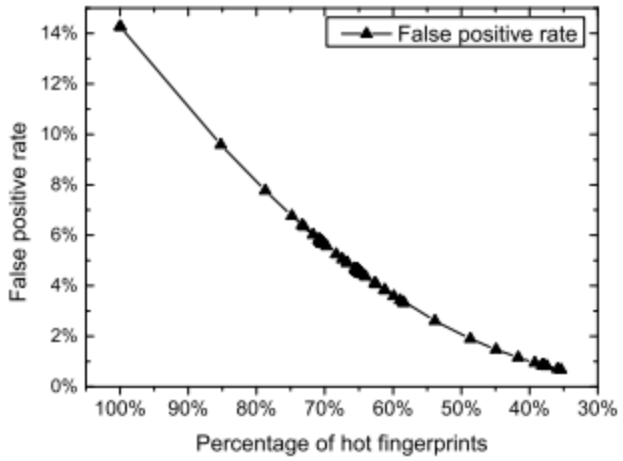


Fig. 5. False positive rate vs. percentage of hot fingerprint entries

实验评估

评估环境

在本节中，我们进行了一组实验来评估我们的热指纹条目提取方法的性能。我们在一个名为destor[11]的开源项目上实现了HID和EHID。为了定量评估性能，我们利用了四个指标，包括索引表的内存开销、备份性能、恢复性能和存储开销（比重复数据消除率更直观）。精确重复数据消除（ED）方法识别并删除每个重复的块，并且不使用任何重写算法。它是绩效评估的基准。此外，将HID和EHID的性能与最先进的算法HAR[10]进行了比较。由于HID和EHID只在备份性能上有所不同，因此省略了HID实验结果的其他三个指标（索引表的内存开销、恢复性能和存储开销）。

实验性操作系统是CentOS 7.4版（Linux版本3.10.0–693.11.1–el7.x86–64）。容器大小为4MB（包含大约1000个数据块）。稀疏阈值是HAR和EHID的一个非常重要的参数。

稀疏容器的数量随着稀疏阈值的增加而增加。这会导致更多的碎片块、无用块和冷指纹条目。因为EHID和HAR都通过重写分段块来提高恢复性能，所以更高的稀疏阈值导致EHID和HAR都重写更多的分段块，因此它们都实现了更高的恢复性能和更低的重复数据消除率。此外，和HAR相反，EHID将冷指纹条目从索引表中分离出来。因此，更高的稀疏阈值导致EHID分离出更多的冷指纹条目，这将进一步降低索引表的内存开销。结果，EHID进一步提高了备份性能，而HAR不能提高。当阈值降低到一个较低的值时，它将产生相反的过程。HAR已经证明，将阈值设置为50%可以很好地平衡重复数据消除率和恢复性能。因此，为了与HAR进行公平的比较，我们在本文的所有实验中都将阈值设置为50%。

Linux数据集和FSL数据集[1，2]是常用的公共数据集[30]。这两个数据集的特征如表1所示。由于FSL跟踪没有数据块的字节流，我们无法像Linux跟踪那样评估FSL跟踪的实际吞吐量。我们分析了重复数据消除过程中由索引查找引起的磁盘I/O行为，并使用磁盘I/O的数量来衡量吞吐量，并评估我们在FSL数据集上的方法的效率。

Table 1. Characteristics of workloads used for evaluation

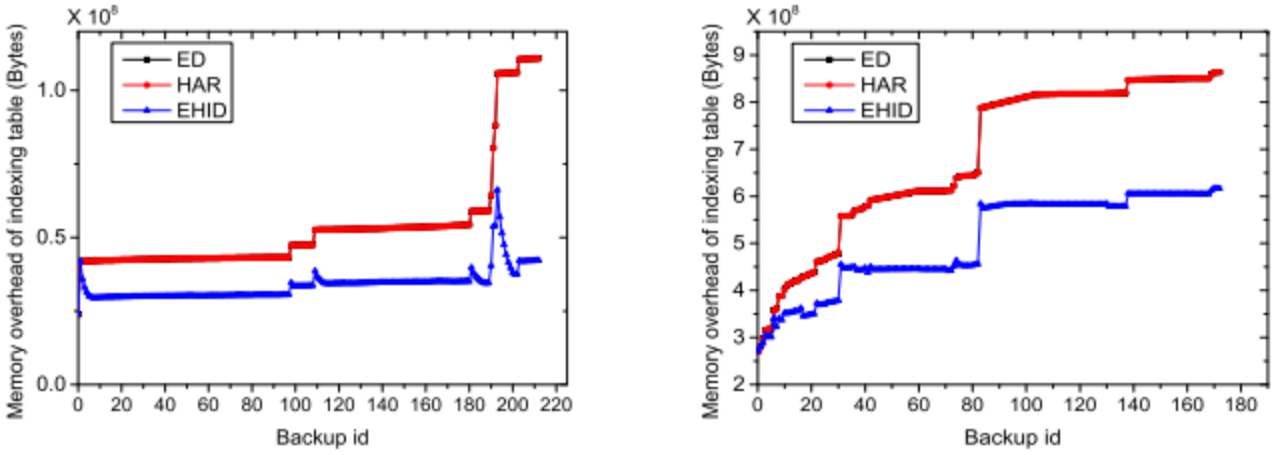
dataset name	TS^1	DR^2	ACS^3	# of versions
Linux	99GB	97.85%	3.1KB	213
FSL	1.95TB	98.73%	4.4KB	173

[1] TS stands for the total size of the trace.
[2] DR stands for the deduplication ratio of the trace.
[3] ACS stands for the average chunk size of the trace.

我们使用速度因子[19]作为评估恢复性能的指标。速度因子表示可以通过平均从磁盘读取容器来恢复的原始数据大小。速度因子的值越高，表示恢复性能越好。

索引表的内存开销评估

我们评估了Linux和FSL数据集上索引表的内存开销。请注意，内存开销意味着将索引表的所有指纹条目读取到内存中，以完全避免基于磁盘的索引访问。在图6（a）中，y轴表示用于执行重复数据消除的索引表的大小，x轴表示备份版本的ID。图6（a）所示的ED和HAR曲线完全重叠。这种趋势是意料之中的，因为这两种方法不会将冷指纹条目从索引表中分离出来。EHID可以有效地降低索引表的内存开销。此外，随着备份版本的增加，EHID变得更加高效。例如，当备份版本达到213时，EHID引起的索引表的内存开销仅为其他两种方法（ED和HAR）的37.80%。图6（b）描述了当我们对FSL轨迹进行实验时的实验结果。图6（b）显示了类似于Linux跟踪的性能。例如，当备份版本达到173时，EHID生成的指纹条目仅占其他两种方法的71.38%。这些实验证实，EHID可以有效地降低索引表的内存开销，避免系统老化导致的索引表臃肿。



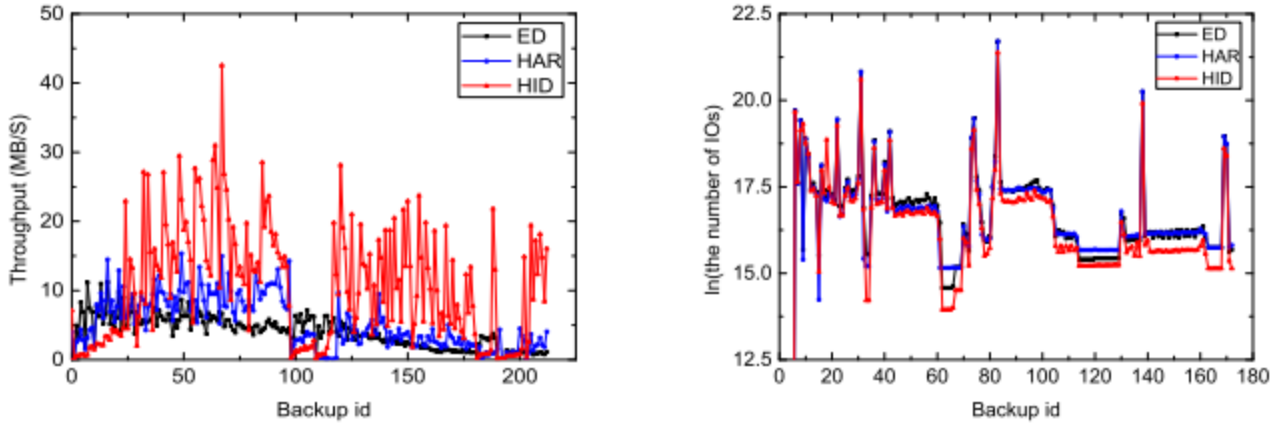
(a) Linux dataset. (b) FSL dataset.

Fig. 6. Memory overhead of indexing table.

图6（a）和图6（b）表明，当使用Linux数据集时，EHID可以比FSL数据集节省更多的内存。这背后有两个原因。第一个是Linux数据集的稀疏容器部分高于FSL数据集。这是因为Linux数据集的很大一部分是自引用的[10]。第二个是，与FSL数据集的173个备份版本相比，Linux数据集有213个备份版本。随着备份版本的增长，EHID可以节省更多内存，这是一个非常重要的功能。此外，对于Linux数据集，我们使用Linux-3.x进行191版本备份前的评估。下一个备份版本192使用Linux-4.x。由于Linux内核进行了重大修订更新，新的备份数据会产生大量唯一的块和相应的指纹条目。这就是我们在图6（a）中观察到尖峰的原因。然而，这两个数字都表明了相同的趋势，即随着备份版本的增加，ED和HAR生成的索引表的内存开销显著增加。相比之下，EHID引起的索引表的内存开销比ED和HAR增长得慢得多。

评估备份性能

我们评估了在两个数据集（即Linux和FSL数据集）上禁用Bloom过滤器时的备份性能。图7（a）描述了使用Linux数据集时ED、HAR和HID的备份吞吐量。结果表明，HID的吞吐量是最高的。HAR和ED分别排名第二和第三。前23个备份版本的HID吞吐量较低的原因是无法缓存HID的稀疏容器会损害提取热指纹条目的效率。

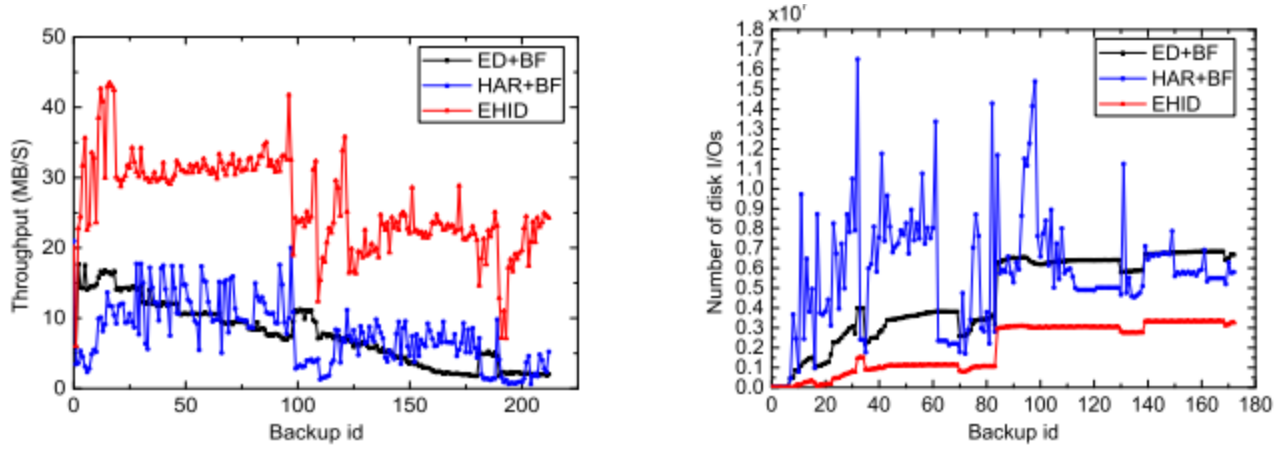


(a) Backup throughput with Linux dataset. (b) Number of disk I/Os with FSL dataset.

Fig. 7. Backup performance of ED, HAR and HID.

由于FSL跟踪不包含数据块的字节流，因此无法像在Linux跟踪上那样评估FSL跟踪上的实际吞吐量。在这部分研究中，我们研究了这些方法的I/O行为。图7（b）显示了ED、HAR和HID在FSL数据集上的备份性能。由于I/O的波动较大，为了使结果更直观和易于理解，我们对I/O的数量（图7（b）中的y轴）执行ln（I=Os）运算，图7（b）显示，与其他两种方法（即ED和HAR）相比，HID生成的I/O数量最少。

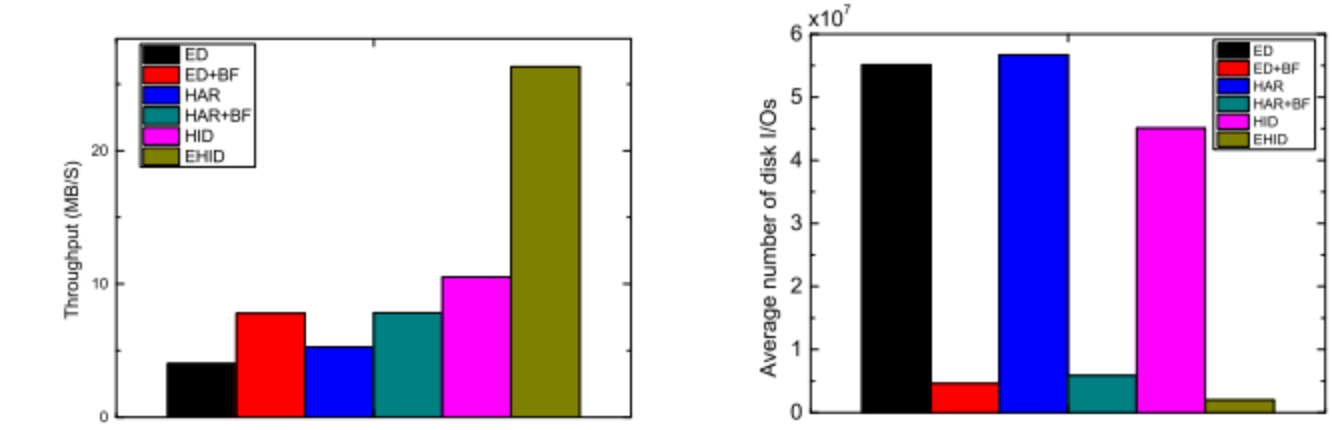
现在，我们可以评估EHID——HID的一个进化版本，它集成了Bloom过滤器，以进一步提高备份性能。为了进行公平比较，我们还为ED和HAR方法配置了Bloom过滤器。图8（a）显示了ED+BF（Bloom filter）、HAR+BF和EHID的吞吐量。图8（a）的结果表明，EHID的吞吐量远高于其他两种方法。有两个因素促成了EHID的高吞吐量。首先，当配置相同的内存大小时，EHID提高了内存的效率，并将更有用的指纹条目从磁盘读取到内存中。其次，由于内存中保存的所有指纹条目都是热指纹条目，因此基于内存的索引查找的命中率显著提高，从而显著避免了访问基于磁盘的索引的查找请求。正如预期的那样，ED+BF的吞吐量随着备份版本的增加而逐渐降低。这是因为数据块的碎片随着备份版本的增加而逐渐增加。因此，必须使用基于磁盘的索引来满足更多的查找请求，这样的影响会导致性能下降。观察到HAR+BF的吞吐量围绕ED+BF的曲线波动是非常有趣的。吞吐量波动背后的原因是HAR必须更新所有碎片块的指纹条目，这涉及到额外的磁盘I/O。当碎片块的数量增加时，由于生成了大量的磁盘I/O，HAR的吞吐量相应降低。



(a) Backup throughput with Linux dataset. (b) Number of disk I/Os with FSL dataset.

Fig. 8. Backup performance of ED+BF, HAR+BF and EHID.

图8（b）表明，当使用FSL数据集时，EHID为索引查找发布的磁盘I/O数量明显低于启用Bloom filter的其他两种方法。原因是EHID显著减轻了发送基于磁盘的索引的查找请求。ED+BF和HAR+BF生成的磁盘I/O说明了FSL数据集上与Linux数据集上类似的行为。请注意，这三种方法都从Bloom filter中受益匪浅，因为Bloom filter有效地避免了访问因识别唯一块而产生的基于磁盘的索引。然而，EHID受益最大。以上六种方法的平均性能如图9所示，这证实了EHID明显优于其他五种方法。



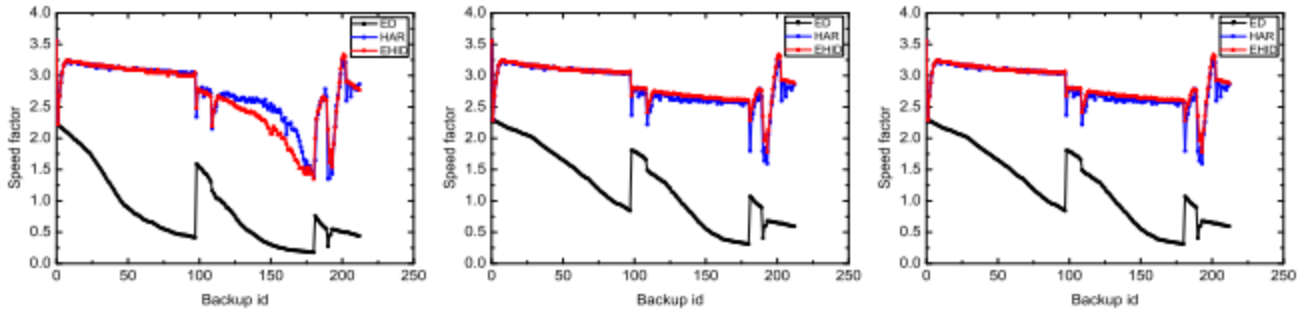
(a) Average backup throughput with Linux dataset. (b) Average disk I/Os with FSL dataset.

Fig. 9. Average backup performance.

评估恢复性能

现在我们关注恢复性能。图10显示了使用Linux数据集时的速度因素。由于HAR使用了最优缓存替换（OPT）算法，为了公平起见，我们实验中的ED、HAR和EHID三种方法都配置了OPT算法。OPT的缓存大小设置为30、60和90，缓存单元是一个容器。从图10中可以看出，ED的速度因子是最低的。随着备份版本的增加，速度系数会降低。原因是ED没有采用重写算法来减轻碎片。HAR和EHID的速度因子非常

相似。更具体地说，当缓存大小为30时，HAR和EHID的平均速度因子分别为2.75和2.68（见图10（a））。当备份版本从115增长到175时，HAR的性能略优于EHID。这是因为在这些备份版本中，HAR比EHID重写更多的碎片块，这导致了更多的存储空间使用。然而，随着高速缓存大小的增加，EHID的速度因子变得比HAR的速度因子更好。例如，当高速缓存大小为60时，HAR和EHID的平均速度因子分别为2.8327和2.8806。当缓存大小为90时，HAR和EHID的平均速度因子分别为2.8328和2.8807。

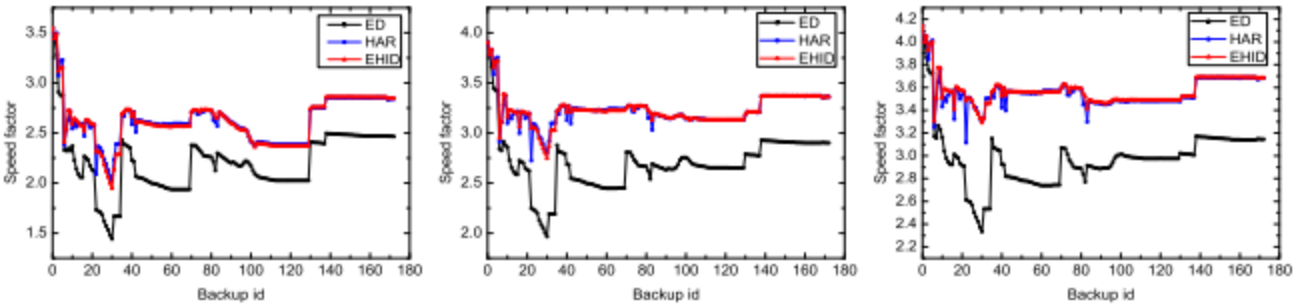


(a) OPT 30. (b) OPT 60. (c) OPT 90.

Fig. 10. Speed factor with Linux dataset.

此外，我们可以看到，在60和90缓存大小的情况下，EHID的性能略优于HAR。这一变化是因为EHID只重写一次分段块，而HAR可以多次重写一个分段块。例如，假设存在指向稀疏容器C0的分段指纹条目I1，则新备份块J1的指纹与分段指纹条目的指纹重复。新的组块J1将被重写到新的容器C1，以减轻数据碎片。由于新容器通常不是稀疏容器，因此系统应及时更新分段指纹条目I1并将其指向容器C1。如果在需要备份另一个新的组块J1之前没有更新该分段指纹条目I1，则系统将把新的组元J1重写到另一新的容器C2。在原始的HAR中，由于多线程和脏缓冲区的同步，偶尔会出现碎片指纹条目无法及时更新的情况。这样做，HAR中重写的块在恢复数据时会浪费存储空间和磁盘带宽。与HAR相反，不需要EHID来更新分段指纹条目，因为所有分段指纹条目都已从索引表中分离出来。因此，EHID不可能多次重写相同的碎片块。

图11描述了使用FSL数据集时的速度因子。实验结果与Linux数据集的结果非常相似。正如预期的那样，图11表明ED的速度因子仍然是最低的，EHID的速度因子与HAR的速度因子非常接近。更具体地说，随着高速缓存大小的增加，EHID在速度因子方面变得优于HAR。例如，首先，当高速缓存大小为30时，HAR和EHID的平均速度因子分别为2.6272和2.6240。接下来，当高速缓存大小增加到60时，HAR和EHID的平均速度因子分别变为3.2273和3.2311。最后，当缓存大小达到90时，HAR和EHID的平均速度因子分别变为3.5595和3.5738。值得注意的是，在紧急恢复场景中，使用更大的缓存大小来恢复数据是非常有价值的。



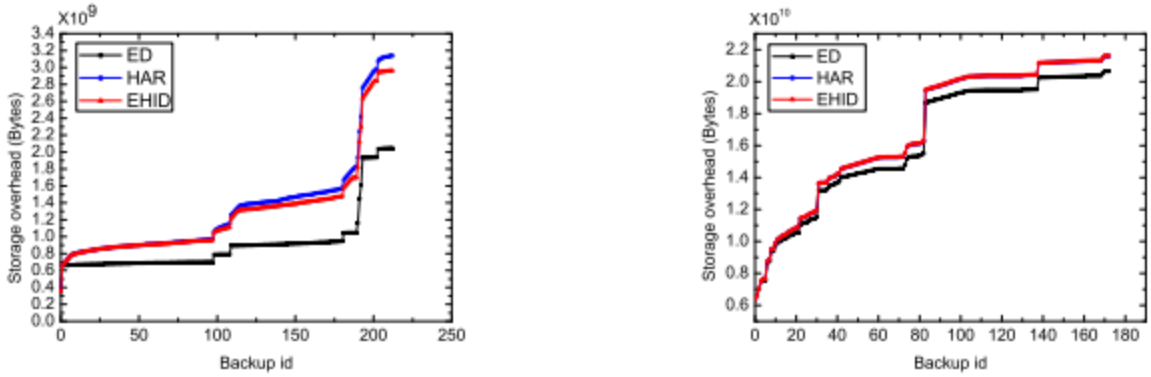
(a) OPT 30. (b) OPT 60. (c) OPT 90.

Fig. 11. Speed factor with FSL dataset.

评估存储开销

我们的实验结果表明，在Linux和FSL数据集上，三种方法（即ED、HAR和EHID）的平均重复数据消除率在96.73%到98.73%之间。由于这三种方法的重复数据消除率非常相似，为了使性能比较更直观、更清晰，我们评估的是存储开销，而不是重复数据消除比率。这里，存储开销是指在基于重复数据消除的备份系统完成前x次（即备份id+1）备份之前累积的存储块的总大小。

图12（a）显示了这三种方法与Linux数据集之间存储开销的比较。从图12（a）可以看出，ED使用的磁盘空间是最小的，这是因为ED不考虑恢复性能，只存储唯一的块。HAR的存储开销略高于EHID，这是因为HAR存在过重写现象。这种现象意味着碎片块在备份中被重写两次或两次以上。这种过度重写操作降低了容器的利用率，并生成了更稀疏的容器。由于稀疏容器数量的增加，重写算法重写了更多的碎片块并浪费了存储空间。图12（a）的实验结果表明，EHID的重写率平均比Linux数据集的HAR低2.83%。这意味着，当HAR重写10GB的数据块时，EHID可以节省约290MB的存储空间。



(a) Linux dataset. (b) FSL dataset.

Fig. 12. Storage overhead evaluation.

图12（b）显示了数据集为FSL时ED、HAR和EHID的存储开销。同样，ED的存储开销也是最小的。由于HAR在FSL数据集情况下的过重写影响不如Linux数据集情况那么强，因此HAR和EHID的存储开销非

常接近。例如，在第173版本的备份中，HAR的存储开销达到21590404782字节，而EHID的存储开销则达到21617550154字节。EHID的存储开销仅比HAR高0.13%，可以忽略不计。

结论

在这项研究中，我们发现传统基于重复数据消除的备份系统索引中的指纹条目可以分为三类，即无用指纹条目、碎片指纹条目和热指纹条目。无用的指纹条目和碎片化的指纹条目会浪费内存空间并增加磁盘I/O。此外，分离碎片指纹条目使基于重复数据消除的备份系统能够直接重写碎片块，从而减少数据碎片。为了解决由冷指纹条目（即无用和碎片化的指纹条目）引起的性能瓶颈，我们首先提出了HID，该HID分离冷指纹条目，并仅在内存中保留热指纹条目。与最先进的方法HAR相比，HID提高了内存利用率和备份性能。由于减少了数据碎片，HID还提高了恢复性能。

我们引入了FTU特性来弥补Bloom滤波器的不足。为此，我们提出了一种进化的HID策略EHID。EHID嵌入了仅映射热指纹的Bloom过滤器，从而降低了误报率。因此，EHID表现出两个显著的优点：EHID在识别唯一块时避免了基于磁盘的索引访问。EHID降低了集成Bloom过滤器的误报率，从而进一步减少了磁盘访问。这些显著特征使EHID保持在高效模式。此外，EHID可以有效地提高基于重复数据消除的备份系统的吞吐量。原因有两方面。首先，EHID只在内存中存储热指纹条目，这可以提高指纹查找的命中率。其次，EHID显著减少了基于磁盘的索引查找所产生的磁盘I/O数量。更具体地说，为了加快指纹查找，EDID将索引表分离，并在备份完成后将其存储为两个文件，即热索引文件和冷索引文件。如果发生内存丢失，EHID将在基于磁盘的索引中查找丢失的指纹条目。由于EHID中基于磁盘的索引只包含热索引文件，而热索引文件是整个索引表的一小部分，因此EHID的基于磁盘索引的大小远小于传统方法。因此，就在基于磁盘的索引中查找指纹条目而言，EHID比传统方法快得多。

我们进行了大量的实验来定量评估EHID的性能。实验结果和分析表明，EHID可以显著提高基于重复数据消除的备份系统的性能。

未来工作的可能方向包括将EHID和近乎精确的重复数据消除相结合，以进一步探索EHID的性能行为。

> 原文：《Improving the Performance of Deduplication-based Backup Systems via Container Utilization based Hot Fingerprint Entry Distilling》

摘要

重复数据消除技术构建一个由指纹条目组成的索引，以识别和消除重复数据的重复副本。基于磁盘的索引查找的瓶颈和消除重复数据块导致的数据碎片是重复数据消除中的两个具有挑战性的问题。基于重复数据消除的备份系统通常使用将连续块与其指纹一起存储的容器来保留数据位置，以缓解这两个问

题，但这仍然不够。为了解决这两个问题，我们提出了一种基于容器利用率的热指纹条目提取策略，以提高基于重复数据消除的备份系统的性能。我们将索引分为三个部分，即热门指纹条目、碎片指纹条目和无用指纹条目。利用率小于给定阈值的容器称为稀疏容器。指向非稀疏容器的指纹条目是热指纹条目。对于剩余的指纹条目，如果指纹条目与即将到来的备份块的任何指纹匹配，则将其分类为碎片指纹条目。否则，它将被归类为无用的指纹条目。我们观察到，热指纹条目只占索引的一小部分，而其余指纹条目占索引的大部分。这一有趣的观察结果启发我们开发一种名为HID的热门指纹条目提取方法。HID从索引中分离无用的指纹条目，以提高内存利用率并绕过磁盘访问。

此外，HID将碎片指纹条目分离，使基于重复数据消除的备份系统直接重写碎片块，从而减轻不利的碎片化。此外，HID还引入了一个功能，将碎片块视为唯一块。这一功能弥补了Bloom过滤器无法直接识别某些重复块（即碎片块）的缺点。为了充分利用上述特性，我们提出了一种称为EHID的进化HID策略。

EHID包含Bloom过滤器，只有热指纹才会映射到该过滤器。在这样做的过程中，EHID表现出两个显著的特征：（i）EHID避免了磁盘访问以识别唯一块和碎片块；（ii）EHID大幅降低集成Bloom滤波器的假阳性率。这些显著特征将EHID推向高效模式。我们的实验结果表明，当使用Linux数据集和FSL数据集时，我们的方法分别将索引的平均内存开销减少了34.11%和25.13%。此外，与最先进的方法HAR相比，使用Linux数据集，EHID将平均备份吞吐量提高了2.25倍；当涉及到FSL数据集时，EHID将平均磁盘I/O流量减少了高达66.21%。EHID还略微提高了系统的恢复性能。

ACM模式引用

Datong Zhang, Yuhui Deng, Yi Zhou, Yifeng Zhu, and Xiao Qin. 2020. Improving the Performance of Deduplication–based Backup Systems via Container Utilization based Hot Fingerprint Entry Distilling. J. ACM 37, 4, Article 111 (August 2020), 25 pages.
https://doi.org/10.1145/1122445.1122456

介绍

数据的爆炸性增长成为数据存储系统面临的日益严峻的挑战[13，27]。重复数据消除是一种现代技术，可以识别和存储唯一的块，从而显著减少对存储空间的需求[4，12]。此外，重复数据消除可以最大限度地减少网络存储系统中冗余数据的网络传输[23]。由于其高效性，重复数据消除已被广泛研究。新兴的重复数据消除应用，如I/O重复数据消除[18]、文件系统重复数据消除[33]和内存系统重复数据删除[15，16，32]，都在宣扬重复数据消除的日益普及和重要性。基于重复数据消除的备份系统首先将备份流划分为固定或可变大小的块[26，29]，然后为每个块计算SHA–1摘要[28]（即指纹）。为了保留备份流的位置，容器[19，34]用于存储连续的数据块及其指纹。当备份块时，系统向（指纹）索引发出查找请

求，以将当前块的指纹与存储块的指纹进行比较。如果发生匹配，则块是重复副本，将从备份流中删除。否则，块是唯一的，并且将被存储。

由于每个区块备份都会导致索引查找来定位匹配的指纹，因此当索引的大小太大而无法存储在内存中时，由于基于磁盘的索引访问性能较差，重复数据消除的性能会大幅下降[14]。先前的研究表明，访问磁盘驻留索引的速度至少是访问内存驻留索引的1000倍[9]。频繁的基于磁盘的索引查找是基于重复数据消除的备份系统中最重要的性能瓶颈之一[31]。例如，假设数据块的平均大小为8KB，备份800TB的数据集将产生至少2TB的指纹，这将太大而无法存储在存储器中[31]。有一种重复数据消除系统将25GB的内存用于索引[14]，如此巨大的索引将对内存造成巨大压力，并导致大量磁盘I/O。基于重复数据消除的备份系统中的另一个瓶颈是数据碎片[17，24]。碎片化表示一种现象，即重复数据消除后，备份流中逻辑上连续的数据块在物理上分散在不同的容器中。严重分散的数据块被视为碎片块。数据碎片会降低恢复备份流的性能，因为碎片会破坏空间局部性[21]。

为了避免上述对索引的昂贵查找请求，将容器元数据（即存储在容器中的指纹）从磁盘预取到存储器，以快速识别备份流中的连续重复块。因此，形成了由多个指纹条目组成的索引，以便于容器元数据预取；并且每个指纹条目将指纹映射到一个或多个容器的地址[11]。图1简要显示了基于重复数据消除的备份系统中的几个关键概念（例如，指纹条目、索引表和容器）及其关系。请注意，索引表由两部分组成：基于内存的索引和基于磁盘的索引。在虚线框中，第一行中的数字表示存储的数据块的指纹，而第二行中的号码表示容器的ID，每个容器存储特定的数据块。

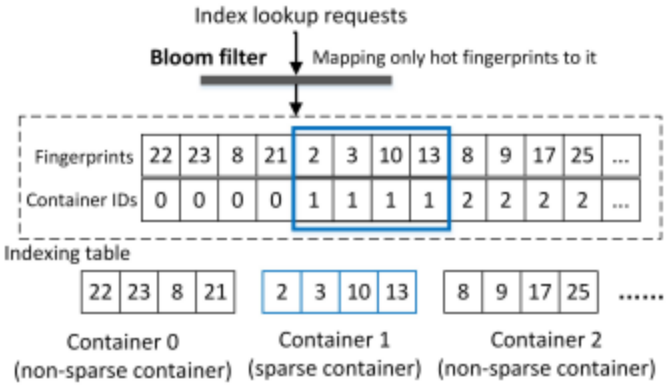


Fig. 1. The relationship of fingerprint entries and containers.

动机1：分析重复数据消除过程表明，索引表中的指纹条目可分为三类：热指纹条目、碎片指纹条目和无用指纹条目。让我们用图1中的例子来详细说明这些术语。在图1的底部是三个容器，其中容器0和2是非稀疏容器，容器1是稀疏容器。在本研究中，利用率（详见第3节）小于给定阈值的容器称为稀疏容器；指向非稀疏容器（即容器0和2）的指纹条目是热指纹条目，而冷指纹条目指向稀疏容器，因此在本例中我们有四个冷指纹条目2、3、10和13。如果冷指纹条目的指纹与未来备份流中包含的任何数据块的指纹相同，则该冷指纹条目被分类为碎片指纹输入。否则，这个冷指纹条目是无用的指纹条目，因为在未来的备份流块中没有匹配的指纹。将指纹条目分为三类的基本原理有三个方面。首先，内存中的热指纹条目有助于避免基于磁盘的索引访问，这会在重复数据消除过程中加快备份流中重复的块标识。其次，在内存中保留无用的指纹条目浪费了稀缺的内存空间，因为无用的指纹条目的重复块识别没有贡

献。第三，碎片化指纹条目反映了碎片化的数据块（请参阅第3节中的详细讨论），应该将其重写为新的容器，以缓解数据碎片化。

动机2：从索引表中删除碎片指纹条目使基于重复数据消除的备份系统将碎片块视为唯一的块，并直接将其重写到新的容器中，从而缓解数据碎片化。现在，我们用图1中的上述示例演示了删除碎片指纹条目如何使基于重复数据消除的备份系统直接重写碎片块，以减轻数据碎片。让我们首先回顾一些术语如下。**碎片化是指在重复数据消除后，备份流中逻辑上连续的数据块在物理上分散在不同的容器中的现象，这种块被称为碎片化块。**稀疏容器是指利用率小于给定阈值的容器。因为图1中的容器1是一个稀疏容器，所以指纹2、3、10和13表示的相应数据块几乎不会出现在后续的备份流中。为了缓解数据碎片化，即使稀疏容器的一些块出现在随后的备份流中，它们也会被视为碎片化的块，并与其他唯一的块一起重写到新的容器中。例如，我们假设后续备份流包含四个块2、A、B和C，每个块向索引表（全局指纹条目）发出索引查找请求；那么，只有指纹2匹配。因此，后续备份流中的新块2是重复的块，因为在稀疏容器1中存在新块2的重复副本。我们不是对新块2进行重复数据消除，而是将新块2视为分段块，并将其与备份流中的其他唯一块一起重写到新容器中，以减轻数据分段。注意，传统的基于去重的系统首先将新的块2识别为重复的块，因为分段的指纹条目（例如指纹2的条目）没有被移除，然后使用额外的重写算法来确定重复的块是否是分段的块，这增加了计算开销。如果我们从索引表中删除碎片化的指纹条目，则新的块2被标识为唯一的块，并直接重写到新的容器中，以减轻数据碎片化。请注意，在本文中，稀疏容器1中存储的块2和备份流中的新块2都被称为碎片块。分段块的指纹条目（例如指纹2的条目）被称为分段指纹条目。

受上述动机的启发，在本研究中，我们首先提出了一种基于容器利用率（详见第3节）的热指纹条目提取方法HID，以缓解磁盘I/O瓶颈并减少数据碎片。HID利用容器利用率来识别热指纹条目和冷指纹条目。HID在备份过程结束时从索引表中分离冷指纹条目。然后在下一个备份过程中，只使用热指纹条目来构建索引表（包含基于内存的索引和基于磁盘的索引）。通过这样做，HID提高了基于内存的索引查找的命中率，因为索引表中的所有指纹条目都是热指纹条目，从而避免了频繁的基于磁盘的索引访问。此外，HID减少了数据碎片，因为碎片指纹条目的分离使基于重复数据消除的备份系统能够将碎片块视为唯一块，直接将其重写到容器中。

动机3：值得注意的是，我们提出了一个显著的特征，称为FTU——碎片块被视为唯一块。FTU的机制是HID只在索引表中保留热指纹条目，在索引表上不留下碎片指纹条目。当对分段块执行索引查找时，索引表中没有匹配的条目，因此分段块被视为唯一块。FTU的引入弥补了Bloom滤波器的不足。Bloom过滤器是一种重复数据消除优化技术，用于识别备份流中的唯一块。然而，由于误报问题（即哈希冲突），Bloom过滤器无法识别备份流中的重复块[5，22，34]。如果没有FTU，碎片块可能会降低基于重复数据消除的备份系统的性能。更具体地说，分段块是重复块的子集。Bloom过滤器不会识别备份流中的每个碎片块，并且会向索引表发出索引查找请求（见图1），这可能会导致基于磁盘的索引访问。幸运的是，与FTU匹配的Bloom过滤器（通过仅将热指纹映射到该Bloom过滤器）将碎片块视为唯一的块，并且可以有效地识别这些碎片块，从而避免索引表查找。

为了充分利用FTU和Bloom滤波器的优势，我们最终提出了一种基于容器利用率的进化热指纹条目提取策略——EHID——来改进我们的初始HID。EHID嵌入Bloom过滤器，并仅将热指纹映射到Bloom过滤

器中。在这样做的过程中，EHID表现出三个优点。首先，EHID在识别唯一块和碎片块时有效地避免了基于磁盘的索引访问。其次，通过仅将热指纹映射到集成Bloom过滤器中，降低了Bloom过滤器的误报率（请参见第5.3节）。第三，EHID在基于重复数据消除的备份系统中实现了相当大的性能改进。综上所述，本文的贡献如下：

- 我们获得了一些令人信服且有价值的发现。首先，为了减少基于磁盘的索引查找并缓解数据碎片化，我们将全局指纹条目分为三类：热指纹条目、无用指纹条目和碎片指纹条目。无用的指纹条目对于识别备份流中的重复块是无用的，所以将它们缓存在内存中是不明智的。分离无用的指纹条目可以缓解基于磁盘的索引查找问题。碎片指纹条目的分离使基于重复数据消除的备份系统能够将碎片块视为唯一的块，直接将其重写到新的容器中，从而减轻数据碎片。其次，我们揭示了容器利用率是促进指纹条目的3类分类的有效指标。指向非稀疏容器的指纹条目是热指纹条目，而指向稀疏容器的指印条目是无用的或碎片化的指纹条目。
- 我们提出HID来处理基于磁盘的索引查找瓶颈，并缓解基于重复数据消除的备份系统中的数据碎片。
- 我们提出了一种称为EHID的进化HID。EHID嵌入Bloom过滤器，并且仅将热指纹映射到嵌入的Bloom过滤器。通过这样做，EHID显著避免了昂贵的基于磁盘的索引访问。大量实验结果表明，EHID实现了相当高的性能。

本文的其余部分组织如下。第2节总结了相关工作。第三部分介绍了相关的背景知识。第4节介绍了我们对指纹录入类别转换的动机和分析。第5节演示了我们的设计和实现。进行了综合实验，以评估第6节中提出的方法。第7节是本文的结语。

相关工作

已经进行了许多研究来处理基于磁盘的索引查找的瓶颈。例如，Bloom滤波器[5]被精确的重复数据消除方法[22，34]用于识别唯一的块。然而，Bloom过滤器的效率随着备份数据量的增加而下降。这是因为Bloom滤波器的假阳性率[5，22，34]随着其空间使用量的增长而变大。接近精确的重复数据消除方法通过对所有指纹条目的一部分进行粗略采样（例如，随机采样、均匀采样）来执行重复数据消除[3，14，20，31]。由于采样指纹项的数量远小于指纹项的总数，因此近乎精确的重复数据消除方法可以减少索引查找导致的磁盘访问次数。尽管精确和近乎精确的重复数据消除方法减少了磁盘I/O的数量，但这两种方法都涉及大量无用的指纹条目和碎片化的指纹条目，这可能会导致频繁的数据丢失。为了减轻数据碎片，HAR（即历史感知重写算法）利用容器利用率来识别和重写碎片块[10]。请注意，重写碎片块是指允许在磁盘上存储少量重复的块。不幸的是，HAR忽略了这样一个事实，即容器利用率可以促进对指纹条目进行分类。在这项研究中，我们利用容器利用率对指纹条目进行分类，旨在加快索引查找并缓解数据碎片。为了使相关工作更加清晰和全面，以下两段将更详细地介绍每项工作。

为了减少对基于磁盘的索引的访问，已经提出了许多重复数据消除方法。朱等人[34]使用Bloom过滤器来识别唯一的块，并使用容器来保留备份流的位置，从而克服了磁盘瓶颈。MFDedup[35]通过使用数据分类方法来生成最佳数据布局，从而维护备份工作负载的位置。然而，随着Bloom滤波器的空间使用

量的增长，Bloom滤波器[5]的假阳性率急剧增加。稀疏索引方法[20]从存储块的所有指纹条目（密集索引）中随机抽取一定百分比的指纹条目，以构建稀疏索引。稀疏索引方法减少了索引的内存开销，提高了备份吞吐量。但是，其重复数据消除率取决于备份流的位置和采样率。郭等[14]将采样方法从逻辑局部性应用到物理局部性。Extreme Binning[3]和ChunkStash[8]利用了文件的相似性理论，而不是备份流的局部性。但是，它对文件的相似性很敏感。Xia等人利用局部性和相似性，它适用于相似性较弱或较低的工作负载[31]。然而，由上述构造的索引表包含大量的冷指纹条目，并且冷指纹条目导致在执行索引查找时频繁地访问基于磁盘的索引。

由于碎片化大大降低了恢复性能，因此已经提出了许多用于减轻碎片化的重写算法。大多数重写算法分配小尺寸存储器来缓冲备份流中的连续数据块。缓冲区中的数据块序列称为逻辑序列。相反，位于磁盘上的数据块序列称为物理序列。如果逻辑序列与物理序列明显不同，则重写算法确定缓冲区具有许多分段块并对其进行重写。然而，由于分配内存只缓冲备份流的一小部分，因此它们无法准确地识别碎片块。为了解决这个问题，HAR[10]使用先前备份流的全局信息来预测后一备份流的碎片。HAR克服了其他重写算法的不足，提高了恢复性能和去重率。不幸的是，HAR没有考虑到容器利用率这一重要指标反映了指纹条目的类别。此外，在HAR中更新碎片指纹条目的开销降低了备份性能。

背景

分段：图2显示，通过处理第一个备份流，将唯一的块按顺序聚合到容器中。第一备份流的数据块存储在容器1、2、3和4中。当第二备份流到达时，块A被标识为重复的。因此，A不被存储，而是被指向位于容器1中的共享块A的指针所代替。下一个组块L被标识为唯一组块，因此在该示例中它被存储在容器5中。在此过程之后，将对备份流的其余数据块逐一进行重复数据消除。最后，第二个备份流的所有连续数据块都分散在不同的容器中，这称为数据碎片。

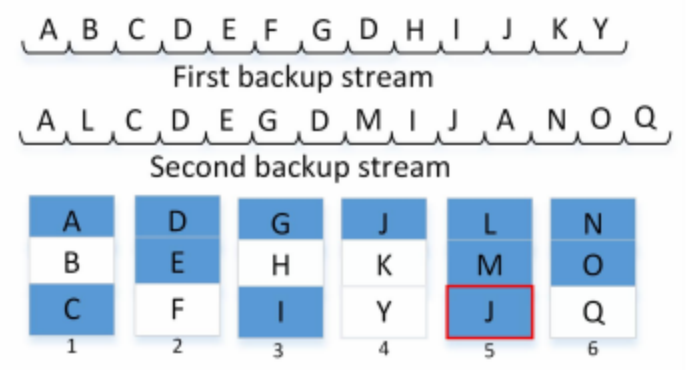


Fig. 2. Example of deduplication for two backup streams.

容器利用率：图2表明，对于第一个备份流，容器1有三个共享块A、B和C。然而，对于第二个备份流而言，容器1只有两个共享块C和A。对于所有后一个备份流来说，容器1很有可能只有不到两个的共享块，因为备份数据被连续地修改（删除/添加/修改数据块）。换句话说，随着备份和归档系统中备份版

本的增长，容器中共享块的数量会逐渐减少。这种效果通过容器利用率来反映，并且可以表示为以下等式1。容器利用率定义为：

$$CU_i^k = R_i^k / S, \quad (1)$$

其中k表示第k个备份流，i表示第i个容器，S表示容器大小（数据块的数量），R_i^k 表示第k备份流的第i个容器中的共享块的数量，CU_i^k 表示用于第k个备用流的第i个容器的容器利用率。

如图2所示，由于CU^1_1=3/3=100%；CU^2_1 = 2/3 = 66.66%，我们可以得出结论，容器利用率越高，在数据备份和恢复过程中访问该容器的频率就越高。容器利用具有历史传承特征[10]。对于容器i，其容器利用率随着备份版本（流）的增加而降低。即$CU^1_i \geqslant CU^2_i \geqslant \cdots \geqslant CU^n_i$。因此，对于典型的备份场景，除非发生罕见的回滚，否则容器利用率不会增加[10]。尽管在极少数情况下，容器利用率的增长对整体系统性能的影响微乎其微。

此外，备份内重复对容器利用率没有影响。例如，如图2所示，容器2包含三个数据块D、E和F。尽管第一个备份流包含数据块E、F和D的两个重复副本，但R^1_2仍计算为3。这表明，无论E、F或D在第一备份流中重复出现多少次，R^1_2都保持不变。由于S是固定值，根据等式1，容器利用率CU^1_2也保持不变。

稀疏容器：稀疏容器表示利用率低于给定阈值的容器。例如，如果阈值P设置为70%，对于第二个备份流，容器1是稀疏容器，因为CU^2_1＝66:66% < P。根据其历史继承特性，如果特定容器在前一个备份流中被标识为稀疏容器，则该容器在随后的备份流中仍将是稀疏容器。这是我们方法的基本前提。

重写碎片块：为了提高恢复性能，HAR[10]重写碎片块并减少稀疏容器的数量。如图2所示。例如，假设稀疏阈值P＝50%，由于CU^2_4＝1 / 3＝33.33% < P，所以容器4是稀疏容器。当处理第二备份流时，由于共享块J的指纹条目指向稀疏容器4，所以复制的块J（第二备份串流中的块J）被识别为分段块。因此，分段的块J与唯一的块L和M一起被写入到新的容器5。当恢复第二备份流时，块J直接从缓存在存储器中的容器5获得，因为在恢复前一个块L时容器5已经从磁盘读取到存储器。相反，如果不重写分段的块J，读取容器4不可避免地会获得块J，这会导致磁盘I/O并损害恢复性能。

详细的重复数据消除过程：如图3所示，索引表（包含基于内存的索引和基于磁盘的索引）由存储块的所有指纹条目组成。每个指纹条目都将指纹映射到相应的容器ID（逻辑地址）。容器位于磁盘存储器中，每个容器都保存相应的指纹。配方用于记录用于恢复数据的逻辑块序列。活动容器用于存储唯一的块和重写的块。字节流（备份流）已经被划分为块，并且指纹已经被计算出来。当备份新的区块时，系统首先转到容器元数据缓存以定位相同的指纹。如果指纹没有在缓存中命中，那么系统将生成索引表的查找请求，这可能会导致访问基于磁盘的索引。在搜索索引结束时，确定新块的属性。如果新的区块是唯一的区块，系统会将其写入活动容器，并生成一个新的指纹条目并将其插入索引表。如果新区块是重复的区块，则不会对其进行存储。如果新区块是分段区块，系统会将其重写到活动容器，然后更新分段指纹条目（使条目指向活动容器而不是稀疏容器）。请注意，更新碎片指纹条目可能需要访问基于磁盘的索引，因为它必须再次查找索引表以定位碎片指纹条目，然后更新条目的值（容器ID）。

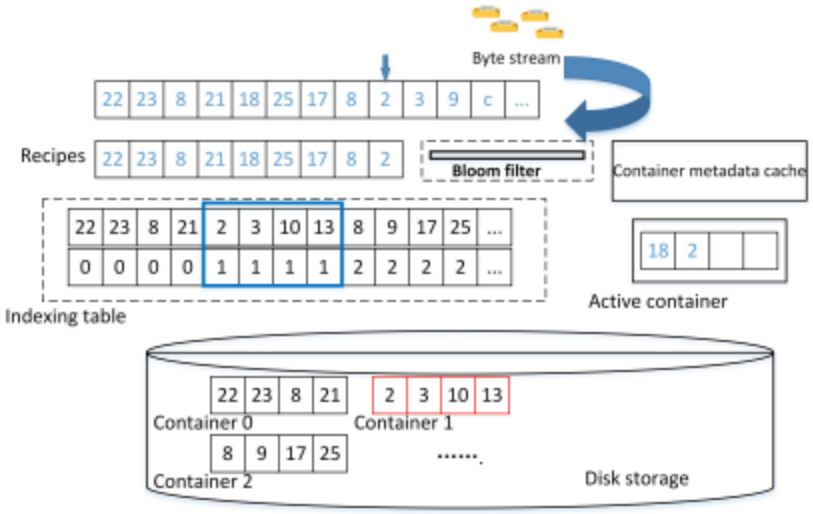


Fig. 3. The detailed process of deduplication.

恢复过程：在重复数据消除过程之后，文件的数据块可能会完全混乱，分散在所有磁盘的各个容器中，索引表中的指纹条目也会混乱。因此，由于缺乏数据块的序列信息，无法利用索引表来恢复数据。使用存储在磁盘中的文件配方来恢复数据，而不是使用索引表。文件配方由指纹和相应数据块的物理地址组成。文件配方中的指纹序列与备份流中的数据块序列一致。每个备份版本都会生成一个特定的文件配方。如果需要第k个备份版本进行恢复，系统将首先分析文件配方的元数据，获得第k个文件配方的存储路径，将文件配方加载到内存中，并从磁盘读取数据块，利用文件配方信息恢复数据。由于文件配方独立于索引表，因此从索引表中删除任何指纹条目都不会对恢复过程产生任何影响。

动机探讨与指纹录入类别转换分析

动机讨论

查找索引表和不断增加的碎片块分别会导致频繁访问基于磁盘的索引和数据碎片，从而导致基于重复数据消除的备份系统出现主要性能瓶颈。我们的关键思想是，分离无用的指纹条目对识别和消除重复的块没有影响，而分离碎片化的指纹条目使系统直接重写碎片化的块并减少数据碎片。我们的主要观察结果是，稀疏容器反映了无用和碎片化的指纹条目。例如，图3显示了索引表中指纹为2、3、10和13的指纹条目是冷指纹条目，因为它们都指向容器1，假设容器1是稀疏容器。

这些事实使我们能够将指纹条目分为热门、碎片和无用，而碎片和无用的指纹条目构成了冷指纹条目。例如，在图3中，索引表中指向稀疏容器的指纹条目被分类为冷指纹条目，其余的被分类为热指纹条目。更具体地说，指纹为10和13的指纹条目是无用的指纹条目，因为由于数据备份场景的特点，它们在后续备份过程中不会被访问（详见第3.2节）。指纹条目2和3是碎片指纹条目，因为在随后的字节流中存

在重复的指纹（块）2和3，但是它们的共享块位于稀疏容器1中。在本文中，我们将重复块2和3以及共享块2和共享块3等块称为碎片块。

分离无用的指纹条目可以有效地节省内存空间，因此它总是被执行的。但是，分离碎片指纹条目是可选的。在HID的设计中，我们选择分离碎片指纹条目，原因有以下三个：

- 分离碎片指纹条目引入了FTU功能。FTU功能表示，分离碎片指纹条目使基于重复数据消除的备份系统将碎片块视为唯一块。例如，在图3中，指纹2和3的条目被分离后，字节流中的分段块2和3通过遍历索引表被识别为唯一块。
- 分离碎片指纹条目使得基于重复数据消除的备份系统直接重写碎片块，这种直接重写机制减轻了数据碎片，避免了更新碎片指纹条目的开销。例如，由于FTU的特性，字节流中的分段块2和3（见图2）被转换为唯一块（FTU的唯一块）。基于重复数据消除的备份系统直接将这些FTU的唯一块写入活动容器，并为这些块生成新的指纹条目，而不是更新分离的碎片指纹条目。
- 将指向稀疏容器的指纹条目（即，冷指纹条目）划分为无用指纹条目和碎片指纹条目的操作是昂贵的。为了避免这种开销，指向稀疏容器的指纹条目直接与所有指纹条目分离。

集成Bloom过滤器和FTU：FTU是对Bloom过滤器的补充。基于重复数据消除的备份系统中的Bloom过滤器可以通过将所有指纹映射到其自身中来有效地识别唯一的块。然而，由于其假阳性，它缺乏识别重复块的能力[5，22]。具体来说，字节流中的重复块必须查找索引，这可能导致基于磁盘的索引访问。幸运的是，通过集成FTU，Bloom过滤器能够识别一些重复的块，因为FTU将所有碎片块视为唯一的块。请注意，分段块属于重复块，而分段块只是重复块的一小部分。总之，Bloom过滤器和FTU的组合进一步防止了基于重复数据消除的备份系统进行基于磁盘的索引访问。在代码实现层面，只要我们将热指纹而不是所有指纹映射到集成的Bloom过滤器，Bloom过滤器和FTU就可以很好地集成。

基于上述分析，我们提出了EHID，通过集成Bloom过滤器和FTU来进一步提高基于重复数据消除的备份系统的备份性能。EHID将Bloom过滤器放在容器元数据缓存和索引表之间，它只将热指纹映射到这个Bloom过滤器中（见图3）。在这样做的过程中，大多数唯一块（正常和FTU的唯一块）由Bloom过滤器而不是索引表来标识。例如，由于冷指纹条目（即指纹2、3、10和13的条目）没有映射到Bloom过滤器，所以当处理字节流中的新块2和3时，Bloom过滤器将这些块识别为唯一的块。因此，这避免了查找索引表来确定这两个块的属性（唯一、重复或分段）。注意，传统方法将继续搜索索引表，因为它们将冷指纹（即，2、3、10和13）映射到Bloom过滤器，结果是Bloom过滤器不能将新块2和3作为唯一块来处理。除了FTU和Bloom滤波器的相互增益之外，仅将热指纹而不是所有指纹映射到Bloom滤波器中可以降低误报率。

指纹录入类别转换分析

HID和EHID仍然需要解决两个关键问题。也就是说，随着工作负载的发展，无用或碎片化的指纹条目是否可以重新分类为热门指纹条目？当这种情况发生时，HID和EHID如何处理？为了回答这两个问

题，我们进行以下分析。请注意，创建索引表（全局指纹条目）是为了快速识别备份流中的重复块。考虑以下典型的数据备份场景：

工程师创建一个文件，该文件由{A1；A2；A3；A4}$_0$表示，下标0表示该文件是初始版本，“{}”中的每个字符表示一个数据块，{A1；A3；A4}$_0$表示文件包含四个数据块A1、A2、A3和A4。为了保护文件，工程师使用基于重复数据消除的备份系统进行第一次备份。由于此时备份存储中没有存储块，因此索引表为空。索引表由（⌀）0表示，其中下标0表示索引表尚未更新，“（）”中的每个字符表示条目的指纹。当数据块A1–A4被备份时，系统搜索索引表（⌀）0。由于索引表为空，这些数据块被标识为唯一块，并存储到磁盘中。然后，系统为存储的四个块创建指纹a1–a4的条目。由于索引表为空，这些数据块被标识为唯一块，并存储到磁盘中。然后，系统为存储的四个块创建指纹a1–a4的条目。此时，索引表从（⌀）0更新为（a1—a4）$_1$。

稍后，工程师修改文件，文件变为{A1；A2；B1；A4}$_1$。此时，他进行第二次备份。系统遍历（a1—a4）$_1$，并分别找到指纹a1、a2和a4。因此，数据块A1、A2和A4被识别为重复块。数据块B1被识别为唯一块，因为在（a1—a4）1中没有对应的指纹。然后，系统为数据块b1创建指纹条目b1，并将b1插入索引表。现在，索引表从（a1—a4）1更新为（a1—a4；b1）$_2$。我们可以发现，在第二次备份期间，只有（a1—a4）1中的指纹条目a1、a2和a4有助于识别{a1；a2；B1；a4}$_1$中的重复块a1、a2和a4。换句话说，从索引表（a1—a4）1中删除指纹a3的条目对重复数据消除过程没有影响。

之后，工程师再次修改文件，文件变为{A1；A2；B1；C1；C2}$_2$。此时，她进行第三次备份。同样，只有（a1—a4；b1）$_2$中的指纹a1、a2和b1的条目有助于识别{a1；a2；b1；C1；C2}$_2$中的重复块a1、a2和b1。因此，从（a1—a4；b1）$_2$中删除指纹条目a3和a4不会产生副作用。

在上面的备份示例中，索引表按以下顺序更新：

$$\begin{aligned}(\varnothing)_0 \rightarrow (a1, \underline{a2}, a3, a4)_1 \rightarrow (a1, \underline{a2}, \underline{a3}, a4, b1)_2 \\ \rightarrow (\underline{a1}, \underline{a2}, \underline{a3}, a4, b1, c1, c2)_3 \rightarrow \ldots \rightarrow (\underline{a1}, \underline{a2}, \underline{a3}, \\ \underline{a4}, \underline{b1}, c1, c2, \ldots, xm)_n,\end{aligned} \tag{2}$$

其中n是备份的数量，带下划线的字母表示无用的指纹条目。此示例表明，尽管指纹条目的总数随着备份版本的增加而增加，但无用指纹条目的数量也相应增加。先前的研究[6，7，11]表明，随着备份的增加，碎片块的数量会增加，碎片指纹条目的数量也会增加。因此，我们可以推断，热指纹条目的数量保持在稳定水平，或者随着备份版本的增加而缓慢增长。

在典型的数据备份场景中，随着备份版本的增加，三个指纹条目的变化趋势如下：

- 热指纹条目：如果指纹条目在当前备份中被归类为热指纹条目，则在后续备份中，它可能是热指纹条目或成为无用或碎片指纹条目。
- 无用指纹条目：如果指纹条目在当前备份中被归类为无用，那么在后续备份中它仍然是无用指纹条目。

- 碎片指纹条目：如果指纹条目在当前备份中被归类为碎片指纹条目，那么在后续备份中，它仍然是碎片指纹条目。

由于典型数据备份场景的特点，冷指纹条目（即无用指纹条目和碎片指纹条目）不再被重新归类为热指纹条目。因此，HID和EHID在任何备份版本中都能有效工作。

设计和实现

体系结构概述

图4描述了我们的EHID策略的总体架构和工作流程。整个系统可分为三个大模块，包括指纹查找模块（FLM）、容器利用率监测模块（CUMM）和索引分离模块（ISM）。这些模块通过重复以下三个步骤协同工作。

- 在步骤1，当备份每个新的数据块时，FLM负责识别唯一的块和重复的块。同时，CUMM通过记录备份过程中哪个数据块属于哪个容器来计算每个容器的利用率，然后获得稀疏容器的列表。该列表用于区分热指纹条目和冷指纹条目。
- 在步骤2，ISM通过检查指纹条目是否指向列表上记录的稀疏容器，从所有指纹条目中识别冷指纹条目和热指纹条目。如果指纹条目指向稀疏容器，则它是冷指纹条目。否则，天气会很热。通过这种方式，ISM分离冷指纹条目和热指纹条目。
- 在步骤3，热指纹条目（热索引文件）成为由基于内存和基于磁盘的索引组成的索引表。热指纹用于在下一次备份开始时初始化Bloom过滤器。最后，系统返回步骤1并重复上述过程。

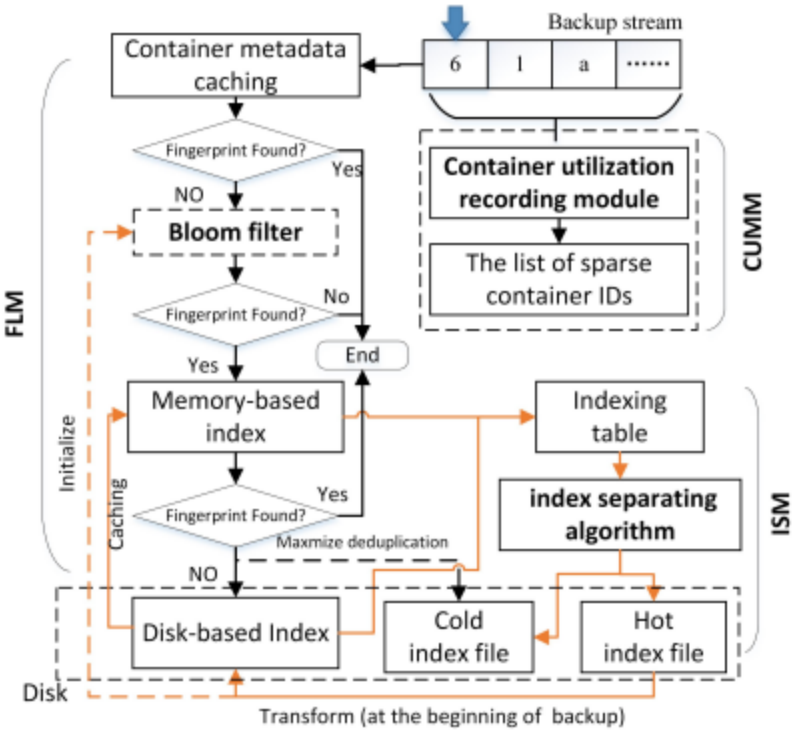


Fig. 4. Architecture overview of EHID based deduplication-based backup system.

关键过程：如图4中的橙色箭头所示，基于内存和基于磁盘的索引构成了索引表。随着备份的数据越来越多，索引表中所有指纹条目的数量也会越来越多，即使一部分指纹条目逐渐变为冷指纹条目。索引分离算法（见图4）在每个备份过程结束时将索引表中的指纹条目划分为冷索引文件和热索引文件。以这种方式，越来越多的冷指纹条目被保存到冷索引文件，并且热索引文件中的热指纹条目的数量保持较小。在每次备份开始时，只有热索引文件用于构建索引表（基于磁盘的索引和基于内存的索引），该文件还用于初始化Bloom过滤器。Bloom过滤器在每次备份开始时都会重新初始化。除非我们的设计要求最大化重复数据消除率，否则FLM不会查询冷索引文件（请注意，传统的基于重复数据消除的备份系统仅提供FLM模块）。

冷索引文件和热索引文件存储在磁盘中。在特定备份的初始阶段，系统将热索引文件作为基于磁盘的索引，并将基于磁盘的一部分索引读取到内存中。此时，索引表等于基于磁盘的索引，并且索引表和基于磁盘的索引来包含基于内存的索引中的指纹条目。在备份过程中，会不断生成新的指纹条目，并将其插入到基于内存的索引中。如果分配的内存大小是固定的，则插入的指纹条目将触发内存中指纹条目的替换。在这个阶段，基于磁盘的索引可能不包括所有基于内存的索引。因此，用于重复数据消除的索引表应同时包含基于磁盘的索引和基于内存的索引，如图4所示。当前备份完成后，ISM将索引表分离为冷索引文件和热索引文件，并将它们再次存储在磁盘中。因此，每个备份版本都会更新热索引文件和冷索引文件。此外，在备份过程中，索引表会为后续备份流不断更新。

CUMM和ISM的开销较低。CUMM使用64位整数数组来记录稀疏容器。在我们的实验中，一个备份有大约3500个容器。假设所有的容器都是稀疏容器，CUMM只需要27.34KB的内存空间和一点计算开销即可操作。ISM通过只遍历索引表一次来分离冷指纹条目和热指纹条目。假设所有指纹条目的数量为N，

则该分离过程的时间复杂度为O（N）。值得注意的是，对于访问基于磁盘的索引的查找请求，时间复杂度也是O（N）。此外，在开始下一次备份之前，可以离线进行索引分离，以最大限度地减少开销。

热指纹条目的提取算法

Algorithm 1 Hot Fingerprint Entries Distilling Algorithm

Input: $list_n$; //list of sparse container IDs in nth backup
$glIndex_n$; // all fingerprint entries in nth backup stream
Output: $hIndex_n$; // hot fingerprint entries in nth backup stream
$coIndex_n$; // cold fingerprint entries in nth backup stream

1: **function** INIT()
2: $glIndex_n \leftarrow hIndex_{n-1}$;
3: **end function**
4: **function** LOOKUP( )
5: **while** $(new\_chunk \neq end_{backupStream})$ **do**
6: **if** $(new\_chunk = uniq\_chunk)$ **then**
7: $insert(glIndex_n, entry_{uniq\_chunk})$;
8: **end if**
9: $new\_chunk \leftarrow new\_chunk->next$;
10: **end while**
11: **end function**
12: **function** SEPARATES()
13: get $first$ in $glIndex_n$;
14: **while** $(first \neq end_{glIndex_n})$ **do**
15: **if** $first->data.value$ in $set(list_n)$ **then**
16: $write(first->data, coIndex_n)$;
17: **else**
18: $write(first->data, hIndex_n)$;
19: **end if**
20: $first \leftarrow first->next$;
21: **end while**
22: **end function**

算法1演示了HID的基本工作过程。它概括为以下三个重要步骤：

- 当第n次备份开始时，由第（n−1）次备份中的separate函数提取的热指纹条目（例如 $hIndex_{n-1}$）会在当前备份（例如glIndexn）中构建索引表，而不考虑$coIndex_{n-1}$。
- 在备份过程中，查找功能将唯一块的指纹条目（例如$entry_{uniq_chunk}$）插入索引表（例如$glIndex_n$）。同时，索引表中所有指纹条目的一小部分被转换为冷指纹条目。
- 当备份过程完成时，separate函数提取热指纹条目（例如$hIndeX_n$），并将它们存储到磁盘中以备下次备份。当下一次备份开始时，算法1再次返回到第一步。

EHID

图4概述了EHID的体系结构和工作流程。EHID有两个优点：

- EHID充分利用FTU和Bloom过滤器来有效地识别正常的唯一块和FTU的唯一块（即碎片块）。此设计可防止基于重复数据消除的备份系统在识别备份流中的唯一块时访问基于磁盘的索引。
- EHID采用的Bloom过滤器工作效率高。Bloom滤波器的假阳性率随着其空间使用的增长而增加。幸运的是，EHID及时地分离了冷指纹条目，同时最大限度地减少了热指纹条目的数量。这样的效果减少了Bloom滤波器的热指纹占用的空间。

访问基于磁盘的索引的分析

下面正式分析EHID的效率。假设一个备份流中的数据块数量是固定的，并且每个唯一的数据块必须触发查询基于磁盘的索引的请求。相反，每个重复的块可以以相似的概率触发对基于磁盘的索引的搜索，因为重复的块的索引查找请求可以在容器元数据高速缓存或基于存储器的索引中提供。每个重复块触发基于磁盘的索引查找请求的平均概率应该相等，并用P_{find}表示。假设uniqs和dups分别表示唯一块和重复块的数量。然后，对于传统的基于重复数据消除的备份系统，在一个备份流中访问基于磁盘的索引的查找请求数定义为：

$$disk\_lookups_{tradis} = uniqs + dups * p_{find}. \tag{3}$$

我们假设Bloom滤波器的假阳性率为0；对于配置了Bloom筛选器的传统基于重复数据消除的备份系统，基于磁盘的索引查找请求数定义为：

$$disk\_lookups_{tradis+Bloomfilter} = dups * p_{find}. \tag{4}$$

对于EHID，基于磁盘的索引查找请求的数量定义为：

$$disk\_lookups_{EHID} = dups^{'} * p_{find}. \tag{5}$$

请注意，dups'＜dups，因为EHID将一些重复的块（即碎片块）视为唯一的块（如FTU功能）。换言之，EHID识别的唯一块的数量大于传统的基于重复数据消除的备份系统（即uniqs′≥uniqs）识别的块的数量。相应地，EHID识别的重复块的数量小于传统的基于重复数据消除的备份系统（即dups′≤dups）。因此，我们可以推导出$disk_lookups_{tradis} \geq disk_lookup_{tradis+Bloomfilter} \geq disk_lookups_{EHID}$，这个表达式表明EHID访问基于磁盘的索引所需的查找请求数量最小。

我们将举一个例子来说明为什么EHID策略可以利用Bloom过滤器来识别碎片块，避免磁盘中的索引访问，从而提高备份吞吐量。请注意，传统方法使用的Bloom过滤器无法识别碎片块。假设磁盘中存储

有一个分段块J1，则新的备份块J2是J1的重复块。为了减轻数据碎片，尽管J2是一个重复的块，但它必须被写入磁盘。由于EHID不将碎片指纹映射到Bloom过滤器（以集成Bloom过滤器和FTU），因此系统无法找到块J1的指纹的映射位。因此，新的组块J2被识别为唯一的组块，并且将被写入磁盘。如果碎片指纹被映射到Bloom过滤器，则块J1的映射比特将被定位。由于Bloom过滤器的误报，系统必须进一步按顺序检查基于内存的索引和基于磁盘的索引，以识别块J2的属性，包括唯一块、重复块和碎片块。查找指纹（条目）可能会在以下步骤中触发磁盘I/O。

EHID Bloom滤波器的假阳性率

EHID的另一个迷人特征是它降低了集成Bloom滤波器的假阳性率。Bloom滤波器的假阳性率（FPR）的计算公式可以表示为：

$$FPR = (1 - e^{(-kn/m)})^k, \tag{6}$$

其中k表示散列函数的数量，m是Bloom滤波器的大小（比特），n是映射指纹的数量。假设具有k个散列函数和250000个指纹的128KB Bloom过滤器，FPR为14.28%，这意味着大约有14个查找请求用于访问索引表以处理每100个块。EHID只将热指纹（占Linux跟踪上所有指纹的37.8%）映射到Bloom过滤器，因此FPR降低到0.78%。这个例子表明，对索引表的访问显著减少。图5显示了FPR和热指纹条目的百分比之间的关系。结果表明，当百分比从100%降低到37.8%时，Bloom filter的FPR从14.2%降低到0.8%。请注意，x轴上的热指纹条目的百分比是由Linux数据集的1到213的不同备份版本生成的。

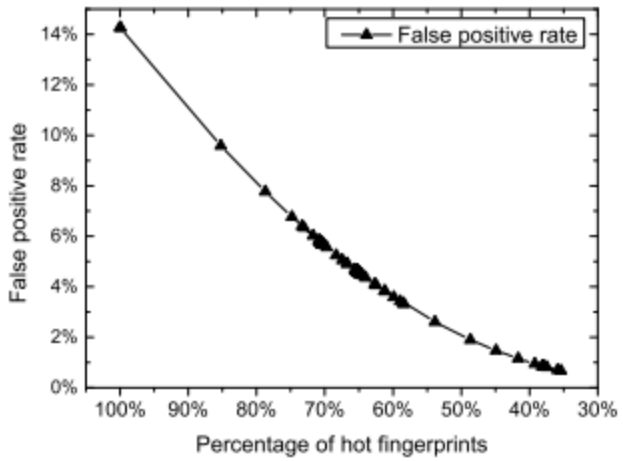


Fig. 5. False positive rate vs. percentage of hot fingerprint entries

实验评估

评估环境

在本节中，我们进行了一组实验来评估我们的热指纹条目提取方法的性能。我们在一个名为destor[11]的开源项目上实现了HID和EHID。为了定量评估性能，我们利用了四个指标，包括索引表的内存开销、备份性能、恢复性能和存储开销（比重复数据消除率更直观）。精确重复数据消除（ED）方法识别并删除每个重复的块，并且不使用任何重写算法。它是绩效评估的基准。此外，将HID和EHID的性能与最先进的算法HAR[10]进行了比较。由于HID和EHID只在备份性能上有所不同，因此省略了HID实验结果的其他三个指标（索引表的内存开销、恢复性能和存储开销）。

实验性操作系统是CentOS 7.4版（Linux版本3.10.0–693.11.1–el7.x86–64）。容器大小为4MB（包含大约1000个数据块）。稀疏阈值是HAR和EHID的一个非常重要的参数。

稀疏容器的数量随着稀疏阈值的增加而增加。这会导致更多的碎片块、无用块和冷指纹条目。因为EHID和HAR都通过重写分段块来提高恢复性能，所以更高的稀疏阈值导致EHID和HAR都重写更多的分段块，因此它们都实现了更高的恢复性能和更低的重复数据消除率。此外，和HAR相反，EHID将冷指纹条目从索引表中分离出来。因此，更高的稀疏阈值导致EHID分离出更多的冷指纹条目，这将进一步降低索引表的内存开销。结果，EHID进一步提高了备份性能，而HAR不能提高。当阈值降低到一个较低的值时，它将产生相反的过程。HAR已经证明，将阈值设置为50%可以很好地平衡重复数据消除率和恢复性能。因此，为了与HAR进行公平的比较，我们在本文的所有实验中都将阈值设置为50%。

Linux数据集和FSL数据集[1，2]是常用的公共数据集[30]。这两个数据集的特征如表1所示。由于FSL跟踪没有数据块的字节流，我们无法像Linux跟踪那样评估FSL跟踪的实际吞吐量。我们分析了重复数据消除过程中由索引查找引起的磁盘I/O行为，并使用磁盘I/O的数量来衡量吞吐量，并评估我们在FSL数据集上的方法的效率。

Table 1. Characteristics of workloads used for evaluation

dataset name	TS^1	DR^2	ACS^3	# of versions
Linux	99GB	97.85%	3.1KB	213
FSL	1.95TB	98.73%	4.4KB	173

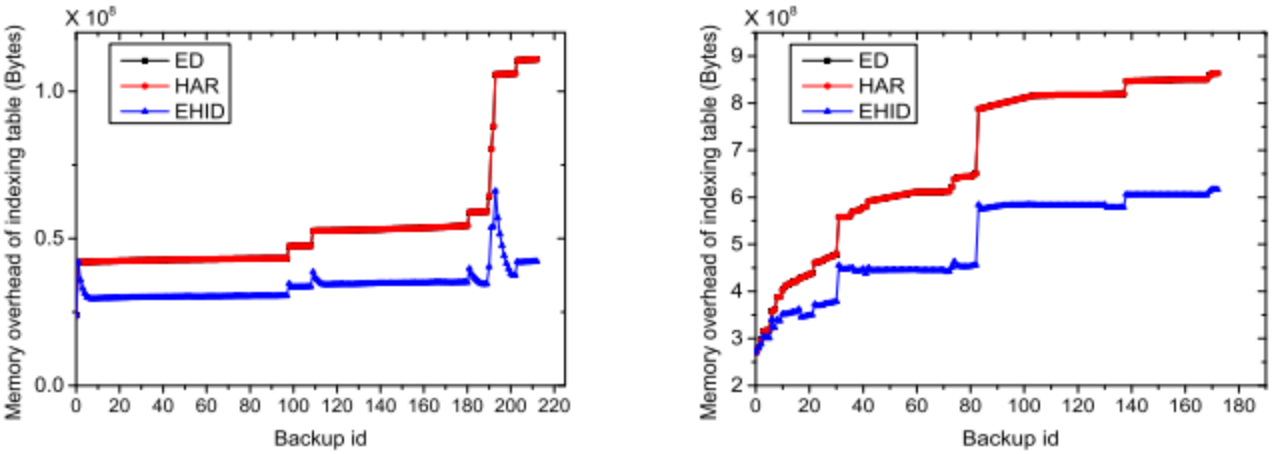
[1] TS stands for the total size of the trace.
[2] DR stands for the deduplication ratio of the trace.
[3] ACS stands for the average chunk size of the trace.

我们使用速度因子[19]作为评估恢复性能的指标。速度因子表示可以通过平均从磁盘读取容器来恢复的原始数据大小。速度因子的值越高，表示恢复性能越好。

索引表的内存开销评估

我们评估了Linux和FSL数据集上索引表的内存开销。请注意，内存开销意味着将索引表的所有指纹条目读取到内存中，以完全避免基于磁盘的索引访问。在图6（a）中，y轴表示用于执行重复数据消除的索引表的大小，x轴表示备份版本的ID。图6（a）所示的ED和HAR曲线完全重叠。这种趋势是意料之中的，因为这两种方法不会将冷指纹条目从索引表中分离出来。EHID可以有效地降低索引表的内存开销。

此外，随着备份版本的增加，EHID变得更加高效。例如，当备份版本达到213时，EHID引起的索引表的内存开销仅为其他两种方法（ED和HAR）的37.80%。图6（b）描述了当我们对FSL轨迹进行实验时的实验结果。图6（b）显示了类似于Linux跟踪的性能。例如，当备份版本达到173时，EHID生成的指纹条目仅占其他两种方法的71.38%。这些实验证实，EHID可以有效地降低索引表的内存开销，避免系统老化导致的索引表臃肿。



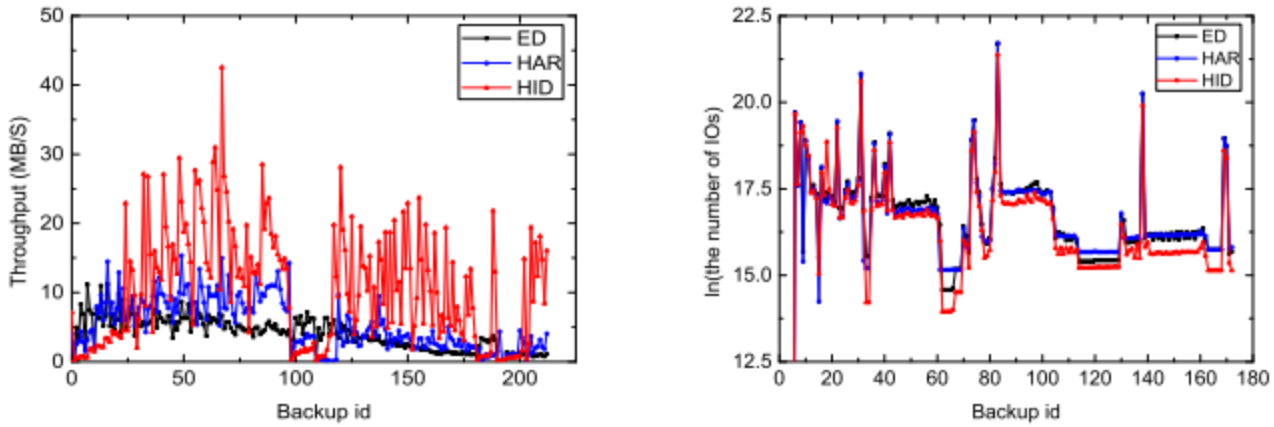
(a) Linux dataset. (b) FSL dataset.

Fig. 6. Memory overhead of indexing table.

图6（a）和图6（b）表明，当使用Linux数据集时，EHID可以比FSL数据集节省更多的内存。这背后有两个原因。第一个是Linux数据集的稀疏容器部分高于FSL数据集。这是因为Linux数据集的很大一部分是自引用的[10]。第二个是，与FSL数据集的173个备份版本相比，Linux数据集有213个备份版本。随着备份版本的增长，EHID可以节省更多内存，这是一个非常重要的功能。此外，对于Linux数据集，我们使用Linux-3.x进行191版本备份前的评估。下一个备份版本192使用Linux-4.x。由于Linux内核进行了重大修订更新，新的备份数据会产生大量唯一的块和相应的指纹条目。这就是我们在图6（a）中观察到尖峰的原因。然而，这两个数字都表明了相同的趋势，即随着备份版本的增加，ED和HAR生成的索引表的内存开销显著增加。相比之下，EHID引起的索引表的内存开销比ED和HAR增长得慢得多。

评估备份性能

我们评估了在两个数据集（即Linux和FSL数据集）上禁用Bloom过滤器时的备份性能。图7（a）描述了使用Linux数据集时ED、HAR和HID的备份吞吐量。结果表明，HID的吞吐量是最高的。HAR和ED分别排名第二和第三。前23个备份版本的HID吞吐量较低的原因是无法缓存HID的稀疏容器会损害提取热指纹条目的效率。

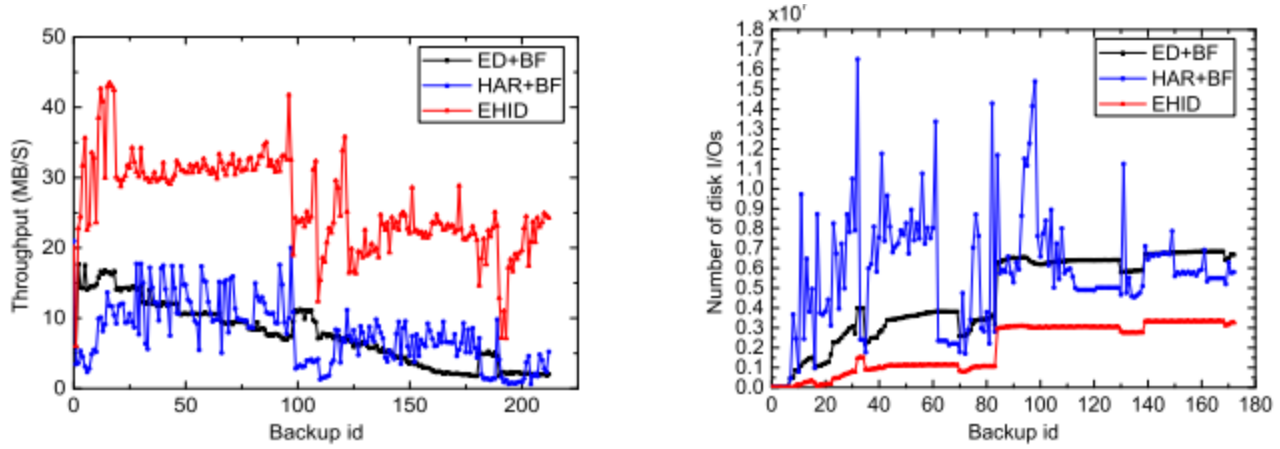


(a) Backup throughput with Linux dataset. (b) Number of disk I/Os with FSL dataset.

Fig. 7. Backup performance of ED, HAR and HID.

由于FSL跟踪不包含数据块的字节流，因此无法像在Linux跟踪上那样评估FSL跟踪上的实际吞吐量。在这部分研究中，我们研究了这些方法的I/O行为。图7（b）显示了ED、HAR和HID在FSL数据集上的备份性能。由于I/O的波动较大，为了使结果更直观和易于理解，我们对I/O的数量（图7（b）中的y轴）执行ln（I=Os）运算，图7（b）显示，与其他两种方法（即ED和HAR）相比，HID生成的I/O数量最少。

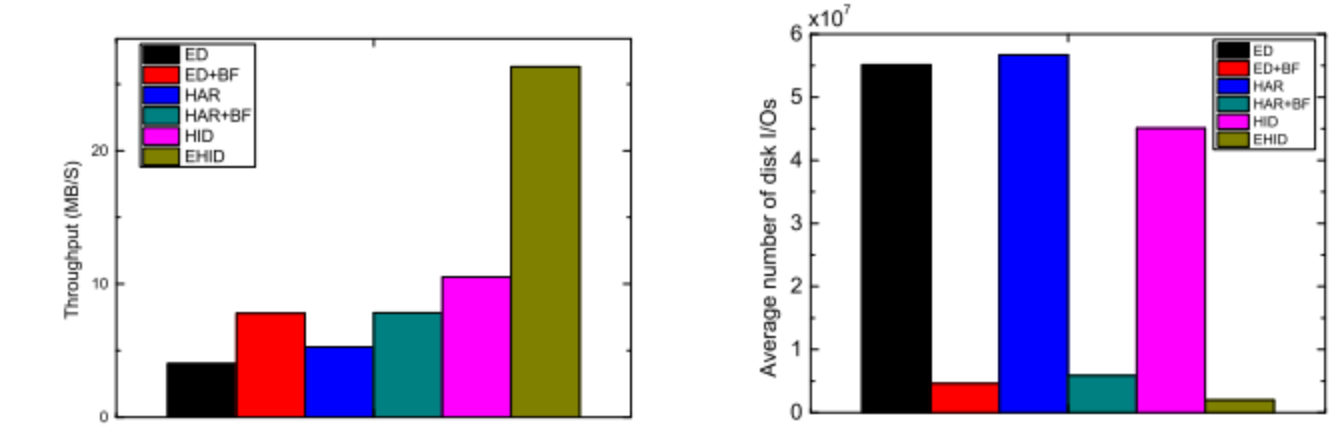
现在，我们可以评估EHID——HID的一个进化版本，它集成了Bloom过滤器，以进一步提高备份性能。为了进行公平比较，我们还为ED和HAR方法配置了Bloom过滤器。图8（a）显示了ED+BF（Bloom filter）、HAR+BF和EHID的吞吐量。图8（a）的结果表明，EHID的吞吐量远高于其他两种方法。有两个因素促成了EHID的高吞吐量。首先，当配置相同的内存大小时，EHID提高了内存的效率，并将更有用的指纹条目从磁盘读取到内存中。其次，由于内存中保存的所有指纹条目都是热指纹条目，因此基于内存的索引查找的命中率显著提高，从而显著避免了访问基于磁盘的索引的查找请求。正如预期的那样，ED+BF的吞吐量随着备份版本的增加而逐渐降低。这是因为数据块的碎片随着备份版本的增加而逐渐增加。因此，必须使用基于磁盘的索引来满足更多的查找请求，这样的影响会导致性能下降。观察到HAR+BF的吞吐量围绕ED+BF的曲线波动是非常有趣的。吞吐量波动背后的原因是HAR必须更新所有碎片块的指纹条目，这涉及到额外的磁盘I/O。当碎片块的数量增加时，由于生成了大量的磁盘I/O，HAR的吞吐量相应降低。



(a) Backup throughput with Linux dataset. (b) Number of disk I/Os with FSL dataset.

Fig. 8. Backup performance of ED+BF, HAR+BF and EHID.

图8（b）表明，当使用FSL数据集时，EHID为索引查找发布的磁盘I/O数量明显低于启用Bloom filter的其他两种方法。原因是EHID显著减轻了发送基于磁盘的索引的查找请求。ED+BF和HAR+BF生成的磁盘I/O说明了FSL数据集上与Linux数据集上类似的行为。请注意，这三种方法都从Bloom filter中受益匪浅，因为Bloom filter有效地避免了访问因识别唯一块而产生的基于磁盘的索引。然而，EHID受益最大。以上六种方法的平均性能如图9所示，这证实了EHID明显优于其他五种方法。



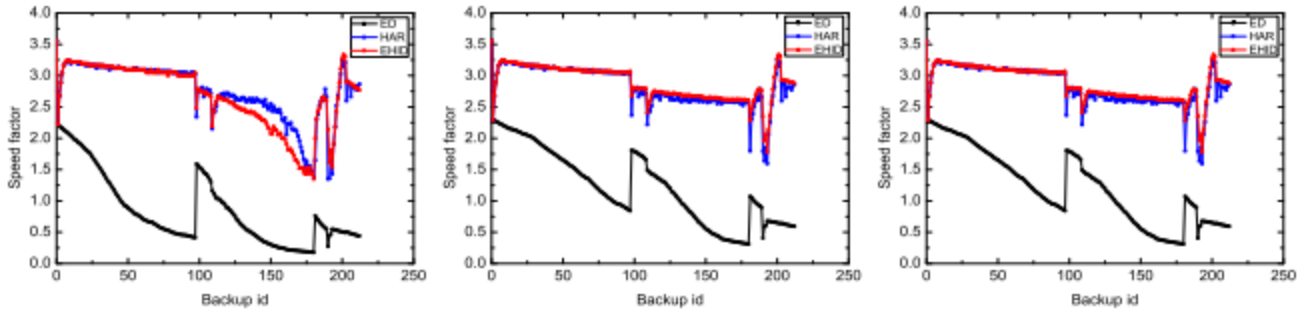
(a) Average backup throughput with Linux dataset. (b) Average disk I/Os with FSL dataset.

Fig. 9. Average backup performance.

评估恢复性能

现在我们关注恢复性能。图10显示了使用Linux数据集时的速度因素。由于HAR使用了最优缓存替换（OPT）算法，为了公平起见，我们实验中的ED、HAR和EHID三种方法都配置了OPT算法。OPT的缓存大小设置为30、60和90，缓存单元是一个容器。从图10中可以看出，ED的速度因子是最低的。随着备份版本的增加，速度系数会降低。原因是ED没有采用重写算法来减轻碎片。HAR和EHID的速度因子非常

相似。更具体地说，当缓存大小为30时，HAR和EHID的平均速度因子分别为2.75和2.68（见图10（a））。当备份版本从115增长到175时，HAR的性能略优于EHID。这是因为在这些备份版本中，HAR比EHID重写更多的碎片块，这导致了更多的存储空间使用。然而，随着高速缓存大小的增加，EHID的速度因子变得比HAR的速度因子更好。例如，当高速缓存大小为60时，HAR和EHID的平均速度因子分别为2.8327和2.8806。当缓存大小为90时，HAR和EHID的平均速度因子分别为2.8328和2.8807。

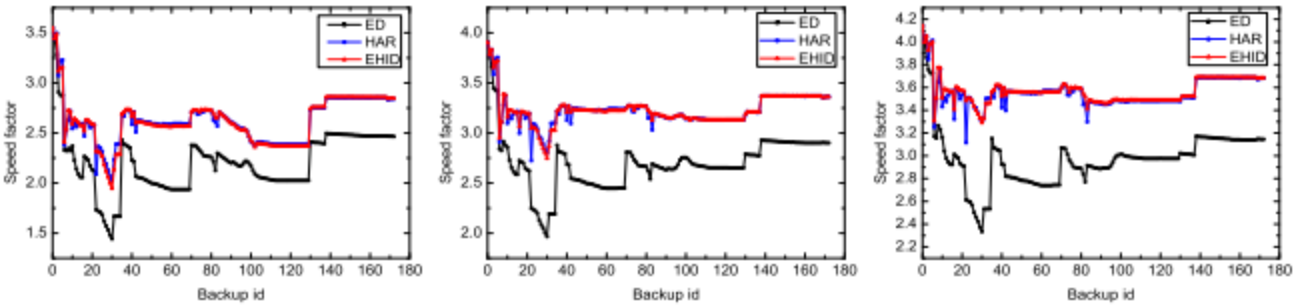


(a) OPT 30. (b) OPT 60. (c) OPT 90.

Fig. 10. Speed factor with Linux dataset.

此外，我们可以看到，在60和90缓存大小的情况下，EHID的性能略优于HAR。这一变化是因为EHID只重写一次分段块，而HAR可以多次重写一个分段块。例如，假设存在指向稀疏容器C0的分段指纹条目I1，则新备份块J1的指纹与分段指纹条目的指纹重复。新的组块J1将被重写到新的容器C1，以减轻数据碎片。由于新容器通常不是稀疏容器，因此系统应及时更新分段指纹条目I1并将其指向容器C1。如果在需要备份另一个新的组块J1之前没有更新该分段指纹条目I1，则系统将把新的组元J1重写到另一新的容器C2。在原始的HAR中，由于多线程和脏缓冲区的同步，偶尔会出现碎片指纹条目无法及时更新的情况。这样做，HAR中重写的块在恢复数据时会浪费存储空间和磁盘带宽。与HAR相反，不需要EHID来更新分段指纹条目，因为所有分段指纹条目都已从索引表中分离出来。因此，EHID不可能多次重写相同的碎片块。

图11描述了使用FSL数据集时的速度因子。实验结果与Linux数据集的结果非常相似。正如预期的那样，图11表明ED的速度因子仍然是最低的，EHID的速度因子与HAR的速度因子非常接近。更具体地说，随着高速缓存大小的增加，EHID在速度因子方面变得优于HAR。例如，首先，当高速缓存大小为30时，HAR和EHID的平均速度因子分别为2.6272和2.6240。接下来，当高速缓存大小增加到60时，HAR和EHID的平均速度因子分别变为3.2273和3.2311。最后，当缓存大小达到90时，HAR和EHID的平均速度因子分别变为3.5595和3.5738。值得注意的是，在紧急恢复场景中，使用更大的缓存大小来恢复数据是非常有价值的。



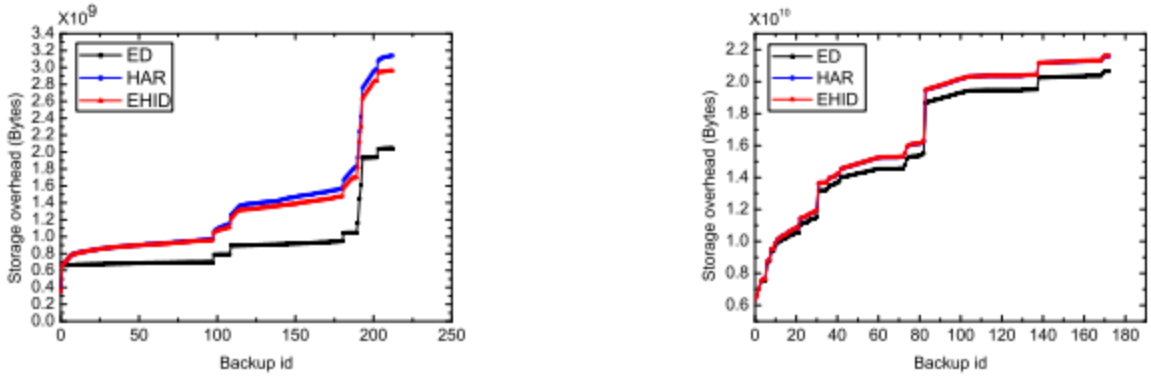
(a) OPT 30. (b) OPT 60. (c) OPT 90.

Fig. 11. Speed factor with FSL dataset.

评估存储开销

我们的实验结果表明，在Linux和FSL数据集上，三种方法（即ED、HAR和EHID）的平均重复数据消除率在96.73%到98.73%之间。由于这三种方法的重复数据消除率非常相似，为了使性能比较更直观、更清晰，我们评估的是存储开销，而不是重复数据消除比率。这里，存储开销是指在基于重复数据消除的备份系统完成前x次（即备份id+1）备份之前累积的存储块的总大小。

图12（a）显示了这三种方法与Linux数据集之间存储开销的比较。从图12（a）可以看出，ED使用的磁盘空间是最小的，这是因为ED不考虑恢复性能，只存储唯一的块。HAR的存储开销略高于EHID，这是因为HAR存在过重写现象。这种现象意味着碎片块在备份中被重写两次或两次以上。这种过度重写操作降低了容器的利用率，并生成了更稀疏的容器。由于稀疏容器数量的增加，重写算法重写了更多的碎片块并浪费了存储空间。图12（a）的实验结果表明，EHID的重写率平均比Linux数据集的HAR低2.83%。这意味着，当HAR重写10GB的数据块时，EHID可以节省约290MB的存储空间。



(a) Linux dataset. (b) FSL dataset.

Fig. 12. Storage overhead evaluation.

图12（b）显示了数据集为FSL时ED、HAR和EHID的存储开销。同样，ED的存储开销也是最小的。由于HAR在FSL数据集情况下的过重写影响不如Linux数据集情况那么强，因此HAR和EHID的存储开销非

常接近。例如，在第173版本的备份中，HAR的存储开销达到21590404782字节，而EHID的存储开销则达到21617550154字节。EHID的存储开销仅比HAR高0.13%，可以忽略不计。

结论

在这项研究中，我们发现传统基于重复数据消除的备份系统索引中的指纹条目可以分为三类，即无用指纹条目、碎片指纹条目和热指纹条目。无用的指纹条目和碎片化的指纹条目会浪费内存空间并增加磁盘I/O。此外，分离碎片指纹条目使基于重复数据消除的备份系统能够直接重写碎片块，从而减少数据碎片。为了解决由冷指纹条目（即无用和碎片化的指纹条目）引起的性能瓶颈，我们首先提出了HID，该HID分离冷指纹条目，并仅在内存中保留热指纹条目。与最先进的方法HAR相比，HID提高了内存利用率和备份性能。由于减少了数据碎片，HID还提高了恢复性能。

我们引入了FTU特性来弥补Bloom滤波器的不足。为此，我们提出了一种进化的HID策略EHID。EHID嵌入了仅映射热指纹的Bloom过滤器，从而降低了误报率。因此，EHID表现出两个显著的优点：EHID在识别唯一块时避免了基于磁盘的索引访问。EHID降低了集成Bloom过滤器的误报率，从而进一步减少了磁盘访问。这些显著特征使EHID保持在高效模式。此外，EHID可以有效地提高基于重复数据消除的备份系统的吞吐量。原因有两方面。首先，EHID只在内存中存储热指纹条目，这可以提高指纹查找的命中率。其次，EHID显著减少了基于磁盘的索引查找所产生的磁盘I/O数量。更具体地说，为了加快指纹查找，EDID将索引表分离，并在备份完成后将其存储为两个文件，即热索引文件和冷索引文件。如果发生内存丢失，EHID将在基于磁盘的索引中查找丢失的指纹条目。由于EHID中基于磁盘的索引只包含热索引文件，而热索引文件是整个索引表的一小部分，因此EHID的基于磁盘索引的大小远小于传统方法。因此，就在基于磁盘的索引中查找指纹条目而言，EHID比传统方法快得多。

我们进行了大量的实验来定量评估EHID的性能。实验结果和分析表明，EHID可以显著提高基于重复数据消除的备份系统的性能。

未来工作的可能方向包括将EHID和近乎精确的重复数据消除相结合，以进一步探索EHID的性能行为。