海量存储技术论文

学　　号：2017282110253

姓　　名：张瑞

专　　业：计算机技术

任课教师：何水兵

海量存储中重复数据的删除

摘要：业界和学术界的数据生成和处理价值链的巨大增长已经大大增加了备份环境的压力。数据本身已经成为一项宝贵的资产，因此使用高度可靠的备份系统来保护数据非常重要。大多数备份系统都是基于磁带环境，直到 21 世纪后期，磁带仍然是存储数据的最具成本效益的方式之一。基于磁盘的备份环境从重复数据删除技术中获益，这往往会显着减少存储的数据量。这些技术使磁盘成为经济适用的备份技术。

本文着重讲述了利用惰性的确切重复数据删除和以高并发为目的从同时处理数千个备份数据流角度出发提出了分类的重复数据删除方法。“懒惰”的重复数据删除方法，可以缓存传入的指纹，并批量执行磁盘查找，以减少磁盘瓶颈；而基于分类的重复数据删除则提供了一种新的精确重复数据删除方法，设计用于在同一指纹索引上同时处理数千个备份数据流。该基本的方法破坏了传统上被利用的时间块局部性，并通过对指纹进行排序来创建新的块。排序导致备份服务器上完美顺序的磁盘访问模式，而只是稍微增加了客户端的负载。

关键字：海量存储、重复数据删除

一、懒惰的确切重复数据删除方法

一个懒惰的重复数据删除方法，它将内存中的指纹缓存到散列桶中。当哈希桶中的指纹数量达到用户定义的阈值时 T 系统读取磁盘并一起搜索这些指纹。重要的是，懒惰的方法执行单个磁盘查找 T 指纹。这减少了磁盘指纹查找的磁盘访问时间。尽管本文提出的缓存查找策略在备份流方面效果较好，但懒惰方法适用于主工作负载和备份工作负载。

1.1 技术背景

重复数据删除技术用于减少存储需求和网络传输，并且在计算和 I/O 密集型方面都是如此。由于分块，指纹计算，压缩等，计算量很大。

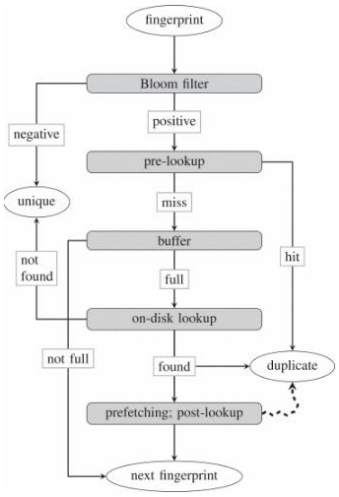
对于大规模重复数据删除系统，由于主存不够大，无法容纳所有的指纹，大部分指纹都存储在磁盘上。因此，重复数据消除也是磁盘 I/O 密集型的，导致磁盘瓶颈，这会严重影响吞吐量。磁盘瓶颈随着数据大小（以及指纹数量）的增长而增加，而计算时间通常保持稳定。因此，以前的大多数工作都集中在消除重复数据删除中的磁盘瓶颈上。虽然在一些严格的重复数据删除系统中磁盘瓶颈可以减少 99%，但它仍然是一个瓶颈。我们的目标是通过将多个指纹查找组合到单个磁盘访问中来进一步减少这个组件。

重复数据删除系统利用数据流中的局部属性，通过使用内存中缓存来减少磁盘访问。在磁盘上找到指纹时，将调用预取，从而将存储在磁盘上的相邻指纹传送到缓存。由于指纹经常以与它们先前到达相同的顺序到达（因此与它们存储在磁盘上的顺序相同），所以这种预取策略导致高的缓存命中率，这显着减少了磁盘存取时间。

1.2 懒惰的重复数据删除

我们可以绕过缓存和缓冲，并立即写入磁盘。通过布隆过滤器的指纹首先在缓存中查找，我们将其称为预查找，并且缓存不在缓存中的指纹。查找指纹作为磁盘查找的结果触发预取，之后在缓存中查找指纹中的一些指纹，称为查找后。

预先查找利用在指纹流内紧密接近发生的重复指纹，而后查找则利用整个指纹流中的指纹循环模式。



A. 指纹管理

延迟重复数据删除旨在通过推迟和合并磁盘上的指纹查找来减少磁盘访问时间。需要在磁盘上查找的指纹最初存储在内存中的散列表（缓冲区）中。它们被存储直到哈希桶中的指纹数量达到阈值，我们称之为缓冲器指纹阈值（BFT）。当达到阈值时，在磁盘上搜索哈希桶内的所有指纹。图 2 说明了懒惰方法的基本思想。系统使用指纹桶功能，在同一个桶 ID 中的桶内桶内查找缓存桶内指纹，逐桶进行。找不到的指纹是唯一的，这是 Bloom 过滤器的“误报”。

指纹以两种方式存储在磁盘上：

唯一的指纹存储在磁盘哈希表中，用于方便查找。磁盘哈希表和缓冲区使用相同的哈希函数。对于磁盘哈希表，我们使用单独的链接来解决桶溢出问题。

唯一指纹和重复指纹都存储在日志结构化的元数据数组中。它们按照它们到达的顺序进行存储，从而保留了局部性。磁盘哈希表中的指纹指向相应的元数据条目，并且当在磁盘哈希表中找到一个指纹时，相邻的元数据条目被预取到高速缓存中。

这种方法很容易适用于像 ChunkStash 或 BloomStore 这样的系统，因为它们也使用散列表来组织指纹。具体而言，我们可以类似地在由哈希值限制的搜索空间内批量执行磁盘上的搜索。

为了缓冲指纹，延迟方法需要额外的存储空间，并且由于我们不能先验地知道哪些块是重复的，所以数据块也需要被缓冲。假设一个哈希表 n 桶被用来缓冲指纹，每个指纹的大小是 Sfp BFT 被设置为 T 和平均块大小是 Schk。然后缓冲指纹所需的内存是 Spacefp:=nSfpT 并且相应的块所占用的平均内存是 Spacechk:=nSchkT。所以所需的额外空间是由

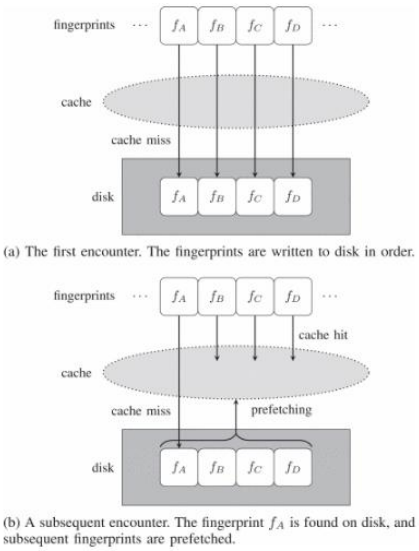
Space:=Spacefp+Spacechk=(Sfp+Schk)nT.

B.缓存策略

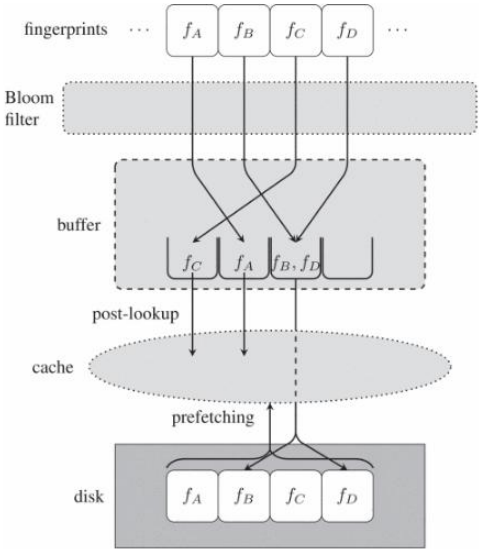
指纹缓存已被证明是重复数据删除系统中的重要因素。备份数据流中的重复模式已经被用来设计有效的缓存策略来最小化磁盘访问。

对于渴望的方法，图 3 说明了如何在缓存中利用本地。数据块通常以与之前相似的顺序到达，所以当在磁盘上找到指纹时，随后的磁盘指纹被预取到高速缓存中。当后续传入的指纹到达时，通常在这些预取指纹中找到它们，导致缓存命中。

在懒惰的方法中，指纹将被缓冲，所以我们不能使用与急切重复数据删除相同的缓存策略。图 4 修改了图 3b，显示了在重复数据删除中使用的缓存方法。指纹在处理数据流时被缓冲，直到其对应的散列桶已满，才会在磁盘上查找指纹。随后的传入指纹将在预取发生之前到达，因此也将被缓存。



图三



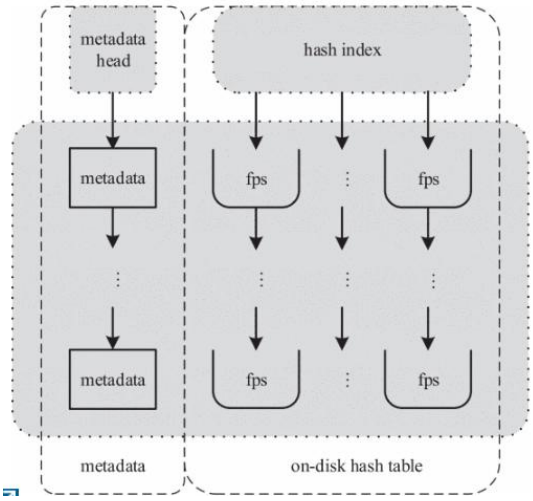
图四

通常会有许多缓冲区周期（其中之一是“当前”缓冲区周期，指纹被添加到缓冲区周围），并且缓冲区中的每个指纹通常属于唯一的缓冲区周期。当一个哈希桶变满时，其中的指纹将在磁盘上被搜索。在磁盘上找不到的指纹是唯一的，并写入磁盘。在磁盘上找到的指纹是重复的，找到时会触发预取。在预取之后，在高速缓存中搜索同一循环中的指纹（即，后查找）。我们使用最近最少使用（LRU）驱逐策略来更新缓存; 新增加的指纹停留时间更长，以方便流动的预查。

一些指纹将在缓存中被搜索几次。这发生在独特的指纹（通过布隆过滤器），指纹没有指纹在相同的缓冲区周期。为了减轻这一点，我们将缓冲区中每个指纹的缓存查找次数限制为 10 次，之后系统从其缓冲区中删除指纹。对于缓冲区周期以外的指纹，在磁盘上发现预取不会被触发，从而避免预取无指纹的磁盘 I / O。

C.磁盘管理

三个主要的磁盘组件是：数据，元数据和哈希表。元数据和磁盘哈希表的磁盘布局如下图所示。



1.3 原型系统设计及性能评估

1）数据

数据以日志结构布局存储，分为固定最大大小的数据段。如果发现传入的块是唯一的（即，以前没有看到过），则将其添加到“当前”数据段中，当数据段满时，将其添加到磁盘中。数据段是链接的，即每个数据段都有一个指向下一个数据段的指针。

2）元数据

指向数据块的指针（8 字节偏移量，4 字节长度）与指纹一起以元数据的形式存储在日志结构的布局中，分成固定最大大小的元数据段（为简单起见，我们选择相同的最大大小至于数据）。对于每个传入的指纹，无论是否唯一，都将条目存储到数据中相应块的位置。我们使用元数据来跟踪组块到达的时间顺序。元数据存储关于文件由哪个块组成的信息；他们很小，有重复的元数据条目。

3）磁盘哈希表

当缓冲区中的其中一个散列桶已满时，其中的指纹将在磁盘上查找为批处理。磁盘上的指纹存储在磁盘上的哈希表中。指纹与指向相应元数据条目的指针一起存储。磁盘上的哈希桶被链接在一起，以方便在磁盘上查找指纹。对于每个唯一的块，在插入其元数据条目之后，将一个哈希表条目添加到相应的存储桶。一个条目包括指纹，一个 8 字节的指针来显示元数据条目的位置，和一对（8 字节的偏移量，4 字节的长度）给出块信息。桶中的条目将被逐个存储，直到桶已满。

我们使用多个 CPU 线程和一个 GPU 来加速分块和指纹。系统创建 16 个分块线程，每个线程用于处理 64MB 块中的数据。在指纹计算中，系统将 4096 个批次的批次传送给 GPU。GPU 为批次中的每个数据块分配一个线程以计算其指纹。然后系统将一批指纹从 GPU 传送到主存储器。

实验结果表明，这种方法可以显着减少磁盘指纹查找的时间，在固态硬盘上高达 70%，在硬盘上高达 85%。

二、同时处理数千个备份数据流的分类重复数据删除方法

2.1 技术背景

大多数重复数据删除系统将备份数据集分成小块，并通过比较块的指纹识别数据中的冗余。只有在指纹尚未包含在所谓的块索引中时才存储新的数据块。对于以后的数据恢复，系统会创建包含重建存储数据所需的全部信息的配方。

在块索引查找 UPS 一直在重复数据删除系统的主要性能瓶颈完整的指标通常是太大，无法在主存储和索引举行访问生成随机 I/O。因此，重复数据删除系统设计人员通常至少使用以下技术之一：

他们减少索引大小，以便它适合主内存和交易这减少重复检测率。

他们利用块局部性，即备份数据流中块的趋势再次发生，并从捕获该局部的内部数据结构中分块预取块指纹。对于单个备份流，预取可以产生近乎顺序的磁盘访问。

但是，即使每个备份流都具有本地属性，当并行处理更多备份时，磁盘友好访问方案也会随机降级。另外，这些流竞争可用的内存空间，因为每个额外的流减少了流本地高速缓存大小，使得 I/O 访问的数量增加。结果是写入性能下降，特别是当系统负载很高时。由于用户在工作日期间更喜欢默认的调度窗口 ，所以公司自然会产生较高的系统负载，导致夜间活动爆发。在线备份的重复数据消除系统面临同样的挑战。在这里，用户可以在一天中或多或少均衡地备份他们的数据，但是用户的数量可以容易地达到 10-100K，并在高峰时间产生 1-10K 的数据流。因此，需要系统能够有效地处理数千个流而不需要繁重的硬件要求。

在这项工作中，我们提出了一个重复数据删除方法，可以处理许多流，并避免内存问题。它通过按排序顺序处理所有流来实现这一目标，以便所有流同时访问相同的索引区域。这创造并利用了高指数地区。这种方法的 I/O 数量仅取决于全局唯一数据量，但几乎与并发数据流的数量无关。

2.2 排序块索引

所有重复数据删除系统都试图避免大块查找磁盘瓶颈。这个系统的主要方法各不相同，但是大多数系统都试图利用块区域。本地用于预取和缓存块识别信息以保存 I/O 操作。

但是，这个地方是流本地的，一个处理成千上万个流的系统应该利用一个全球的地方。我们通过维护一个已排序的块索引并按照排序顺序处理每个流来协调创建。这创建了一个完美的块索引位置，减少了整个磁盘 I/O，并创建了顺序磁盘访问模式。

在下文中，我们将详细描述这个排序的重复数据删除和一个样本系统，排序块索引（Sorted Chunk Indexing，SCI）。

A.排序重复数据删除

传统重复数据删除系统在预取期间可能会由于以下三个原因之一而生成非顺序磁盘访问模式：

•块共享

一些块在同一个备份流中出现多次[6]。这导致块和其邻居的指纹的重复预取。另外，块共享也可以发生在不同的流之间。

•独立

即使流之间的块共享较低，系统也会生成随机 I / O，因为并行处理所有流，并且所有流本地预取都会生成全局随机访问。

•老化

备份会随着时间而改变。对于某些系统，这会减少内部数据结构的局部性，并可能增加预取的数量。

排序的重复数据删除可以消除随机访问，如下所示：对于每个已排序的流，系统遍历已排序的块索引，并在流中快速标识流中的块。这消除了由流本地块共享导致的随机 I / O，因为根据定义，排序的顺序强制没有块被重新访问。另外，由于备份流的改变改变了组块的集合，但不改变其外观顺序，所以没有随机的 I / O。

但是，备份流的独立处理并不能消除流间共享和独立造成的随机 I / O。为此，系统还必须对每个流执行相同的处理速度，使得每个流同时访问相同的块索引位置。这保证了相同的 I / O 访问可以服务于任意多个流。由此产生的 I / O 模式成为一个完全连续的运行。

B. 系统

作为概念验证和测试排序的重复数据删除方法，我们开发了一个称为 Sorted Chunk Indexing（Sorted Chunk Indexing）的原型实现。在 SCI 中，客户端执行分块和指纹识别，而服务器识别所有的块。

1）服务器

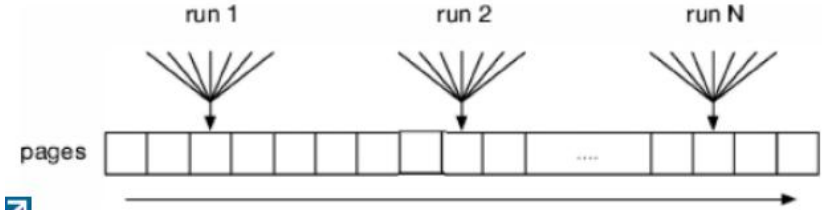
服务器负责识别备份流中的新块。为了简单起见，我们首先假设来自客户端的所有流同时开始。稍后，我们放松一下，展示如何自发地处理数据流。

服务器按排序顺序接收块指纹。它通过在两级日志结构合并树（LSM 树）[8]中组织块索引来利用顺序。LSM 树按顺序在磁盘上的叶节点中按顺序存储数据。

这允许在树的叶子上以单个顺序运行同时处理所有的流。对于每个加载的叶子，服务器在进入下一个叶子之前处理落入该叶子所覆盖的区间的所有指纹。这确保了服务器端上所有流的处理速度相同。由于 LSM 树只在将基于内存的第一级别合并到第二级别时才修改叶子，因此每个叶子是只读的，并且可以由每个流并行访问。为了进一步减少 I / O 访问，叶子被分组成相同大小的页面。因此，页数构成备份运行期间生成的 I / O 数量的上限。当 LSM 树的基于内存的第一级结构的大小达到预定义阈值时，将添加新页面[8]。当服务器处理一个页面时，它将预取下一个页面，因为它很可能会用于足够的备份数据量。请注意，指纹击中页面的概率与页面的指纹范围成正比，因此这些概率不均匀。在 III-B 部分，我们进一步调查这个概率。

并发数据流

新客户不会被迫等待索引运行完成。相反，他们可以利用他们可以在排序列表中的任何点开始传输指纹，因此，钩在当前的索引运行。为此，每个客户机首先向服务器请求当前正在处理的页面的最后一个指纹，并用下一个更大的指纹开始传输。另外，系统允许客户从不同位置开始发送多个流，以克服不同客户的处理速度。这将在页面上引入多个独立的运行，如图 1 所示。但是，这些运行的次数必须受到限制，因为太多运行的磁盘访问模式变得随机。



2）客户端

与 DDBoost 类似[9]，客户端首先执行备份数据的分块和指纹。每个客户端使用相同的分块和散列函数方法来使服务器能够检测流间冗余。接下来，它对指纹进行排序，并与服务器进行通信以检测新的指纹，如图 2 所示。为此，它首先发送排序的指纹列表。在发送过程中，它将跳过同一备份中冗余的块。这些块可以按排序顺序直接检测，因为指纹的所有副本

都排列在列表中的一个块中。服务器回复一个包含存储位置（容器）的列表 ID），或者如果块是新的，则为零值。存储位置是强制性的，因为客户端还负责创建用于恢复备份数据的配方信息。服务器无法执行此任务，因为它没有关于块的原始顺序的信息。最后，客户端将配方和新块的原始数据发送给服务器，服务器随后将新块的存储位置添加到配方中。原始块以原始顺序发送以保留恢复情况的块局部性。在恢复过程中，排序的顺序将创建一个随机访问容器。请注意，服务器只使用相同流的块填充容器，与其他系统类似。

客户端操作的开销如下：分块和指纹需要读取所有备份数据。排序是一个标准的排序算法要求[数学处理错误]，在哪里 [数学处理错误]是任何重复数据删除之前备份数据中的块数。建立服务器的消息可以在[数学处理错误]。网络 I / O 操作的数量取决于客户端必须发送的消息数量。这个数量又取决于每个客户端可用的服务器缓冲空间。我们假设缓冲区可以保持[数学处理错误]指纹并且服务器在处理前一个时收到一条消息。因此，客户端发送和接收[数学处理错误]消息。下一步是配方的更新。这需要[数学处理错误]操作，如果客户端另外维护一个存储每个块指针的索引到它在配方中的出现。另外，客户端执行[数学处理错误]复制操作，其中 n new 是新块的数量。

C.流间冗余

在同一个索引运行期间，可能发生两个或多个客户端发送相同的块指纹。这种情况需要特殊处理，因为系统必须确保容错性，即只要存在至少一个非崩溃客户端，就必须存储块。因此，如果块索引中没有条目，则服务器始终向每个客户端指示该块是新的（图 2 中的第二条消息）。这样，每个客户端都可以发送数据块，并确保数据块到达服务器，只要至少有一个非崩溃客户端即可。

服务器可以简单地多次存储相应的块，即使这种罕见的事件放宽确切的重复数据删除约束。不过，如果服务器保持内存中索引来管理正在运行的块，则可以保持确切的重复数据删除。当新块到达时，它会添加一个侧面索引条目来锁定它，存储块原始数据，并将块索引，边索引和配方更新到存储位置。如果这个块数据第二次到达，那么服务器可以丢弃该块并更新配方。

每个边索引条目还包含一个计数器和一个时间戳。计数器设置为块的剩余进行中版本的数量，而时间戳表示超时。在通常的情况下，没有客户端崩溃，并且服务器在接收到最后一个传入块时删除条目。如果客户端崩溃，服务器将在定期清除边索引期间基于时间戳删除条目。

D.恢复速度改进

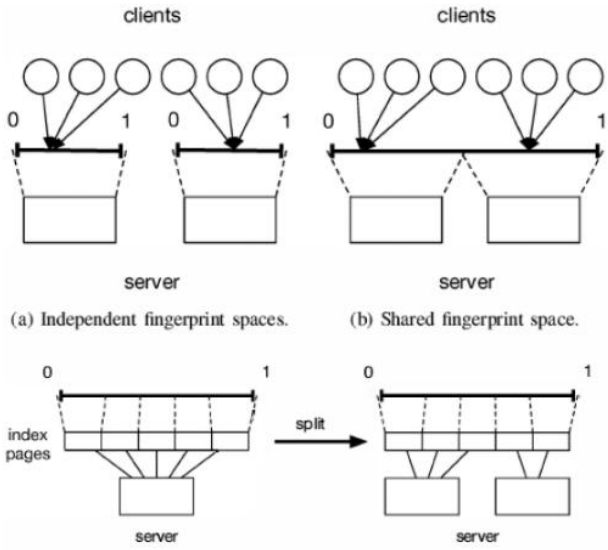
本文提出的 SCI 方法与通过恢复以前的工作介绍了速度的提高兼容[10] ，[11] 。大多数方法通过修复块碎片来提高恢复速度，即，保持原始块信息（容器）的数据结构的捕获位置随时间偏离备份流的实际位置。在处理备份流时，它们允许系统使用数据流中当前位置出现的旧块来填充新的容器。

这种技术需要两个属性：首先，系统必须使用相同备份流的块填充容器。其次，系统必须知道备份流的历史记录，即根据其原始顺序的相应块的位置排序的容器 ID 的列表。每个容器 ID 可以在列表中多次出现，因为每个相应的块可以在流中多次出现。在我们的系统中，前者的财产如上所述保证。后者也是有保证的，因为客户以原始顺序发送配方，并且每个配方必须按其原始顺序保存其指纹列表以允许系统恢复所描述的数据。此外，配方会保存每个指纹的容器标识，以便在还原情况下保存块索引查找。

E.分布式模式

该系统旨在通过一台服务器处理数千个数据流。但是，单服务器设置的可伸缩性受到限制。为了将工作负载分配给多个服务器，可以采用两种方法，如图 5 所示。第一种方法将客户端集合划分为不相交的子集，并在不同服务器上为每个子集维护一个块索引。客户端到服

务器/索引的映射是微不足道的，只能确保将相同的客户端映射到每个备份的相同服务器。



这种方法的缺点是系统执行近似的重复数据删除，也就是说，可能发生一个块被多次存储。当两个客户端处于不相交的集合并将相同的块指纹发送到不同的服务器时，会发生这种情况。在这种情况下，两个索引都会声明这个块是新的，因为这些服务器是独立工作的。近似重复数据删除的影响取决于数据：例如，在我们的测试数据集中，少于 1％的所有数据块在两个或多个数据流之间共享。

另一个缺点是重载服务器的分离是复杂的。在这种情况下，系统将一组服务器的客户端分成两个子集[数学处理错误] 和 [数学处理错误]并将索引条目移动到新的服务器。为了分割块索引，系统执行完整的索引运行并检查每个条目块是否被任何客户端存　　储在中[数学处理错误]。这反过来需要一个新的反向索引来保存指纹→具有块映射的客户机，这需要额外的资源。

第二种方法如图 5b 所示，其中系统划分指纹空间而不是划分客户端。在这里，每个服务器都被分配一个[0,1）间隔的子范围，并且只保留在其子范围内的块索引条目。每个服务器在其自己的子范围上执行自己的索引运行。与第一种方法相比，该系统具有执行准确重复数据删除的优点，并且重载服务器的拆分更简单。例如，如果服务器将其范围[0,0.5]分割成子范围[0,0.25）和[0.25,0.5]，则只需传输属于新服务器范围的块索引页面。这是一个更简单，更直接的方法，因为每个索引页面都覆盖了[0,1）区间内的连续范围。如果系统在索引页边界拆分旧的范围，则只需要传输完整的索引页面而不拆分任何现有的索引页面。然后可以直接在新服务器中使用每个传输的页面而无需修改。

然而，共享空间方法使客户端处理复杂化。系统现在可以在两个选项之间进行选择：客户端可以像以前一样发送指纹，但在每个索引边界停止，直到下一个范围启动新的索引运行;或者将他们的指纹按照服务器的分割进行排序并将它们并行发送到子范围。哪个选项更好取决于客户端和服务器之间的互连：具有较慢互连的客户端应该使用第一种技术，而具有快速互连的客户端则可以使用第二种客户端来利用服务器端的并行处理。

F.食谱和容器 ID

SCI 将容器 ID 保存为配方的一部分。这样可以在还原过程中节省对块索引的随机访问，因为配方包含足够的信息来加载给定的块。但是，这些节省的交易复杂度更高：备份将随着时间的推移而被删除，旧的容器开始存储较少的活动块。重复数据删除系统因此应合并容器以节省存储容量。

新的容器保存所有旧的，但活动的块，也有一个新的 ID。如果没有额外的机制，则合

并会使多个配方无效，因为几个区块的容器 ID 已更改。因此，必须找到并更新所有受影响的食谱，这是一项昂贵的操作。为了减轻这种影响，SCI 可以为容器 ID 维护一个间接层，并维护一个容器索引，将每个容器 ID 映射到磁盘上的位置。在合并期间，SCI 然后用新位置更新旧容器的条目。新索引足够小，可以保存在内存中，在最简单的情况下，一个条目由两个 8 B 条目组成。因此，1 GB 的内存可以容纳大约 2.68 亿个条目。

或者，配方不能包含容器 ID，以便在还原期间必须查询块索引。请注意，这种情况对通信模式只有轻微的影响。图 2 中的第二个通信步骤仍然是强制性的，因为它告诉客户端下一步要发送的原始块数据。服务器只能发送一个位图，而不是容器 ID，因此减少了消息的容量。

G.限制

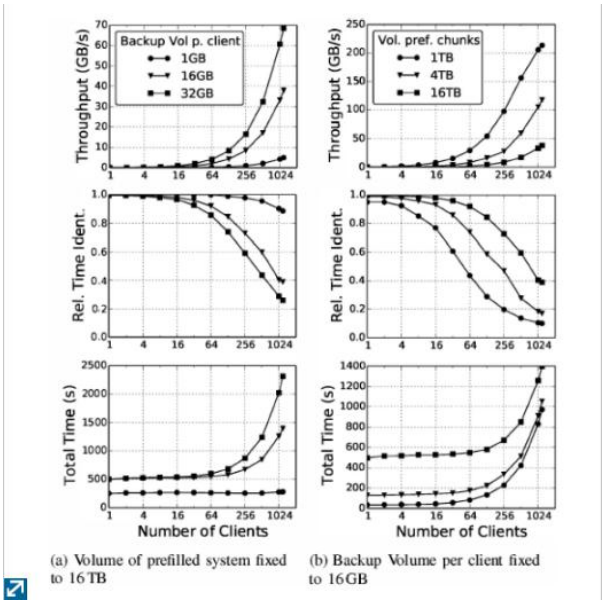
所描述的系统被设计为同时处理多个流。但是，由于索引运行会导致相当大的写入延迟，因此不适合每种情况。SHA1 创建了一个几乎统一的指纹分布，所以即使只涉及很少数据块的小写操作也需要在整个指纹范围内的页面查找。一个极端的例子是，例如，一个备份只包含两个指纹映射到第一个和最后一个索引页面的块。在第一次查找之后，各个客户端将不能直接跳到最后一页，因为其他客户端很可能在其间有块，并强制这个客户端等待。

2.3 实验与评估

我们的系统评估侧重于 SC I 的服务器端，并将 SCI 与 DDFS 和稀疏索引进行比较。我们将首先描述评估方法和使用的数据集，然后我们将为 LSM 树确定合适的页面大小。之后，使用这个页面大小，我们将调查可以并行处理多少个数据流，以及在哪个备份大小下，SCI 的性能优于 DDFS 和 Sparse Indexing。最后，我们将比较三种方法的 I / O 访问模式以及与内存使用相关的性能。

已经实现了一个服务器原型，并对不同数量的客户端进行了评估。客户端通过随机生成块指纹来模拟相同大小的备份。服务器收到指纹，返回存储位置，最后接收配方和新块的原始数据。但是，由于硬件限制，原始数据不会被存储。

服务器运行在配备有 4 个 3.3 GHz（8 个硬件线程），16 GB RAM，10 GBit 以太网网络和一个硬盘的 Intel Xeon CPU 的节点上。对于所有实验，我们生成随机输入数据，同时假设平均块大小为 8 KB，每个客户端的重复数据删除率为 90％，而通过网络发送的块数据的压缩率为 50％。每次运行使用随机预先生成的块索引来模拟老化的系统。我们不使用我们的真实世界的数据集，因为它们只包含多达 597 个客户端。



在该部分中，我们描述了一个用于多个备份流并行处理的新型高性能重复数据删除系统。它对所有流的块进行排序，并协调对指纹索引的访问。这种方法的主要好处是：

它强制所有并发的指纹识别过程在驻留在内存中时使用索引的相同部分。

它确保所有并发的块识别过程只访问一次索引中的相同区域。结合第一个属性，如果传入数据流的数量很大，这是非常有用的，因为它将每个索引“区域”的 I / O 数量减少到 1，而不受数据流数量的影响。

它通过按排序指纹顺序遍历块索引来确保磁盘友好访问模式。

作为概念的证明，我们已经实现了服务器的原型。它已经表明它可以以高达 222GB / s 的速率识别块指纹，即它将每秒 222GB 的备份数据块识别为旧的或新的。此外，我们还将排序块索引（Sorted Chunk Indexing，SCI）与 Zhu 等人的 Data Domain 重复数据删除文件系统进行了比较。和稀疏索引（Sparse Indexing），Lillibridge 等人的近似重复数据删除方法。实验显示，SCI 产生的 I / O 比 DDFS 低 113 倍，比 SI 少 12 倍，同时消耗更少的内存，产生磁盘友好的访问模式，仍然执行准确的重复数据删除。

三、对比总结

惰性的确切重复数据删除是通过将多个指纹查找组合到单个磁盘访问中来进一步减少这个组件，同时重复数据删除系统利用数据流中的局部属性，通过使用内存中缓存来减少磁盘访问。在磁盘上找到指纹时，将调用预取，从而将存储在磁盘上的相邻指纹传送到缓存。以此来避免重复数据删除中磁盘的瓶颈问题和极大的提高了磁盘指纹的访问速度。但是该方法是针对单个备份流读取中重复数据删掉的情景。

而同时处理数千个备份数据流的分类重复数据删除方法则主要是针对在多个甚至数千个备份流在并行处理时实现重复数据的删除，并避免内存问题。它通过按排序顺序处理所有流来实现这一目标，以便所有流同时访问相同的索引区域。最终实验也证明了该种方法能达到极高的指纹访问速度，即它将每秒 222GB 的备份数据块识别为旧或新的数据。

因此在海量存储中，在针对单备份流处理时可以采用惰性的确切重复数据删除，同时在高并发的多备份流处理时可以采用第 2 章中分类重复数据删除，这样综合后整体的效率将会更加明显。