**海量存储技术课程**

**论文阅读报告**

**Building a High-performance Deduplication System**

院（系）名 称：计算机学院

专 业 名 称 ：计算机技术

学 生 学 号 ：2017282110237

学 生 姓 名 ：陈 鹏

# 一 论文基本信息

**Title:**

*Building a High-performance Deduplication System*

**Author:**

*Fanglu Guo Petros Efstathopoulos*  
*Symantec Research Labs  
Symantec Corporation, Culver City, CA, USA*

**Source of publication:**

[*Usenix Conference on Usenix Technical Conference*](http://xueshu.baidu.com/usercenter/data/journal?cmd=jump&wd=confuri%3A%289cebf438a5ab5f8f%29%20Usenix%20Conference%20on%20Usenix%20Technical%20Conference&tn=SE_baiduxueshu_c1gjeupa&ie=utf-8&sc_f_para=sc_hilight%3Dpublish&sort=sc_cited)*, 2011 :25-25*

# 二 论文要解决的问题

## 2.1现代重复删除系统存在的问题

现代重复数据删除技术在消除数据重复方面非常有效，从而使基于磁盘的备份系统的有效容量倍增，并使之成为实际中的磁带替代品。尽管有这些改进，但是单节点原始容量仍然大部分限制在几十或几百TB，迫使用户使用复杂而昂贵的多节点系统，这种系统通常只允许它们扩展到个位数PB。 随着重复数据删除效率优化的机会变得稀少。

## 2.2本文提出的重复删除系统原型

本文展示了高性能重复数据删除原型，通过充分利用节点资源来实现整体单节点性能优化，它还实现了三个重要目标：扩展到大容量，提供良好的重复数据删除效率和接近原始磁盘的吞吐量。本文并未试图改进重复检测算法，而是专注于系统设计方面，并引入新颖的机制。尤其是，本文通过引入渐进采样索引和分组标记-扫描来提高单节点可扩展性，并且还利用事件驱动的多线程客户端 - 服务器交互模型来优化吞吐量。本文的原型实现能够扩展到数十亿的存储对象，吞吐量高，重复数据删除效率基本没有降低。

## 2.3重复数据删除系统的有效性度量

数据公司数量的爆炸式增长需要存储，但是对存储和备份系统施加了很大的压力，迫切需要解决容量，性能和成本难题的新方法，并且通常会提高整体效率。重复数据删除系统的有效性取决于其达到三个相互竞争目标的程度：重复数据删除效率，可扩展性和吞吐量。

重复数据删除效率指的是系统可以检测和共享重复的数据单元，这是其主要的压缩目标。可扩展性是指以一致的性能支持大量原始存储的能力。吞吐量是指数据传入和传出系统的速率，构成主要的性能指标。

所有三个指标都很重要。好的重复数据删除效率降低了存储成本。良好的可扩展性通过减少节点总数来降低整体成本，因为每个节点可以处理更多的数据。高吞吐量尤其重要，因为它可以实现快速备份，最大限度地减少备份窗口的长度。在三个目标中，优化其中任何两个都是容易的，但优化全部就不一样了。为了获得良好的重复数据删除效率，有必要对重复检测执行数据索引。索引元数据大小随着系统的容量而线性增长。保持这个元数据在内存中，会产生良好的吞吐量。但是，可用RAM的大小将会对系统的可扩展性构成严重限制。将索引元数据移动到磁盘将消除可扩展性限制，但会严重损害性能。最后，本文可以像在常规文件服务器中一样优化吞吐量和可扩展性，但是本文会丢失重复数据删除。实现所有三个目标是一项不容易的任务。

另一个不太明显但同样重要的问题是重复引用管理：重复数据共享带来了决定谁将使用特定数据单元以及何时可以回收的需要。这些参考管理机制的计算和空间复杂性随着支持容量的增长而增长。

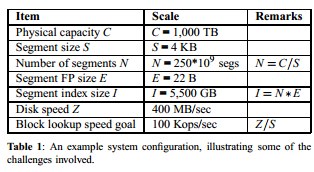
本文应用新的机制来应对重复数据删除挑战，并将其与已有的工程技术结合起来，以考虑所有三个重复数据删除目标来设计和评估系统。渐进采样索引消除了由索引造成的可扩展性限制，同时以O（1）时间复杂度为内存提供大多数查找请求。本文的索引使用抽样来执行细粒度的索引，并且通过需要更少的内存资源来极大地提高可伸缩性。本文通过引入分组标记和扫描来解决参考管理的问题，这种机制可以最大限度地减少磁盘访问并实现接近最佳的可扩展性。

## 2.4遇到的难题

### 2.4.1 索引

大多数重复数据删除系统在子文件级别运行：文件或数据流被分成一系列固定或可变大小的段。 对于每个段，加密散列（MD5，SHA-1/2等）被计算为其指纹（FP），并且用于唯一地识别该特定段。 指纹索引被用作存储在系统中的FP的目录，允许检测重复：在备份期间，如果特定FP的索引中存在形式<FP，磁盘位置>的元组，则对 该段的现有副本已创建。 否则，该段被认为是新的，副本被存储在服务器上并且索引被相应地更新。 在许多系统中，FP索引对于还原过程也是至关重要的，因为索引条目用于查找备份组成部分的确切存储位置。

索引需要具有三个重要的属性：1）扩展到高容量，2）实现良好的索引吞吐量，3）提供高重复检测率 - 即高重复数据删除效率。 表1显示了这些目标对于PB级系统如何变得非常具有挑战性如果系统容量为1 PB，并且段大小为4 KB（对于细粒度重复检测），索引容量至少需要5500 GB，以支持系统中所有2500亿个对象。 这样的索引是不可能在内存中维护的。但是，将其存储在磁盘上会大大降低查询吞吐量。 要达到400 MB /秒的速率，就需要索引（以及整个重复数据删除系统）提供至少100 Kops / sec的查询服务吞吐量。 试图通过在磁盘上存储索引来扩展到1 PB，将无法实现这一级别的性能。 使分段大小更大（例如，128KB）将使得重复数据删除更加粗糙并严重降低其效率，同时仍需要不少于172GB的RAM用于索引。



显而易见的是，高效的可变索引是一个难题。 除了所有其他索引挑战之外，还必须指出，段FP是密码哈希，随机分布在索引中。相邻索引条目不共享任何地址，任何一种简单的预读方案都不能分摊在磁盘上存储索引条目的成本。

### 2.4.2参考管理

与传统的备份系统相反，默认情况下，重复数据删除系统在文件之间共享数据。 参考管理是必要的跟踪段的使用和回收释放的空间。 除了可扩展性和速度之外，可靠性是参考管理的另一个挑战。如果一段文件当其仍被文件引用时释放，则会发生数据丢失，文件无法恢复。 另一方面，如果一个段在实际上不再使用时被引用，则会导致存储泄漏。

为了简单起见，仅考虑了简单的参考计数而没有考虑可靠性和可恢复性。然而，引用计数的可靠性较低，因为它容易丢失或重复更新：当发生错误时，某些段可能会更新，有些可能不会更新。需要复杂的事务回滚逻辑来使引用计数一致。而且，如果一个段被损坏，重要的是知道哪些文件正在使用它，以便通过再次备份文件来恢复丢失的段。不幸的是，引用计数不能提供这样的信息。最后，在大动态系统中几乎没有办法验证参考计数是否正确。

### 2.4.3客户端-服务器交互

即使本文解决索引和参考管理问题，也不能保证端到端的高吞吐量。优化的客户端 - 服务器接口对于获得重复数据消除的好处是必不可少的。典型的重复数据删除客户端在备份过程中执行以下步骤：1）从文件中读取数据，2）形成段并计算FP，3）将FP发送到服务器并等待索引查找结果，以及4）对于每个未命中的索引，传送相关数据到服务器 - 否则创建对现有段的引用。 这个过程可能会遇到三种不同类型的瓶颈。 首先，从磁盘读取文件是一个I / O绑定操作。 其次，计算密码哈希是一项非常耗费CPU资源的任务，客户可能无法以必要的速率计算FP。 最后，高延迟和低通信吞吐量可能成为整体性能的主要瓶颈。

# 三 论文提出的解决方案

## 3.1目标与系统架构

本文将性能目标设定如下：

**•可扩展性：**存储和索引数千亿段。

**•重复数据删除效率：**尽力而为的重复数据删除：如果资源稀缺，可以牺牲一些重复数据删除换取速度和规模。

**•吞吐量：**数据备份，还原和删除的近乎原始磁盘的吞吐量。

文件管理器（FM）负责跟踪重复数据删除服务器上存储的文件。 FM使用三级层次来管理文件信息，如图1所示。底层包含文件，每个文件由一组元数据表示，并由文件FP标识，并由文件组成的所有段FP计算。 中间级别由备份组成，属于同一备份会话的组文件。 在顶层，多个备份被汇总到一个备份组中，允许FM对系统中的文件/备份变更进行粗粒度跟踪，以协助本文的参考管理机制。

段管理器（SM）负责索引和存储原始数据段，并且可以在与FM相同或不同的服务器上运行。分段以大容量（例如16MB）的存储单元存储在磁盘上。容器由原始数据和列出容器中存储的所有FP的目录组成。所有的磁盘访问都是以容器的粒度进行的。通过减少容器I / O并提高索引效率（如3.2.1部分所述），将相邻的段存储在同一个容器中可大大提高重复数据删除性能。 SM还包含重复数据删除索引，并在添加/删除段时对其进行更新。

客户端组件读取文件内容或接收数据流（例如tar数据），执行分段，并计算段FP。查询SM索引后，客户端创建对位于SM中的FP的现有副本的引用，并启动新FP的数据传输。一旦文件被完全处理，文件管理器就会用文件元数据进行更新。

在不失一般性的情况下，本文使用固定大小的4KB段来实现细粒度的重复数据删除，尽管没有一个机制依赖于这个假设。

## 3.2渐进式采样索引

大多数重复数据删除系统在执行备份还原时，都依靠索引（或类似的类目录的结构）来确定每个段的磁盘位置。这迫使对所有FP至少包含一个包含位置信息的完整索引的严格要求，系统将不得不维护和防止崩溃，损坏等，因为错误是不能容忍的。如果由于索引失败而无法确定段的磁盘位置，则整个文件或备份将被损坏。保持这样的数据结构是一项困难且耗费资源的任务，几乎肯定会影响系统的可扩展性和性能，因为索引通常需要存储在内存，性能和磁盘上，以保持持久性。

为了解决索引难题和高性能的数十亿对象，本文不得不通过引入直接可定位的对象来消除这个限制：当文件存储在系统中时，文件段位置信息与文件元数据一起存储，因此去除需要查阅索引文件段的确切位置。例如，如果文件F由分别存储在盘位置1,2和3的具有指纹 A，B和C的段组成，则F将由列表“A，1，B，2，C，3”代替只有“A，B，C”。增加的文件元数据大小不是问题，因为元数据存储在磁盘上，而本文获得的索引自由是非常有价值的。

### 3.2.1重复数据删除效率：预取和缓存

由于索引中插入了“1 out of T”FP，所以索引命中-因此，重复数据删除效率被T因子减少。但是，当查找操作命中一个采样的FP（也称为“钩子”）时，本文找到它所属的容器，并将该容器目录中的所有FP预取到内存缓存中。已经表明[19]由于空间局部性，后续查找命中FP高速缓存的可能性很高：如果钩子FP A之后是FP B，则很可能A和B将按顺序重新出现在将来，在这种情况下，A将会种子预取其容器目录，导致B的缓存命中。容器目录预取对于提高采样索引的重复数据删除效率是非常有效的。然而，预取引入了一个最小采样率：每个容器至少有一个FP（例如，存储在容器中的第一个FP）必须作为钩子在存储器索引中，以便进行预取。因此，如果容器大小为K MB，那么R≥Rmin = S /（K \* 210），并且随后可扩展性不再是“无限的”：最大支持容量现在是C≤（M \* K \* 210） / E。对于4 KB的段和16 MB的容器，至少4096个 FP中有一个需要采样，而64 GB的RAM，如表1的例子，C≤47,662 TB，这个值仍然很高。

**重复数据删除效率。**尽管采样和FP预取的组合通常可以产生高达100％的重复检测，但随机逐出高速缓存条目可以减少重复数据删除。使用简化模型，本文可以估计系统的重复数据删除效率。每个容器目录至多包含（K \* 210）/ S = 1 / Rmin = Tmin条目。如果本文想达到重复数据删除效率为f％，并且本文在一个容器中遭受了x个未命中，那么：

f / 100 = 1 - （x / Tmin）⇒x = Tmin \*（1-（f / 100））。

如果一个特定的容器在一个很长的时间范围内（最有可能的情况下，特别是当使用LRU时）遭受一次驱逐，那么所有的x个未命中将落在击中索引的两个连续的钩子之间，因此：

T = 1 / R = x + 1⇒T = Tmin \*（1-（f / 100））+ 1⇒（E \* C）/（S \* M）= Tmin \*（1-（f / 100） + 1（2）

使用公式2，本文可以计算出，在表1的例子中，有64 GB的内存，重复数据删除效率为f = 97.9％。或者，对于给定的目标重复数据删除效率，本文可以计算出实现它所需的值：例如，如果本文希望f≥95％，并给定E，C和S，则所需的内存量为M≥26.7 GB。

### 3.2.2渐进式取样

采样索引的一个简单而重要的优化是基于以下观察：公式1使用系统的总存储容量，因此计算支持所有C / S数十亿个对象所需的Rtot的值。 然而，在任意给定时间，只有实际存储在系统中的数据量需要被索引，2这使本文能够利用渐进采样率来计算R，使用所使用的存储量而不是最大的原始存储量。 最初本文设置R = 1，并随着更多的存储被使用而逐渐减少。 在本文的工作示例中，具有64 GB的RAM，R = 1可以索引11 TB的存储空间。 当本文接近11TB的限制时，本文可以设定R = 0.5，并对指数进行下采样（例如用FP mod 2 6 = 0下降指数FPs），从而使索引容量加倍。最终，随着使用量达到1000 TB，R将会收敛到Rtot = 0.0116。

## 3.3分组标记-扫描

备份组旨在控制GMS需要管理的条目数量，并由文件管理器创建和管理。当备份很小时，本文将多个小型备份聚合到一个更大的备份组。文件管理器跟踪每个备份组的更改，并为每个更改的备份组进一步跟踪文件是否已被添加或删除。 GMS运行期间，将执行以下步骤：

**1.标记更改组。**只对已更改的备份组进行标记，不对未更改的备份组执行。以图2为例，假设文件管理器的变更跟踪显示，自从上一个GMS周期以来，本文从组Group1中删除了一些文件，向Group3中添加了一些文件，并且没有对Group2进行修改。在这种情况下，本文只需要碰备份组Group1和Group3中的文件。通常情况下，大多数备份组（如Group2）不会更改，这些组中的文件不需要标记。 G1和G3的标记结果是遍历全部重新计算的

Group1和Group3中的文件，并重新计算具有这些文件所使用段的所有容器的G1和G3。一个组的标记结果，比如G1，是一个位图，作为每个容器的文件实现。

**2.将受影响的容器添加到清扫清单。**只有被删除文件的组所使用的容器才需要被扫描，因为只有那些容器可能已经被释放。在图2的示例中，Group1删除了文件，并使用了容器1和2.因此，本文将这两个容器放在清单中。其他容器中的段仍然由未改变的组（例如Group2）中的文件引用，或者由新组（例如Group3）中的新文件引用。

**3.合并，扫除和回收释放的空间。**对于清单中的每个容器，本文合并使用该容器的所有组的标记结果。如果一个细分市场不被使用，它可以被收回。在图2的示例中，对于Container 1，本文合并（旧的）G2和（新的）G1，以确定可能未使用的段。同样，本文合并（新的）G3和（新的）G1，以确定容器2中可能未使用的段。

## 3.4客户端-服务器端交互

即使使用高性能的服务器组件，也不可能实现高吞吐量，除非客户端能够以足够高的速率将数据推送到服务器。 为此，本文的客户端组件基于事件驱动的流水线设计，它使用了一个简单的，完全异步的RPC实现。

本文的RPC协4议是通过在TCP流或系统IPC机制（例如，命名管道）上的消息传递来实现的，这取决于通信是远程的还是本地的。 TCP实现利用多个TCP连接来满足吞吐量要求。 所有的RPC请求都是异步的并且是批处理的，以便最小化往返开销并提高吞吐量。 客户端可以为每种类型的RPC注册不同的回调函数。 回调函数用于在调用者可用时将RPC结果传递给调用者。

基于本文的异步RPC协议，本文实现了一个事件驱动的客户端流水线，如图3所示，其中每个备份步骤都是作为一个单独的流水线阶段来实现的。

首先，阅读器线程R接收备份调度表，读取大块数据（例如，256个段），并将请求排队到散列队列HQ。 散列线程H将来自HQ的请求出队，对每个数据块执行分段，并计算FP。 计算密码散列是一个计算量大的操作，为了充分利用多个CPU内核，H使用n个MD5工作线程（H1，H2，...，Hn）来异步计算FP。 一旦块的段FP被计算，回调函数CB1将更新的请求排入查询队列LQ。查询线程L接收来自LQ的请求，并向服务器发出一个单批处理的异步查找RPC，从而引发所有256个FP的单个RPC往返。回调函数CB2传递RPC回复，并创建对在服务器上找到的FP容器的引用。如果没有找到一个或多个FP，则CB2将更新的请求排入存储队列SQ。

存储线程S接收来自SQ的请求，并通过一个单独的批处理异步RPC将原始数据块发送到后端。回调函数CB3确保写入操作成功，并将每个文件的最后一个请求转发到关闭队列CQ。

最后，关闭线程C，接收来自CQ的最终请求，执行清理，计算文件元数据，并更新文件管理器。

客户端队列允许本文更好地理解系统行为。例如，对于哈希计算吞吐量较低的客户，本文可以观察到HQ大部分时间都是满的，而低的网络性能会导致LQ和SQ大部分时间。在这种情况下，每个流水线阶段可以使用多个线程。例如，通过使用两个存储线程，本文可以以更高的速度处理来自SQ的请求。

# 四 论文解决方案的效果 4.1吞吐量4.1.1备份吞吐量

**索引吞吐量。**在执行任何macrobenchmarks之前，本文使用微基准来确保索引可以支持本文的目标，例如，在表1的例子中，至少为400 MB /秒。在所有的微观基准中，指数可以很容易地处理所需的利率：即使指数占用率超过97％，插入/查找/移除成本也不会分别超过7,619 / 12,020 / 16,836个周期。例如，在3GHz的CPU上，以及在系统中存在所有传入的FP的最坏情况下（并且布隆过滤器没有帮助），索引可以维持大约975 \* TMB /秒的备份速率其中T是采样周期。对于本文的测试配置，T = 101，索引可以维持大约98.5 GB /秒的速率。

**不重复的数据：基准与原型。**图4显示了使用合成数据集的备份吞吐量。为了评估系统的并发能力，本文改变了1,4,16,32,64,128和256步的并发备份数量。为了保持一致性，所有备份都由多个3 GB文件组成，这些文件总共可以达到768 GB。

不重复数据吞吐量测试旨在在没有重复的情况下测量原型的行为。客户端执行初始备份或进行了大量更改时，不重复数据可能很重要。这个测试强调磁盘和网络系统需要传输大量的数据。

为了了解原始硬件的性能，本文首先测量了一个基准吞吐量。 磁盘阵列的基准吞吐量（图4中的“基线”）是通过将相同的合成工作负载写入文件系统来测量的。 对于单个备份，基准吞吐量约为1 GB /秒。 这是存储系统的最大吞吐量。 由于磁盘争用随着并发备份数量的增加而增加，因此基准吞吐量会迅速下降至300 MB / sec左右，以便同时存储多个备份。

使用本文的原型备份相同的数据集（图4中的“唯一数据”）可以实现每秒950MB的稳定吞吐量，因为本文扩展到多个并发备份，这比普通文件服务器要好得多。 这主要是因为本文的原型对所有传入数据执行分段，并管理容器到磁盘的序列化（不管内容源），因此减少了并发磁盘访问。

**重复的数据备份吞吐量。**在使用本文的原型备份不重复的数据工作负载后，本文再次备份相同的文件（图4中的“重复数据 - 冷却缓存”行）。这一次，所有的细分都是重复的，本文的目标是观察本文的原型只需要参考现有的数据，而不是实际存储新的数据。这个测试主要强调索引查找和磁盘预取操作。

最初，对于低级别的并发性，对于小型随机磁盘读取（对于容器FP目录预取）的惩罚主导性能。吞吐量稳步提高，因为本文提高了并发性和重复消除水平，64个并发备份流的聚合磁盘吞吐量达到了6.6 GB /秒以上。当磁盘访问已经是随机的，并发访问不会引入更多的随机性。另一方面，并​​发访问可以充分利用磁盘阵列中的每个磁盘。因此总吞吐量增加。在64个并发数据流之后，磁盘阵列的预取容量已经饱和，而并发开销（索引/缓存锁定，磁盘访问等）的影响变得明显：重复的数据备份吞吐量降至6 GB /秒大多不变。

**备份吞吐量结论。**总之，本文的备份吞吐量实验表明，在备份独特数据时，本文的系统几乎与单流备份的正常文件服务器一样高效（对重复数据删除不会造成损失），而对于多数据流备份，系统速度要快几倍。这表明本文的系统能够更好地组织磁盘上的数据，即使在并发备份的情况下也能实现高吞吐量。当数据大部分是重复数据时，本文可以实现950 MB /秒的单流备份和6 GB /秒的多流备份。多个流有助于提高总吞吐量，因为它们可以最大化容器FP目录预取的吞吐量。

本文观察到的主要限制是由于硬件限制：在本文的端到端性能测试中容器预取吞吐量有限和CPU /网络瓶颈。在配备数百个快速寻找物理磁盘的生产系统上，利用更快的网络连接，本文预计将会看到更高的吞吐量。本文观察到的唯一软件限制是由于pthread锁定，并被认为是次要的，因为它只对最多128个并发备份流的吞吐量影响最小。  
4.1.2参考更新吞吐量

在重复数据删除系统中不经常测试的关键属性是参考更新的性能，特别是当本文需要删除数据时 - 几乎每天都会发生的操作。图5显示了在系统为空或接近满负载时，合成数据集被备份或删除时所测量的参考更新时间。时间与备份或删除数据的大小成线性关系，因为本文需要更新每个使用段的引用。

该行的斜率对应于参考更新的吞吐量，对于数据添加为3.2 GB /秒，对于数据删除为3.1 GB /秒。删除速度稍慢，因为段被删除时，它们也需要从索引中删除。与常规的文件系统相反，重复数据删除系统的删除吞吐量很慢，因为本文支付数据共享的代价。但是，它比新数据的备份吞吐量还要快，这样可以防止备份进程不得不停顿并等待删除机制释放空间。  
4.1.3恢复吞吐量

重复数据删除系统的基准测试主要是由备份测试来决定的，而且对恢复的测试大多被忽略 - 可能是因为恢复过程通常很慢，正确性是主要关心的问题。 但是，恢复是一个重要的操作，本文希望确保本文的原型提供了足够的性能。 在本文的测试中，所有的数据都被正确的恢复，本文的单流恢复吞吐量大约为1 GB /秒，而两个或更多并发恢复流的吞吐量为430 MB /秒。 单流恢复是快速的，因为大多数访问是顺序的，而多个并发恢复流引入了磁盘查找。 使用直接可定位的对象允许本文在不使用索引的情况下执行恢复，从而使整个过程具有很高的可扩展性。  
4.2 重复数据删除效率

虽然本文愿意为高可扩展性牺牲一些重复数据删除准确性，但本文仍然希望确保系统提供足够的重复检测。特别是，由于采样提供了所需的可扩展性，所以重复数据删除效率将主要由预取的有效性决定。

在本文的综合数据集中，真实（“基本事实”）重复是100％。本文的原型始终消除不少于97％的副本。这与基于等式2的理论预期是一致的：当本文从容器目录中预取FP时，并且由于采样率是1/101，所以可能找不到前100个FP。第一次击中之后（最坏的情况下是101个FP），本文预取所有容器中的所有FP。所以从理论上讲，本文可能无法检测出超过4096个FP的100个，即2.4％。

## 4.3与当前最先进技术的比较

在评估重复数据删除系统时，往往是这种情况：定制方法和私人工作量被用来量化所提议的机制的有效性（例如，[19]和[12]）。与其他系统的比较通常比较困难，并且仅限于参考供应商报告的结果，主要是因为没有商定的重复数据删除基准，这将使基准测试和比较成为公平和有意义的基准。此外，当瞄准最先进的系统的性能时，几乎不可能证实获得，部署和基准测试的成本和努力或它们中的一个。在本文的评估中，本文尝试使用能够以有趣的方式运行系统的数据集，而且这些数据集相对容易被其他系统重新创建和测试。

# 五 总结与优化

为了实现提供下一代重复数据删除支持所需的可扩展性，吞吐量和重复数据删除效率，需要解决重要的工程难题。本文已经提出了一个全新的设计，旨在最大限度地提高整体单节点的有效性，并引入新的机制来解决这些难题中最紧迫的问题。与本文所知的任何其他系统不同，本文可直接定位的对象支持使用渐进式采样索引存储器或SSD（提供卓越的单节点可伸缩性和内存使用效率）。本文的分组标记扫描机制以真正可扩展和高效的方式攻击困难且经常被忽视的资源管理和回收问题。另外，本文提出了一个到服务器后端的异步接口，能够以足够高的速度将数据推送到服务器。

本文原型的性能验证了本文设计的有效性。逐行采样索引可以实现非常好的重复数据删除效率，而每200TB的原始存储空间只能使用10GB的内存（在本文的测试中，对于500TB，只有25GB）。另外，本文能够实现从950（所有唯一数据）到6000 MB / sec（所有重复数据）的备份吞吐量，重复数据删除效率不低于97％，而本文的分组标记和清除方法可以处理数据速度高于3.1 GB /秒，这表明通过充分利用可用资源，可大大提高单节点重复数据删除效率。