**海量存储技术结课论文**

|  |
| --- |
| **去重存储系统中物理垃圾收集的逻辑** |

|  |  |
| --- | --- |
| 院（系）名 称 ： | 计算机学院 |
| 专 业 名 称 ： | 计算机软件与理论 |
| 学 生 姓 名 ： | 柳家胜 |
| 授 课 教 师 ： | 何水兵 教授 |

二〇一七年十二月

目录

[1 绪论 3](#_Toc23868)

[1.1 研究概述 4](#_Toc22475)

[2 研究背景和研究动机 5](#_Toc9941)

[2.1 删除算法 6](#_Toc12548)

[2.1.2 标记和扫描 6](#_Toc21054)

[2.2 删除重复数据文件系统中文件表示 7](#_Toc32299)

[2.3 枚举的性能问题 8](#_Toc7876)

[2.4 技术趋势和枚举 8](#_Toc5707)

[3 架构 9](#_Toc4204)

[3.1 逻辑垃圾回收 9](#_Toc31192)

[3.1.1 逻辑垃圾回收步骤 10](#_Toc2137)

[3.1.2 内存优化 11](#_Toc17913)

[3.1.3 对更快枚举速度的需求 12](#_Toc13658)

[3.2 物理垃圾回收 13](#_Toc869)

[3.2.1 更加优化的哈希技术 13](#_Toc17897)

[3.2.2 物理垃圾回收步骤 14](#_Toc13536)

[3.3 枚举示例 15](#_Toc20259)

[3.4 在线垃圾回收 15](#_Toc27349)

[4 实验方法 16](#_Toc14465)

[4.1 部署系统 16](#_Toc3241)

[4.2 控制实验 17](#_Toc14938)

[4.2.1 测试环境 18](#_Toc19896)

[5 总结 19](#_Toc11641)

**1 绪论**

本文名为《去重存储系统中物理垃圾收集的逻辑》，是2017年fast会议的一篇论文，原论文下载地址为：https://www.usenix.org/system/files/conference/fast17/fast17-douglis.pdf，我选择性的截取了论文中的大部分内容进行翻译和理解，主要去除了其中实验评估部分。论文主要讲述在删除重复数据的存储系统中物理垃圾的回收逻辑，作者对删除重复数据存储系统中两种垃圾收集方法进行了详尽的描述，一个是在不包含重复数据的文件上进行操作的逻辑垃圾回收（LGC），还有一个是在底层数据上执行顺序I/O操作的物理垃圾回收（PGC）。然后分析导致先前方法工作负载的变化，并将其替换为新的GC算法，其枚举时间随系统的物理容量而不是逻辑（预重复数据删除）容量或文件数量而变化。接下来又比较LGC和PGC在已部署系统上GC的性能，结果显示PGC相对于LGC在枚举时间上有着二十倍的改进。最后在受控环境下对各种GC算法的性能进行详细比较，结果显示在枚举时间上PGC相对于其它GC有着高达99倍的改进。

大多数以日志结构方式写的存储系统需要一个垃圾收集机制，通过识别磁盘上未使用的区域来回收和巩固存储空间。但是在删除重复数据的存储系统中，垃圾回收往往更加复杂，这是因为在删除重复数据的存储系统中可能涉及到多个相同的底层数据。上文中提到，在商业的删除重复数据的存储系统中有两个变种，分别为LGC和PGC。第二个变种的需求来源于底层工作负载的变化，因为在考虑到底层工作负载的变化时，往往会包含比较高的重复率或者存在数百万单个小文件，从而使在文件级别上进行垃圾回收的速度异常缓慢。之所以速度这么慢，是因为确定块的活性的过程比较缓慢。在这样的工作负载下，确定组块的活性成为逻辑GC的缓慢阶段。相对而言，假如给物理GC一定的额外优化和更高的初始化开销后，作者发现物理GC在极端工作负载的情况下将该阶段的执行时间减少了两个数量级，并且在常见情况下将其改善了约10-60％。总而言之，PGC相对于LGC具有比较大的优势。

**1.1 研究概述**

作者从日志结构文件系统清理存储空间的机制引出目前人们在清理存储空间方面主要的研究方向。对于日志结构文件系统，主要采用调整数据存放的位置从而创建更大区域的连续存储空间。目前大多数的工作都在致力于优化IO成本，因为任何读取和重写数据的工作都会降低新数据的可用吞吐量。但是在删除重复数据的存储系统中都存在一个额外的并发症，即确定哪些数据是活动的。当新数据写入系统的时候，重复的块被替换为先前存储数据的引用，所以跟踪每个引用是非常必要的。此外，随着工作负载的演变，一些系统的使用率与传统的重复数据删除备份存储系统所支持的使用率截然不同。数据域文件系统的设计目的就是用来处理较小数量（几千个）的大文件（GB），完全备份和增量备份几十年以来一直是计算机备份的主流方法。备份数据中的重复数据将会被删除，剩下的部分被包装成“压缩区”，然后“压缩区”通过诸如Lempel Ziv（LZ）的压缩标准进行减少。总的压缩效果由这两个因素共同作用而产生。数据域文件系统针对在20到40范围内的压缩效果进行优化，因为系统通常会存储“完全备份”的数量。

现在是一个数据爆炸的时代，每天每个人都会产生大量的数据，“完全备份”已然不能满足需求，在作者看来，产生这个变化的原因是技术的发展，总的可以归结为两个方面，一个是备份速度的提高，另外一个原因是文件数量的增加。数据域文件系统使用标记和扫描算法来确定从活动文件到达的活动块的集合，然后释放未引用的空间。 还有其他的选择，如引用计数，通过对引用的计数，确定何时该释放块空间。

人们最初都是采用LGC的方式来进行垃圾回收，系统必须分析每个活动文件以确定存储系统中的活动块集合。转向使用单个文件级备份而不是焦油状的聚合文件，意味着某些系统中的文件数量急剧增加。 这会导致标记阶段的垃圾回收开销较高，尤其是由于所需的随机I/O数量太多。 同时，某些系统中的重复数据删除率较高，导致同一个活动块被重复识别，同样会大大增加垃圾回收开销。 在这样的系统中完成垃圾回收的单个循环的时间可能是几天的时间。 由于与垃圾回收的同时备份数据会导致争用磁盘I/O和磁盘处理，因此对于如此长的垃圾回收周期有重大的性能影响; 此外，一个完整的系统在等待回收空间时可能会耗尽容量。

基于上面LGC的缺陷，作者重新设计了垃圾回收以在物理层面上工作：垃圾回收并不枚举所有活动文件及其引用的块，而是随机访问，垃圾回收执行一系列包含大量块的存储容器的顺序传递。 因为I/O模式是连续的，并且因为它随着物理容量而不是重复数据删除比率或文件数量而变化，所以开销是相对恒定的，并且与系统的大小成正比。

该论文描述了垃圾收集子系统的两个版本，逻辑和物理回收子系统（分别是LGC和PGC）。 并且对PGC子系统进行了创新和极大地改进。 PGC已经经历了一个变革，从改变容器访问顺序开始，然后进一步优化，以降低内存使用量，并避免数据传输的需要。 作者将LGC与较早部署在客户现场的PGC的较早实现和较新的“阶段优化物理GC”（PGC +）进行了比较，其中包含了额外的优化。

**2 研究背景和研究动机**

在任何存储系统中，肯定都包含释放为引用空间的操作。目前存储系统的空间管理有多重方式，包括标记空闲位图或更新空闲列表来立即释放从其节点引用的块等，但是对于重复数据删除存储系统，这些操作却比较复杂，比如确定引用哪些数据块，因为数据块可能在单个文件内以及在不同时间写入的许多文件中都有大量引用。 虽然一些重复数据删除研究假设了FIFO删除模式，但文件删除通常可以以任何顺序。 设计和评估垃圾收集算法时需要考虑许多特性：

•必须保留所有引用的块，以便可以检索用户数据。

•大多数未被引用的块必须被移除以释放空间。

•垃圾收集期间，系统必须支持客户端读取和写入。

•系统开销应该最小化。

作者还指出，大多数未被引用的块必须被删除，而不是全部删除，因为将清理集中在大多数未引用的容器上通常会更高效，假如重新分配大量不连续的活动块，结果仅仅回收少量的空间，这显然是得不偿失的。这在通常用在重复数据删除存储的日志结构存储系统中尤其相关，这些存储系统倾向于呈现双峰分布的容器活性，其中容器往往大部分已经死亡或大部分存活。因为需要重新分配的块比较集中，通过重新分配这些块，系统就可以得到大量的连续空间。

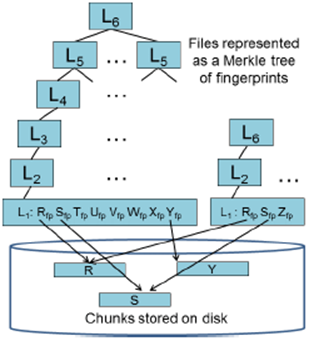
**2.1 删除算法**

**2.1.1 引用计数**

通过维护每个块的引用计数，就知道某个块该在什么时候释放。当在写入路径中发现重复的块时，计数器递增，并且当文件被删除时，计数器递减; 在零时，块可以被释放。

**2.1.2 标记和扫描**

另一种方法是异步算法，以确定实时文件引用哪些块。分标记和扫分两个阶段进行。第一阶段遍历所有活文件，并在数据结构中标记它们。第二期扫描容器和扫未引用的块。该算法中扫描阶段通常是最慢的阶段。在删除重复数据的存储系统中，文件被组织成指纹树的形式，示意图如下所示：



由于活动块是从磁盘读取并写入新的容器，因为要枚举活动指纹涉及随机磁盘I/O，作者发现标记阶段可能是新系统的瓶颈。

**2.2 删除重复数据文件系统中文件表示**

接下来论文讲解了文件在系统中的表示形式，作者把用户直接写成的块称为L0，意思是树的最低层。连续的L0块通过L1块的指纹阵列被引用，L1块本身由指纹识别。L1指纹的数组由L2块参考，继续到树的根;即使文件足够小，不需要像图右侧的示例那样的中间节点，为方便起见，根总是标记为L6。论文中称L1-L6块称为Lp块，其中p是一个范围从1到6的参数，并指示表示文件的元数据。发生重复数据删除是因为可以多次引用一个块。文件系统是Merkle树的森林，但这些树不是分离的，特别是在最底层。

尽管在上图中没有示出，但是Lp块本身被存储在容器中的磁盘上，容器中包括相对较小（数百KB）的元数据部分，容器中的块的指纹列表。因此，它们可能比整个容器更快地被读取。例如，考虑一个具有100TB物理容量的系统，20\*TC，8KB L0块和20字节指纹。逻辑容量，即可由应用程序写入的存储量为2PB（20-100TB）。由于客户端逻辑写入的每个8KB需要存储在L1中的20字节指纹，所以L1块是5TB。树的上层（L2-L6）也存储在容器中，但较小。这个例子强调标记阶段不能完全在内存中执行，因为L1引用必须从磁盘读取。 Data Domain支持fastcopy [7]系统命令，以使用相同的基础Merkle树有效地复制现有文件。它使用一个新名称创建新文件，并因此创建一个新的树的L6根，但该树则引用相同的LP块。由于这个操作只涉及到树的根，所以它的速度非常快，并且不会增加包含L6的一个块以外的物理空间。

**2.3 枚举的性能问题**

标记和扫描的标记阶段涉及枚举从活动文件中引用的所有活动指纹并为每个指纹标记数据结构。一些问题会影响性能：

•重复数据删除和压缩。枚举成本随逻辑空间增长;一些数据集具有极高的TC，使得逻辑空间非常大，列举速度不合理。请注意，重复数据删除可能会有很大差异，而LZ压缩通常在一个很小的范围内（2-4）。

•系统中的文件数量和文件大小分布。在枚举文件时，每个文件都有元数据开销，对于小文件，开销可能与枚举时间本身一样大。另外，如果系统在文件级发出并行枚举线程（一次处理多个文件），那么一个无法进一步并行化的非常大的文件可能会减慢整体枚举速度，尽管这样的偏差是很少见的。

•Lp树的空间局部性。由于所有L0引用都记录在L1块中，所以遍历Lp树在L1级停止。磁盘上的Lp块碎片增加了枚举所需的随机I/O数量[ 12,13]。虽然顺序写入的文件倾向于具有较高的Lp块位置，但是从增量更改创建新文件会损害局部性，因为枚举必须在Lp树的分支之间跳转。

**2.4 技术趋势和枚举**

在前面概述中提到，随着技术的发展，导致了枚举时间的增加，主要体现在两个方面。首先，客户端通过写入自上次备份以来更改的块或文件等增量更改来创建许多频繁的备份。由于只传输增量更改，所以备份速度更快。通过复制以前的完整备份并应用创建后的用户更改来创建综合完整备份，这是在重复数据删除存储中提供支持的高效操作。通过引用先前备份的未修改部分来生成新文件，从而导致重复数据删除率提高。这允许用户查看和恢复完整备份，而无需恢复先前的完整备份，并应用中间增量更改：每日增量备份和每周完整备份的传统备份策略。对于给定的时间段，这意味着备份存储系统现在可能包含大量完整备份（相应的高重复数据删除率），而不是大量增量备份（重复数据删除率较低）。（与fastcopy不同，合成完整备份会创建一个全新的LP树;因此，用于存储元数据的开销约为数据逻辑大小的1％）。随着备份完成速度的提高，数据库等应用程序需要更多频繁的时间点备份，这些备份变得更加频繁（每天多次）。同样，应用程序可能会在经过重复数据删除的存储上频繁采用快照，从而有效地制作数据的虚拟副本[2]。由于这些技术趋势，一些系统的重复数据删除率高达100-300倍或更高（相应的高TC），而10倍重复数据删除是传统备份环境的典型特征。

第二个技术趋势是文件数量的增加。这是由于上述使用方式的变化以及使用重复数据删除来支持单个文件的备份，而不是像整个备份（类似于tar文件）那样将整个备份表示为单个文件而造成的。用户也可能通过另一个接口（如NFS）访问重复数据删除设备，将其视为比主存储更像备份。随着TC和小文件数量的增加，枚举时间成为整体垃圾收集的一个重要组成部分。为了支持这些新的工作负载并继续扩展性能，作者开发了一种新的标记算法，用深度优先的顺序替换逻辑枚举，使用顺序磁盘扫描以宽度优先的顺序进行物理枚举。

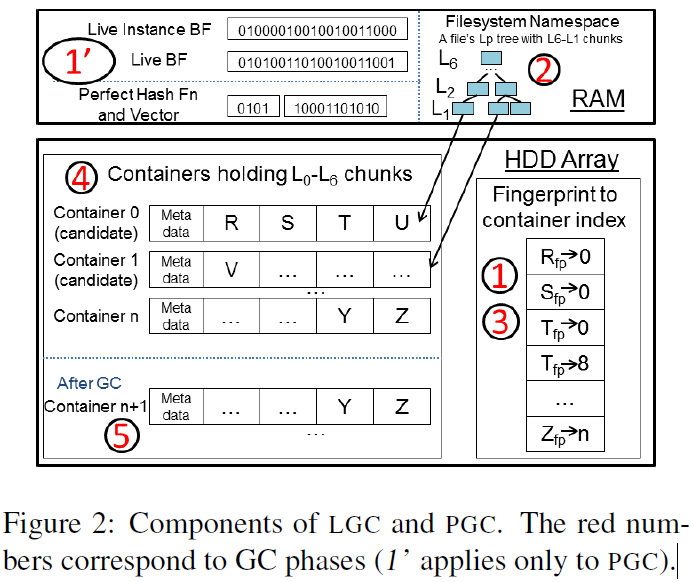
**3 架构**

在本节中，作者主要介绍对原始的LGC算法以及启用PGC和PGC +的更改。 这两种形式的GC都有一个共同的问题，那就是如何处理在线更新文件系统的状态。

**3.1 逻辑垃圾回收**

Data Domain的原始垃圾收集技术使用标记和扫描算法。在更详细地描述每个阶段之前，作者简要地给出一个算法的概述。第一步涉及遍历所有的文件Lp树，并将Bloom块过滤器中的块指纹记录为活的块。之所以使用Bloom过滤器，因为与指纹索引相比，它降低了内存需求，尽管Bloom过滤器也相当大。然后，GC通过读取容器元数据区域来“扫除”任何未参考的块，并确定块是否记录在布隆过滤器中。容器元数据区域如图2所示，包含multimegabyte容器内的块的指纹列表。实时块被复制到新的容器，旧的容器被删除。

布鲁姆过滤器中的指纹匹配可能是假阳性，这导致死块被错误地识别为存活和保留，但是这样的例子将是罕见的;可以通过筛选布鲁姆过滤器来限制其误报率。此外，Bloom过滤器使用的散列函数在运行之间更改，因此下次可能会删除误报。请注意，活块将永远不会被误认为是死的。



**3.1.1 逻辑垃圾回收步骤**

LGC按图2所示的五个阶段进行：合并，枚举，筛选，选择和复制。前四个在复制阶段扫描时共同执行标记。

1. 合并：这将最近的索引更新从内存同步到磁盘，以便下一个阶段可以遍历磁盘索引。在这个阶段，作者还会对文件系统进行快照，以便从一致的角度进行清理，这包括复制文件系统名称空间的根目录。
2. 枚举：为了标识实时块，作者列举了从根引用的所有文件。枚举按照从顶层（L6）开始的深度优先顺序进行，并向下推进到底部内部节点（L1）。枚举在L1处停止，因为它由引用数据块（L0）的指纹组成，这些数据块是树的叶子。作者在Live Bloom Filter中记录活动文件的指纹，指示相应的块应该被保留。
3. 过滤器：因为存储系统可能包含少量存储在容器中的重复块（即，没有完全重复数据删除，例如图2的指纹索引中的Tfp），作者接下来确定要保留哪个实时指纹实例。指纹到容器索引在系统中有完整的指纹列表;其条目被组织成大小为512字节至384KB的桶。然后作者遍历索引。虽然可以保留任何重复指纹的实例，但是作者的默认策略是将实例保留在最近写入的容器中，这会优先考虑最近写入的数据在较早的备份上的读取性能[12,13]。作者使用Live Bloom Filter来跟踪系统中是否存在块，以及活动实例布隆过滤器来跟踪存在重复项的每个块的最新实例。因此，如果Live Bloom过滤器中存在指纹，作者在Live实例布隆过滤器中记录<指纹，容器ID>（具体而言，这两个值的XOR）的组合。
4. 选择：作者接下来估计容器的活性，将清洁重点放在那些能够以最小的努力回收最多空间的容器上。由于集装箱往往大部分是活的或大部分是死的，作者可以通过关注大多数死容器来回收大部分空间[20]。作者通过读取元数据区域的容器遍历容纳块指纹。作者计算<指纹，容器ID>值并检查实例实例布隆过滤器。计算每个容器的活性作为活指纹的数量除以容器中指纹的总数量。作者对内存中的活跃记录进行排序，并根据作者的目标回收空间选择动态设置的动态阈值下的容器。
5. 复制：新的容器由从先前的容器复制出来的实时块形成。当一个新的容器已满时，它被写入磁盘，死容器被释放。
6. 总结：在使用全系统Bloom过滤器来避免磁盘上的索引查找浪费的系统中，Bloom过滤器被重建以反映当前的活动块。它的时间与选择阶段大致相同，通常由复制阶段占主导地位。 由于在未来版本的DDFS [1]中已经完全消除了汇总阶段，所以作者省略其进一步讨论。

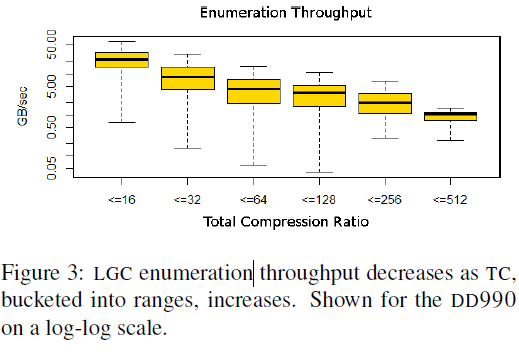
尽管作者已经将这种算法呈现为线性，但在阶段内还存在并行机会，例如在枚举过程中读取多个文件，在过滤期间使用多个索引桶以及在选择和复制阶段使用多个容器。

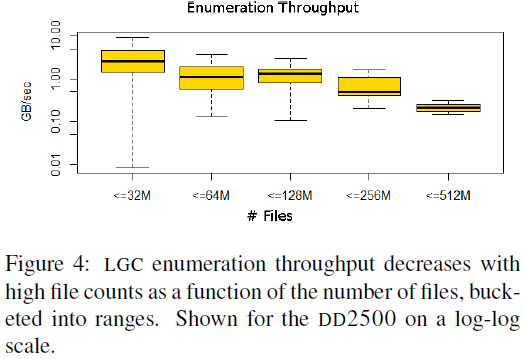
**3.1.2 内存优化**

如上所描述的，该算法假定Bloom存储器具有足够的存储空间以具有低的误报率。取决于系统配置，内存可能有限;根据所需的误报率，跟踪200B（亿）条目的布隆过滤器的大小将为数百GB。对于这样的系统，作者开发了一种技术，将清洗算法集中在最死的块上。

在运行之前描述的四个阶段之前，作者根据布隆过滤器可用的内存和当前存储在系统中的块的数量设置一个采样率。然后，作者运行标记阶段（枚举，过滤器和选择），但仅在插入Bloom过滤器之前考虑与采样标准相匹配的指纹。在选择阶段，作者选择大多数已经死亡的候选容器，但是作者也会根据内存限制来限制容器的总数。然后，作者通过读取候选容器的容器元数据区域并将指纹插入候选布隆过滤器来创建覆盖这些容器的候选布隆过滤器。除了Select之外，上面的步骤被重复，尽管受限于候选布隆过滤器2。例如，如果指纹在候选布隆过滤器中，枚举阶段仅向活布隆过滤器添加活指纹。虽然这引入了第三个Bloom过滤器，但作者只需要两个，因为候选Bloom过滤器可以在Enumeration阶段之后被释放。结果是，作者将清洁工作集中在候选集上。

然而，虽然实际上复制和清洁容器的扫描阶段通常占据头顶，但是对两组标记阶段的需求增加了相当多的处理时间。通常，设备上存储的数据越多，系统就越有可能需要两组标记阶段。大部分为空的系统仍然会有足够的DRAM支持加载系统上的GC，因此标记阶段的Bloom过滤器可能适合内存。经验上，作者发现大约一半的GC运行需要两个阶段：通常，系统启动的负载较少，随着时间的推移，他们存储更多的数据并超过阈值。 Chamness在2011年对EMC Data Domain系统的研究中发现，在所研究的大约六个月内，有一半的系统预计将达到满负荷状态。





**3.1.3 对更快枚举速度的需求**

虽然逻辑枚举已经在生产系统中被成功地使用了十多年，但是研究概述中讨论的技术趋势正在提高总压缩比和小文件的数量。这将创建一个更大的名称空间并增加枚举时间。图3显示了使用作者最大的可用模型DD9903（对较小系统的分析显示类似的结果），数千个部署的系统的LGC枚举吞吐量作为TC的函数。作者将枚举吞吐量定义为系统中使用的物理存储的GB，除以GC枚举时间;这将根据使用容量对GC性能进行归一化。作者给出了一个盒子的枚举范围：晶须显示第90和第10个百分点，盒子显示第75和第25个，中间的结果在框内画成水平线。吞吐量随着TC的增加而稳步下降，说明LGC枚举时间与数据集的逻辑大小有关，而不是物理大小。作者在下面实验中更明确地显示了这一点。

在图4中，作者显示了枚举吞吐量与文件数量的关系。作者选择一个较小的系统DD2500，它是文件数最多的model4;作者看到吞吐量随着文件数量的增加而稳步下降。因此，作者开发了PGC，用逻辑枚举的随机I/O替换物理枚举的顺序I/O。

**3.2 物理垃圾回收**

PGC在几个方面与LGC不同。 首先，作者使用完美的哈希[3，14]（PH）作为辅助数据结构来驱动活动文件的森林的广度优先遍历。 其次，一个新的枚举方法从容器中识别出实时块，而不是遍历每个文件。目前普遍使用的GC版本运行PGC，正如作者下面将要描述的那样，它改善了新使用模式，高重复数据删除和大量单个文件的枚举开销。 然后，为了消除传统用例的性能下降，作者增加了额外的优化。

**3.2.1 更加优化的哈希技术**

确定指纹是否被活动文件引用可以表示为成员资格查询，并且PH是用于最小化存储器要求的数据结构。简而言之，PH由一个哈希函数组成，这个哈希函数可以唯一地从一个密钥映射到一个哈希向量中的一个位置，称为完美哈希向量（Perfect Hash Vector，PHV）。散列函数是通过分析所考虑的静态指纹集来生成的，并且PHV用于通过设置分配给每个指纹的特定位来记录成员资格。以前的工作集中在生成一个PH函数，同时减少内存/分析的开销[3,4,14]。作者的技术直接由Botelho等人提出，他们使用Data Domain中的PH来平衡分析时间和内存以实现安全删除。

如图2所示，作者为PH功能和PHV分配内存。一组指纹有一个PH功能;使用该功能，作者得到PHV中指纹的位的位置。在枚举过程中，作者使用PH代表文件树中的L6到L1层，原因有很多。首先，与布隆过滤器相关的误报率会导致在不正确标记的Lp块下的整个分块分支留在系统中。其次，作者的PH比Bloom过滤器更紧凑，因此它使用更少的内存来表示这些级别。第三，PH协助逐级校验匹配，确保没有腐败。

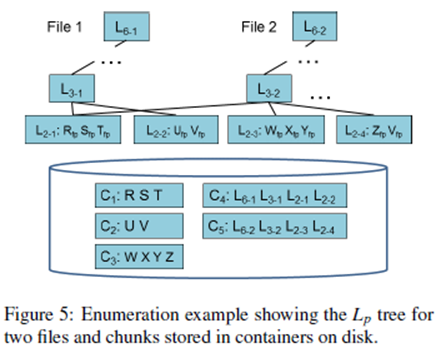
**3.2.2 物理垃圾回收步骤**

PGC有一个新的分析阶段（在图2中标记为阶段1'），但是与LGC具有相同的阶段，并且改变了枚举阶段。

1. 合并：这与LGC相同。分析：作者通过分析磁盘索引中的指纹来创建PH功能。结果是在PHV中从指纹到偏移量的唯一映射。 PH函数和矢量总共使用每个Lp指纹小于4位。
2. 枚举：与LGC不同，作者执行一系列连续的容器扫描，而不是漫步文件树结构。作者首先通过设置漫游位，根据名称空间中引用的文件在PHV中记录活动的L6指纹。作者的系统有一个内存结构，记录每个容器中的Lp类型（L6，L5，... L0），因此作者可以专门用L6块扫描容器。如果在PHV中将L6标记为步行，则作者标记确认位，然后解析存储在L6中的L5指纹，并为PHV中的那些条目设置步行位。然后，作者重复这些步骤扫描L5块，L4块等，直到读取L1块。当为任何块（L6-L1）设置漫游位时，作者也在LGC中记录Live Bloom Filter中的指纹。虽然可以多次读取一个容器，因为它拥有多个类型的块（例如，L6和L5），主要成本来自L1容器，其只读取一次。因此，与文件的逻辑枚举相比，作者可以减少随机I / O和整体运行时间，尤其是对于高文件数和重复数据删除率。
3. 筛选，4.选择，5.复制：这些和LGC相同。

垃圾收集算法的正确性对于防止数据丢失至关重要。首先，GC必须确保在枚举期间不会丢失大块。其次，如果已经存在腐败，GC应该停止并提醒管理员尝试数据恢复。由于重复数据删除，问题更为严重，因为损坏的Lp块可能会导致大量的文件由于共享Lp树而损坏。为了满足这些要求，物理枚举通过使用PHV来计算每LP级别的校验和。由于空间的考虑，作者没有介绍细节，知识简要介绍了一下这个方法。

作者为每个级别使用两个校验和，一个用于父块（P），另一个用于子块（C）。当在Lk级处理一个块时，作者对其Lk-1级别的引用散列数组进行XOR并将其添加到级别Lk-1的子级校验和C.当处理Lk-1级别的块时，作者将块的散列添加到Lk-1的P校验和中。如果没有损坏，则在Lk-1扫描结束时，Lk-1的P和C应匹配。校验和是针对所有L1-L6级别进行计算和匹配的，并且使用PH功能可以防止重复包含散列值。在计算XOR之前，作者通过检查PHV中的相应位来检查一个块是否已经包含在校验和中。作者使用一个PH值函数，将块哈希值映射到两个比特，代表1）找到一个块作为其父项散列数组的成员，2）在下一个较低级别找到块本身。



**3.3 枚举示例**

下面作者举一个逻辑和物理枚举的例子，如图5所示。显示两个文件的Lp树，以及磁盘上容器（C1-C5）中块的位置。文件2是通过复制文件1并在块T之后插入新的内容而创建的。在本例中，Lp树被简化了。从逻辑枚举开始，作者从文件1开始。作者首先读取容纳文件的根块的容器C4，L6-1也读取L3-1，L2-1和L2-2到存储器中。访问L2-1和L2-2引用的L1块包括读取容器C1和C2。按照逻辑枚举File 2的步骤相似。因为L3-2是指L2-1，第二次读取容器C1。物理枚举在L5块继续到L1块之前读取所有的L6块。在这个例子中，容器C4和C5被读取以访问L6块。再次读取这些容器以访问L3和L2块（取决于容器是否仍在内存高速缓存中）。接下来，L1块被读取，所以容器C1，C2和C3被读取一次以访问它们的L1块。

**3.4 在线垃圾回收**

在GC期间将系统置于离线状态是不合理的，因此在GC运行时，作者的客户也正在进行备份。因为作者的重复数据删除过程是针对系统中的任何块进行检查的，所以如果一个客户端写入了一个与其他未被引用的（死的）块相同的块，它就会将未被引用的块切换到被称为块的复活状态。还有一点是打开的文件可能还不能从根目录访问，所以作者特别跟踪打开的文件并枚举它们的Lp树。

作者添加了一个进程，在GC运行时通知GC传入的块，因此块可以添加到实例（实例）向量中。当在复制阶段复活发生时，存在复活块可能正在被删除的危险。作者看到复活是否在目前正在处理的集装箱范围内;如果是这样的话，为了安全起见，作者写一个（可能）重复的块，但是在复制阶段写入的重复项可以在下次GC运行期间被移除。如果复活发生在容器范围之外，那么作者将<指纹，容器ID>添加到活动实例向量，或将指纹添加到PGC +的活动向量。

GC不仅需要支持在线摄取（在GC运行时向系统添加新数据），还必须支持其他任务，如恢复和复制。因此，作者添加了控制来限制GC的资源使用情况，这些可以在每个系统的基础上配置。作者把这个控制叫做节流，但是GC节流的方式已经从LGC变成了PGC。由于PGC以LGC的方式限制GC性能，所以两者之间的直接比较是有问题的。

作者采取两个小节来解决性能上的差异。首先是考虑在PGC打算解决的极端情况下，对LGC和PGC进行比较。如果PGC比LGC提供更好的性能，即使PGC可能更多地进行节流，作者可以看到PGC严格优于LGC。其次是在所有GC算法中手动禁用节流的实验室内实验。

**4 实验方法**

本节介绍作者的实验方法：作者使用了部署系统的遥测和受控实验的组合。

**4.1 部署系统**

像许多供应商[2,10]，作者的产品可以发送系统行为的可选报告。作者使用这些报告从每个报告中提取以下信息：序列号，型号，系统版本，日期，使用中的逻辑和物理容量，文件计数和GC时间。

作者排除任何显示<1TB使用中的物理容量，<1小时或> 2周GC执行或其他异常值的报告。这给作者留下了每个系统一个或多个报告，这可能反映出LGC或PGC发生。

作者的评估侧重于运行LGC的大量部署系统。作者使用第二套运行PGC（PGC +尚未提供）的已部署系统进行极端情况的比较（高重复数据删除率或文件计数）。作者研究了从LGC升级到PGC的系统，并没有将容量，文件数量或重复数据删除率改变超过5％。作者发现，对于更“典型”的系统，运行PGC而不是LGC时出现中度（偶尔显着）的降级，但不可能将这种影响归因于节制效应或一次性开销升级的过程。作者在后面报告的实验室实验中排除了节流。时间为2014年1月至2016年9月。两套（LGC和PGC）包含数千个系统。

评估GC性能有多种方法，特别是评估特定系统环境的变化。作者通常的比较类似系统的方法是用一些度量标准来排序，比如物理容量，然后把它们分成桶。请注意，这些桶被标记为？一些价值，但它们仅反映那些大于下一个较低桶的点（即，它们不是在所有较低桶中累积的）。另外作者通常会发现，以对数形式显示桶会强调异常值。由于PGC +只是最近才向客户提供，所以作者不能使用遥测来评估使用PGC +的系统。

**4.2 控制实验**

为了以可控的方式比较LGC，PGC和PGC +，作者在实验室环境中进行了许多GC实验。为了确保实验中的数据相同，作者获取或生成测试数据集，然后在系统修改时重置为测试数据集的状态。对于大多数实验，作者使用一个负载生成器，用于评估DDFS中的内容消毒;这与塔拉索夫等人的工作类似。该工具使用随机和可预测内容的组合来创建指定大小的文件，以获得所需级别的重复数据消除和压缩。每个新文件都包含来自上一代相同“血统”的重要内容，而添加新内容时会删除一些内容。不应该跨独立的谱系进行重复数据删除，可以并行编写来实现预期的吞吐量。

对于一些实验，作者通过一些额外的步骤来创建作者所谓的规范数据集。当作者写新数据时，作者多次运行GC来老化系统并降低局部性：例如，摄取到5％的容量并使用GC重置到容量的3％，然后再摄入到7％，再次运行GC。（作者注意到，在这个过程中删除的未经过滤的副本的数量高于通常在现场出现的副本的数量（10％或更少）。当作者达到最大系统的物理容量的10％时，摄取就停止了，DD990。规范化数据集是将1.1 PiB的逻辑数据进行重复数据删除并压缩到30.1 TiB的物理数据（36：6\*TC）。作者使用这个数据集来比较不同的设备，也作为评估高重复数据删除率的起点。规范数据集大约是作者用于其他实验的DD860容量的25％。

为了评估使用中的物理容量的影响，作者希望从容量较低开始，所以作者一次吞下了20％的物理容量。对于这个实验，作者没有运行破坏性的GC（物理重组数据和删除额外的重复）在摄入。该实验的TC从17：9\*-18：3\*变化。

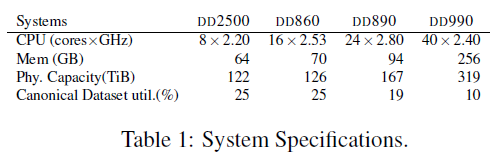
为了开始使用相同的可预测内容，作者使用集合复制[7]在另一个设备上创建数据的完整副本。要关注GC的标记部分，作者可以确定应该回收哪些数据，但在启动扫描和修改文件系统之前终止GC。可以在不修改系统状态的情况下测量多个标记运行（任何GC类型）。如果作者包含复制阶段，作者从保存的副本中复制以将系统恢复到原始状态。每个4MB容器以及索引都是相同的，尽管容器可能存储在磁盘上的不同物理位置。作者使用复制来评估高文件数量，从已经播种了900M文件的质量保证系统复制。

由于作者发现许多GC运行需要两个标记阶段，因此使用一个标记传递和两个标记来评估运行很重要。作者通过收集两组时间并通过堆积条形图在结果中分别指示它们来做到这一点。 LGC-1和PGC-1是较低的（实心）棒;这些可以直接与单个PGC +条进行比较，或者可以考虑标记为LGC-2和PGC-2的较高阴影条。一般来说，较少的完整系统需要一次通过，而更多的完整系统需要两次。

**4.2.1 测试环境**

作者使用各种备份设备进行测试。 作者在四个独立的系统上报告作者的标准数据集的标记阶段花费的时间。 然后，作者使用特定的系统来评估工作负载的某些方面，如复制阶段变化，使用容量，高文件数量或重复率的变化。

表1显示了作者在测试中使用的系统的规格。 它显示了核心，内存，可用物理容量以及规范数据集使用的容量，范围从10-25％。 在比较一个数据集上的所有四个系统之后，作者使用DD860在不同的工作负载上进行广泛的比较。



**5 总结**

工作负载的转移需要在重复数据删除存储系统中采用新的垃圾收集方法。作者不是从深度优先的标记和扫描GC算法，从单个文件的角度来跟踪实时数据，而是采用了广度优先的方法来利用数据的物理布局。 PGC将大量的随机I/O转换为整个存储系统的一系列顺序扫描。当重复数据删除率很高时（100-1000\*而不是10-20\*），它可以很好地工作，当存储系统用于数以亿计的相对较小的文件而不是数千个大型的tar类备份文件时，效果很好。由于PGC的其他开销，包括使用空间高效和准确完善的哈希函数所必需的分析[3,14]，原始的PGC方法并不比它所取代的LGC方法更快。改进的PGC算法（称为PGC +）使用PH来另外减少内存需求，足以避免GC期间的两个标记序列。作者已经证明，它与传统工作负载上的原始LGC具有可比性，这些传统工作负载的指纹适合内存（需要一个标记阶段），在需要两次通过时速度要快得多，而在有问题的工作负载上要好得多。

通过对这篇论文的学习，加深了我对数据存储的理解，让我了解到可以在两个不同的层面对存储系统进行垃圾回收。特别是对存储空间的清理，垃圾回收等方面，更加拓宽了我的视野。