计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2020.20190543
57(2):269-280,2020

新型存储设备上重复数据删除指纹查找优化

何柯文　张佳辰　刘晓光　王　刚
(南开大学计算机学院　天津　300350)
(天津市网络与数据安全技术重点实验室(南开大学)　天津　300350)
(hekw@nbjl.nankai.edu.cn)

Fingerprint Search Optimization for Deduplication on Emerging Storage Devices

He Kewen, Zhang Jiachen, Liu Xiaoguang, and Wang Gang
(*College of Computer Science, Nankai University, Tianjin* 300350)
(*Tianjin Key Laboratory of Network and Data Security Technology (Nankai University), Tianjin* 300350)

Abstract Fingerprint search part is I/O intensive, and the performance of the external storage device is the bottleneck of fingerprint search. Therefore, this paper focuses on the fingerprint search of data deduplication system. This paper compares the traditional eager deduplication algorithm with lazy deduplication algorithms that reduce the number of disk accesses, and studies deduplication algorithm on the emerging storage devices: Optane SSD and persistent memory, and gives optimization suggestions. In this paper, we model the fingerprint search delay of the eager deduplication algorithm and the lazy deduplication algorithm, and three conclusions under the new storage device are obtained through the modeling results: 1) The number of fingerprints for batched search should be reduced; 2) The local ring size should be reduced on faster devices, and the local loop size has an optimal value; 3) On fast devices, the eager fingerprint lookup is better than the lazy fingerprint lookup. Finally, the experimental results verify the correctness of our model on the actual HDD, Optane SSD and emulated persistent memory. The eager algorithm is better than the lazy algorithm on the emerging storage devices, and the locality ring optimal value is advanced, which basically conforms to the conclusions of the proposed model.

Key words deduplication; persistent memory; fingerprint index; emerging storage device; data spatial locality

摘　要 指纹查找部分是I/O密集型工作负载,即外存存储设备的性能是指纹查找的性能瓶颈.因此关注重复数据删除系统的指纹查找部分,对比了传统的勤奋指纹查找算法和致力于减少磁盘访问次数的懒惰指纹查找算法,分析了2种方法在傲腾固态硬盘(Optane solid state drive, Optane SSD)和持久性内存(persistent memory, PM)两种新型存储设备上的性能表现,并给出了优化建议.对勤奋指纹查找算法和懒惰指纹查找算法的时间进行建模,分析得出了指纹查找算法在新型存储设备下的3点优化结论:1)应减少统一查找的指纹数;2)在较快设备上应减少懒惰指纹查找中局部性环的大小,并且局部性

收稿日期:2019-08-15;**修回日期**:2019-11-29
基金项目:国家自然科学基金项目(U1833114,61872201,61702521,61602266);天津市自然科学基金项目(17JCYBJC15300,16JCYBJC41900);天津市人工智能重大专项项目(18ZXZNGX00140,18ZXZNGX00200);中央高校基本科研业务费专项资助项目
This work was supported by the National Natural Science Foundation of China (U1833114, 61872201, 61702521, 61602266), the Natural Science Foundation of Tianjin (17JCYBJC15300, 16JCYBJC41900), the Artificial Intelligence Major Project of Tianjin (18ZXZNGX00140, 18ZXZNGX00200), and the Fundamental Research Funds for the Central Universities.
通信作者:王刚(wgzwp@nbjl.nankai.edu.cn)

环大小存在一个最优值;3)在快速设备上,勤奋指纹查找的效果要优于懒惰指纹查找.最终,在实际机械硬盘(hard disk drive, HDD)、Optane SSD 和 PM 模拟器上实验验证了模型的正确性.实验结果显示,快速设备上指纹查找的时间相较于 HDD 减少 90%以上,并且采用勤奋算法要优于懒惰算法,局部性环最优值前移的现象,也与模型理论优化结果吻合.

关键词 重复数据删除;持久性内存;指纹索引;新型存储设备;数据空间局部性

中图法分类号 TP391

随着大数据时代的到来,如何高效、可靠地存储海量数据成为业界所关注的一个重点.微软[1]和易安信公司[2]的研究称在他们的数据中 50%~85%都是重复数据,如果能将这些重复数据进行删除,就能节约出大量空间来存储更多数据并且能提升云备份系统的带宽.但重复数据删除系统既是计算密集型系统,也是 I/O 密集型系统,外存的存储介质的性能很大程度影响到了重复数据删除系统的指纹查找效率,最终影响到了重复数据删除系统的性能.其中,我们称 Data Domain 重复数据删除文件系统(data domain deduplication file fystem, DDFS)[3]采用的指纹查找算法为勤奋指纹查找算法,是通过预取整个数据块的指纹,利用重复数据块中指纹的空间连续性减少对外存的访问. 而懒惰指纹查找算法[4]则是在勤奋指纹查找算法的基础上,将通过合并 I/O 的方式进一步减少外存的访问来增加指纹查找效率,并且通过局部性环的方式保证了其指纹查找算法的 Cache 命中率接近勤奋指纹查找算法.

近年来,高性能固态硬盘和持久性内存(persistent memory, PM)的出现,使得外存的随机写延迟能够接近顺序写延迟,随机读写并不会成为外存访问的瓶颈.所以,原来针对机械硬盘(hard disk drive, HDD)优化的重复数据删除指纹查找算法需要进行改进.本文以提升重复数据删除系统指纹查找性能为目的,对新型存储设备上的勤奋指纹查找算法和懒惰指纹查找算法的表现进行了研究.

本文的主要贡献包括 3 个方面:

1) 分析了重复数据删除系统中的勤奋指纹查找算法和致力于减少外存访问的懒惰指纹查找算法[4],并实验说明这 2 种指纹查找算法在 HDD、傲腾固态硬盘(Optane solid state drive, Optane SSD)和 PM 上的性能表现;

2) 对勤奋指纹查找算法和懒惰指纹查找算法进行数学建模,理论分析新型存储设备对 2 种指纹查找算法的影响,并得出在新型存储设备上的指纹查找算法优化结论;

3) 通过在 HDD,Optane SSD,PM 上进行指纹查找算法的实验,验证了模型的正确性以及建模得到的优化结论的有效性.

由于 PM 并未大范围生产,市场上无法获得,所以本文对持久性内存的实验,采用 Quartz 模拟器[5]来进行模拟实验.

1 相关工作

面对全世界的数据量不断增加的挑战,如何处理重复数据、减少数据冗余,逐渐成为学术界和工业界关注的目标.从 1950 年利用 Huffman 编码的基于编码的静态模型[6]、基于字典的字符串压缩模型[7]再到现在的基于数据块或者文件级别的重复数据删除[8],处理的重复数据的量越来越大,对于处理重复数据的带宽、延迟要求也越来越高.

重复数据删除系统既是计算密集型的系统,也是 I/O 密集型的系统.由于重复数据删除系统在外存磁盘中更多是存储指纹以及数据块结构,因此一些相关研究着眼于设计能减少磁盘访问的索引结构和相关算法,例如 DDFS[3]利用 Bloom Filter[9]和重复数据块中指纹的空间连续性来达到此目的. 在此基础上,懒惰指纹查找算法[4]采用合并 I/O 的延迟访问策略,同时该算法也通过局部性环等保持数据块局部性的 Cache 策略来提高 Cache 命中率.除了利用重复数据块中指纹的空间连续性,还有利用文件之间相似性的工作[10],其思想是如果 2 个文件中具有代表性的数据块相同,则这 2 个文件大概率会重复,基于这一特性,通过判断 DRAM 中的关键块信息和文件信息摘要,减少对外存的访问.最后,还有同时利用数据块空间局部性和相似性的 SiLo[11],其基本思想是将相似性高的小文件组成 1 个 Segment,然后再将空间局部性高的 Segments 组织成 1 个 Block,以此来同时利用空间局部性和相似性.

Optane SSD 和 PM 等快速存储设备的出现,以及基于持久性内存的新编程模式的提出[12],对加速

指纹索引带来了新的可能性.学术界一些研究针对这些性能更佳的新型存储设备上的数据结构、文件系统和应用程序进行改进,例如为解决叶子结点排序所导致的写放大问题而改进 B^+ 树[13],再如修改扩容机制来降低散列的写次数和一致性问题[14],以及设计持久性内存和 DRAM 混合的新文件系统(NOVA)[15]等.另一方面,也有工作专门对 SSD 进行指纹索引优化[16-18]和对 PM 文件系统研究重复数据删除方法[19].

本文探究的新型存储设备下的重复数据删除系统指纹查找算法优化,是为了比较在新型存储设备下,原来针对 HDD 进行优化的重复数据删除系统指纹查找算法是否有效,并提出在新型存储设备上的重复数据删除系统指纹查找算法的优化方案.

2 重复数据删除系统指纹查找算法

本节主要介绍重复数据删除系统的结构、针对 HDD 优化的懒惰指纹查找算法以及对应的 Cache 改进策略.

2.1 重复数据删除系统基本结构

重复数据删除系统作为一个针对数据流进行重复数据删除的系统,在目前的海量数据场景下显得非常重要.重复数据删除系统不同于常见的压缩算法,传统的压缩算法针对的粒度是字符或者字符串级别的,但是重复数据删除系统所针对的粒度是块级别(一般为 4~12 KB 的块)或者文件级别.重复数据删除系统的主体流程图如图 1 所示:

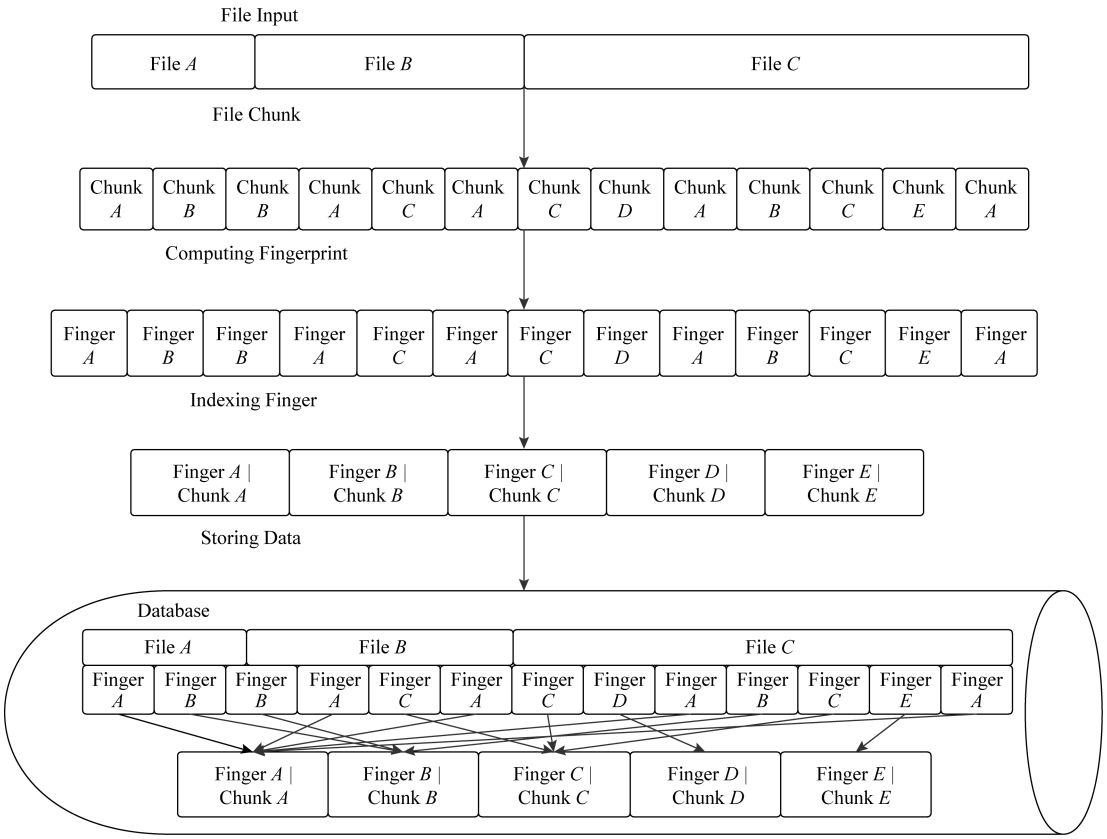


Fig. 1 The deduplication process

图 1 重复数据删除流程

其中,重复数据删除算法的基本流程描述如下:

1) 数据分块.一般采用 4~12 KB 的定长块分块或者采用 Rabin-Hash 的非定长块分块.

2) 指纹计算.对每个数据块进行指纹计算,一般计算的方法采用安全散列算法(secure Hash algorithm-256, SHA-256)或者消息摘要算法(message digest algorithm-5, MD5)等方法进行计算.

3) 指纹查找.将得到的指纹针与外存中的指纹进行比对,判断指纹是否重复.

4) 数据压缩.主要针对重复删除过后的数据块进行压缩,本部分为可选部分.

5) 数据存储.将非重复指纹和对应磁盘存入外存或者传输给远端的数据中心,实现重复数据的合并.

在上述 5 个环节中,数据分块、指纹计算、数据压缩都是计算密集型任务,剩下的指纹查找、数据存储则是 I/O 密集型的任务.对于计算密集型任务我们可以通过多核并行计算或者 GPU 设备来进行加速.

但是,随着数据量的增长,指纹查找的环节将会频繁访问 HDD,HDD 的延迟相较于 DRAM 有万倍以上的差距.所以,利用数据的局部性等特点在指纹查找环节减少对慢速 HDD 设备的访问次数是至关重要的.

2.2 勤奋指纹查找算法

类似 DDFS[3] 采用的指纹查找算法为勤奋指纹查找算法,其基本思想是利用 Bloom Filter 进行初次筛选,然后再利用重复数据块中指纹具有连续性的特点,预取同一重复块中所有指纹装载进 Cache,来减少对外存的访问.本文只聚焦讨论指纹查找的效率,所以并未使用 DDFS 上层的文件接口.该算法也将作为与懒惰指纹查找算法的一个基准线来进行比较,勤奋指纹查找算法如下所示.

算法 1. 勤奋指纹查找算法.

输入:指纹数据流;

输出:指纹是否重复并进行存储.

1) 输入指纹进行 Bloom Filter 初次筛选,被判断为不存在于 Bloom Filter 的指纹说明为非重复指纹,存入外存;判断为重复的指纹,进行下面的环节继续查找.

2) 指纹进入 DRAM 的 Cache 进行查找,如果查找该指纹,说明指纹是重复的;否则,进入第 3 环节进行外存查找.

3) 指纹进行外存查找,判断为相同指纹后,预取进 Cache;没有找到相同指纹,则说明该指纹不重复,存入外存.

2.3 懒惰指纹查找算法

为了尽可能地减少对外存的访问,Ma 等人[4] 提出了一种懒惰指纹算法.其基本思想就是将多次指纹查找合并成一次指纹查找,在 Buffer(内存缓冲区)中建立散列桶来暂存待查找指纹,即,请求的指纹在 Cache 中查找不到时,并不马上进行外存查找,而是保存在 Buffer 的散列桶中,当散列桶中积累的待查找指纹数超过阈值时,再将其统一在磁盘中进行查找.懒惰指纹查找的合并查找方式如图 2 所示:

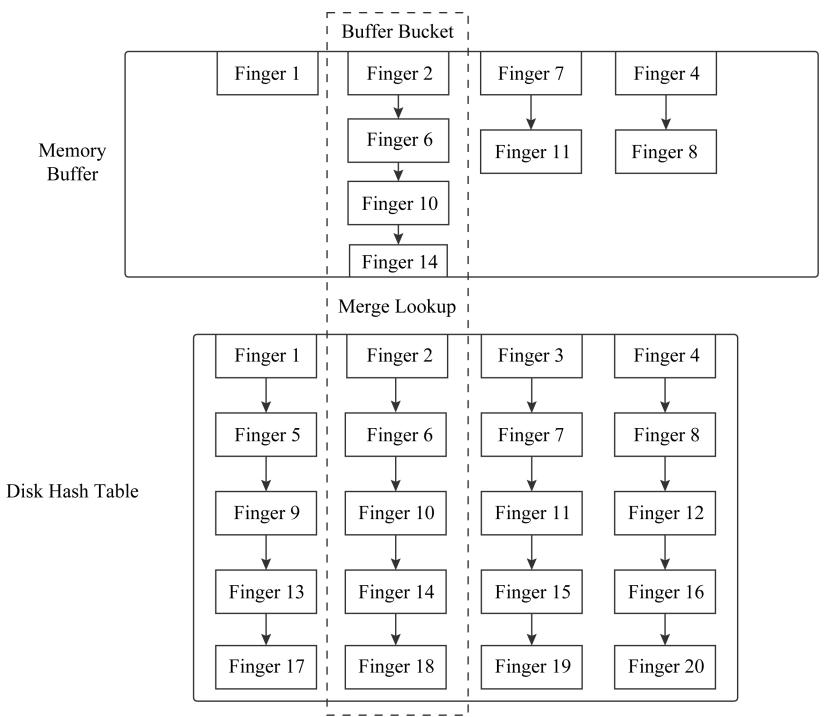


Fig. 2 The lazy deduplication fingerprint group search

图 2 懒惰指纹算法批查找

图 2 中的 Buffer Bucket 的指纹 2,6,10,14 四个指针达到了 Buffer 的阈值,则说明 Buffer 中的指纹存储满了,再将这些指纹一并在磁盘中查找,即 4 次外存访问合并成 1 次外存访问(阈值可进行调整),减少了读取磁盘的次数.

懒惰指纹查找算法这种基于外存访问延迟的算

法会破坏指纹的空间局部性.空间上相邻的指纹通过散列函数映射到不同的桶中,所以 Buffer 中同一个桶中的指纹并不为相邻的指纹.这就导致了 Cache 命中率低的问题,大部分的指纹都不会在 Cache 中找到,这样延迟外存访问的操作又增加了外存访问的次数,对于性能较差的 HDD 来说,降低了指纹查找的性能.因此,懒惰指纹查找算法采用局部性环和标记指纹秩的顺序数来改进懒惰指纹查找算法的 Cache 命中率.局部性环和秩本质上是利用了指纹的空间局部性.例如,在备份系统中,如果每天都进行全量备份,则会出现大量的相同连续数据块.

局部性环的基本思想就是用一个循环链表将在空间上连续的指纹进行连接.在重复数据删除开始时,局部性环为一个空环,然后当每次进行指纹查找,通过 Bloom Filter 筛选之后,认定为重复的指纹会被加入到局部性环当中,来保证局部性环中相近的指纹具有局部性,当局部性环中的指纹超过阈值(阈值一般为 Cache 中指纹的容量),则重新开启一个新的局部性环.

同时,Buffer 中的指纹还保留每个指纹的秩,即指纹到来的顺序.在局部性环中,第 1 个到来的指纹的秩为 0,后面的指纹依次进行递增(0,1,2,…).由于指纹的秩目的是为了保持指纹的相对位置关系,所以被 Bloom Filter 判断为非重复的指纹,仍需要记录其指纹的秩.局部性环和秩的结构如图 3 所示.

在加入局部性环和秩之后,懒惰指纹查找算法的流程如下所示,具体流程图如图 4 所示.

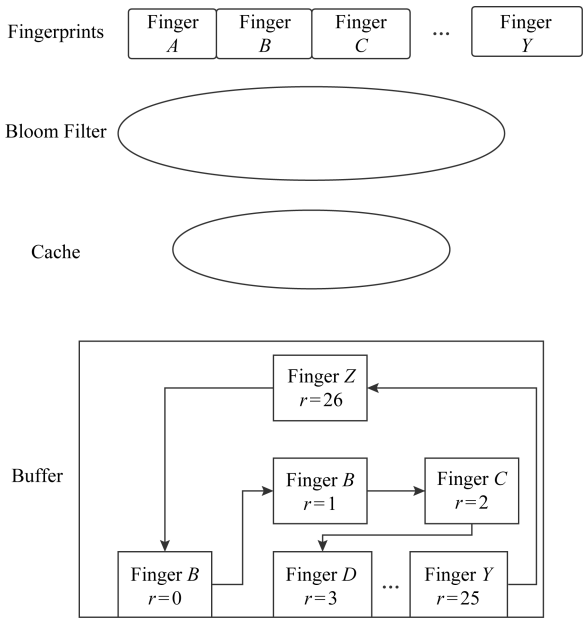


Fig. 3 Local ring and rank

图 3 局部性环和秩示意图

算法 2. 懒惰指纹查找算法.

输入:指纹数据流;

输出:指纹是否重复并进行存储.

1) 指纹输入到 Bloom Filter 进行初次筛选,被判断为不存在于 Bloom Filter 中的指纹说明为非重复指纹,存入磁盘;判断为重复的指纹,进行后续查找.

2) 指纹进行 Pre Lookup(先验查找),利用指纹间存在时间局部性来进行 Cache 查找.

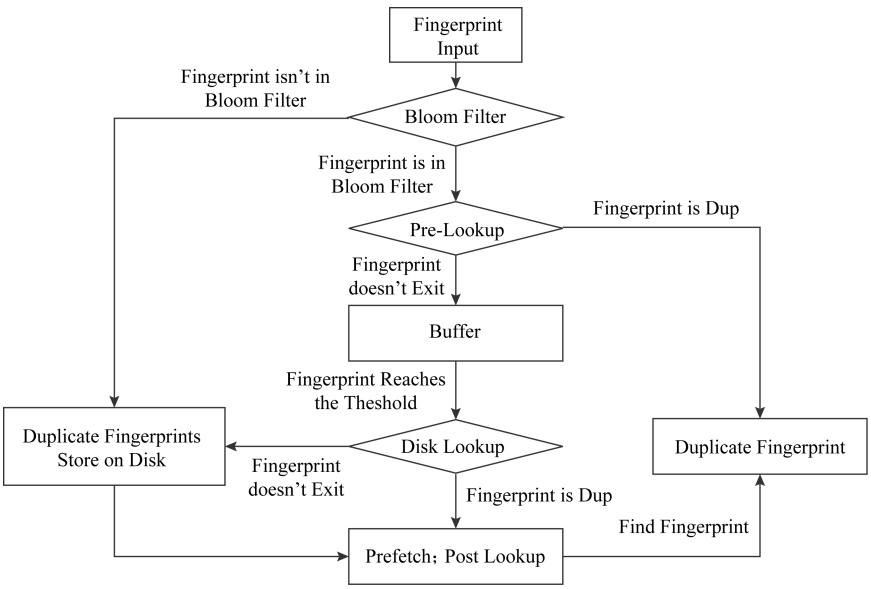


Fig. 4 Lazy deduplication fingerprint search

图 4 懒惰指纹查找算法流程

3) 指纹进入 Buffer 的对应散列桶中,当散列桶的指纹数量达到阈值,则将散列桶的指纹全部进行磁盘进行查找,指纹不存在,则说明非重复指纹;指纹如果存在,则说明是重复指纹.

4) 在磁盘查找为重复指纹时,将磁盘中该指纹周围存在空间局部性的指纹进行预取.然后,再对 Buffer 中同一局部性环的指纹进行 Post Lookup(后验查找)来达到提升 Cache 命中率的目的.懒惰指纹查找环节结束.

其中,Pre Lookup 是在指纹进入 Buffer 前进入 Cache 查找,为的是利用指纹的时间局部性,即上一次被访问的指纹下一次还可能被访问到.而 Post Lookup 是在指纹进入外存进行查找之后,将具有空间局部性的指纹进行预取并在 Cache 中进行查找,目的是利用指纹的空间局部性,让局部性环中的指纹尽可能命中,让 Buffer 中的桶尽可能不满,从而减少外存访问.二者都是通过增加对 Cache 的访问减少外存磁盘的随机访问,从而提高指纹查找的性能.

3 研究的动机和意义

本节将针对第 2 节所讨论的懒惰指纹查找算法和勤奋指纹查找算法,讨论新型存储设备(Optane SSD 和 PM)的低延迟特性对指纹查找算法的影响,同时说明本文对新型存储设备上的重复数据删除指纹查找算法的研究意义.

3.1 指纹查找算法在新型存储设备的性能表现

本文所研究的新型存储设备主要是 Optane SSD 和持久性内存,Optane SSD 和持久性内存相对于传统的 HDD 而言具有高带宽、低延迟的特性,由于我们的研究对象是指纹查找,这里主要讨论 Optane SSD 和持久性内存的延迟,各存储设备的读延迟如表 1 所示:

Table 1 Device Access Delay

表 1 存储设备访问延迟 μs

Storage Device	Latency
HDD	11494
SSD	141
Optane SSD	25
PM	0.80
DRAM	0.13

HDD 的延迟是 Optane SSD 的 400 多倍,是持久性内存的一万倍左右,并且持久性内存的延迟只有 DRAM 的 6～7 倍.这意味着 10 000 次的持久性内存访问的时间才相当于 1 次的 HDD 访问,而 7 次持久性内存的访问代价相当于 1 次 DRAM 访问.这使得当我们外存采用更快存储设备时,懒惰指纹查找算法的优化不再高效,可能勤奋指纹查找算法反而更加高效.

其中,HDD,SSD,Optane SSD 都是用 Fio[20] 以 1 KB 块大小测试的平均延迟,DRAM 则是以 1 KB 块大小利用函数 memcpy 测试的平均延迟,持久性内存是通过 Quartz 进行模拟[21],模拟的延迟是 DRAM 的 7～8 倍.

我们还测试了第 2 节介绍的勤奋指纹查找算法和懒惰指纹查找算法在 HDD、Optane SSD、持久性内存上的性能,指纹查找延迟时间如图 5 所示,其中总数据量为 104 GB,Eager 代表勤奋指纹查找,Lazy 代表懒惰指纹查找.

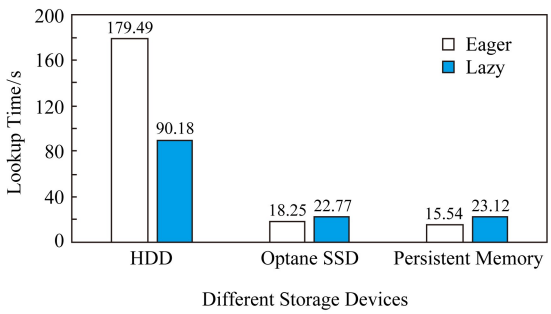


Fig. 5 The fingerprint lookup time

图 5 指纹查找时间

从图 5 可以看出,在慢速的外存设备 HDD 上,懒惰指纹查找算法相较于勤奋指纹查找算法降低了 50%的指纹查找时间,这说明了懒惰指纹查找算法的查找请求延迟策略有效地减少了对外存 HDD 的访问次数,并且利用局部性环以及秩有效地提升了 Cache 命中率,这在较慢速的 HDD 上带来了显著的性能提升.但是,在快速外存介质 Optane SSD 和持久性内存上,懒惰指纹查找的总时间略大于勤奋指纹查找,这说明懒惰指纹查找优化策略在 Optane SSD 和 PM 已经不明显,懒惰指纹查找策略的效率需要在新型存储设备上重新分析以及优化.

3.2 新型存储设备对指纹查找的影响分析

懒惰指纹查找方法通过延迟查找以及批提交来重复利用磁盘的带宽,减少对磁盘的访问,并且利用局部性环和秩保持指纹和指纹之间的时间与空间局部性,利用 Cache 来减少对磁盘的访问,但是 Cache 访问次数变多,相当于增加了对 DRAM 的额外访问.

在外存为 HDD 时,根据图 5 可知,增加 DRAM 访问次数来减少对外存的访问是有效的;但是当外存设备为更加快速的 Optane SSD 或者持久性内存时,大幅度增加 DRAM 的访问来减少对快速外存设备的访问就不再有效.

懒惰指纹查找算法基本可以分为 Bloom Filter, Pre Lookup, Buffer, Disk Lookup, Post Lookup 这 5 个阶段,其中 Bloom Filter 全在 DRAM 中进行查找,并且只和数据集大小有关,其时间与外存和 DRAM 的访问延迟无关.而 Buffer, Disk 都可以认为受外存的访问延迟影响,Pre Lookup, Post Lookup 可以认为受 DRAM 的访问延迟影响.而懒惰指纹查找算法相当于是增加 Pre Lookup, Post Lookup 的 DRAM 访问次数来减少 Buffer, Disk Lookup 的访问次数,而 Pre Lookup 在勤奋指纹查找算法和懒惰指纹查找算法中均存在,代价相同.因此,懒惰指纹查找算法是否优于勤奋指纹查找算法的关键在于 Post Lookup 增加的 DRAM 访问代价是否少于减少的外存访问代价.这也是本文所研究新型存储设备下的指纹查找性能的动机和意义.

4 指纹查找算法建模与优化

本节将把不同存储设备的延迟进行量化,通过数学建模分析外存设备的延迟对指纹查找的影响.并根据建立的模型分析,得出如何在快速的 Optane SSD 设备持久性内存上优化本文第 2 节所提到的懒惰指纹查找算法.本文将对懒惰指纹查找算法与勤奋指纹查找算法进行比较,得到对应的优化结论.

4.1 指纹查找数学建模

本节建模的目的是探究在考虑外存设备延迟等系统参数为自变量的前提下,勤奋指纹查找算法会优于懒惰指纹查找算法的条件.即,在输入数据量相同的情况下,勤奋指纹查找算法的时间会大于懒惰指纹查找算法的时间,可表达为

$$T_{\text{eager}} > T_{\text{lazy}}, \tag{1}$$

其中,T_{eager}为勤奋指纹查找算法的时间,T_{lazy}为懒惰指纹查找算法的时间.

那么对应勤奋指纹查找算法模型,指纹查找总时间表示为

$$T_{\text{eager}} = T_{\text{bloom}}^{\text{eager}} + T_{\text{cache}}^{\text{eager}} + T_{\text{disk}}^{\text{eager}}, \tag{2}$$

其中,$T_{\text{bloom}}^{\text{eager}}$代表指纹查找在 Bloom Filter 上所花费的时间,$T_{\text{cache}}^{\text{eager}}$代表勤奋指纹查找算法在 Cache 查找上所花费的时间,$T_{\text{disk}}^{\text{eager}}$代表勤奋指纹查找算法在外存磁盘上查找所花费的时间.

为了具体讨论外存访问延迟对总体指纹查找的影响,可以将式(2)转换为

$$T_{\text{eager}} = T_{\text{bloom}}^{\text{eager}} + V_{\text{DRAM}} \times F_{\text{Dup}} \times R_{\text{cache}}^{\text{eager}} + V_{\text{disk}} \times F_{\text{disk}}^{\text{eager}}, \tag{3}$$

其中,V_{disk}代表在外存中访问一个指纹的延迟,V_{DRAM}代表在 DRAM 中访问一个指纹的延迟,F_{Dup}代表重复的指纹量,$R_{\text{cache}}^{\text{eager}}$代表勤奋指纹查找算法的 Cache 命中率,$F_{\text{disk}}^{\text{eager}}$代表勤奋指纹查找算法在外存中查找的指纹量.

接下来与采用 Post Lookup, Pre Lookup 的懒惰指纹查找算法进行对比,懒惰指纹查找的顺序是 Bloom Filter, Pre Lookup, Buffer, Disk Lookup, Post Lookup.由于 Pre Lookup, Post Lookup,懒惰指纹查找环节变多,所以除了正常 Cache 命中所带来的 Cache 访问,还有多次遍历局部性环的 Post Lookup 带来 Cache 访问.于是,我们懒惰指纹查找的访问时间为

$$T_{\text{lazy}} = T_{\text{bloom}}^{\text{lazy}} + T_{\text{pre-lookup}} + T_{\text{disk}}^{\text{lazy}} + T_{\text{post-lookup}}^{\text{cache}}, \tag{4}$$

其中,$T_{\text{bloom}}^{\text{lazy}}$代表懒惰算法在 Bloom Filter 所耗费的时间,$T_{\text{pre-lookup}}$代表懒惰算法在进行 Pre Lookup 所耗费的时间,$T_{\text{disk}}^{\text{lazy}}$代表懒惰算法在磁盘上所耗费的时间,$T_{\text{post-lookup}}^{\text{cache}}$表示懒惰算法在 Post Lookup 所耗费的 Cache 时间,而 Post Lookup 的 Cache 时间又可以分为 2 个部分:一个是 Post Lookup 访问局部性环和秩带来的多余 DRAM 访问,另一个是 2 次查找 Cache 的时间.

为了讨论具体的存储介质访问延迟对指纹查找的影响,将存储介质的延迟等参数加入,得到:

$$T_{\text{lazy}} = T_{\text{bloom}}^{\text{lazy}} + V_{\text{DRAM}} \times F_{\text{Dup}} \times R_{\text{first\_cache}} + V_{\text{disk}} \times F_{\text{disk}}^{\text{lazy}} + V_{\text{DRAM}} \times F_{\text{cache}}^{\text{post-lookup}} + V_{\text{DRAM}} \times F_{\text{Dup}} \times R_{\text{second\_cache}}, \tag{5}$$

其中,V_{disk}代表在外存中访问 1 个指纹的延迟,V_{DRAM}代表在 DRAM 中访问 1 个指纹的延迟,F_{Dup}代表重复指纹总数,$R_{\text{first_cache}}$代表 Pre Lookup 的 Cache 命中率,$R_{\text{second_cache}}$代表第 2 次查找的 Cache 命中率,$F_{\text{cache}}^{\text{post-lookup}}$代表 Post Lookup 所带来的额外 Cache 访问指纹数量,$F_{\text{disk}}^{\text{lazy}}$代表懒惰算法在磁盘进行查找的指纹量.

将式(3)和式(5)代入式(1),可得:

$$T_{\text{bloom}} + V_{\text{disk}} \times F_{\text{disk}}^{\text{eager}} + V_{\text{DRAM}} \times F_{\text{Dup}} \times R_{\text{cache}}^{\text{eager}} > T_{\text{bloom}} + V_{\text{DRAM}} \times F_{\text{Dup}} \times R_{\text{first\_cache}} + V_{\text{disk}} \times F_{\text{disk}}^{\text{lazy}} + V_{\text{DRAM}} \times F_{\text{cache}}^{\text{post-lookup}} + V_{\text{DRAM}} \times F_{\text{Dup}} \times R_{\text{second\_cache}}. \tag{6}$$

由于 T_{bloom} 的时间只跟数据集的数量有关,在此比较过程中,勤奋和懒惰的 Bloom Filter 时间相同,所以可以抵消,并且有:

$$R_{\text{cache}}^{\text{lazy}} = R_{\text{first\_cache}} + R_{\text{second\_cache}}, \tag{7}$$

得到最后的比较式为

$$\frac{V_{\text{disk}}}{V_{\text{DRAM}}} > \frac{F_{\text{Dup}}(R_{\text{cache}}^{\text{lazy}} - R_{\text{cache}}^{\text{eager}}) + F_{\text{cache}}^{\text{post-lookup}}}{F_{\text{disk}}^{\text{eager}} - F_{\text{disk}}^{\text{lazy}}}, \tag{8}$$

其中,V_{disk} 和 V_{DRAM} 分别代表外存的延迟和 DRAM 的延迟,F_{Dup} 代表重复指纹的数量,$R_{\text{cache}}^{\text{lazy}}$ 代表懒惰指纹查找算法的 Cache 命中率,$R_{\text{cache}}^{\text{eager}}$ 代表勤奋指纹查找算法的 Cache 命中率,$F_{\text{cache}}^{\text{post-lookup}}$ 代表懒惰指纹查找算法 Post Lookup 的额外指纹访问,$F_{\text{disk}}^{\text{eager}}$ 代表勤奋指纹查找算法在磁盘查找的指纹量,$F_{\text{disk}}^{\text{lazy}}$ 代表懒惰指纹查找算法在磁盘查找的指纹量.

只有当式(8)成立时,才会有式(1)成立,接下来我们来讨论式(8)成立的条件.

4.2 建模分析及优化结论

式(8)的不等式左边可以理解为外存的速度与 DRAM 速度的比值,而公式右边可以理解为采用 2 种方案在外存中的访问数据量的差值和在 Cache 中访问数据量的差值的比值.采用懒惰策略的优化方案是否比勤奋方案好,取决于这 2 个比值的大小关系.

对于 $F_{\text{cache}}^{\text{post-lookup}}$ 而言,它的访问数据量与局部性环的长度为正相关关系,我们用 $L_{\text{local-ring}}$ 表示局部性环的长度,如果它为 1,说明则相当于不采用 Post Lookup,局部性环长度越大,则 $F_{\text{cache}}^{\text{post-lookup}}$ 访问的数据量越大.对应的 Cache 命中率的大小与局部性环也成正相关的关系.而桶的阈值也与合并 I/O 操作的磁盘访问量成负相关关系.

根据以上的相关关系可以针对不同的外存配置场景进行分析,一般的存储设备如 HDD 的延迟都是 DRAM 设备的 80 000 倍左右(SSD 为 1 000 倍左右),由式(8)可知,公式左边设备延迟的比值会远大于公式右边访问数据的差值的比值.所以,采取懒惰指纹查找算法会使得指纹查找时间小于勤奋指纹查找算法.

但是当存储设备采用更加快速的设备如 Optane SSD,甚至是随机写时间接近 DRAM 设备的持久性内存,式(8)的左边值会降低到几十或者极端情况接近 1,这个时候式(8)的左边就不会出现恒大于右边的情况,这时候就要考虑式(8)的右边部分,这里观察式(8)的分子部分,由于懒惰指纹查找算法的 Cache 命中率会低于勤奋指纹查找算法的 Cache 命中率,所以式子 $F_{\text{Dup}} \times (R_{\text{cache}}^{\text{lazy}} - R_{\text{cache}}^{\text{eager}})$ 为负值,后面的项 $F_{\text{cache}}^{\text{post-lookup}}$ 为正,如果 Post Lookup 所带来的局部性环访问量不断增大,会使得式(8)右边增大,造成式(8)不成立,勤奋指纹查找算法延迟会比懒惰数据删除的指纹查找延迟要小,采用勤奋指纹查找算法更合适.

为了让式(8)成立,公式右边尽可能大,即我们指纹查找的 I/O 优化方案有效,可以给出 3 个建议:

1) 可以减少合并 I/O 的散列桶的阈值.

2) 减少局部性环大小,即减少 Post Lookup 所带来的额外 Cache 查找的时间,局部性环存在一个最优值.

3) 当外存的存储介质足够快能够接近 DRAM 速度时,针对于 Cache 命中方面的优化作用变小,这时候采用勤奋指纹查找算法的效果要比懒惰指纹查找算法好.

5 实 验

本节针对第 4 节对重复数据删除指纹查找算法建模分析得到的优化结论进行实验验证,探究这 2 种指纹查找算法在不同存储设备上的性能如何.

5.1 实验基本条件

本文实验平台的基本配置如表 2 所示:

Table 2 The Configuration of the Experimental Platform

表 2 实验平台配置情况

Hardware	Parameters
CPU	16 Intel® Xeon® CPU E5-2609 v4 @ 1.70 GHz
Memory	48 GB DDR4
HDD	WDC WD30EZRZ-00GXCB0 3 TB 5400rpm
SSD	DELL PERC H730 Adp 250 GB
Optane SSD	Intel Optane 900P 480 GB
OS	Linux (kernel version 4.16.0)

实验将使用 Quartz 模拟器[21] 作为持久性内存,设置持久性内存的延迟为 DRAM 的 7 倍左右.为了更好地探究在真实环境下重复数据删除系统性能,实验数据集中将采用与文献[4]中类似的数据集——MIRROR,该数据集来自是浙江大学镜像站① 的镜像文件.数据集中包含从 2010 年开始到

① http://mirrors.zju.edu.cn

2019 年 6 月的 Arch Linux,CentOS,Cygwin,Debian,Deepin,Docker CE 等系统的所有镜像文件,每个系统的版本数量在 5～20.将系统软件的不同版本镜像根据时间顺序依次写入磁盘,可以重现这些系统软件每个版本的归档过程,是一个典型的备份系统动态数据写入过程.数据集的大小、文件数量、文件时间范围、重复数据、重复数据情况如表 3 所示:

Table 3 The Dataset Description

表 3 数据集描述

Workloads	Description
Dataset	Zhejiang University Mirror Station
Total File Number	44 923
Archive Date	2010-02—2019-06
Data Size/GB	104.42
Dup Data Size/GB	58.87
Duplication Rate/%	56.38

由于本文的实验更关注指纹查找的部分,所以在下面的实验过程中,我们将只讨论指纹查找部分的时间,下面实验结果给出的时间都是处理整数据集所有指纹查找的总时间.

5.2 不同存储介质的指纹查找算法各部分时间

下面的实验中,Bloom Filter 的大小为 1 MB,局部性环的大小为 2 048,Cache 和缓存区占用的空间在 256 MB,分块是采用 Rabin-Hash 加上滑动窗口的变长分块算法[22],平均的块大小为 4 KB,而指纹摘要算法利用 SHA-1,即每一个 4KB 块对应的指纹大小为 160 b.外存采用 3 种存储设备:HDD,Optane SSD,PM,PM 是通过 Quartz 模拟器[21]来模拟,其延迟为 DRAM 的 7～8 倍.

为了说明勤奋和懒惰指纹查找算法在 HDD,Optane SSD 和 PM 上的查找总时间以及 2 种指纹查找算法的各部分延迟.实验统计了 2 种指纹查找算法的各部分时间.指纹查找大致可以分为 3 部分时间:外存访问时间、Cache 访问时间和 Bloom Filter 查找时间,实验结果如图 6、表 4 所示.

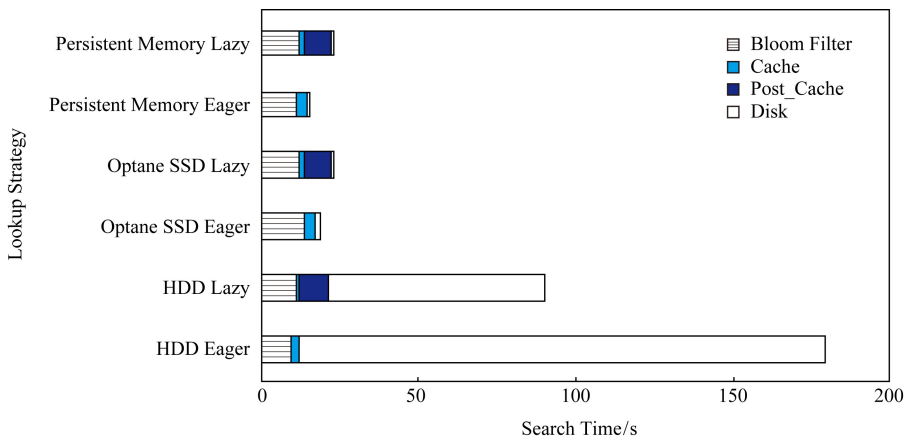


Fig.6 The time of each step in eager and lazy fingerprint search

图 6 勤奋和懒惰指纹查找的各部分时间

Table 4 Specific Time of Fingerprint Search

表 4 指纹查找具体时间 s

Time	HDD	Optane SSD	PM
T^{eager}_{bloom}	8.81	13.68	11.26
T^{eager}_{cache}	3.05	3.57	3.28
T^{eager}_{disk}	167.63	1	0.998
T^{lazy}_{bloom}	10.53	11.74	11.37
$T_{pre\text{-}lookup}$	1.57	1.72	1.73
T^{lazy}_{disk}	69.36	0.62	0.993
$T^{cache}_{post\text{-}lookup}$	8.72	8.69	9.03

从图 6 可以看出,在 Optane SSD 和持久性内存上,勤奋和懒惰指纹查找算法的时间是 HDD 上的勤奋和懒惰指纹查找算法的总时间的 10%左右,并且 HDD 上的指纹查找算法时间占比最大的是在外存的查找时间,懒惰指纹查找算法优于勤奋指纹查找算法也就是在外存查找上节约了时间.但是在 Optane SSD 和持久性内存上,2 种算法的外存查找时间相差不多,而懒惰指纹查找算法增加的 Post Lookup 时间使得懒惰算法性能比勤奋算法的总时间更多.各部分的具体时间如表 4 所示.

为了验证式(8)的正确性,我们将一些参数值代入式(8).式(8)的左边是外存与 DRAM 的延迟比值,当外存分别是 HDD,Optane SSD、持久性内存

时,其延迟与 DRAM 延迟的比值分别为 88 415.38, 192.31,6.15.再讨论式(8)的右边,我们统计得到了处理 MIRROR 数据集产生的总指纹量,因为外存设备快慢本身不会影响处理的指纹量与 Cache 命中率,所以式(8)右边的值为 554.06.当使用 HDD 时, 88 415.38 远大于 554.06,所以懒惰指纹查找算法优于勤奋指纹查找算法;而当使用 Optane SSD 和持久性内存时,192.31,6.15 远小于 554.06,所以勤奋指纹查找算法优于懒惰指纹查找算法,图 6 中的指纹查找总时间也验证了该计算分析的结果.同时也验证了第 4 节的结论 3.

5.3 局部性环大小对整体性能的影响

根据第 4 节的分析,局部性环的阈值对整体指纹查找的性能有着较大的影响,当局部性环的阈值为 1 时,即没有采用 Post Lookup,Cache 命中率会降低,但是节约了查找局部性环的时间.为了更好地探究在不同局部性环大小的影响下,指纹查找时间在不同存储设备上将会如何变化,下面针对局部性环不同的取值进行实验,以下实验都是基于懒惰指纹查找算法来做的.

首先分析理论上能够增加 Cache 命中率的局部性环取值.因为在 Post Lookup 的过程是查找到一个指纹,然后在将这个指纹的局部性环上其他指纹都进行 Cache 查找.所以,局部性环的大小设定为和 Cache 中桶大小相同比较合适(默认设置为 2 048).

本文将局部性环的大小配置为 2,4,8,16,32,64,512,1 024,2 048,其他的配置与 5.2 节相同.查看对于指纹查找时间的影响.实验结果如图 7、图 8 所示.

图 7 为 HDD 上采用不同局部性环大小时的各部分时间,图 8 表示在 Optane SSD,PM 上采用不同局部性环大小时的各部分时间.

从图 7 可知,对于 HDD 而言,随着局部性环的增长,外存访问时间不断减少,Cache 访问时间增加,局部性环的访问时间增长.但是由于在 HDD 上,指纹查找的性能主要受限于外存访问时间,所以局部性环大小越大,指纹查找的性能越好.但是局部性环仍有一个最优值,超过这个最优值,就不能再明显

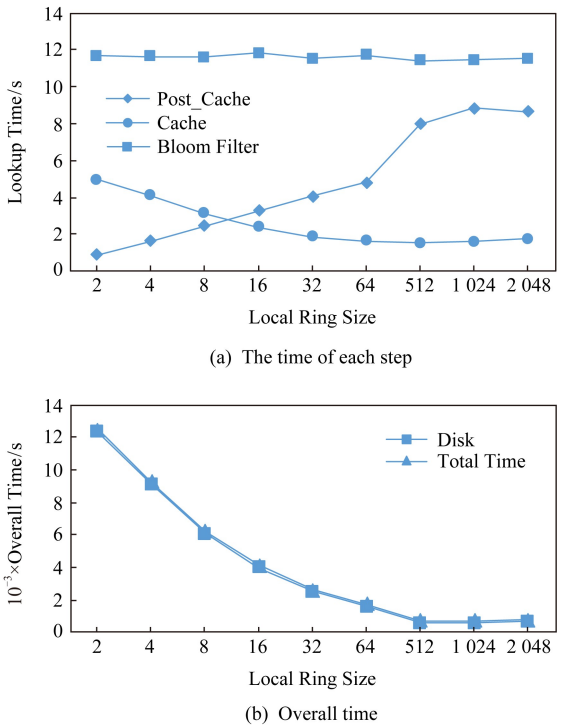


Fig. 7 The impact of local ring size on lazy fingerprint lookup time (HDD)

图 7 局部性环大小对懒惰指纹查找时间的影响(HDD)

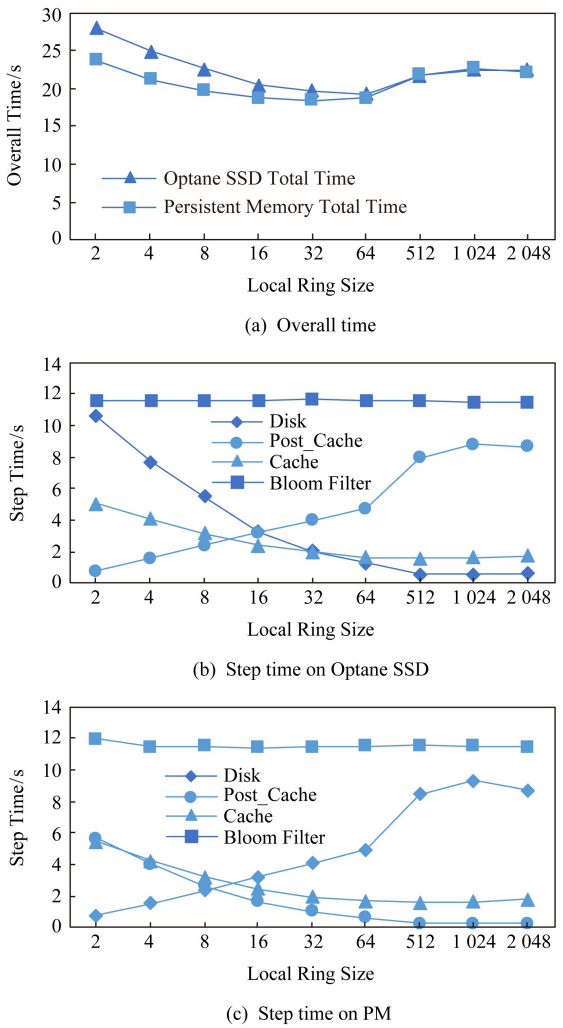


Fig. 8 The impact of local ring size on lazy fingerprint lookup time (Optane SSD,PM)

图 8 局部性环大小对懒惰指纹查找时间的影响(Optane SSD 和 PM)

地减少磁盘访问时间,指纹查找时间基本没有变化.在 HDD 上的局部性环最优值为 512,查找时间为 68.82 s.

而通过图 8(a)可以看出,在采用更快的外存设备 Optane SSD 和持久性内存,当局部性环增大时,懒惰指纹查找的总时间会有一个增加.并且观察 Optane SSD 和持久性内存的局部性环最优值,会发现其最优值相较于 HDD 有明显的前移,从 HDD 的 512,前移到 64,32,这也符合第 4 节的结论 2.为了更加细致地分析在不同设备上局部性环大小对懒惰指纹查找的影响,我们还对指纹查找的不同环节进行了时间变化的记录.从图 8(b)、图 8(c)可以看出,指纹查找的 4 个部分时间,Disk Lookup,Cache_Post,Cache,Bloom Filter 的基本趋势在 Optane SSD 和持久性内存上相近.Bloom Filter 基本没有大幅度的变化,说明局部性环大小并不影响 Bloom Filter 的时间.Post Lookup 的时间都随着局部性环大小的增大而增大,Pre Lookup 的时间和外存查找的时间都随着局部性环大小的增大而减小.在局部性环较大时,Post Lookup 的时间已经成为影响指纹查找总时间的主要因素.

下面说明 3 种存储设备上的最优局部性环大小和最优指纹查找时间,如表 5 所示.

Table 5 The Optimal Local Ring Size and Corresponding Fingerprint Lookup Time

表 5 局部性环最优值和对应指纹查找时间

Parameter	HDD	Optane SSD	PM
Best Local Ring Size	512	64	32
Best Fingerprint Search Time/s	68.82	19.22	18.44

设备速度越快,局部性环大小降低,符合第 4 节结论 1 和 2.

综合上述实验可以得到如下结论:

1) 当外存存储设备采用 Optane SSD 这种新型 SSD 时,这种存储设备的优化针对于重复数据删除系统的指纹查找效率足够,再采用更快的存储设备(例如 PM),性能提升并不明显.

2) 当外存存储设备的性能越来越好时,最优局部性环的值会越来越小.

3) 当使用 Optane SSD 或者持久型内存时,甚至不采用局部性环的相关优化,采用勤奋指纹查找算法的效率更高.

6 结 语

本文针对新型存储设备上的重复数据删除指纹查找算法进行研究.首先介绍 2 种典型的指纹查找算法:一种是传统的勤奋指纹查找算法,另一种是针对于 HDD 等慢速设备优化外存访问的懒惰指纹查找算法.然后将外存存储介质的延迟作为变量,对 2 种指纹查找算法进行建模,分析并得到 3 点的优化结论:降低 Buffer 的大小、降低局部性环的大小、在高性能设备上勤奋重复数据删除算法性能优于懒惰重复数据删除算法.在实际数据集上的实验结果验证了模型的正确性以及 3 点优化结论.

参 考 文 献

[1] Meyer D T, Bolosky WJ. A study of practical deduplication [J]. ACM Transactions on Storage, 2012, 7(4): No.14

[2] Shilane P, Huang M, Wallace G, et al. WAN-optimized replication of backup datasets using stream-informed delta compression [J]. ACM Transactions on Storage, 2012, 8(4): No.13

[3] Zhu B, Li Kai, Patterson R H. Avoiding the disk bottleneck in the data domain deduplication file system [C] //Proc of the 6th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2008: 1-14

[4] Ma Jingwei, Stones R J, Ma Yuxiang et al. Lazy exact deduplication [J]. ACM Transactions on Storage, 2017, 13(2): No.11

[5] Volos H, Magalhaes G, Cherkasova L, et al. Quartz: A lightweight performance emulator for persistent memory software [C] //Proc of the 16th Annual Middleware Conf. New York: ACM, 2019: 37-49

[6] Huffman D A. A method for the construction of minimum-redundancy codes [J]. Proceedings of the IRE, 1952, 40(9): 1098-1101

[7] Ziv J, Lempel A. A universal algorithm for sequential data compression [J]. IEEE Transactions on Information Theory, 1977, 23(3): 337-343

[8] Xia Wen, Jiang Hong, Feng Dan, et al. A comprehensive study of the past, present, and future of data deduplication [J]. Proceedings of the IEEE, 2016, 104(9): 1681-1710

[9] Bloom B H. Space/time trade-offs in Hash coding with allowable errors [J]. Communications of the ACM, 1970, 13(7): 422-426

[10] Bhagwat D, Eshghi K, Long D D, et al. Extreme binning: Scalable, parallel deduplication for chunk-based file backup [C] //Proc of 2009 IEEE Int Symp on Modeling, Analysis & Simulation of Computer and Telecommunication Systems. Piscataway, NJ: IEEE, 2009: 1-9

[11] Xia Wen, Jiang Hong, Feng Dan, et al. SiLo: A similarity-locality based near-exact deduplication scheme with low RAM overhead and high throughput [C] //Proc of 2011 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2011: 26-30

[12] SNIA. NVM programming model (NPM) SNIA technical position [OL]. Intel, [2019-11-09]. https://www.snia.org/sites/default/files/NVMProgrammingModel_v1.pdf

[13] Chen Shimin, Jin Qin. Persistent b+-trees in non-volatile main memory [J]. Proceedings of the VLDB Endowment, 2015, 8(7): 786-797

[14] Zuo Pengfei, Hua Yu, Wu Jie. Write-optimized and high-performance hashing index scheme for persistent memory [C] //Proc of the 13th Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2018: 461-476

[15] Xu Jian, Swanson S. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories [C]//Proc of the 14th Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2016: 323-338

[16] Meister D, Brinkmann A. dedupV1: Improving deduplication throughput using solid state drives (SSD) [C]//Proc of the 26th IEEE Symp on Mass Storage Systems and Technologies. Piscataway, NJ: IEEE, 2010: 1-6

[17] Debnath B K, Sengupta S, Li Jin. ChunkStash: Speeding up inline storage deduplication using flash memory [C] //Proc of USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2010: 1-16

[18] Lu Guanlin, Nam Y, David H, et al. BloomStore: Bloom-filter based memory-efficient key-value store for indexing of data deduplication on flash [C]//Proc of the 26th IEEE Symp on Mass Storage Systems and Technologies. Piscataway, NJ: IEEE, 2012: 1-11

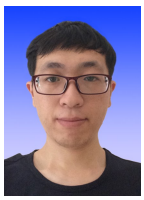
[19] Wang Chundong, Wei Qingsong, Yang Jun et al. Nv-dedup: High-performance inline deduplication for non-volatile memory [J]. IEEE Transactions on Computers, 2017, 67(5): 658-671

[20] Axboe J. Fio [OL]. [2019-11-10]. https://github.com/axboe/fio

[21] Volos H. Quartz: A DRAM-based performance emulator for NVM [OL]. Hewlett Packard Labs, [2019-11-10]. https://github.com/HewlettPackard/quartz

[22] Rabin M O. Efficient dispersal of information for security, load balancing, and fault tolerance [J]. Journal of the ACM, 1989, 36(2): 335-348

He Kewen, born in 1997. Master candidate. His main research interests include deduplication, persistent memory, key-value store.



Zhang Jiachen, born in 1994. Master candidate. His main research interests include storage virtualization and persistent memory.

Liu Xiaoguang, born in 1974. PhD, professor and PhD supervisor. Senior member of CCF. His main research interests include parallel computing, storage systems, and search engines.

Wang Gang, born in 1974. PhD, professor and PhD supervisor. His main research interests include storage systems and parallel computing.