DOI：10.7652/xjtuxb201810004

云环境下应用感知的动态重复数据删除机制

贺秦禄[1]，边根庆[1]，邵必林[2]，贾雷刚[1]
（1.西安建筑科技大学信息与控制工程学院，710055，西安；2.西安建筑科技大学管理学院，710055，西安）

摘要：针对传统在线/离线重删对云存储系统中重删效率不高的问题，采用混合重复数据删除(Hy-Dedup)机制，通过融合在线和离线两种方式进行有效的数据重删。该方案在线重删阶段根据负载类型对指纹索引进行聚类分组，设置不同重删阈值来评估数据流的空间局部一致性，提高了缓存命中率；离线重删阶段采用延迟敏感的方法，对在线阶段缓存没有命中的重复块进行精确重删。通过这种混合方式在保持系统的 I/O 性能和吞吐量的前提下，显著减少了写入云存储的重复数据量。实验结果表明，与 iDedup 机制相比，Hy-Dedup 机制可将在线重删率提高 35.9%，磁盘空间需求减少 41.36%，并且能够在云存储系统中实现高准确率的重删，提升重删效率，节省存储空间。

关键词：在线重删；离线重删；缓存；云存储
中图分类号：TP393.1 **文献标志码**：A **文章编号**：0253-987X(2018)10-0024-07

A Dynamic Deduplication Method with Application-Aware in Cloud Environment

HE Qinlu[1], BIAN Genqing[1], SHAO Bilin[2], JIA Leigang[1]
(1. School of Information and Control Engineering, Xi'an University of Architecture and Technology, Xi'an 710055, China;
2. School of Management, Xi'an University of Architecture and Technology, Xi'an 710055, China)

Abstract: A hybrid deduplication method (Hy-Dedup) is adopted to solve the problem that the deduplication efficiency in the cloud storage system is not high for traditional mode online/offline deduplication, and the method performs effective data deduplication by combining online and offline modes. This method clusters fingerprint indices according the type of loads in online deduplication stage by adopting the fingerprint caching technology. The temporal local consistency of the duplicated data in data stream is estimated and the spatial local consistency is evaluated by setting different deduplication thresholds to reduce the disk fragments. The problem that the cache cannot be hit because lack of local consistency in the offline deduplication phase will be solved. The duplicated data is significantly reduced by this method while maintaining the I/O performance and the system throughput. Experimental results and a comparison with iDedup show that Hy-Dedup improves the online deduplication ratio by up to 35.9% and the disk capacity requirement reduces by 41.36%. It is concluded that the proposed method can achieve high-deciding deduplication in the cloud storage system, improve deduplication efficiency, and save storage space.

收稿日期：2018-01-27。 作者简介：贺秦禄(1982—)，男，博士，讲师；边根庆(通信作者)，男，副教授，硕士生导师。 基金项目：国家自然科学基金资助项目(61672416)；陕西省自然科学基础研究计划资助项目(2018JM6105)；西安建筑科技大学人才基金资助项目(RC1707)。
网络出版时间：2018-08-16 网络出版地址：http://kns.cnki.net/kcms/detail/61.1069.T.20180816.1705.002.html

Keywords：online deduplication；offline deduplication；cache；cloud storage

重复数据删除技术是将数据分解成块，使用数据块的哈希指纹来检测和删除重复块以节省存储空间。重删技术在备份及归档系统中得到广泛应用[1-2]。根据重复数据删除执行的方式可分为两类：在线重复数据删除[3]和离线重复数据删除[4-5]。前者在 I/O 写请求路径上执行重复数据删除，以立即检测和删除重复数据，而后者在后台进行重删，以避免对 I/O 性能的影响。然而，这两种方式都存在效率不高的问题。

对于在线重复数据删除，指纹查找是主要的性能瓶颈，因为指纹索引表通常超过内存的大小，只能部分存储在内存，剩余部分存储在磁盘，虽然备份系统可以容忍基于磁盘指纹查找的延迟，但是在云存储系统上可以采用缓存来解决应用程序的延迟问题。iDedup[6]和 POD[7]机制进行选择性在线重删。iDedup 机制实现了在不同阈值下 I/O 性能、空间效率和资源消耗之间的均衡；POD 机制采用恒定阈值长度为 3 进行分析验证。本文根据数据流的特征动态调整阈值，以提高系统重删的性能。

离线重复数据删除技术存在两个缺点：第一，在重删之前将所有的数据块写入磁盘，这对于采用 SSD 作为缓存介质的系统，重删将影响 SSD 的使用寿命[8]；第二，当需要进行大量的重复数据删除时，离线重删技术存在着与其他应用竞争系统资源的问题，影响整个系统的性能。AA-Dedupe 机制[9]是应用源端感知的重删方法，基于不同的分块和哈希方案将整个索引划分为更小的子集进行重删指纹匹配。与 AA-Dedupe 机制不同，本文机制根据应用类型将整个索引分为不同的子集进行数据重删，并利用空间位置来解决读取放大问题。

将在线和离线处理融合在一起可以使每个阶段相互补充并解决单独处理的局限性。在线阶段可以删除具有良好时间局部性的重复块，而将其余重复块保留到离线阶段进行精确重删[10]。在这种方式下，负载的指纹缓存扮演着重要的角色，在线阶段高效的缓存机制能够识别大量的重复块，减少写入磁盘的数据量[11]，从而提高 I/O 性能并减少离线重删阶段的工作量。

基于此，本文结合在线重删和离线重删各自的特点，提出了一种在线/离线混合重复数据删除(Hy-Dedup)机制，可以实现云环境下数据重删后良好的读写性能，获得 I/O 效率和存储空间之间的平衡。在线重删阶段，采用基于应用感知的指纹索引聚类分组方法来区分不同数据流局部一致性，通过周期性动态调整不同数据流的阈值，可以更好地发挥局部一致性处理数据流的优势。该机制显著地提高了在线重删的缓存命中效率，并减少离线重删阶段的负载量，实现高效能的在线重删率；离线重删阶段仅处理在线重删阶段缓存未命中的重复块，这些数据块相对数量较小。与传统的离线重删机制相比，本文机制通过高效的在线重删大大降低了对存储空间的需求和对系统资源的竞争。

1　Hy-Dedup 机制的设计

传统的在线或离线处理重复数据删除难以满足云存储系统对的重删率和延迟要求。本文提出了 Hy-Dedup 机制，结合在线重删和离线重删的两种方式，能够实现系统 I/O 性能和重删率之间的平衡，这对云存储系统的在线重删是至关重要的。

1.1　Hy-Dedup 机制系统架构

图 1 给出了 Hy-Dedup 机制系统架构。Hy-Dedup 机制部署在存储节点上，数据流的 I/O 读写操作通过与文件系统接口进行交互，可以兼容任何类型存储系统并进行性能优化。此外，与全文重复数据删除的 iDedup[6]和 POD[7]机制相比，Hy-Dedup 机制又独立于文件系统，具有较好的灵活性和可扩展性。Hy-Dedup 机制也可以部署在虚拟机管理程序中，例如 VDI、XEN 等。虚拟机镜像中存

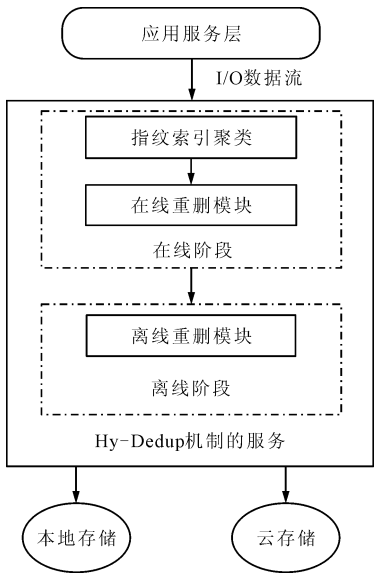


图 1　Hy-Dedup 机制系统架构

在大量重复数据块，对于在同一主机上运行的多个容器，可以在块设备上部署 Hy-Dedup 机制，进行重复数据删除。

在线重删阶段首先根据应用类型对写数据流中的数据块的指纹索引进行聚类分组，之后定期评估不同数据流的时间局部性，优先考虑将缓存分配给具有良好时间局部性的数据流。通过这样的方式，在线重删阶段可以有效地减少写入数据量，实时更新指纹索引，保证数据一致性，减少存储和查询哈希索引表所引起的内存和 CPU 开销，同时减少离线重删阶段的工作量。其次，离线重删阶段仅处理在线重删阶段中未检测出的重复块，采用延迟触发的思想进行设计，当重复数据块达到设定的阈值时才会进行离线重删，根据重删结果置换内存中相应的索引项。与单纯离线处理的重删系统相比，高效的在线重删流程大大降低了对存储容量和系统资源的占用。此外，Hy-Dedup 机制根据数据流特征动态调整阈值的策略，不仅可以保证重删率，同时也减少了数据碎片的产生，不影响后续数据恢复的性能。

1.2 在线重复数据删除阶段

在线重删阶段，Hy-Dedup 机制维护一个数据块指纹缓存和物理块地址(PBA)映射，以避免指纹表检索的磁盘瓶颈，同时也建立数据块的逻辑块地址(LBA)和 PBA 之间的映射关系。此外，具有相同内容和局部一致性的 I/O 请求仅在映射表中记录一项，这将减少实际应用中的内存开销。首先，计算数据流中的数据块哈希值，在重删引擎的缓存中查询块指纹进行匹配；其次，数据流局部一致性估计器负责监视和估计来自不同应用数据流的时间和空间局部一致性。时间局部一致性估计用于优化缓存的命中率，而空间局部一致性用于估计动态调整数据流的重删阈值以减少磁盘碎片。

当缓存命中了输入数据块指纹、这个数据块不存在于映射表时，则将创建 LBA 和对应的 PBA 项，并将其添加到 LBA 映射表中；若缓存中没有命中数据块指纹，则将数据块直接写入底层存储，该数据块相关联的数据块指纹、LBA 和 PBA 映射以及引用计数的对应元数据在 3 个表中同步更新。缓存未命中的数据块将在离线阶段再次进行重删。

1.2.1 *数据流的时间局部一致性估计* 重复块的时间局部一致性特征表示在数据流中重复块可能达到系统的时间。良好的时间局部一致性表示重复块通常彼此接近，而弱局部一致性表示重复块通常彼此远离或数据流中几乎不存在重复块。为了计算重复块的时间局部一致性，本文引入局部重复集合的度量(LDS，定义为 L)表示数据流在给定时间之前多个连续数据块中重复块的数量，根据缓存分配情况，通过 L 预测在未来即将到达数据流中重复块。常见的方法是使用历史 L 来进行预测，为了从数据流中获得历史 L，通常是对数据流中所有的指纹及其在估计间隔内出现的次数进行计数，然而，记录所有的数据指纹将产生很高的内存开销。为了解决这个问题，数据流局部一致性估计器使用存储采样算法[12]从数据流中采样指纹，然后使用不可见的估计算法[13]对样本进行 L 估计。

假设系统处理 M 个数据流，估计间隔大小为 a，时间局部一致性估计的目标是从最近 m 个写请求中收集 k 个指纹样本，每个数据流基于指纹样本计算其对应的 L。在采样之后，用 N_i 表示在估计区间内来自 i 数据流写请求的数量，估计在数据流中最后 t 个写请求中可实际写入的数量。通过空间局部一致性估算 L_i 的过程如下：首先定义指纹采样函数(FFH)设为 F，用来估算数据流 i 的 L_i；用 $H_{i'}$ 表示样本的采样结果，H 表示数据流 i 在整个估计间隔的 F，计算任意一项在特定时间间隔内被采样的概率。采样数据块预期结果 $H_{i'}$ 可以通过 $H_{i'}=MH/(T+H)$ 计算得到，之后通过计算 H_i 和 $H_{i'}$ 之间的距离平均值得到 H，进而代入计算就可以得到数据流的 L_i。对于在估计间隔期间只有少量写请求的情况，不需要通过上述过程来估计 L，只需要将 L 设置为一个较小的定值即可。

1.2.2 *数据流的空间局部一致性估计* 为解决数据流重删后引起数据碎片问题，Hy-Dedup 机制定义两个变量 V_w 和 V_r。V_w 记录连续重复块最大长度值出现的次数，V_r 记录顺序读的长度值出现的次数。例如，$V_w[3]=100$，表示存在长度为 3 的 100 个连续重复块；如果 $V_r[3]=100$，则表示有 100 次长度为 3 顺序读请求。初始阶段阈值定义为 z_0，初值设定为 20，两个变量在请求到达时收集数据。当阈值更新被触发时，阈值 Z 的计算公式为

$$Z=\sum(RL_d+(1-r)L_r)/N \tag{1}$$

式中：L_d 和 L_r 分别是重复序列的平均长度和平均读取长度；Z 是读和写延迟的平衡点；r 是所有请求之间的写比率；N 为估算间隔。L_d 和 L_r 分别根据在 V_w 和 V_r 中收集的数据进行计算，为了应对每个数据流重复模式的变化，自上一个阈值更新以来重删率减少超过 50%时，两个变量将全被重置为 0。

1.3　离线重复数据删除阶段

在离线重删阶段，通过匹配磁盘上存储的指纹索引表进行数据块重删。与传统离线重删不同，Hy-Dedup 机制以延迟方式运行，只有当重复数据块达到设定的阈值时，才会触发重删操作。这种策略为读请求节省空间，对于延迟敏感云存储负载有利。首先，对于在线阶段没有缓存命中的数据块在索引表中创建一个新条目，属性包含数据块 ID、哈希值和块大小，当新增加的条目数达到预设值时，离线重删模块会生成一个名为 Targets[]的列表，同时，离线重删模块会将每个数据块的 LBA 和最后访问时间戳(LAT)插入到其相应的条目中。其次，根据重删优先级来重新分配最近访问的数据块的次序，当缓存区存储的指纹数大于设定的阈值时，重删模块才允许完全遍历 Targets[]中所有的目标；否则仅处理 Targets[]中的一半项即可。最后，按照哈希值的匹配进行数据重删。

2　Hy-Dedup 机制的实验分析

首先描述实验参数和方法，然后通过实验来验证 Hy-Dedup 机制的性能。

2.1　实验环境

在 ADMAD 平台[14]的基础上实现了 Hy-Dedup 机制原型。Hy-Dedup 机制是云存储环境中应用感知的重复数据删除机制，通过聚类应用中的数据指纹索引以实现高重删效率，它适用于不同的分块算法，如变长分块(CDC)、静态分块(SC)、整个文件分块(WFC)以及不同类型文件和应用的哈希函数，如 SHA-1、MD5 和 Rabin 哈希。Hy-Dedup 机制是在 Lessfs 系统[15]上进行搭建，在实验中使用真实数据集重放来验证 Hy-Dedup 机制的缓存命中率和重删率。具体实验环境由 10 个节点组成，每个节点配置信息见表 1。

表 1　实验环境配置信息

组件	描述
CPU	Intel(R) Xeon(R) E5502 @ 1.87 GHz (Intel(R) Xeon(R) E5-2697 V2 @ 2.7 GHz)
内存	16 GB(32 GB) DDR SDRAM
硬盘	ATA Hitachi HTS54501,150 GB * 2
SSD	Intel DC S3700 200 GB
网络	1 GB 以太网卡
操作系统	Ubuntu 12.04.3 LTS,Linux 3.8.0-29(x86_64)

表 2 给出了 3 个实际应用数据集(VM 镜像[16]、Firefox 安装镜像[17]和 Linux 内核源码[18])的存储数据大小和当数据块大小为 16 KB 时数据集的内部重复率。

表 2　3 种数据集的数据特征

应用类型	存储数据大小/GB	内部重复率/%
VM 镜像[16]	332	23.7
Firefox 安装镜像[17]	556	42.0
Linux 内核源码[18]	657	84.7

表 3 给出了这 3 种应用数据之间的冗余特性，每个单元格表示共享重复率，由垂直应用和水平应用之间的冗余数据除以水平应用的总数据计算得到。如果单元格中的两个应用相同，则表示应用程序中的数据冗余。从表 3 可以看出，任何两种不同的应用之间，只有少于 1%数据块是重复块。因此，当这 3 种负载根据请求到达时间的顺序进行排序和合并时，生成的数据集与原始数据集具有相同的 I/O 模式，这表明对于从相同数据源生成的数据集，设置内容随机重复率为 0～36%，将代表用户使用的典型负载类型。

表 3　3 种不同应用的数据局部一致性

应用类型	共享重复率/%		
	VDI	Firefox	Linux 内核
VM 镜像[16]	23.7	0.025	0.019
Firefox 安装镜像[17]	0.025	42.0	0
Linux 内核源码[18]	0.019	0	84.7

通过上述结果表明，不同应用之间共享的冗余数据量可以忽略不计，原因是不同应用具有不同的数据内容和数据格式，因此可以将相同应用的指纹索引有效地组合在一起，并根据应用类型将整个指纹索引进行聚类分组。

实验的整体思路如下：在初始状态下，后端存储是基于 HDD 的 RAID10，包括具有 128 MB 缓存和 16 KB 数据块大小的 4 个 HDD。通过在 iDedup 和 Hy-Dedup 机制中重放 3 个数据集来验证系统吞吐量，同时通过改变缓存和负载的局部性比率来验证重删机制的性能敏感度。

2.2　写性能对比

在由 4 个 HDD 组成的 RAID10 系统上设置不同大小的缓存分别进行实验。图 2 给出了 Hy-Dedup、iDedup 和 POD 机制随着缓存的增加对系统写吞吐量的影响。与 iDedup 和 POD 机制相比，Hy-Dedup 机制将写吞吐量最高提高了 6.9 倍，写

吞量平均提高了 3.2 倍，这表明 Hy-Dedup 机制的写吞吐量对缓存不太敏感，由于 Hy-Dedup 机制所需的索引缓存小，在云储存环境下应用 Hy-Dedup 机制时将会获得更好的可扩展性。O 分析原因有两个方面：①iDedup 和 POD 机制写吞吐量由指纹索引缓存中数据冗余度决定，虽然 iDedup 机制将相同分块方案的指纹分组在一起，但索引缓存仍然太大，无法存储在内存中，后端存储和内存之间的页面置换将影响系统性能；②由于 Hy-Dedup 机制只将应用聚类分组后的指纹索引加载到内存中，重删时所需的缓存和查询大小都减少了，即使使用较小的索引缓存，写吞吐量也呈线性增长。

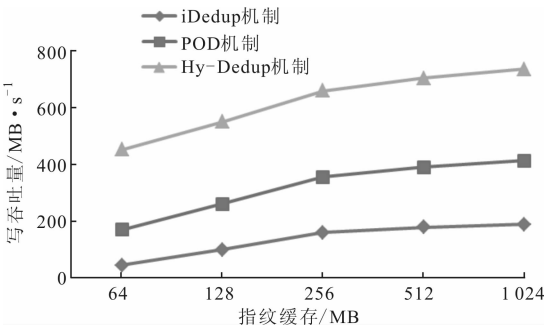


图 2 3 种方法写吞吐量随缓存的变化

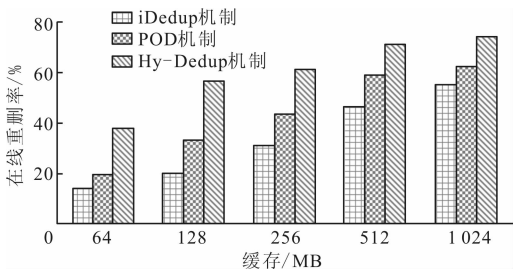
2.3 在线重删缓存命中率比较

图 3 给出了 Hy-Dedup、iDedup 和 POD 机制在线重删率与缓存的关系。对于每种类型数据流都存在不同的缓存替换策略，文献[5]实验分析得出，LRU 算法是 iDedup 和 POD 机制较好的缓存替换方法，因此 Hy-Dedup 机制也使用 LRU 算法作为缓存替换方法，方便进行实验对比。对于负载 A、B 和 C，具有良好局部性分部(G)和弱局部性能部分(N)之间的数据大小比例分别为 3∶1、1∶1和 1∶3。

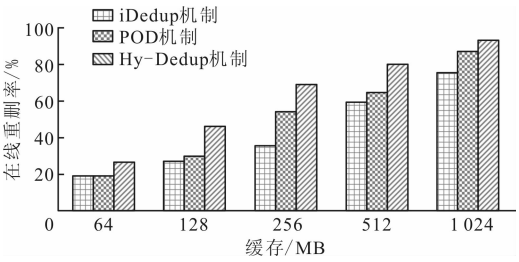
如图 3 所示，随着局部一致性差的负载量增加，iDedup、POD 和 Hy-Dedup 机制之间的重删率的差距增加，Hy-Dedup 机制显著地提高在线重删率。对于负载 A，Hy-Dedup 机制比 POD 和 iDedup 机制重删率提高了 7.14%～22.18%；对于负载 B，提高了 8.32%～23.9%；对于负载 C，提高了 16.7%～35.9%。随着非本地负载量的增加(从负载 A 到 C)，负载的弱局部一致性导致在线重删率较低。Hy-Dedup 机制利用局部一致性估计算法，并结合重复块指纹索引聚类动态分配缓存，可以提高云存储中多个应用重删的整体缓存命中率。

2.4 磁盘空间需求分析

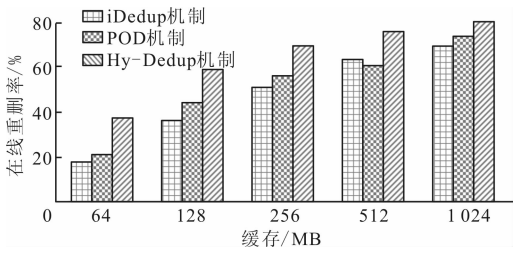
如图 4 所示，对比 Hy-Dedup、AA-Dedupe 机制



(a)负载 A(G∶N=3∶1)



(b)负载 B(G∶N=2∶2)



(c)负载 C(G∶N=1∶3)

图 3 3 种机制重删率随缓存的变化

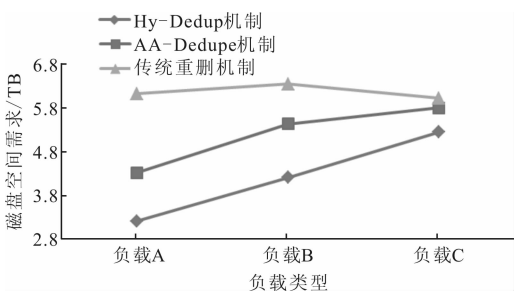


图 4 3 种机制磁盘空间需求随不同负载类型的变化

和传统重删机制在重删过程中对磁盘空间的需求。在实验中，Hy-Dedup 机制在线重删的缓存设置为 200 MB，使用 LRU 算法作为缓存替换方法。

由图 4 可以看出，针对不同负载 Hy-Dedup 机制明显降低了磁盘空间需求，与 AA-Dedupe 机制相比，磁盘空间需求分别下降了 41.36%、27.88%和 11.95%。结果表明，负载的局部性越好，通过 Hy-Dedup 机制两阶段的重删可以发现更多的重复块，同时减少数据块的写入量。Hy-Dedup 机制这种混合重删架构，只需要维护 200 MB 的指纹索引缓存

就可以减少数百 GB 的数据写入。

2.5　系统开销分析

2.5.1　计算的开销　Hy-Dedup 机制的计算开销主要来自时间局部性估算，包含两部分：指纹生成函数 Y 的时间和时间局部性估计算法执行时间。为了计算 Y，需扫描采样缓冲区并计算指纹发生的情况，时间复杂度是 $O(n)$，其中 n 是样本数。如图 5 所示，当采样率为 20%时 1 000 个数据块的估计间隔计算时间少于 2 ms；对于每个估计间隔，每个数据流时间局部性估计大约需要 17 ms。该进程在后台执行并且不影响系统写性能，因此这样的计算开销是可接受的。

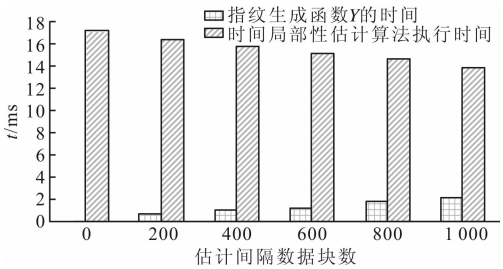


图 5　Hy-Dedup 机制计算开销

2.5.2　内存开销分析　Hy-Dedup 机制的内存开销主要来自样本采样缓存。利用估算间隔 N 和采样率 p，内存开销 q 计算如下

$$q = Np(f_{\text{size}} + c_{\text{size}}) \tag{2}$$

式中 f_{size} 和 c_{size} 分别是用于存储指纹和发生计数的内存开销。例如，当缓存为 200 MB 时，将有大约 2.8 MB 的缓存项。对于 20%的采样率，即使选择较大的估计间隔因子（例如负载 C，间隔因子为 0.6），内存开销只有 4.6 MB（缓存为 2.89%）。在实验中，对于具有较好的时间局部一致性的数据流，可以将估计间隔因子设置为较小的值，内存开销减少很多（例如负载 A 为 2.13 MB，负载 B 为 2.87 MB），与具有 200 MB 缓存的两个负载的在线重删率（18.36%～22.51%）相比，Hy-Dedup 机制的内存开销是完全可以接受的。

2.6　离线重删阶段的效率

对比 Hy-Dedup、AA-Dedupe 和 iDedup 机制不同后端存储的写吞吐量，在由 4 个 HDD（HDD-RAID10）、4 个 HDD（HDD-RAID5）和 4 个 SSD（SSD-RAID5）分别组成的 RAID5 和 RAID10 系统上进行实验，结果如图 6 所示。

从图 6 可以看出，HDD-RAID5 写吞吐量高于 HDD-RAID10。原因是 4 个 HDD 组成的 RAID5 系统比 4 个 HDD 组成的 RAID10 系统具有更好的

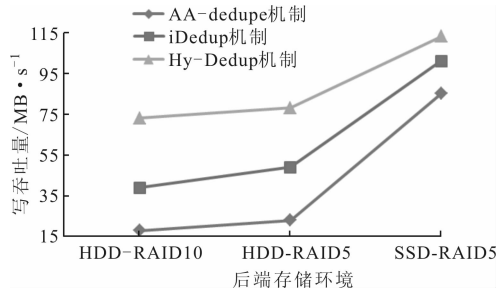


图 6　写吞吐量随不同后端存储环境的变化

访问并行性，因此可提供更高的写吞吐量。SSD-RAID5 的写吞吐量是情况最好的，其原因为如下两点：①SSD 的性能优于 HDD，从而具有较高的性能；②内存和 SSD 之间的置换操作比内存和 HDD 之间的置换操作快得多，从而提高了索引存储和查询的效率，文献[4]的研究结果已经证明了这一结论。另一方面，与 iDedup、AA-Dedupe 机制相比，Hy-Dedup 机制在 HDD-RAID10、HDD-RAID5 和 SSD-RAID5 上的性能高出 1.8～3.6 倍。

3　结　论

本文提出了一种混合重删机制（Hy-Dedup），根据应用负载类型对指纹索引进行聚类分组，估计数据流中重复块的局部一致性，使用动态局部一致性估计算法来提高在线缓存命中率，并将没有被缓存命中的相对数量较少的重复块在离线重删阶段进行处理。通过这种方式，Hy-Dedup 机制能够在云存储系统中实现精确的重删，提升系统 I/O 性能，节省存储空间。与现有的在线重删机制相比，Hy-Dedup 机制显著地提高在线缓存效率，从而实现了高效在线重删率。未来还需要在以下两个方面展开研究：设计具有高查询效率的兼容性索引结构；利用机器学习实现自适应调整控制参数的策略。

参考文献：

[1]　王龙翔，张兴军，朱国峰，等. 重复数据删除中的无向图遍历分组预测方法 [J]. 西安交通大学学报，2013，47(10)：51-56.
WANG Longxiang，ZHANG Xingjun，ZHU Guofeng，et al. A grouping prediction method based on undirected graph traversal in de-duplication system [J]. Journal of Xi'an Jiaotong University，2013，47(10)：51-56.

[2]　NISHA T R，ABIRAMI S，MANOHAR E. Experimental study on chunking algorithms of data deduplication system on large scale data [C]// Proceedings of

the International Conference on Soft Computing Systems, Advances in Intelligent Systems and Computing. Berlin, Germany: Springer, 2016: 91-98.

[3] 付印金，肖侬，刘芳，等. 基于重复数据删除的虚拟桌面存储优化技术 [J]. 计算机研究与发展，2012，49(S1)：125-130.
FU Yinjin, XIAO Nong, LIU Fang, et al. Deduplication based storage optimization technique for virtual desktop [J]. Journal of Computer Research and Development, 2012, 49(S1): 125-130.

[4] SUDHAKARAN S, TREESA M. A survey on data deduplication in large scale data [J]. International Journal of Computer Applications, 2017, 165(1): 1-4.

[5] XIA W, JIANG H, FENG D, et al. Similarity and locality based indexing for high performance data deduplication [J]. IEEE Transactions on Computers, 2015 64(4): 1162-1176.

[6] SRINIVASAN K, BISSON T, GOODSON G, et al. iDedup: latency-aware, inline data deduplication for primary storage [C] // Proceedings of the USENIX Conference on File and Storage Technologies. New York, USA: USENIX, 2012: 24.

[7] MAO Bo, JIANG Hong, WU Suzhen, et al. POD: performance oriented I/O deduplication for primary storage systems in the cloud [C] // Proceedings of the 2014 IEEE International Parallel and Distributed Processing Symposium. Piscataway, NJ, USA: IEEE, 2014: 767-776.

[8] KAISER J, SÜß T, NAGEL L, et al. Sorted deduplication: how to process thousands of backup streams [C] // Proceedings of the 2016 32th Symposium on Mass Storage Systems and Technologies. Piscataway, NJ, USA: IEEE, 2016: 07897082.

[9] FU Yinjin, JIANG Hong, XIAO Nong, et al. AA-dedupe: an application-aware source deduplication approach for cloud backup services in the personal computing environment [C] // Proceedings of the IEEE International Conference on Cluster Computing. Piscataway, NJ, USA: IEEE, 2011: 112-120.

[10] LI Wenji, JEAN-BAPTISE G, RIVEROS J, et al. Cachededup: in-line deduplication for flash caching [C] // Proceedings of the 14th USENIX Conference on File and Storage Technologies. New York, USA: USENIX, 2016: 301-314.

[11] PARK D, RAN Ziqi, NAM Y J, et al. A lookahead read cache: improving read performance for deduplication backup storage [J]. Journal of Computer Science and Technology, 2017, 32(1): 26-40.

[12] VITTER J S. Random sampling with a reservoir [J]. ACM Transactions on Mathematical Software, 1985, 11(1): 37-57.

[13] VALIANT G, VALIANT P. Estimating the unseen: improved estimators for entropy and other properties [J]. Advances in Neural Information Processing Systems, 2013, 64(6): 2157-2165.

[14] LIU Chuanyi, LU Yingping, SHI Chunhui, et al. ADMAD: application-driven metadata aware de-duplication archival storage system [C] // Proceedings of the 2008 5th IEEE International Workshop on Storage Network Architecture and Parallel. Piscataway, NJ, USA: IEEE, 2008: 29-35.

[15] NG C H, LEE P P C. RevDedup: a reverse deduplication storage system optimized for reads to latest backups [C] // Proceedings of the 4th Asia-Pacific Workshop on Systems. New York, USA, ACM, 2013: 1-7.

[16] LI Wenji, JEAN-BAPTISE G, RIVEROS J, et al. Cachededup: in-line deduplication for flash caching [C] // Proceedings of the 14th USENIX Conference on File and Storage Technologies. New York USA: USENIX, 2016: 301-314.

[17] MANDAL S, KUENNING G, OK D, et al. Using hints to improve inline block-layer deduplication [C] // Proceedings of the 14th USENIX Conference on File and Storage Technologies. New York, USA: USENIX, 2016: 315-322.

[18] FAN Ziqi, WU Fenggang, PARK D, et al. Hibachi: a cooperative hybrid cache with NVRAM and DRAM for storage arrays [C] // Proceedings of the 2017 33th IEEE Symposium on Mass Storage Systems and Technologies. Piscataway, NJ, USA: IEEE, 2017: 1-12.

[本刊相关文献链接]

杨一晨，张国和，梁峰，等. 一种基于可编程逻辑器件的卷积神经网络协处理器设计. 2018，52(7)：153-159. [doi：10.7652/xjtuxb201807022]

褚伟波，王丽芳，蒋泽军，等. 面向资源使用的TTL缓存服务计费机制与存储策略. 2018，52(6)：42-47. [doi：10.7652/xjtuxb201806007]

王鑫，高家明，梁煜，等. 一种高效存储多级二维9/7离散小波变换结构. 2018，52(4)：111-116. [doi：10.7652/xjtuxb201804016]

李杨，王劲林，叶晓舟，等. 面向嵌入式处理器的优化Montgomery模乘算法. 2017，51(2)：47-52. [doi：10.7652/xjtuxb201702008]

（编辑 武红江）