存储系统重复数据删除技术研究综述

谢 平
(青海师范大学计算机学院 西宁 810008) (华中科技大学计算机科学与技术学院 武汉 430074)

摘 要 目前企业对数据量不断增长的需求使得数据中心面临严峻的挑战。研究发现,存储系统中高达60%的数据是冗余的,如何缩减存储系统中的冗余数据受到越来越多科研人员的关注。重复数据删除技术利用CPU计算资源,通过数据块指纹对比能够有效地减少数据存储空间,已成为工业界和学术界研究的热点。在分析和总结近10年重复数据删除技术文献后,首先通过分析卷级重删系统体系结构,阐述了重删系统的原理、实现机制和评价标准。然后结合数据规模行为对重删系统性能的影响,重点分析和总结了重删系统的各种性能改进技术。最后对各种应用场景的重删系统进行对比分析,给出了4个需要重点研究的方向,包括基于主存储环境的重删方案、基于分布式集群环境的重删方案、快速指纹查询优化技术以及智能数据检测技术。

关键词 重复数据删除,重删率,体系结构,元数据结构,I/O优化
中图法分类号 TP311 **文献标识码** A

Survey on Data Deduplication Techniques for Storage Systems

XIE Ping
(Department of Computer, Qinghai Normal University, Xining 810008, China)
(Department of Computer, Huazhong University of Sci. and Tech., Wuhan 430074, China)

Abstract With the ever-increasing data volume in enterprises, the needs of massive data storage capacity currently become a grand challenge in data centers, and researching shows that there are about 60% redundant data in storage systems. Therefore, the problems of high redundancy in data storage systems are paid much more attentions by researchers. Exploiting CPU resource to compare the data block's fingerprint which is unique, data deduplication techniques can efficiently accomplish data reduction in storage systems, thus data deduplication techniques have become a hot topic in both industry and academia fields. Based on adequately analyzing and summarizing literatures on data deduplication techniques appeared in recent ten years, this paper first presented the principle of representative data deduplication systems, implementation mechanisms as well as evaluation methodologies after analyzing volume-level data deduplication system architecture. Second, we also focused on existing deduplication optimizing techniques with consideration of both the characteristics of data and scale of data deduplication systems. Finally four new research directions were given as follows by comparatively analyzing various application scenarios of data deduplication systems, including research of primary-Storage-Level data deduplication approaches, research of distributed data deduplication scheme for clustered storage systems, research of highly-efficient fingerprint searching techniques and research of intelligent data detection techniques.

Keywords Data deduplication, Deduplication ratio, System architecture, Metadata structure, I/O optimization

1 引言

随着IT技术的不断发展,许多行业(如:金融、通信、互联网等)正呈现出数字化迅猛发展趋势,信息存储的应用领域越来越广泛,加之云技术、云存储的应用,企业数据中心存储需求量越来越庞大,数据量呈指数级增长,已从之前的TB级上升到PB级,甚至EB级。图灵奖获得者Jim Gray提出了一个新的经验定律:网络环境下每18个月产生的数据量等于有史以来数据量之和。2011年10月12日IT研究与顾问咨询公司Gartner对8个国家的1004家大型企业的最新研究结果[1]显示数据增长仍然是大型企业部署IT设施的最大挑战,47%企业将数据增长列为第一挑战,62%企业拥有扩充现有数据中心硬件容量的规划,30%企业计划新建数据中心;随着存储规模的增大,一方面需要企业投入巨额资金购置存储容量,另一方面数据中心的运营成本也将显著增加,运营开销大约为1500美元/TB[2]。如何缓解企业数据中心存储容量成本和运营成本已成为当前网络存储领域面临的严峻挑战。

研究表明[3],在应用系统所保存的数据中,高达60%的数据是冗余的,而随着时间推移,冗余数据的比例也将上升,而为了确保能够可靠而持久地保存数据,可能要花费超过10

到稿日期:2013-03-23 返修日期:2013-06-22 本文受国家973重点基础研究发展计划(2011CB302303)资助。
谢 平(1979—),男,博士生,副教授,主要研究方向为网络存储等,E-mail:qhnuxp@gmail.com。

倍的存储空间和管理成本。因此存储系统中数据高冗余问题受到越来越多研究人员的关注，如何缩减存储系统数据存储容量已成为一个热门的研究课题，而重复数据删除技术(Deduplication)是其中一种容量优化(Capacity Optimization)技术，它通过消除存储系统中重复的数据，缩减系统中实际存储的数据或通过网络传输的数据，在备份、长期归档和数据灾难恢复等方面得到了广泛的应用。在工业界，DataDomain DDFS，IBM Diligent，EMC 的 Avarma，Veritas 的 PureDisk 以及 CommVault 的 Simpana 是比较知名的重复数据删除产品，这些产品通常可以达到 20:1 的重复数据删除率；同时学术界也进行了深入研究，包括美国的 MIT、OSU、UC 和 Stanford 等大学，德国的 Paderborn 大学，英国的剑桥大学，以及中国的 CUHK、清华大学[5]、国防科大[6]和华中科技大学等高校。目前主流重删系统针对备份/归档的二级存储系统，备份/归档存储系统[4]中有 80%～90%的数据都是冗余的，这也是现有重删系统之所以获得如此高重删率的原因。近年来，随着虚拟化技术、专用处理器技术和新型存储介质的出现，结合实时联机重复数据处理和降低 IT 成本的考虑，重复数据删除技术面临新的机遇和挑战。

重复数据删除作为一项应用于存储系统上的数据管理技术，有必要结合数据特征和存储规模来探讨重删技术对存储系统性能的影响，从而获取相关的性能优化技术，因此本文从重复数据删除技术工程设计与实现的角度，分析并概括了重复数据删除技术的基本原理、实现机制、现状以及发展方向。第 2 节介绍重复数据删除技术的概念、层次、分类体系和与重删技术相关的问题；第 3 节阐述重复数据删除系统的体系结构、重删系统设计中的关键操作、评价标准和数据规模分析；第 4 节深入探讨提升重删系统性能的各种加速技术；第 5 节总结和讨论工业界和学术界重删系统项目；最后结合海量存储系统负载特征的复杂性和应用场景的复杂性等特点，给出了 4 个重点研究方向。

2 重复数据删除系统介绍

2.1 重复数据删除技术

重复数据删除技术是一种数据缩减技术(Data Reduction)，通常用于基于磁盘的备份系统，旨在减少存储系统中使用的存储容量。重复数据删除技术通过对源数据的对比分析，将数据中重复的数据块“去掉”，即重复的数据仅在磁盘介质上写一份，其它重复的副本建立“索引”，这样可以有效地节省备份介质。重复数据删除技术支持在已有的磁盘设备上存储更多的备份数据。因此采用重复数据删除技术可以增加保存备份数据的时间，减少数据中心设备数量及电力消耗，降低运营成本。

2.2 重删 I/O 层次分析和总结

从系统结构的层次模型看，对于磁盘的一次 I/O 请求，首先应用层下发 I/O 请求，经过虚拟文件系统层，其次是文件系统层，接下来是 Cache 层、通用块层、I/O 调度层、块设备驱动层，最后是物理块设备层，现有的重删系统主要位于应用层、文件系统层和块层实施重删检测功能，比如现有的开源重删系统 Lessfs[7]和 OpenDedup[8]就是基于应用层，通过用户态文件系统 Fuse[9]实施重删，从而避免了用户对底层的管理，显然实现容易，但与传统的 Linux 文件系统 EXT2、EXT3 和 XFS 相比其性能更低[7]，另外，有的重删系统也可以通过调用用户态的块设备驱动 SCST 模块[10]来实现重删。基于内核态的重删系统是工业界和科研机构研究的重点，内核态则是以修改内核代码或者以模块的形式动态植入内核代码，LiveDFS[11]就是在现有的文件系统 EXT3 中实施的重删功能，文件系统层重删系统一般是对实际文件的内容进行分块和重删，把每个文件分为若干个块，针对每个文件块进行重删，而对文件元数据及目录不做重删，文件系统层的重删只是针对挂载了该重删功能的文件系统对应的分区才具有重删功能，因而也称为局部重删；块层重删系统 I/O Deduplication[12]在文件系统层之下，独立于文件系统，能够支持多个文件系统，因而能够实施全局的重删，块接口层语义简单，从而使得 I/O 容易截获、操作和重定向，块接口是一个通用的抽象，如操作系统的块设备实现、RAID 和 ISCSI 存储设备都是基于块接口的，因此基于块的抽象更能被移植和部署到多个平台。综上所述，重删系统所在的 I/O 层次从上到下的实现是从易到难，语义信息逐渐减弱，但重删性能会更好。

2.3 重删系统分类介绍和总结

根据实施重复数据删除操作位置的不同，分为源和目标端重复数据删除；根据实施重复数据删除操作时机的不同，分为离线 (Off-Line)、近线(Out-of-Line)和在线重删系统(On-Line)；根据实施重删操作粒度的不同，分为文件级和块级重删系统；根据重删系统所服务数据实体的存储与访问方式的不同，分为主存储和从存储重删系统。

2.3.1 基于源和目标端的重删系统

在网络环境中，为了确保数据的可用性与可靠性，数据一般都会备份到远端存储节点。因而数据重删既可用于本地客户端节点(源重删系统)，也可用于远端存储节点(目标端重删系统)。在源重删系统里，副本被消除在每一个本地节点里，仅有唯一的数据通过网络传输到远端存储阵列进行存储，这种方式节省了网络带宽，在客户端安装应用程序，可以充分利用源端节点的资源；在目标端重删系统里，所有的数据都通过网络传输到远端存储节点进行识别并消除副本，这种方式的缺点是所有的数据都是通过网络传输的，因而占用了更多的网络带宽。由于多个客户端节点的数据最终都汇聚到了目标端节点，因此在目标端发现副本的概率更高，两者的结合将会提供更多的时间和资源来布置重删操作，比如，NetApp 的 Snapvault 使用了两者的一个结合。

2.3.2 主存储与从存储重删系统

存储系统可分为主(Primary)存储系统和从(Secondary)存储系统[15]。从存储重删主要针对归档和备份系统负载，数据具有局部性，备份是典型的批量写操作，一般通过自动化的进程处理。备份过程中，系统的性能会暂时下降，或者出现系统资源不足的情况，备份过程几乎没有读操作，多数用户并不关注数据恢复的速度，而对吞吐率敏感，因此从存储重删系统主要关注 I/O 吞吐率的提升。主存储系统则完全是另一种情况，数据缺乏局部性特征，数据会随时写入，主存储系统由于要及时响应客户端的请求，因此对延迟敏感。因而主存储重

删系统关注于如何提高 I/O 的平均响应时间，由于写可以采用写回操作，因此可以批量地更新写缓存，而不影响前台应用写请求的平均响应时间，但是对于读请求，必须从存储系统中读取，而且主存储系统读请求的比例一般大于 75%，因此主重删系统更应该关注读请求的性能提升。

2.3.3 离线、近线和在线重删系统

目前在线重删系统、离线重删系统和近线重删系统既可以应用于主存储系统，也可以应用于从存储系统，主要是基于不同应用场景的重删系统对性能的考虑。在线重删在数据写到存储系统之前在写路径中完成重删工作。由于重删过程在写关键路径中，因此增加写延迟，数据规模较小时，重删对系统延迟影响不大，但是当存储系统所处理的数据规模逐渐增大时，会加长重删的路径，从而会延长重删的处理时间，降低重删系统的性能。如果考虑逐个比较数据块，数据规模比较大，而且又在主存储系统环境，可以想象处理一个副本块将非常耗时。针对这个问题，有研究者提出牺牲重删率方式来提高系统的性能，如 iDedup[13] 只删除达到一定阈值的连续串，有效地缩短了指纹查询延迟时间，但缺点是只能实现部分重删。离线重删系统是一种后处理方式(Post-Process)，数据先存储到存储系统中然后实施重删功能，这种方式能够隐藏重删过程的延迟对用户体验的影响，因而对响应时间敏感的主存储环境可以采用这种离线的方式进行处理，这种类型的覆盖写会增加写延迟，也会增加读碎片。近线重删系统(Out-of-Line)倾向于等待系统空闲时完成对以前写数据的重删操作，是介于在线与离线之间的一种重删系统，只是其对时间要求不那么苛刻，如：DDE[14] 在主存储系统处于轻载或空闲时实施重删。

2.3.4 文件级与块级重删系统

文件级重删系统也称单一实例存储 SIS (Single Instance Store)，它以文件为粒度作为处理的对象，比如 Microsoft 的 wss2000[16]，对于输入的任意一个文件采用 Hash 算法计算其指纹，并将这个指纹与已有文件的指纹库进行比配，如果相同，系统只保存一个指向原始文件的指针，而不是再一次存储到另外一个地方，适合于由小文件构成的数据集；由于不需要对数据文件再次进行分块划分，相应其划分数据的 CPU 开销相对低，RAM 开销相对低，因而其执行高效、简单快捷。在实际工作负载中，尽管有大量的文件为小文件，但占用了绝大部分存储空间的是占少数的大文件。由于以文件为粒度的重删只能进行文件粒度的重复数据检测，不能发现文件内部以及文件之间更小粒度的重复数据，因此数据集的重删率往往不高，基于此有研究者提出了块级的重删系统。块级重删系统对输入的数据文件以块为粒度进行划分，如：DBLK[17] 以 4k 为粒度对块进行划分，块级重删系统的数据重复检测在指纹库中的查找和匹配过程与文件级重删系统类似；与按文件粒度划分相比，由于其粒度更小，检测出重复块的概率会更高，如果在文件更新的过程中，一个大文件内部只有部分数据块发生变化，这种情况在块级重删系统中只保存发生变化的块，因而更能节省磁盘空间；相应地，以块粒度进行划分，计算开销会更大，会占用更多 CPU 资源，块级重删系统中其元数据表会变长，从而降低 RAM 的利用率，花费更多的磁盘空间用于存储元数据表；在块划分的过程中有的系统按 FSP 算法[18] 进行固定块大小划分，这种分块算法有利于减少分块在时间上的开销，由于主重删系统必须要求低延迟，因此可以考虑使用固定分块算法，这种方法的优点是简单高效，适合于更新操作少的数据集(如，图片服务器和视频服务器)，如果一个文件中的内容稍有变化，便会导致其后数据块内容与先前数据块内容稍有不同，从而不能智能地根据文件内容的变化进行调整，降低了系统重删率，因此有人提出重删系统按 CDC 算法[18] 进行变长块划分(适合于经常更新的场合，如博客)，CDC 算法是应用 Rabin 指纹将文件分割成长度大小不一的分块策略，主要采用滑动窗口技术(Sliding Window Technique)，后来有研究者提出了 Fingerdiff[19] 和 Sliding Block Technique 技术[18] 对其进行了进一步的改进，提高了重删率，对于大文件系统，变长块进行划分会获得更高的存储效率，但相应地会增加计算的开销。综上所述，文件划分粒度影响重删系统的性能：a)数据重删率；b)读写速度。由于每一个块都需要相同大小的元数据项，块越小，数据重删率会越高，但指纹项会越多，从而导致检索的 Hash 表会越长，表越长，更新将变得更越难，块的大小决定了系统的开销。划分粒度对于重删系统而言在时间开销、空间开销上是一个权衡。

2.4 重删与压缩的关系

目前数据缩减(Data Reduction)[15] 技术是存储系统领域非常重要的技术，包括重复数据删除技术和数据压缩技术，压缩技术的优点是非常成熟，并且易于理解，它的缺点是处理机制仅限于单个文件之内，而无法做到跨文件处理。压缩通常是针对那些存取频率不是很高的数据，这是因为数据的压缩和解压缩需要 CPU 进行非常密集的计算处理，计算开销较大，这样往往会影响数据的访问。目前使用专用芯片实现压缩和解压缩，有一些产品诸如 Storewize 公司的 STN-2100 和 STN-6000，以及 Ocarina Networks 公司的 ECOsystem，都支持在数据写入/读取过程中同步进行压缩/解压缩，如果它们能达到线速，就不会影响主存储系统的读写性能。重复数据删除采用指纹技术来处理数据块的内容，有相同 Hash 值的块只存储一次来实现数据的消重，在存储层使用重复数据删除技术并不意味着不再需要数据压缩等其它数据缩减技术。重复数据删除是一种补充型技术，能够与包括业界标准 LZS 压缩算法在内的现有数据压缩技术形成互补，以提供双重数据缩减性能。重删系统必须先实施重删而后再压缩，因为压缩会使即使非常相似的内容经压缩后变得非常不同。

3 重删系统结构及关键操作

3.1 卷级重删系统体系结构

卷级重复数据删除系统体系结构如图 1 所示，企业级存储阵列通过高速存储网络 Switch(FC/Ethernet)与多个 Host 连接，为 Host 提供物理存储资源，是一种典型的 SAN 存储区域网结构。为了提高存储系统的性能和可靠性，磁盘阵列上都部署有 RAID5 或 RAID6，其可以被看作 Host 端的虚拟存储盘，每个 Host 通过 LUN 访问相应的存储资源。重删模块处在 LUN 逻辑卷层之上，共享整个物理磁盘资源，保存每个客户 LUN 和存储资源之间的实际分配关系，所以多个主机

端的存取数据流需要在重复数据删除层汇集，实现在单一物理存储资源上的重复数据检测及其删除，从而在最终的物理存储子系统上仅保留一份数据拷贝。存储资源池和磁盘阵列模块管理实际的磁盘布局拓扑和物理存储虚拟化，物理上通过高速通道(FC 或 SAS)和存储区域网连接到磁盘柜中的磁盘上。

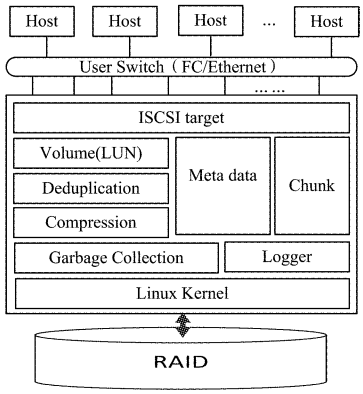

图 1　卷级重复数据删除系统体系结构

从系统结构层次关系看，重复数据删除层的重复数据处理流程分为文件/数据流的变长分块算法 Chunking、块指纹哈希算法 ChunkHashing、指纹查询检索 FPindexing 和存储元数据和唯一数据 DataStoring 4 个阶段。如图 1 所示，它是一种典型的块层重删系统体系结构，重复数据删除层的顶端为 ISCSI 访问协议层，在访问协议层以下的就是重复数据删除层中的各功能子模块，主要有 Volume 模块、Deduplication 模块、Compression 模块、Garbage Collection 模块、Meta data 模块、Chunk 模块和 Logger 模块。重复数据检测的流程首先根据重删系统数据块分割算法将某个 LUN 卷的文件或数据流分割成若干个 Chunk，并计算每一个 Chunk 的 Hash 值(也叫指纹 FingerPrint，简称 FP)，Deduplication 模块根据生成的 FP 到 Meta data 模块中的指纹索引表(简称 FPIndex 表，记录了所有 Chunk 的 FP 信息)中去查找匹配，如果查找到，说明该数据块 Chunk 是副本，因此不用写入磁盘，代替的是通过 Logger 将该 FP 对应 counter 加 1，如果没找到，说明该数据块是新的，可以直接写入磁盘并将该 FP 更新到 FPIndex 表中，在 Meta data 的地址索引表 AddrIndex(保存了所有 LBA 与 PBA 的映射关系)中更新地址项。数据和元数据最终在磁盘上都是按 Container 大小存放的。为了进一步对数据实施缩减，都要使用某种压缩算法 Compression 对每一个 Container 实施压缩。由于数据和元数据在磁盘上都是以覆写方式存放，因此适时动态地回收不再使用的数据块(Counter 为 0 的数据块)将是非常必要的，垃圾回收技术 Garbage Collection 将会有效地提升重删系统 FP 索引管理的可靠性、查询速度和可扩展性。

3.2　重复数据判定操作

重复数据的判定技术就是将输入的数据与已有数据进行比较并消重的技术，目前通用的做法就是从输入的每个 Chunk 的内容中选择一个特征值即一个 FP 来标识它，然后将该新 FP 与 FPIndex 表中已有的 FP 进行比对，期望的特征值能唯一标识 Chunk，该特征值一般选择抗冲突加密 Hash 值作为其特征，如 SHA 和 MD5 等算法；尽管有研究者因为 Hash 函数存在碰撞和生日悖论对此表示怀疑，但很多人认为相比于硬件出错，Hash 碰撞引起的出错概率更小。

该问题符合概率论中的生日悖论模型，SHA-1 的两个不同数据块拥有同一 Hash 值的概率为 $1/2^{160}$；随着输入的数据块的个数逐渐增多，其 Hash 冲突的概率 FPIR[22] 会逐渐升高，定义为：

$$FPIR=1-EXP(\frac{-N^2}{2\times 2^m}) \tag{1}$$

式中，N 为输入数据块的个数，m 为 Hash 值位数，块大小为 32kB，磁盘空间 1PB，FPIR 为 4.038×10^{-28}，块越大其冲突率会越低，典型磁盘的位失效率最小为 10^{-27}；可见 1PB 数据空间 Hash 冲突率与磁盘位失效率相当，因此在系统中可以不考虑 Hash 冲突造成的影响。

3.3　重删流程读、写和删除操作

重删系统根据 I/O 过程分为读、写和删除操作 3 个流程，其中指纹库的查询过程是重点优化步骤。由于整个哈希索引表太大而不能完全放入内存，因此有一部分会放在磁盘上，而磁盘的 I/O 开销太大，会严重影响性能，所以要通过合理的设计来提高内存缓冲区的命中率，从而减少对磁盘 I/O 的需求，同时尽量减少内存的占用，并使查找的过程尽可能快，以减少写请求的响应时间。Data Cache 中的数据项采用 LUN＋LBA 直接索引，但不对每个 LUN 单独分配 Cache 空间，统一管理。

3.3.1　写请求

首先通过哈希函数计算数据块的哈希值，然后在哈希指纹库中查询数据块是否在哈希索引表(FPIndex)中。如果哈希值不在表中，则申请新的磁盘空间来存放这个数据块，然后将数据块写入数据容器 DataLog，再更新哈希索引表，把新数据块的 FP 到 PBA 的映射关系加入到哈希索引表中，再在地址索引表 AddrIndex 中加入 LBA 到 PBA 的映射关系，并更新数据缓存、地址索引缓存和指纹索引缓存。如果哈希值在表中被查到，说明该数据块是重复的，更新计数器 Counter，再在地址索引表中查找相应的 PBA，并更新数据缓存、地址索引缓存和指纹索引缓存中的 LBA 到 PBA 之间的映射关系。

3.3.2　读请求

直接在地址索引表 AddrIndex 中查找，找到后就在 DataLog 中读取相应的数据块，并返回读取的数据块。对于部分放不进内存的地址索引表，要通过磁盘调度进入内存，再进行读取。

3.3.3　删除请求

先在地址索引表中删除相应的映射关系，再在哈希指纹库中删除相应的映射关系，最后根据引用计数情况来确定是否去 DataLog 中删除相应的数据块。

3.4　重删系统的评价标准

重删系统量化评价指标体现在 3 个方面：I/O 吞吐率、请求的平均响应时间和重删率，下面结合图 1 介绍的块层重删系统量化了这 3 个指标。

3.4.1　平均响应时间

重复数据删除系统是在正常读写流程中加入重删模块，

一方面在写入数据的过程中由于需要对重删的数据块元数据进行查找匹配，当元数据比较大时，不可能在 cache 中实现全命中，从而增加了额外的磁盘 I/O，增加了查找延迟，因而会延长写入延迟时间；另一方面由于重删破坏了原有数据的连续性存放，多个文件共享一个唯一的副本，从而造成磁头更多来回移动，增加了读数据延迟，这些时间开销就叫做延迟时间，也叫响应时间，延迟时间＝内存查询开销＋磁盘查询开销。

(1)写入延迟时间

$$T_{write}=(T_{pa}+T_{fp}+T_r\times h_r+T_d\times(1-h_r)+T_{dw})\times N \tag{2}$$

T_{write}：写入总延迟时间；T_{pa}：数据分块对齐的时间开销；T_{fp}：数据块生成的指纹时间开销；T_r：写数据块在内存中的查询时间开销；T_d：写数据块在磁盘中的查询时间开销；h_r：写查询内存命中率；T_{dw}：写入数据和元数据开销；N：写入数据规模。

(2)读数据延迟时间

$$T_{read}=T_m\times h_m+T_{dd}\times(1-h_m)+T_{dr}\times w \tag{3}$$

T_{read}：读数据总延迟时间；T_m：读数据块在内存中的查询时间开销；T_{dd}：读数据块在磁盘中的查询时间开销；h_m：读查询内存命中率；T_{dr}：读数据和元数据开销；W：读取数据规模。

3.4.2 重删率

重复数据删除技术实现了数据的缩减，其量化标准就是重删率[20]，重删率的定义可以通过重删之前输入重删检测模块的字节数(Bytes In)和重删之后输出重删检测模块的字节数(Bytes Out)来定义：

$$f_{ddr}=\frac{Bytes\ In-Bytes\ Out}{BytesIn}\times 100\% \tag{4}$$

就目前的重删系统而言，数据的重复率 f_{dr}(冗余度)由数据集(Data Set)自身的特征以及具体应用场景、数据分块算法和平均数据分块大小决定；当数据规模较小时，能实现 100% 副本消重，这时重删率就等于重复率，但是随着数据规模的不断增大，其元数据表的长度也在增加，因而为了考虑性能等因素，不同的重删系统针对同一个数据集的重删率可能不同，为了体现重删系统消重的效率定义为重删效率 f_{def}[21]，重删率定义如下：

$$f_{ddr}=f_{dr}\times f_{def} \tag{5}$$

采用查询优化是最关键的，很大程度上决定了重删效率，理想的情况下重删效率接近 100%是最好的情况。影响重删系统重删效率有两个因素：1)系统的数据规模；2)采样算法。重删效率 f_{def} 与数据规模因子 β 成反比，与采样算法因子 α 成正比，可表示为 $f_{def}=\alpha/\beta$。在相同的数据规模下，采样算法越好，FP 的内存命中率就越高，重删效率就越高；在相同的采样算法的情况下，数据规模越大，其重删效率会越低。

3.4.3 I/O 吞吐率

存储介质吞吐率 Th_{dr} 指单位时间内传输的数据量。由于重删增加了正常的数据读写延迟时间，因此重删系统的 IO 吞吐率比不实施重删的系统吞吐率更低，数据集大小一定，与延迟时间 $T_{latency}$ 成反比，定义为：

$$Th_{dr}=Bytes\ In/T_{latency} \tag{6}$$

3.5 重删系统数据规模分析

在重删系统中随着数据规模的不断扩大，其 FP 查询管理开销将逐渐增加，严重地影响了重删系统的性能和重删效率，重删率与 workload 有关系。比如 100TB 的数据，如果每一个数据块大小为 4kB，则有 100TB/4kB＝25G 个数据块，如果每个数据块的指纹大小为 20B，则 FPIndex 表总共为 25G×20B＝500GB 的容量；如果重删率为 50%，则 FPIndex 表的大小为 500GB×50%＝250GB，显然不适合存放于内存，如果数据量达到 PB 级，则该 FPIndex 表还会更大。因此随着数据规模的增大，不可能将所有的 FP 放入内存，也不可能对每一个 FP 逐一查询，目前提高 FP 查询优化的技术主要是预取和采样两种技术[23]，其可实现指纹的快速查询匹配。

如图 1 所示的块层重删系统，如果每一个 LUN 最大容量为 64TB＝2^{46} B，块大小为 4kB＝2^{12} B，则有 $2^{46}/2^{12}=2^{34}$ 个块，如果每一个块都用一位来表示，则共有 2^{34} 个位，即 $2^{34}/8$＝2GB 的 LUNLBA_bitmap 空间，如果总共有 1024 个 LUN，则占用空间为 1024×2GB＝2048GB；对于 1PB＝2^{50} B 的磁盘空间，块大小为 4kB＝2^{12} B，共计 2^{50-12} 即 2^{38} 个 Chunk，同样每一个块都用一位来表示，则共有 2^{38} 个位，即 $2^{38}/8$＝32GB 的 PBA_bitmap 空间，同理所需的 PBA 的字节数为 5，如果考虑 LBA 所需字节数假设为 10，为方便计算并将标志位等可能 PBA 所需字节数设定为 6，则一个元数据项 LBA→PBA 共需 16 个字节，则需要 2^{38}×16B＝4TB 的 AddrIndex 空间。

综上所述，通过数据规模分析，重复数据删除技术使数据路径加长，数据量达到阈值后，逻辑地址 LUNLBA_bitmap、地址映射表 AddrIndex、指纹库 FPIndex、物理地址 PBA_bitmap 都无法全部放到内存，引起查询延迟，从而导致了重删系统性能的瓶颈，因而其数据规模不可能无限地增大，而内存不变，通过规模分析得出如下结论：1)减小查找的范围，数据块小，查找范围大；块大，查找范围小，从而导致重删率降低。2)尽量减少对内存的占用，提高内存使用的效率。3)如何使操作尽可能不访问磁盘，在不得不访问磁盘的情况下，如何减小带来的时间开销。

4 提高重删系统性能的加速技术

重删系统 I/O 路径上分为 Chunking、ChunkHashing、FPindexing 和 DataStoring 4 个阶段，针对其中的每一个阶段既可以采用硬件加速技术又可以采用软件加速技术来提升重删系统性能；就目前现有技术而言，分块算法 Chunking 和 ChunkHashing 算法的复杂性的提高会增加 CPU 计算开销，所采用的算法相对成熟易实现，对算法的改进提升较难，现有的文献和资料显示可采用多核技术来加速其执行；DataStoring 阶段一般受硬件接口性能所限制，可以采用读写性能更好的器件来提升，所有文献资料显示指纹查询检索(FPindexing)是加速提升的重点。为了提升系统 FP 索引管理的可靠性、查询速度和可扩展性，重删系统都会采用灵活的垃圾回收技术 Garbage Collection、Striping 技术与纠删码技术，以进一步提升系统的性能和可靠性。

4.1 FPindexing 查询优化技术

指纹库查询过程的优化是重点，目前针对较大数据量的内存查询优化策略，采用挖掘局部性特征的技术和位置特性(相似性技术)来减少内存的占用和磁盘 I/O 等等；采用 FP 快速判别算法 Bloomfilter 技术把常用的热指纹放到内存中，

所做的工作都是在重复数据删除率、I/O 吞吐率、响应时间之间进行平衡和折中。另外还可以根据具体数据流的特点进行优化，如 DDFS[24] 系统。

4.1.1 局部性技术

数据重删系统随着数据规模的增加，其指纹索引表 FPindex 会逐渐加长，不可能全部放于内存，指纹的查询导致额外的磁盘 I/O，增加了索引查询的时间。为了提高在 RAM 中指纹查询的命中率，降低指纹查询开销，在重删系统中提出了利用局部性技术（Locality Preserved Caching）来提升指纹的查询优化技术，局部性技术分为空间局部性（Spatial Locality）和时间局部性（Temporal Locality）。

Venti[25] 是最早给出在备份系统中实施重删的设计方案，现有重删系统的基本架构和功能来源于此，使用了 Block Cache、Index cache 和 Disk Striping 3 者的结合来提高写吞吐率。即使在使用 8 个高端 Sheetah SCSI 磁盘的条带化后，最终的写速度为 6.5MB/ Sec。Cache 的低命中率导致了如此低的速率，这表明对于大数据流单纯使用内存 Cache 局部性技术来提高随机磁盘 I/O 吞吐率是非常有限的。DDFS[24] 注意到传统 Cache 技术存在的问题，从而在基于 LPC 技术的基础上又提出了 SISL（Stream-Informed Segment Layout）技术，SISL 将无特征的输入备份数据整理为有特征的连续数据块，这些数据块在磁盘的物理空间上尽量连续保存，其对应的 FP 也是连续保存的，即相互关联的数据和元数据在存储空间上放入磁盘的同一个 Container 里，该 Container 具有自我描述，数据的压缩是针对每一容器实施的，不同的容器对应不同的数据流，当有多个数据流时，可同时打开多个容器。当容器满时才一次性地 Flush 到磁盘，也叫一次 I/O，以避免磁盘瓶颈。正是由于 SISL 技术确保了输入的不同数据流在空间上保持的局部性，才使得 LPC 技术的命中率大为提高，SISL 是确保 LPC 命中率提高的前提条件，因而可将两者统称为 LPC 技术，这种技术靠装入或逐出基于一个容器的指纹来保持局部性，即一个指纹被装入，那么该指纹所在的容器中的指纹全部都装入，反之在内存逐出一个指纹，那么该指纹所在的容器也被踢出，如果在内存中未命中，意味着在磁盘上的该指纹所包括的容器将被调入内存，进而利用 LRU 算法将在内存中的相应容器踢出，代替的不是根据指纹值的一个范围来装入或逐出，这个范围并不代表局部性。实验结果表明，Bloomfilter 技术加 LPC 技术能够获得 100MB/sec 吞吐率并显示磁盘的 I/O 在索引的查询操作中不再是瓶颈。

在备份存储重删系统中 LPC 技术获得了较好性能，由于备份存储系统其行为是比较固定的，即备份固定目录的数据，因此数据流可以被解析为连续的、有意义的。若命中某一备份数据流的第一个 Chunk，可以预见，接下来的 Chunk 也会和上一次的备份数据流中的下一个 Chunk 相同。但对于主存储系统局部性技术显然不会有太好的效果，因为主存储系统的数据流是比较随机的，缺乏局部性特征。

4.1.2 相似性技术

相似性检测技术在目前的重删系统中被普遍采用，由于重删系统是在原有正常 I/O 路径的流程中加入了一个具有重删功能的模块，该模块的主要功能就是检测输入的新块是否是重复的块，在重删之初，元数据的查找比对可能非常快，但是当数据量达到 TB 级别时，这个查找就非常耗时了，再者内存的容量是有限的，不能完全容纳元数据，因而就会将一些元数据放入磁盘，因此增加了延迟时间；基于这种情况引入了相似性技术，相似性技术就好比将一个大范围的 Chunk 表缩小成一个小范围的 Chunk 表一样，这个小范围的 Chunk 表能够满足大多数的命中，比如 90% 的命中率，或者称为热点的 Chunk 表，因此这个小范围的 Chunk 表就能够放入内存，从而实现快速的查找比对。在比对数据块的时候只比对满足某一阈值的连续块[13]，这样在重删的时候只删除连续的数据块，这里存在一个不完全重删的问题，牺牲了重删率，但是它能够有效地提高重删系统的响应时间，这对于主重删系统是非常重要的。

Sparse indexing scheme[26] 在一个 D2D 的在线数据备份系统里使用采样指纹索引而不是整个指纹索引，从而进一步减少了重删系统的内存使用率。该方案的观点是副本数据块的长度是可连续变动的，在高概率的情况下一定范围内的采样指纹值的匹配可能代表了整个范围内的匹配，在所有的匹配范围内，一个冠军被作为重删的依据，采样率是在重删率与内存使用率之间的一个权衡。重删率的高低很大程度上由采样率来决定，选择一个合适的采样率是一个关键问题，比如有两个极端：一种是将所有的 Hash 值调入内存，当然都能找到，但不可能全部放入内存；另一种情况是只调入一个 Hash 值进入内存，显然 I/O 开销小，但命中率低，因而怎么采样是一个关键问题。

Extreme-bin[27] 方法利用文件相似性，这种重删方法的主要目的是基于由小或非局部性的文件构成的非传统的备份载荷，对于每一个文件，Extreme-bin 选择最小的 Chunk 的 Hash 值作为代表指纹，有相同“代表指纹”的文件被归为一个 bin，bin 是块级重删的基本范围，所有 bin 的有代表性的指纹都编入一个 Primary Index 中，Primary Index 是相当小的，被放入 RAM，因此对于每一个输入的文件只有一次对相应 bin 的磁盘访问。由于 Extreme-bin 的采样，每一个文件只有一个指纹，被归入相同桶的相似文件的概率很大程度上取决于它们的相似度，根据 broder 理论，不同文件有相同代表性指纹的概率随着文件数的增长将会降低。

Fanglu 等人[23] 提出渐进式采样索引，典型的块被归入 Segment，指纹的查询索引被采样以确保只有重要指纹才代表最大程度的存储段，也确保了整个索引适合放入主存储器。建议在采样的时候，采样的方案基于剩余的有效 Memery，而不是使用整个主存储器的容量。如此的一个渐进式的采样方案能动态地改变采样率和产生更好的重删率。

4.1.3 BloomFilter 技术

BloomFilter 是一种空间效率很高的随机数据结构，它利用位数组来很简洁地表示一个集合，并能判断一个元素是否属于这个集合，BloomFilter 算法的核心思想就是利用多个不同的 Hash 函数来解决“冲突”。在最早的重删系统 Venti[25] 中采用传统段桶式的 FPindex 查询方式，但是当数据规模逐渐增大时，其 FPindex 不可能全部放于内存，BloomFilter 技术是一种利用空间换时间的技术，因此引入 BloomFilter 技术可以有效提高元数据的查找速度。

DDFS[24] 首次将 BloomFilter 应用于重删系统来加速指

纹的查找，称之为 Summary Vector，其实现的过程包括 3 个阶段：初始化阶段、插入指纹和查询指纹阶段。DBLK[17] 在 DDFS 的基础上提出了 MBL(Multi Blooming Filter)的思想，在主存储重删系统里，针对 8TB 的磁盘，每一个块为 4kB，构造了 1664×1664 两层的 BloomFilter，其占用内存空间 11G，实现了 1500IOPS 的随机写速率和 700IOPS 的随机读速率。

综上所述，采用 Blooming Filter 之前，FPindex 体积很大，即使采用较大的 Chunk，1TB 原始数据对应的 FPindex 也要几个 GB，所以 FPindex 无法完全保存在内存里，那么就会导致 FPindex 查询时频繁的 I/O 操作，处理能力肯定低下。而 Blooming Filter 的体积比完整的 FPindex 小很多，在 DDFS[24] 文中，16TB 的原始数据，只需要 1G 的空间，那么 Blooming Filter 可以完全保存到内存中。但是 Blooming Filter 不是代替 FPindex 的查询，在 DDFS[24] 中，FPindex 会保存在磁盘中，Blooming Filter 保存在内存中。只有那些全新的 Chunk 可以通过 Blooming 判定不存在相同的 FP，从而避免了指纹查询过程中的磁盘 I/O。但对非全新 FP，还是需要进行 FPindex 的查询操作，磁盘 I/O 无法避免。按照 DDFS[24] 给出的比率，Blooming Filter 可以在 FPindex 查询过程中减少 16%左右的磁盘 I/O。MBL 比起单层 BloomFilter 则使用更大的内存空间，也是一种空间换时间的思路，比起单层 Bloomfilter 其能够更快速精确地定位 FPindex，从而避免读取不相关的指纹集合 FPSET。

目前重删系统普遍采用上述 3 种技术实现重复数据的删除，影响重删系统性能的关键在于索引的查找，目前加快索引查找的策略可归为采样和预取技术[23]。DDFS 采用的 BloomFilter 技术和局部性技术能够有效地保证 90%的内存命中率，但是其中存在小的随机磁盘 I/O，严重地影响了磁盘的性能，针对这个小随机磁盘 I/O，Chunkstash[34] 采用 Flash 存储器来加快对索引的查找。

4.2 硬件加速技术

从计算机系统结构的观点出发，除了上面基于软件角度的考虑外，重删系统流程的每一个功能部分都可以采用硬件加速技术来提升重删系统的性能。

基于数据流的变长分块算法 Chunking 和块指纹哈希算法 ChunkHashing 其计算开销都比较大，延长了系统的响应时间，考虑将数据流分段实施分块和指纹哈希有利于实现上的并行性，从而提升系统的响应时间，如 P-Dedupe[28] 采用多核处理器或多处理器来实现重删系统的流水线和并行性有助于降低系统延迟时间，该思想的实施将会使主存储重删系统采用变长分块技术成为可能。重删流程中的指纹检索(FPindexing)和保存元数据和唯一数据(DataStoring)是整个 I/O 优化的重点，一方面，如果不考虑 RAM 扩展限制，可以无限增加 RAM，将所有的 FP 都存放于 RAM 中加快指纹检测过程，但是由于 CPU 型号和主板资源的限制，这一想法肯定不可能，目前单节点终端机 RAM 一般都是 2G 或 4G，如果考虑将一个集群 PC 的 RAM 构建成 RAMCloud[29] 将会成倍地扩大内存，从而加快指纹查询检索时间；另一方面影响指纹检索和写入元数据及唯一数据的性能瓶颈在于磁盘 I/O 性能的限制，目前单磁盘的性能很难达到 500 个 IOPS，而新型的存储介质 SSD(Solid State Disk)和 PCM(Phase Change Memory)都具有较高的 IOPS 数、高吞吐率和低访问延迟的特性[30,31]，dell[32] 和 NETAPP[33] 企业级的单个 SSD 的性能能达到 6k～18k 个 IOPS 随机读性能，具有 80～250MB /s 的顺序读写吞吐率，随机写性能仅能达到 100～140 个 IOPS，访问时间为 0.05ms～0.5ms，正是由于 SSD 良好的性能，目前重删系统采用 SSD 来存储元数据，如 Microsoft 针对在线二级存储重删系统使用 Flash 来提升重删性能，提出的 ChunkStash[34] 主要是在传统的 RAM 与 DISK 之间增加一级 Flash Memory，由于 Flash 良好的读写速度，使得元数据的查找速度得以提升。以吞吐率进行测试，结果显示：其比基于磁盘索引的重删系统快 7X～60X，在磁盘阵列里可以采用 RAID 技术来进一步提升 I/O 性能。

Chunkstash[34] 注意到在 Data Domain 公司理想的情况下确保了大多数的查询发生在内存，但是这里不可避免地存在一些小的随机写磁盘 I/O，由于这些随机的写磁盘 I/O 导致性能的降低，如果能改善随机磁盘 I/O 的命中率将会显著地提升整个吞吐率。因此提出了使用 Flash 存储器和 Flash 存储器感知的算法和数据结构来获得更快的索引查询操作。Flash 存储器的读写性能好于 SCSI 磁盘而低于 RAM。但是 Flash 的随机更新是相当低的，通过将 Flash 的随机更新转变为 Log 结构的扩展操作来克服对 Flash 的随机更新。评估结果显示，使用 Flash 磁盘能得到 200MB/Sec 的吞吐率。

4.3 垃圾回收技术

垃圾回收技术在存储系统中用于判断某个物理块是否应该被回收，具体来讲就是维护元数据表中的物理块组的使用情况，目前主要有 3 种通用的思路[35]：标记和扫描技术(Mark and Sweep)、基于引用计数(Reference Count based)和基于时效期的技术(Expiry Time based)。

标记和扫描技术(Mark and Sweep)就是对已使用的每一个物理块进行标记，然后对所有的物理块进行扫描以回收未作标记的物理块，但是在标记阶段需要冻结元数据表从而不允许用户访问，另外标记物理块的开销也将是非常大的，不可能将其放于 RAM，因此更新 PBA 将会导致低磁盘 I/O 性能。Hydrastor[36] 提出只读标记与扫描(Read Only Mark and Sweep)，解决了标记阶段冻结系统的问题，回收是通过块引用数来实现的，其引用数的增加是定期更新的，从而使随机更新转变为顺序更新。Fanglu[23] 提出分组标记和扫描(Grouped Mark and Sweep)机制，其主要思想是：为了避免在标记阶段处理每一个文件和扫描阶段处理每一个容器，通过使用文件管理器将备份数据流归为不同容器的不同层次，因此标记阶段仅仅标记一个组中发生改变的文件，而扫描也仅仅扫描标记阶段所处理组所在的容器，从而大大降低了标记所用时间开销，标记和扫描的工作量是随着系统载荷的变化而变化的，因而其具有可扩展性、高可靠和快速的特点。基于引用计数(Reference Count based)简单来讲就是对每一个物理块使用一个计数器来记录其引用的次数，每一次创建一个新块和删除一个块时，其计数器就相应地加 1 或减 1，每一次更新的时候检测其计数器为 0，说明该块未被使用可以回收，比起标记和扫描的方式来讲避免了耗时的扫描，但系统规模比较大时，其更新的过程也会降低系统的性能。基于时效期的技术(Expiry Time based)通过 PBA 时效期的方式来替代

引用计数，因而可进一步地减少更新次数，但是当覆写物理块时需要更新其时间，在垃圾回收时需要扫描所有的物理块，看其是否过期。目前有的系统可能使用的是其中的一种方式，或是两种混合以取得更好的时间和性能开销。

4.4 striping 技术与纠删码技术

重复数据删除造成了数据在磁盘上的不连续存储，从而降低了磁盘 I/O 的性能，Venti[25] 为了提升 I/O 性能最早采用了 Striping 技术[49]，Striping 技术就是一种自动地将 I/O 的负载均衡到多个物理磁盘上的技术，在读数据时以并行的方式从不同的磁盘上获取数据块，从而获得非常好的性能。另外重删存储系统中数据的可靠性也是非常值得关注的，单纯地靠使用副本的机制一方面存储空间效率比较低，另一方面也不能确保数据失效时能够完整地恢复。目前提高存储系统可靠性普遍采用的是纠删码技术[50]，如 Data Domain 公司的 DDFS[24] 采用能够同时容双盘失效的 RAID-6 编码来纠错，现有的支持多盘容错的纠删码，如 Reed-Solomon 编码、Hover 编码、WEAVER 编码和 STAR 编码等都可以用来提高存储系统的可靠性；虽然纠删码比起副本机制能够获得更高的空间效率，但是需要大量的 CPU 计算开销。目前大多数提高可靠性的方案都是单纯从副本或纠删码的角度出发考虑可靠性和性能，因此如何结合副本和纠删码两种方案来提高系统的性能和可靠性将有待进一步研究。

5 重删系统分析与讨论

重删系统性能主要体现在 I/O 吞吐率、平均响应时间和重删率 3 个方面，重删系统所服务数据对象的数据构成特征和规模决定了该重删系统的性能考虑，不同应用场景的重删系统总是在这 3 者之间权衡。

表 1 各种重删系统的对比

重删系统	存储系统类型	重删时机	重删粒度
Venti[25] Symantec[23]	从存储	在线	固定块
REBL[41]	从存储	离线	变长块
Deep Store[42] TAPER[43]	从存储	在线	全文件分块
Foundation[44] Dedupv1[45] NEC HYDRAstor[36] GreenBytes[48] SiLo[47] EMC Cluster[46] ChunkStash[34] EMC Centera[21]	从存储	在线	变长块
IBM N Series[37] NetApp ASIS[38] EMC Celerra[39] StorageTank[40]	主存储	离线	固定块
DDE[14] I/O Deduplication[12] LiveDFS[11]	主存储	近线	固定块
iDedup[13] DBLK[17]	主存储	在线	固定块

表 1 对各种重删系统进行了对比和总结，迄今为止大部分重复数据删除研究都是为了提高备份和归档等从存储系统的重删能力。Venti[25] 是最早给出的在备份系统中实施重删的设计方案，由于归档和备份系统负载数据集的特点，很多数据都是冗余的，数据有明显的局部性特征，语义信息多，主要处理大数据级的流式写，对延迟不敏感，因而采用在线方式处理，但其对吞吐率比较敏感，因而可以采用 CDC 算法来实现变长分块以获得更高的重删率。很多论文中提出的，包括 Venti[25]、SiLo[47]、Sparse Indexing[26]、ChunkStash[34] 和 Foundation[44] 等都是属于备份和归档的存储系统，作者主要对归档吞吐率比较敏感，单个请求的延迟不是其主要的考虑因素。主存储重删强调针对业务负载实现重删，数据缺乏局部性特征。为了不影响前台用户体验，一般考虑响应时间敏感型，吞吐率和重删率不是主要考虑的因素。为了获得及时的响应，绝大多数采用离线方式实施重删，如 IBM N Series[37]、NetApp ASIS[38]、EMC Celerra[39] 和 StorageTank[40] 系统，有的主存储重删系统会牺牲重删率来换取用户体验，或采用一种近线的方式来实施重删，如 DDE[14]、I/O Deduplication[12] 和 LiveDFS[11]。为了进一步提升在线主存储环境重删系统的 FPindexing 查询优化，iDedup[13] 和 DBLK[17] 提出了很好的思路，为了尽量减少延迟时间，一般采用固定分块算法实施重删，因而主存储重删系统的重删效果相对于从存储重删系统不明显，随着各种加速技术的进一步提升，缩短了系统响应延迟时间，主重删系统重删时机逐渐从离线向近线或在线方式转变。近年来，随着虚拟化技术、集群技术和分布式文件系统的不断融合，云存储作为一个备份或归档目标允许用户异地备份或归档数据，因而云存储重复数据删除技术成为了目前重删系统研究的热点，如 DDE[14] 和 LiveDFS[11] 都是云存储环境下的重删实例：一方面可以将缩减的海量数据存储在云端；另一方面又能充分利用云端的计算资源缓解数据分块和数据指纹计算占用过多的 CPU 资源。将重复数据删除技术应用在云存储上可以发挥云存储虚拟化和云计算的优势，从而构建集群重删系统，提供多倍的吞吐能力和处理能力。

结束语 本文从计算机系统结构层次模型角度出发，对当前存储系统重复数据删除技术的研究现状和进展进行了深入分析和总结，包括重删系统的体系结构和所在层次的分析，以及与重删系统应用场景相关的分类体系，量化了重删系统的评价指标，分析了重删与压缩的关系，阐述了重删系统的关键操作技术，深入分析了数据规模行为引起重删系统性能瓶颈的原因，进而重点介绍和总结了从 I/O 路径的各个环节提升重删系统性能的各种加速技术，通过研究分析，重复数据删除技术实现了数据缩减，节省了能耗，保障了数据的可用性及可靠性，但是随着重删系统应用场景的变化和数据规模的不断扩大，原有加速技术的性能无法得以提升，随着网络存储技术的发展，重复数据删除系统面临许多挑战。总体来说，目前存储系统重复数据删除技术研究呈现出以下的发展趋势：

(1)随着云存储技术、专用处理器技术的进步，加之实时联机数据处理和存储预算压力的考虑，重复数据删除技术正面临从备份/归档存储向主数据存储环境的转变，数据构成特征缺乏局部性，随机小写比较多，读数据占绝大多数，因此如何提升主存储重删系统读请求的性能是重复数据删除技术急需解决的问题。

(2)随着存储系统容量的不断扩大，单节点存储系统存在单点性能瓶颈和单点失效的问题，重复数据删除技术面临从单节点向分布式集群环境的转变，如何协调和保障分布式环境下数据完整性、一致性和系统可扩展性仍然有待进一步研究。

(3)随着存储系统数据规模的不断扩大，数据集缺乏局部性特征，其 FP 查询管理开销将逐渐增加，严重地影响了重删系统的性能，现有的 FPindexing 查询优化技术有待进一步提

升，如何通过元数据的布局来提升重删系统性能将会成为重删技术研究亟待解决的问题。

(4)针对不同应用场景数据构成特征的复杂性，如何通过结合统计学和数据挖掘领域的各种技术，对数据特性进行充分的分析和挖掘，开发适合不同数据特征的检测技术还有待进一步深入研究。

由于重删系统良好的数据缩减率，近年来企业界和学术界对重复数据删除技术进行了深入的研究和讨论，上述的4个趋势和挑战将是未来一段时间重复数据删除技术的主流发展方向，随着云存储技术的不断发展，重复数据删除技术将在云存储环境下显示出新的生命力。

参 考 文 献

[1] Gartner:IT 数据量平均增长 40%至 60% [EB/ OL]. http://www.199it.com/archives/16863.html,2011-10-13/2012-06-05

[2] Greenan K M, Long D D E, et al. A spin-up save- d is energy earned: achieving power-efficient, erasurecoded storage[A]// Proceedings of the 4th Conference on Hot Topics in System Dependability[C]. Berkeley: USENIX, 2008: 4-4

[3] 郭平. 消除冗余解放容量[EB/OL]. http://www2.ccw.com.cn/07/0710/c/0710c24_4.html,2007-03 -19/2012-06-07

[4] McKnight J, Asaro T, et al. Digital archiving: end-user survey and market forecast 2006-2010 [EB/OL]. http://www.esg-global.com/research-reports/digital-archiving-end-user-survey-market-forecast-2006-2010/,2006-03-15/2012-06-07

[5] 敖莉，舒继武，李明强. 重复数据删除技术[J]. 软件学报，2010，21(5):916-929

[6] 付印金，肖侬，刘芳. 重复数据删除关键技术研究进展[J]. 计算机研究与发展，2012，49(1):12-20

[7] Lessfs: Open source data deduplication[EB/OL]. http://www.lessfs.com/wordpress/,2009-03-25/2012-07-05

[8] OpenDedup: Deduplication with OpenDedup [EB/ OL]. http://www.tuxlanding.net/deduplication- with-opendedup/,2011-07-13/2012-05-05

[9] FUSE: File systems using FUSE[EB/OL]. http:// fuse.sourceforge.net/,2012-08-23/2012-08-25

[10] SCST: GENERIC SCSI TARGET SUBSYSTEM FOR LINUX [EB/OL]. http://scst.sourceforge.net /index.html, 2012-03-20/2012-06-25

[11] Ng C-H, Ma Ming-cao, et al. Live Deduplication Storage of Virtual Machine Images in an Open-Source Cloud[A]// Proceedings of the 12th ACM/IFIP/USENIX International Conference on Middleware[C]. Berlin: Spinger-Verlag, 2011: 81-100

[12] Koller R, Rangaswami R. I/O Deduplication: Utilizing Content Similarity to Improve I/O Performance[J]. ACM Transactions on Storage, 2010, 6(3): 13

[13] Srinivasan K, Bisson T, et al. iDedup: Latency-aware, inline data deduplication for primary storage[A]// Proceedings of 10th USENIX Conference on File and Storage Technologies [C]. CA, USA: USENIX, 2012: 299-312

[14] Hong Bo, Plantenberg D, et al. Duplicate data elimination in a SAN file system[A]// Proceedings of the 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies[C]. College Park, MD: IEEE, 2004: 301-314

[15] 去重和压缩[EB/OL]. http://articles.e-works.net.cn/storage/article79873.htm, 2010-08-24/2012- 07-03

[16] Bolosky W J, Corbin S, et al. Single instance storage in windows 2000 [A]// Proceedings of the 4th USENIX Windows System Symposium[C]. Washington: USENIX, 2000: 13-24

[17] Tsuchiya Y, Watanabe T, et al. DBLK: Deduplication for Primary Block Storage[A]// Proceedings of the 27th IEEE Symposium on Mass Storage Systems and Technologies[C]. Piscataway: IEEE, 2011: 1-5

[18] Denehy T E, Hsu W W. Duplicate management for reference data[R]. IBM Research Report, RJ 10305(A0310-017). IBM Research Division, 2003

[19] Bobbarjung D R, Jagannathan S, et al. Improving Duplicate Elimination in Storage Systems[J]. ACM Transaction on Storage, 2006, 2(4): 424-448

[20] Understanding data deduplication ratios [EB/OL]. http://www.snia.org/sites/default/files/Understanding_Data_Deduplication_Ratios-20080718.pdf, 2008-07-18/2012-03-15

[21] Tan Yu-juan, Jiang Hong, et al. SAM: A Semantic-Aware Multi-Tiered Source De-duplication Framework for Cloud backup [A]// Proceedings of the 39th International Conference on Parallel Processing[C]. Los Alamitos, CA, USA: IEEE, 2010: 614-623

[22] Hash Collisions: The Real Odds[EB/OL]. http://www.backupcentral.com/mr-backup-blog-mainmenu-47/13-mr-backup-blog/145-de-dupe-hash-collisions.html, 2007-10-14/2011-12-05

[23] Guo F, Efstathopoulos P. Building a high performance deduplication system[A]// Proceedings of the 2011 USENIX Annual Technical Conference [C]. Berkeley: USENIX, 2011: 25-25

[24] Zhu Benjamin, Li Kai, et al. Avoiding the disk bottleneck in the Data Domain deduplication file system[A]// Proceedings of the 6th USENIX Conference on File and Storage Technologies [C]. Berkeley: USENIX, 2008: 269-282

[25] Quinlan S, Dorward S. Venti: A new approach to archival storage[A]// Proceedings of the FAST'02 Conference on File and Storage Technologies[C]. Berkeley: USENIX, 2002: 89-101

[26] Lillibridge M, Eshghi K, et al. Sparse indexing: Large scale, inline deduplication using sampling and locality[A]// Proceedings of the 7th USENIX Conference on File and Storage Technologies[C]. Berkeley: USENIX, 2009: 111-123

[27] Bhagwat D, Eshghi K, et al. Extreme Binning: Scalable, parallel deduplication for chunk-based file backup[A]// Proceedings of the 17th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems [C]. London: IEEE, 2009: 1-9

[28] Xia Wen, Jiang Hong, et al. Accelerating Data De- duplication by Exploiting Pipelining and Parallelism with Multicore or Manycore Processors [EB/OL]. http://static.usenix.org/events/fast12/ poster_descriptions/Xiadescription.pdf, 2012-3-2/2012-7-6

[29] Ousterhout J K, Agrawal P, et al. The case for RAMClouds: scalable high-performance storage entirely in DRAM[J]. Operating Systems Review, 2009, 43(4): 92-105

(下转第 42 页)

Linux 中的差异，然后提出并研究了实现 64 位 Windows ABI 虚拟化的关键问题，分析了在 Linux 用户空间和内核空间虚拟 64 位 Windows ABI 的优劣，最后基于 Linux 用户空间虚拟 64 位 Windows ABI 的解决方案，在开源软件 Wine 和 64 位 Ubuntu 10.04 基础上，实现了一种兼容 64 位 Windows 应用程序的操作系统 KgdLinux。实验测试表明，64 位 KgdLinux 已能支持部分 Win64 应用程序的运行，由此证明 64 位 Windows ABI 虚拟化方法是可行的。

参 考 文 献

[1] Vallee G, Naughton T, Engelmann C. System-Level Virtualization for High Performance Computing[C]//Proceedings of 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing. Washington: IEEE Computer Society, 2008: 636-643

[2] Hazelwood K. Process-Level Virtualization for Runtime Adaptation of Embedded Software[C]//Proceedings of the 48th Design Automation Conference. New York: ACM, 2011: 895-900

[3] Smith E, Nair R. 虚拟机：系统与进程的通用平台[M]. 安虹，张昱，吴俊敏，译. 北京：机械工业出版社，2009：8-14

[4] Susanta N, Tzi-Cker C. A Survey on Virtualization Technologies [R]. USA: ECSL, 2005

[5] 黄聪会，陈靖，罗樵，等. 面向二进制移植的虚拟化技术[J]. 计算机应用研究，2012，29(11)：4185-4188

[6] Microsoft. Microsoft PE and COFF Specification[EB/OL]. http:// msdn. microsoft. com/library/windows/hardware/gg463119. aspx, 2013-02-06

[7] 宿继奎，吴亚栋，吕必俊. 32 位到 64 位的移植[J]. 计算机应用与软件，2007，24(3)：174-176

[8] Wiki pedia. x86 calling conventions[EB/OL]. http://en. wikipedia. org/wiki/X86_calling_ conventions, 2011-03-20

[9] Bryant A R, Mills R F, Peterson G L. Software Reverse Engineering as a Sensemaking Task[J]. Journal of Information Assurance and Security, 2012, 6(6): 483-494

[10] Jing Yi-ming, Ahn G-J, Hu Hong-xin. Model-based Conformance Testing for Android[C]//Proceedings of the 7th International Workshop on Security. Japan: Springer, 2012: 1-18

[11] Russinovich M E, Solomon D A. Microsoft Windows Internals (the fifth edition)[M]. USA, Microsoft Press, 2009

(上接第 30 页)

[30] Bartizal D. Thomas Northfield. Solid State Drive Performance White Paper[EB/OL]. http://www. csee. umbc. edu/～squire/images/ssd2. pdf, 2008-3-24/2012-6-7

[31] Benefits of SSD vs. HDD [EB/OL]. http://www. amplicon. com/docs/white-papers/SSD-vs-HDD-white-paper. pdf, 2012-3-21/2012-7-8

[32] Solid State Drive vs. Hard Disk Drive Price and Performance Study [EB/OL]. http://www. dell. com/downloads/global/products/pvaul/en/ssd_vs_hdd_price_and_performance_study. pdf, 2011-5-1/2012-8-19

[33] Flash Memory Technology in Enterprise Storage Flexible Choices to Optimize Performance [EB/ OL]. http://www. itdialogue. com/wp-content/ uploads/2010/04/Flash-in-Enterprise-Storage. pdf, 2008-11-1/2012-3-4

[34] Debnath B, Sengupta S, et al. Chunkstash: speeding up inlinestorage deduplication using flash memory[A]//Proceedings of the 2010 USENIX Annual Technical Conference[C]. Boston: USENIX, 2010: 16-16

[35] The Art of Data Deduplication[OL]. http:// www. ecsl. cs. sunysb. edu/tr/rpe21. pdf

[36] Dubnicki C, Gryz L, et al. HYDRAstor: a scalable secondary storage[A]//Proceedings of the 7th USENIX Conference on File and Storage Echnologies [C]. Berkeley: USENIX, 2009: 197-210

[37] IBM System Storage N series Software Guide [OL]. http://www. redbooks. ibm. com/abstracts/sg247129. html, December 2010

[38] Alvarez C. NetApp deduplication for FAS and V-Series deployment and implementation guide[R]. Technical Report TR-3505. NetApp, 2011

[39] EMC. Achieving storage efficiency through EMC Celerra data deduplication[M]. White paper, Mar. 2010

[40] IBM Corporation. IBM white paper: IBM Storage Tank-A distributed storage system[M]. Jan. 2002

[41] Kulkarni P, Douglis F, et al. Redundancy elimination within large collections of files[A]//Proceedings of the 2004 USENIX Annual Technical Conference[C]. Boston: USENIX, 2004: 59-72

[42] You L L, Pollack K T, et al. Deep Store: An archival storage system architecture[A]//Proceedings of the 21st International Conference on Data Engineering[C]. Los Alamitos: IEEE, 2005: 804-815

[43] Jain N, Dahlin M, et al. TAPER: Tiered approach for eliminating redundancy in replica synchronization[A]//Proceedings of the 5th USENIX Conference on File and Storage Technologies [C]. Berkeley: USENIX, 2005: 281-294

[44] Rhea S, Cox R, et al. Fast, inexpensive content-addressed storage in foundation[A]//Proceedings of the 2008 USENIX AnnualTechnical Conference[C]. Berkeley: USENIX, 2008: 143-156

[45] Meister D, Brinkmann A. dedupv1: Improving deduplication throughput using solid state drives (SSD)[A]//Proceedings of the 26th IEEE Conference on Mass Storage Systems and Technologies[C]. Piscataway: IEEE, 2010: 1-6

[46] Dong W, Douglis F, et al. Tradeoffs in scalable data routing for deduplication clusters[A]//Proceedings of the Ninth USENIX Conference on File and Storage Technologies [C]. Berkeley: USENIX, 2011: 15-29

[47] Xia W, Jiang H, et al. Silo: a similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput[A]//Proceedings of the 2011 USENIX Annual Technical Con- ference[C]. Berkeley: USENIX, 2011: 26-28

[48] Zfs deduplication [EB/OL]. https://blogs. oracle. com/bonwick/entry/zfs_dedup, 2009-11-01 /2011. 11. 05

[49] Data striping[EB/OL]. http://en. wikipedia. org /wiki/Data_striping, 2012-08-15/2012-08-23

[50] Reed-Solomon Codes [EB/OL]. http://hscc. cs. nthu. edu. tw/～sheujp/lecture_note/rs. pdf