指导教师评定成绩
(五级制)：优
指导教师签字：陈成彰

SAN 集群文件系统中的分散式重复数据删除

摘要：

承载虚拟机的文件系统通常包含许多重复的数据块，这将导致存储空间的浪费和存储阵列缓存占用的增加。重复数据删除是通过存储每个唯一数据块的单个实例，并在该数据的所有最初来源之间共享从而解决这些问题。虽然对于那些具有集中组件的文件系统，重复数据删除是一种很好的理解。但我们将在分散的集群文件系统中研究它，特别是在 VM 存储的上下文中。

我们提出了一个用于实时集群文件系统的块级重复数据删除系统 DEDE，它不需要任何中央协调，容忍主机故障，并利用了现有集群文件系统的块布局策略。 在 DEDE 中，主机将自己写到集群文件系统的摘要保存在共享的磁盘日志中。每个主机定期独立地处理其锁定文件的摘要，将其与共享的块索引合并，并回收任何重复的块。 DEDE 可以在不了解文件系统实现的情况下使用一般的文件系统接口操作元数据。我们在 VMware ESX Server 的上下文中给出了我们的技术的设计、实现和评估。最终结果显示，对于实际的工作负载，空间方面减少了 80%，性能开销很小。

1 引言

目前使用 Storage Area Networks (SANs : 存储区域网络) 的部署正在增加，这是由于其具备以下优势：数据的普遍访问可来自任何地点、备份的方便性、供应的灵活性和集中式管理。 SAN 阵列通过同时为多台主机提供统一的数据访问，已经构成了现代数据中心的主干。虚拟化技术的发展进一步推动了这一趋势，虚拟化技术依赖于共享存储来支持诸如跨主机的虚拟机实时迁移等功能。

SAN 为多台主机提供对共享存储卷的直接 SCSI 访问服务。常规文件系统假定独占访问磁盘，并会很快损坏共享磁盘。为了解决这个问题，人们开发了许多共享磁盘集群文件系统，包括 VMware VMFS[21]、Redhat GFS[15]和 IBM GPFS[18]，

它们使用分布式锁定来协调多个主机之间的并发访问。

集群文件系统在虚拟化数据中心中扮演着重要的角色，在虚拟化数据中心中，多个物理主机各自运行潜在的数百个虚拟机，这些虚拟机的虚拟磁盘以常规文件的形式存储在共享文件系统中。SAN 以接近本机 SCSI 的性能为主机提供对 VM 磁盘共享存储的访问，同时还支持 VM 跨主机的实时迁移、负载平衡和故障转移等高级功能。

这些共享文件系统为检测和合并重复数据提供了极好的契机。它们会存储来自多个主机的数据，因此它们除了会包含更多的数据外，发生数据冗余的可能性也会更大。虚拟机共享存储是一个成熟的重复数据删除应用程序，因为公共系统和应用程序文件在虚拟机磁盘映像中重复，主机可以自动透明地在虚拟机之间和内部共享数据。 虚拟桌面基础设施(VDI)[24]尤其如此。在 VDI 中，桌面机器被虚拟化，合并到数据中心，并通过瘦客户机访问。我们的实验表明，一个真正的企业 VDI 部署可以在来自 VM 磁盘映像的重复数据上花费高达 80%的总存储占用空间。考虑到降低成本的愿望，这种浪费提供了减少虚拟机的存储需求的动力，尤其是对于 VDI。

现有的重复数据删除技术[1,3-5,8,14,16,17,26]依赖于集中的文件系统，需要跨主机通信来执行关键的文件系统操作，在带内执行重复数据删除，或者使用内容可寻址存储。所有的这些方法在我们的领域都有局限性。集中式技术很难扩展到除了磁盘本身之外没有集中组件的设置。现有的分散技术要求对大多数操作（通常包括读取）进行跨主机通信。在带内执行重复数据删除并写入实时文件系统会降低整个系统的带宽并增加 IO 延迟。最后，内容寻址存储（通过其内容哈希来寻址数据）也受到与昂贵的元数据查找以及空间局部性损失有关的性能问题的困扰[10]。

我们的工作解决了 VMware 的 VMFS 集群文件系统的分散设置中的重复数据删除问题。与现有的解决方案不同，DEDE 协调主机集群以协作地对活动的共享文件系统执行块级重复数据删除。它利用了共享磁盘作为系统中唯一的集中点的优势，并且不需要进行常规文件系统操作的跨主机通信，保留了 SAN 文件系统的直接访问优势。因此，能够停止重复数据删除的唯一故障是 SAN 本身的故障，没有 SAN 本身就没有文件系统可以重复数据删除。由于 DEDE 是一个用于主存储的联机系统，所有重复数据删除都是尽力而为，并作为后台进程执行，从写到带外，以最大限度地减少对系统性能的影响。最后，与其他系统不同，DEDE 在现有文件系统之上构建块级重复数据删除，并利用常规文件系统抽象、布局策略和块寻址。因此，重复数据删除在读取块时不引入额外的元数据 IO，并允许对没有重复项的块进行就地写入。

本文介绍了 DEDE 的设计。我们已经在 VMware VMFS 之上为 VMware ESX Server[23]实现了一个功能化的 DEDE 原型。 通过使用各种综合和实际的工作负载，这其中包括来自一个活动的公司 VDI 安装的数据，我们证实了 DEDE 可以在适度的性能开销下将 VM 存储需求降低 80%以上。

第二部分概述了我们系统的体系结构和我们的目标。 第三部分详细介绍了系统的设计与实现。 我们在第四部分对该系统进行了定量评估，然后在第五部分讨论相关的工作。 最后，我们在第六节中进行总结。

2 系统概述

DEDE 在集群设置中运行，如图 1 所示，其中多台主机直接连接到一个共享的 SCSI 卷并使用一个允许对存储在共享磁盘上的数据进行对称和协作访问的文件系统。DEDE 本身作为文件系统之上的一层在每个主机上运行，其利用文件系统块布局策略并且对写时复制（COW）块本地支持。在本节中，我们将简要概述我们的重复数据删除方法及其所依赖的文件系统支持。一个存储在共享文件系统上并为并发访问而设计的索引允许通过其内容哈希来跟踪文件系统中的所有已知块来进行有效的重复检测。

为了最大限度地减少对关键文件系统操作（如读写文件）的影响，DEDE 通过带外数据（OOB）的方式更新该索引，缓冲更新并以大规模、周期性批处理的方式应用这些更新。在这一过程中，DEDE 可以检测并消除自上次索引更新以来引入的重复项。这可以作为一项不常见的、低优先级的背景任务来完成，甚至可以在活动较少的时候进行调度。与将内容索引直接集成到文件系统存储管理中的内容可寻址存储等重复数据删除方法不同，DEDE 的索引仅用于标识重复块，在一般的文件系统操作中不起作用。

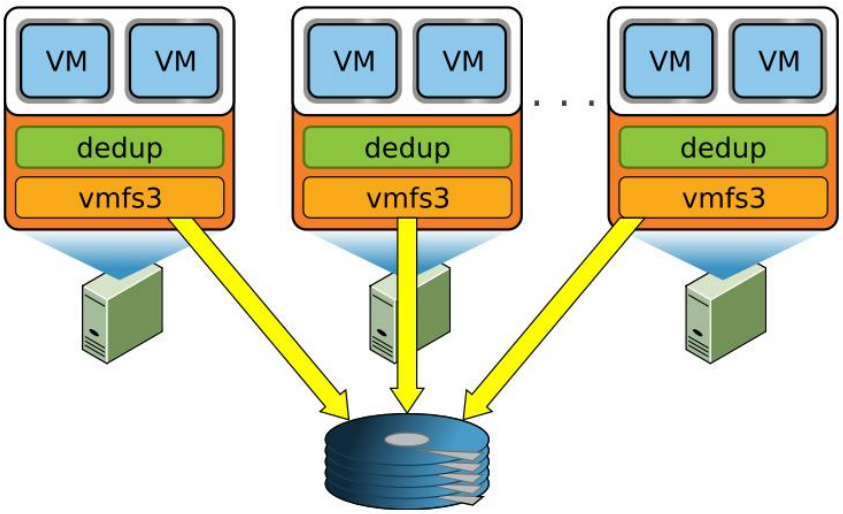


图 1：多台主机并发访问同一个存储卷的集群配置。
每个主机运行 VMFS 文件系统驱动程序(VMFS3)、
重复数据删除驱动程序（DEDUP）和其他进程，如 VM。

DEDE 在主机之间划分这个索引更新过程。每个主机监视自己对集群文件系统中文件的更改，并将最近修改的摘要存储在磁盘写日志中。这些日志包括带内计算的内容散列，日志作为块被写入磁盘。每个主机定期读取它所具有（或可以获得）独占访问权限文件的写日志，并更新共享索引以反映这些记录的修改。在此过程中，主机会发现并回收任何内容与以前索引的块内容相同的块。让每个主机参与索引更新过程可以让主机划分和分配重复数据删除带来的负担，而共享索引允许主机检测重复数据，即使它们是由单独的主机引入的。

带外索引更新意味着 DEDE 必须对陈旧索引条目具有弹性，这些陈旧索引条目不反映最近更新块的最新内容的。事实上，这在分散的环境中基本上是不可避免的，因为通信本身具有延迟。虽然这意味着 DEDE 在更新索引时通常必须验证块内容，但这种弹性有一个重要的含义：DEDE 的正确性不依赖于其监视文件系统每次写入的能力。这具有重要的性能优势。首先，写日志的更新不必与文件内容的更新保持崩溃一致，这既简化了容错，又允许主机缓冲写日志的更新，以最小化额外的 IO。其次，这允许用户在每个文件的基础上权衡写监视的 CPU 和内存开销，以获得文件系统的峰值性能。例如，用户可以简单地禁用高性能要求的或不太可能包含大量重复数据的 VM 的重复数据删除功能。最后，这允许写监视器在系统过载时停止工作。

因为 DEDE 在实时文件系统上运行，所以它专门针对唯一块（没有已知重复的块）进行了优化。与共享块不同，这些块在重复数据删除后保持可变。唯一块的可变性与 DEDE 对陈旧索引信息的弹性相结合意味着这些块可以就地更新，而

无需为副本分配空间或同步更新索引。因此，重复数据删除对写入唯一块的性能没有影响，这是一个非常理想的属性，因为这些恰恰是无法从重复数据删除中受益的块。

与其他一些与虚拟磁盘相关的重复数据删除工作[10,13]类似，DEDE 使用固定大小的块。与面向流的工作负载（如备份）不同，可变大小的块通常可以实现更好的重复数据删除[26]，我们的输入数据应该是块结构的，因为客户文件系统（如 ext3、ntfs)通常会将磁盘分成固定大小的 4 KB 或 8 KB 块。在早期的工作[12]和我们自己的测试结果（参见 4.1 节）中，我们也使用 4 KB 的块大小。

2.1 必要的系统抽象

大多数重复数据删除方法将重复数据删除和存储管理统一起来，完全取代了文件系统。相反，DEDE 作为 VMFS 之上的一个层运行，VMFS 是一个现有的文件系统。该层查找可能相同的块，并将它们标识给文件系统，然后文件系统负责将这些块合并到共享的、即写时复制的块中。

DEDE 要求文件系统是面向块的，并支持文件级锁定。文件系统块大小还必须与重复数据删除块大小对齐，但不幸的是，VMFS 默认的 1 MB 块大小不能满足这一要求。我们对 VMFS 唯一重要的更改是增加了对典型文件系统块大小（即 4 KB）的支持，这将在后面的第 2.2 节中详细说明。

最后，DEDE 需要块级别的写时复制支持，这是一个很好理解的特性，但 VMFS 支持的特性并不常见。具体地说，它需要一个不常见的比较和共享操作，该操作在验证两个块实际上是相同的后（使用按位比较或内容哈希验证），将两个块替换为一个写时复制块。尽管该操作具有特殊性，但它很自然地适合块级写时复制的结构，并且很容易添加到 VMFS 接口中。DEDE 只通过这个接口操作文件系统块，并且不知道底层的文件系统表示形式。

有两个值得注意的功能，DEDE 并不需要文件系统。首先，运行 DEDE 的主机从不修改它们没有独占锁的文件元数据，因为这样做需要跨主机同步，并且会使每个主机的元数据缓存复杂化。因此，发现两个文件之间有重复块的主机，如果其中一个文件被另一个主机锁定，则不能简单地将两个文件修改为指向同一个块。相反，当 DEDE 检测到被不同主机锁定的文件之间的重复时，它使用包含合并请求的第三个文件作为中介。 一个主机创建包含对重复数据删除块的 COW 引用的合并请求，然后将合并请求文件锁的所有权传递给另一个主机，另一个主机反过来用对合并请求携带的块的引用替换其文件中的块。

其次，DEDE 不要求文件系统公开块地址的表示形式。与任何常规应用程序一样，它只通过块在某个锁定文件中的偏移量间接地引用块，文件系统可以将其解析为块地址。这限制了索引的设计，因为它不能简单地直接引用索引块。 但是，这一限制简化了我们的总体设计，因为要求文件系统将块地址公开在文件系统自己的数据结构之外会干扰其释放和迁移块的能力，并可能导致悬空指针。 更糟糕的是，为直接操作块而引入的任何操作都将与文件级锁定和主机元数据缓存发生冲突。

DEDE 引入了一个 virtual arena 文件取代块地址引用块。这是文件系统中的一个常规文件，但它仅由共享块的 COW 引用组成，这些共享块至少存在于另一个文件中的。该文件充当系统中所有共享块的替代视图：DEDE 仅通过 virtual arena 文件中共享块的偏移量来标识共享块，文件系统可以使用常规地址解析在内部将其解析为块地址。

因为 DEDE 构建在底层文件系统上，所以它继承了文件系统的块放置策略。如果基础文件系统保持文件块的顺序，则在重复数据删除后，块通常会保持顺序。共享块可能相对于至少一个文件中的其他块是顺序的，并且共享块的公共序列可能相对于彼此保持顺序。 此外，唯一块的位置和顺序完全不受重复数据删除过程的影响； 因此，重复数据删除不会影响单个唯一块的 IO 性能，因为它们不需要复制，并且它保持跨唯一块跨度的顺序 IO 性能。

2.2 VMFS

DEDE 中的许多设计决策都受到其底层文件系统 VMFS 设计的影响。VMFS 是一种无协调器的群集文件系统[21]，旨在允许主机协同维护存储在共享磁盘上的文件系统。在本节中，我们将简要介绍 VMFS 如何处理和管理对其资源的并发访问，以便为 DEDE 的设计提供更好的上下文。

VMFS 将共享磁盘组织成四个不同的资源池：索引节点、指针块、文件块和子块。 索引节点和指针块与传统 UNIX 文件系统中的作用大致相同，存储每个文件的元数据和指向包含实际文件内容的块的指针。 文件块和子块都存储文件内容，但大小不同，如下文所述。 这些池之间的划分目前在格式化时是固定的，只能通过添加更多的存储来扩展，尽管这不是一个基本的限制。在每个池中，资源被分组为群集。每个集群的头维护关于其包含的所有资源的元数据；最重要的是，这包括每个单独资源的引用计数，并跟踪哪些资源是空闲的，哪些资源是分配的。

为了支持多个主机对文件和资源数据的并发访问，VMFS 使用分布式锁管理器。与使用 IP 网络进行同步的大多数群集文件系统不同，VMFS 使用磁盘锁完全通过共享磁盘本身同步所有文件系统访问。VMFS 使用基于 SCSI-2 的 LUN 保留

来保护读-修改-写关键节，从而确保对磁盘锁结构本身的原子访问。除了利用存储区域网络的可靠性之外，使用相同的方法访问文件系统状态和同步状态可以防止基于 IP 的锁管理器典型的“分裂大脑”问题，在这种问题中，多个主机可以访问文件系统状态，但不能相互通信锁定决策。

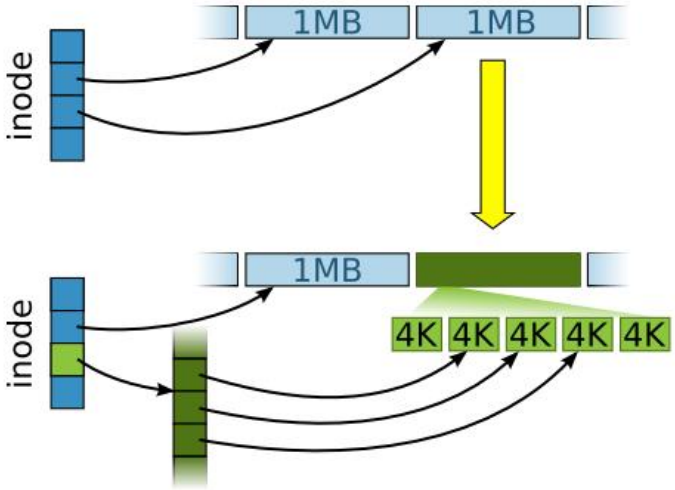


图 2：混合块大小允许将任何 1 MB 的文件块划分为 256 个独立的 4 KB 子块。

VMFS 通过将粗粒度锁与覆盖文件所有元数据（其 inode 和 pointer 块）以及包含文件内容的所有文件块和子块的每个文件相关联，来保护文件数据，避免并发访问。 VMFS 中的文件往往会被锁定很长时间（例如，只要 VM 上电，VM 的磁盘文件就会被锁定）。DEDE 通过根据哪些主机持有哪些文件锁来划分重复数据删除过程，从而尊重文件系统锁定。

VMFS 使用 per-cluster 锁来保护资源元数据。因此，资源的分配和解除分配必须锁定包含所涉及的任何资源的所有群集。每个集群打包的资源数量反映了锁定开销和交叉集群锁争用之间的权衡。更高的集群密度允许主机以更少的锁操作更多的资源，但代价是增加锁争用。由于 DEDE 比典型的 VMFS 使用更强调子块资源池，我们将子块集群密度从每个集群 16 个增加到 128 个资源，但在其他方面使用默认的 VMFS 密度。

VMFS 维护用于存储文件内容的两种不同的资源类型：文件块和子块。 VMFS 中的文件大小通常适合双峰分布。虚拟机磁盘和交换文件通常是几千字节，而配置和日志文件往往是几千字节。因此，VMFS 使用 1 MB 文件块来减少大文件的元数据开销和外部碎片，而对于小文件，VMFS 使用较小的子块来最大限度地减少内部碎片。 DEDE 必须能够寻址单个的 4 KB 块，以便 COW 共享它们，因此我们用 4 KB 子块配置 VMFS。此外，不是简单地回避 1 MB 块的效率并将所有文件内容存储在 4 KB 块中，而是扩展 VMFS 以支持混合块大小，如图 2 所示，这样当需

要共享重复块时，DEDE 可以处理文件的单个 4 KB 块，但在可能的情况下，仍然将文件的独特区域存储在高效的 1 MB 块中。 此更改引入了一个可选的附加指针块级别，并允许将任何文件块大小的区域分成 256 个独立的 4 KB 块，这些块依次加到原始文件块中。 这可以根据重复数据删除决策动态地对任何 1 MB 块进行，并为其他数据保留完整和有效的地址解析。

除了这些不寻常的块大小之外，VMFS 还支持许多其他不常见的特性。对 DEDE 来说，最重要的是支持块级的写时复制(COW)。可以从多个指针块引用每个文件或子块资源，从而允许在多个文件中的多个位置之间共享相同的数据。对共享资源的每个引用都用一个 COW 位标记，这表明任何写入资源的尝试都必须在新分配的资源中创建一个私有副本，然后写入该副本。值得注意的是，这个 COW 位与指向资源的每个指针相关联，而不是与资源本身相关联。否则，每个写操作都需要使用集群锁来检查目标块的 COW 位，即便该块不是 COW。然而，结果就是在两个文件之间共享块需要两个文件上的文件锁，即使只有一个引用会更改。因此，DEDE 必须对所有跨主机合并操作使用合并请求。

VMFS 构成了 DEDE 的底层基础，并处理关键的正确性需求，如专门化 COW 块和验证潜在的重复，使 DEDE 能够专注于重复检测。virtual arena 和合并请求允许 DEDE 在不了解文件系统表示的情况下实现对文件系统结构的复杂、分散的操作，而不是只使用一些通用接口。

3 设计与实现

在本节中，我们详细介绍了 DEDE 尽力而为的写监控子系统的设计和实现，以及带外索引和重复消除过程。

3.1 写监视

每个主机都运行一个写监视器，如图 3 所示，该监视器由一个轻量级内核模块(DEDUP)和一个用户空间守护进程(DEDUPD)组成，前者监视该主机对文件系统中文件的所有写操作，后者将这些信息记录到存储在共享文件系统中的日志中。写监视器是系统中位于文件系统 IO 关键路径的唯一部分，因此写监视器本身必须尽可能少地增加磁盘 IO 和 CPU 开销。

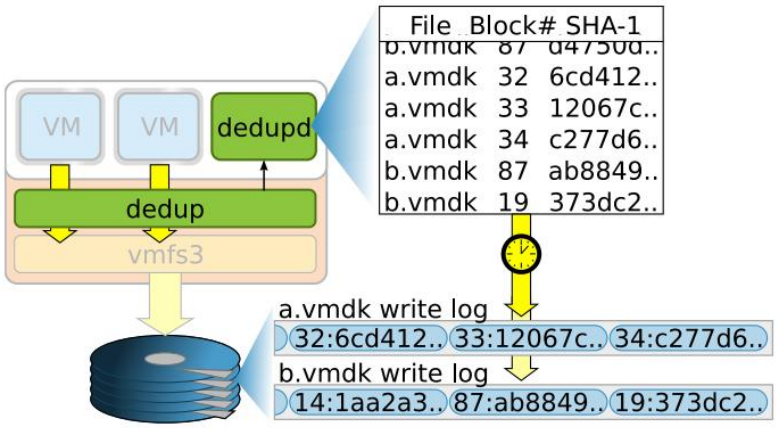


图 3：只有一个轻量级内核模块位于 IO 关键路径中，当块仍在内存中时，它会适时地计算块的哈希值。用户空间守护进程（dedupd）定期向磁盘刷新写日志。重复检测和消除发生在带外。

内核模块为用户空间守护进程提供一个修改流，为主机执行的每次写操作指明：被修改的文件、写操作的偏移量，和所有修改块的 SHA-1 哈希值。虽然监视器的带内 CPU 开销实际上可以通过懒惰地计算这些散列（例如，在索引时）来消除，但这将需要从磁盘中读取修改后的块，从而导致大量额外的随机 IO。 相反，我们选择通过在块位于内存中时计算这些散列来消除额外的 IO，尽管运行时 CPU 开销和重复数据删除时 IO 开销之间的折衷可以通过用户定义的策略动态设置。用户空间守护进程按文件划分修改流，聚合对同一块的重复写入，并将此信息缓冲在内存中，定期将其刷新到与每个常规文件相关联的单个写入日志文件中。这些写日志存储在共享文件系统本身上，因此即使主机出现故障或转移了文件锁的所有权，系统中的任何其他主机都能够读取该主机生成的日志，并将有关修改块的信息合并到索引中。

守护进程可以安全地在内存中缓冲修改流，因为索引更新过程被设计为处理陈旧的信息。如果不这样做，写日志必须与磁盘上的文件状态保持一致，并且对文件系统的每次逻辑写操作至少会导致对磁盘的两次写操作。相反，缓冲允许我们的系统将超过 150 MB 的文件块的写入转化为对日志文件的一次不常见的 1 MB 顺序写入。这是写监视器引入的唯一附加 IO。

类似地，在部分块写入的情况下，我们依赖写监视的尽力而为特性来最小化 IO。如果对文件系统的写入没有覆盖整个块，监视器将忽略该写入，而不是从磁盘读取块的其余部分来计算其哈希值。实际上，当写操作来自虚拟机时，这很少是一个问题，因为客户操作系统通常会写整个客户文件系统块，这些块通常至少为 4 KB。

每个文件都可以启用或禁用写监视。如果某些 VM 的性能非常关键，不会引起写监视的开销，或者如果系统管理员先验地知道某个 VM 的重复率很小，则可以选择不进行重复数据删除。

3.2 索引

共享磁盘上索引通过其内容哈希来跟踪文件系统中所有已知块。 如第 2 节所述，每个主机独立地更新该索引，根据用户定义的策略（例如，仅在非高峰时段），从大量写日志中合并关于最近块修改的信息。 索引中的内容哈希与最近修改块的内容哈希之间的匹配表示可能存在重复，必须对其验证并用共享块的写时复制引用进行替换。

索引充当从散列到块位置的有效映射。 因为 DEDE 对唯一块（只有一个引用的块）和共享块（有多个引用的块）的处理不同，所以每个索引条目同样可以处于两种状态之一，即 Unique（H,f,o）和 Shared（H,a）。索引项通过其包含文件的 inumber f 及其在该文件中的偏移量 o 来标识具有哈希 H 的唯一块。因为索引更新是带外的，而且唯一块是可变的，所以这些条目只是关于块哈希的提示。 因此，由于一个可变块的内容自上次索引以来可能已经发生了变化，因此在用另一个块删除它之前必须验证它的内容。 另一方面，共享块被标记为 COW,因此它们的内容保证是稳定的。索引通过其在索引 virtual arena 中的偏移量 A 来标识每个共享块，将在下一节中讨论。

3.2.1 virtual arena

当发现重复的内容时，DEDE 会回收所有重复的内容(只会保留一个)，并在文件之间共享该块的写时复制。因为主机可以在不更新索引的情况下，随时为每个文件创建共享块的可变副本，我们不能简单地通过重复数据删除文件中的位置来识别共享数据块，就像识别唯一的数据块一样。索引需要一种方法来引用这些共享块，并且这种方法在重复数据删除的文件中发生引用转移时仍然稳定。如前所述，DEDE 不能简单地在索引中存储原始块地址，因为从文件系统暴露这些地址会带来许多问题。相反，我们引入了一个 virtual arena 文件作为一个额外的间接层，它为共享块提供了稳定的标识符，而不违反文件系统抽象。

virtual arena 是一个常规文件,但不像典型的文件,它没有任何数据块分配专门给它(因此,它是虚拟的)。相反，它充当文件系统中所有共享块的备用视图。这样，它与其他重复数据删除系统（如 Venti[16]）中使用的 arena 非常不同，后者存储由内容地址寻址的实际数据块。

为了使块共享，主机从 virtual arena 文件中引入对该块的额外 COW 引用，使用允许块在任何两个文件之间共享的相同接口。除了未收集的垃圾块之外，virtual

arena 仅消耗其 inode 空间和任何必要的指针块。此外，这种方法利用了文件系统的块放置策略：将块添加到虚拟区域而不在磁盘上移动它，因此它很可能与原始文件保持顺序。

然后，索引可以通过其在 virtual arena 文件中的偏移量引用任何共享块，文件系统可以在内部将其解析为块地址，就像对任何其他文件一样。virtual arena 文件的 inode 和指针块结构精确地形成了从索引所需的抽象、稳定的块标识符到文件系统所需的块地址的必要映射。

3.2.2 索引在磁盘上的表现效果

DEDE 将索引存储在磁盘上，作为按哈希排序的索引条目列表。由于 DEDE 总是大批量更新索引，并且由于更新的哈希值没有空间局部性，我们的更新过程只需线性扫描整个索引文件和排序的更新列表，合并两个列表以生成一个新的索引文件。 尽管这种方法简单，但即使更新批大小很小，它也优于为单个随机访问优化的常见索引结构（例如，哈希表和 B 树）。 给定平均索引条目大小为 b 字节，顺序 IO 速率为每秒 s 字节，平均寻道时间为 k 秒，使用随机访问应用 U 更新所需的时间为 Uk，而顺序扫描和重写 I 条目索引的时间为 2I b/s。 如果批大小与索引大小的比值超过 U/I=2b/sk，则顺序重写整个索引比单独应用每个更新更快。例如，假设条目大小为 23 字节，并且假设 SAN 数组具有 150 Mb/s 和 8 ms 的查找能力，批处理大小只需超过索引大小的 0.004%。此外，主机将索引更新推迟到批处理大小超过索引大小的某个固定部分（至少 0.004%)，因此无论索引大小，摊销更新成本都保持不变。

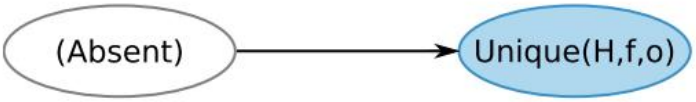
为了实现索引的访问随共享文件系统的主机数量而扩展，同时仍然依赖文件锁定来防止索引访问冲突，主机将索引分解成多个文件，每个文件表示哈希空间的某些细分。一旦主机更新碎片的时间超过某个阈值，下一个更新碎片的主机将把碎片覆盖的哈希范围一分为二，并在单独的文件中写出两个子碎片。 这种技术反映了可扩展哈希[6]的技术，但是我们没有限制哈希桶的大小，而是限制了更新它们所需的时间。结合文件锁定，这将动态调整索引的并发性以匹配需求。

3.3 索引和重复消除

由于索引更新过程合并了关于写入日志中记录的最近修改的块的信息，除了检测指示潜在重复的哈希匹配之外，它还执行实际的 COW 共享操作来消除这些重复。 重复消除过程必须与索引扫描过程交织在一起，因为块内容验证的结果会影响得到的索引条目。

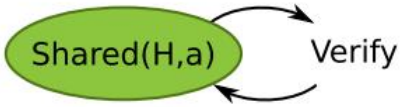
为了更新索引，主机按哈希对最近的写记录进行排序，并与索引中的排序项一起遍历这个排序的写记录列表。两者之间的匹配哈希表示潜在的重复，根据匹配

索引项的状态，处理方式不同。图 4 给出了匹配索引项在当前状态下可以经历的所有可能转换的概述。

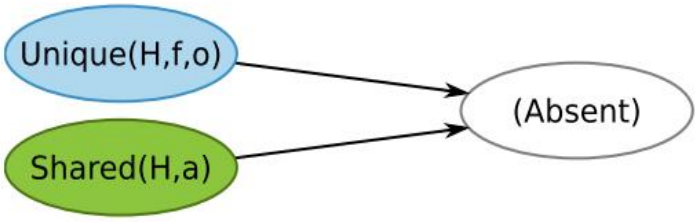


(a)　当文件 F 中偏移量为 o 的块的散列 H 不在索引中时，添加新的唯一条目。

(b)当发现哈希 H 的第二次出现并且块的内容通过验证时，
我们将其放置在 virtual arena 中，并将索引条目设置为共享条目。



(c)当发现共享块的重复时，我们验证其内容，并用对现有共享块的引用替换该块



(d)当索引过程找到具有不同散列的该块的写入记录时，唯一条目被垃圾收集。
当只剩下来 virtual arena 的引用时，共享条目将被垃圾收集

图 4：索引条目的所有可能更新。

当 DEDE 检测到潜在的重复时，它依赖于 2.1 节中描述的文件系统的比较和共享操作，以原子性地验证块的内容没有改变，并用对另一个块的 COW 引用来替换它。基于用户指定的策略，这种验证可以通过读取潜在重复块的内容并确保其与预期的哈希值匹配（即按哈希值比较）来完成，也可以通过读取两个块的内容并执行逐位比较（即按值比较）来完成。 如果后一种策略有效，哈希冲突会降低 DEDE 的有效性，但不会影响其正确性。此外，由于散列仅用于查找潜在的重复项，

如果 SHA-1 被破坏，DEDE 具有独特的能力，可以通过简单地重建其索引来优雅地切换到不同的散列函数。如果主机能够证明块没有改变，则可以完全跳过内容验证步骤； 例如，如果它在生成写记录后的整个期间都持有包含该块的文件的锁，并且没有删除任何写记录。虽然这是一个相当特殊的条件，但它通常在 DEDE 的目标设定中得到满足，因为 VM 磁盘上的锁通常保持很长时间。

3.3.1 单主机索引

我们首先解释索引更新过程，假设只有一台主机独占访问文件系统。在单个主机设计中，主机可以修改任何文件的元数据。我们将在下一节中提出这一假设，我们将扩展该进程以支持多个主机。

任何在索引中没有相应哈希的写记录都表示一个新的唯一块。 尽管这个写记录可能过时，因为唯一块的索引条目只是提示，但安全的做法是简单地将新的唯一块添加到索引中，而不验证块的内容，执行 absent-to-unique 的转换，如图 4(a)所示。在处理新的唯一块时，这种对索引的单次顺序缓冲写入是唯一的 IO。

当写记录的散列对应于唯一块的索引项时，主机尝试共享两个块（在进程中释放其中一个），并将索引项升级为引用共享块。这种唯一共享的转换如图 4(b)所示。但是，由于写记录和索引项都可能过时，所以主机必须在实际共享这两个块之前验证它们的内容。假设验证成功，文件系统用共享块替换这两个块，主机将该块插入 virtual arena，并升级索引项以引用新的共享块。

最后，如果写记录的哈希与共享块的索引项匹配，那么主机会尝试消除这个新检测到的潜在重复，执行 shared-to-shared 转换，如图 4(c)所示。因为写记录可能是陈旧的，所以它首先验证潜在重复的内容没有更改。如果成功，则释放该块，并将对该块的引用替换为通过 virtual arena 找到的对共享块的引用。

3.3.2 多主机索引

将索引更新过程扩展到多个主机，我们不能再假设一个主机可以不受限制地访问每个文件。 特别是，主机只能在它们持有独占锁的文件中验证块和修改块指针。因此，索引必须跨主机分布。 同时，考虑到通过共享磁盘进行通信的开销，我们必须尽量减少主机之间的通信。 因此，即使所涉及的块由不同的主机使用，也可以在没有任何阻塞主机之间通信的情况下完成块的共享。

在多主机设置中，写入日志根据每个主机具有（或可以获得）独占访问的文件在主机之间进行划分。虽然这是必要的，因为主机只能处理它们持有独占锁的文件中的写记录，但它也可以在主机之间分配重复数据删除的工作负载。

absent-to-unique 的转换和 shared-to-shared 的转换在多主机设置中与在单主机设置中相同。 将新的唯一块添加到索引中既不需要块验证，也不需要修改块指针。共享到共享转换只验证和重写当前写日志引用的文件中的块，处理写日志的主机

必须对该文件具有独占锁。

但是，unique-to-shared 的这种转换会由于包含索引引用的惟一块的文件可能被处理写记录的主机以外的其他主机锁定，情况会因此而变得复杂。虽然此主机可能没有访问索引块的权限，但它确实有访问写入日志所引用的块的权限。主机验证此块的内容，并通过将其添加到 virtual arena 并相应地升级索引条目来将其提升为共享块。但是，为了回收最初索引块，主机必须使用关联的合并请求文件将此重复数据删除的机会传递给对包含最初索引块文件持有独占锁的主机。更新索引的主机对包含最初索引块文件发出合并请求。此请求不仅包含唯一块的偏移量，还包含对共享块的另一个 COW 引用。主机定期检查对它们具有独占锁的文件的合并请求，验证它们找到的任何请求，并合并通过验证的块。合并请求中对共享块的 COW 引用允许主机在不访问 virtual arena 时处理请求。

3.3.3 垃圾回收

当主机扫描索引查找哈希匹配时，它还会垃圾收集未使用的共享块和陈旧的索引项，如图 4(d)所示。对于索引中的每个共享块，它检查该块的文件系统引用计数。如果块不再使用，它将只有一个引用（来自 virtual arena），这表明它可以从 virtual arena 移除并被释放。 实际上，这实现了一种简单形式的弱引用，而无需修改文件系统语义。 此外，这种方法允许 virtual arena 在垃圾收集有机会删除未使用的块之前充当牺牲品缓存。

唯一块不需要释放，但它们可以留下陈旧的索引项。主机通过删除引用主机正在处理的任何写记录中的任何块的任何索引项来垃圾收集这些数据。如果存在删除的写记录，这可能不会删除所有过时的索引项，但它将确保每个唯一块最多有一个索引项。 在这种情况下，涉及带有陈旧索引项的块的任何稍后写入或潜在的重复发现都将删除或替换陈旧项。垃圾收集过程还检查文件的截断和删除，并删除任何适当的索引项。

4 评估

在本节中，我们将介绍使用各种微基准和实际工作负载对我们的重复数据删除技术进行评估的结果。我们从 4.1 节开始，使用来自一个真实公司 VDI 部署的数据，通过实验和分析展示重复数据删除可以节省的空间以及由此带来的空间开销。我们还与链接克隆进行了比较，这是一种节省空间的替代方法。

我们已经实现了 DEDE 在 VMware VMFS 上的一个功能原型。虽然我们还没有花太多的时间对它进行优化，但还是值得研究一下它的基本性能特征。 在第 4.2

节中，我们介绍了写监视和重复数据删除对文件系统的其他更改对运行时性能的影响，以及通过改进缓存局部性获得的运行时性能。最后，我们在 4.3 节中查看重复数据删除过程本身的性能。

4.1 虚拟磁盘分析

为了评估重复数据删除在我们的 VDI 目标工作负载段中的有用性，我们分析了来自一个生产企业 VDI 集群的虚拟磁盘，该集群为 32 个 VMware ESX 主机群上的大约 400 个用户的桌面 VM 提供服务。从中，我们随机选择 113 个 VM 来分析重复块，总计 1.3TB 数据（不包括完全由空字节组成的块）。 每个用户 VM 只属于来自市场营销或会计等非技术部门的单个公司用户。这些虚拟机已经使用了 6 到 12 个月，它们都源于一组标准化的 Windows XP 映像。根据我们的经验，这是大多数企业 IT 组织的典型情况，它们限制操作系统的变化以控制管理和支持成本。

图 5 显示了使用 4 KB 和 1 MB 之间的重复数据删除块来减少 VDI 场的存储空间。 正如所料，VDI VM 具有高度的相似性，对于 4 KB 的块大小，VDI VM 的存储占用空间减少了大约 80%，对于 1 MB 的块大小，VDI VM 的存储占用空间从对数下降到大约 35%。 4 KB 块大小的重复数据删除将原来的 1.3TB 数据减少到 235 GB。 考虑到小块大小的显著优势，我们选择为 DEDE 使用默认的 4 KB 块大小。但是，8 KB 块大小可以提供更小的元数据存储和缓存开销，这是一个合理的论点。我们正在探索这以及动态块大小选择作为未来的工作。

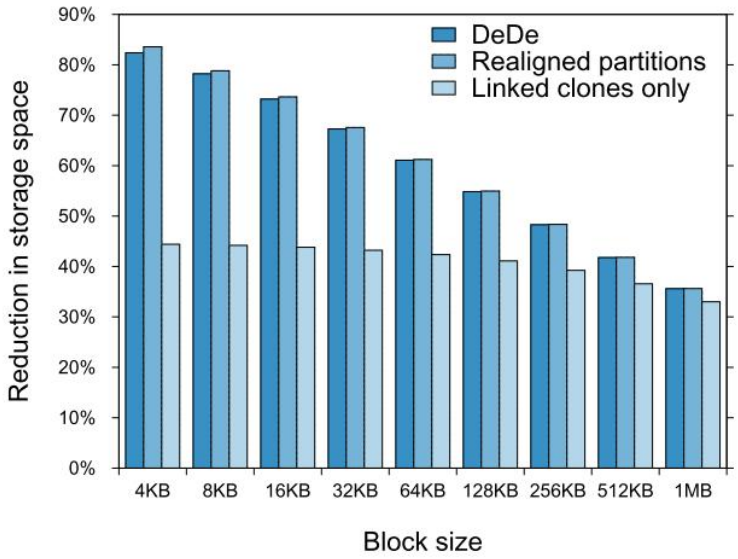


图 5：在不同的块大小和不同方法下的重复。

数据来自 113 个 Windows XP VM 的生产 VDI 部署。

图 6 显示了相同数据的 CDF，根据重复数据删除后对文件系统中每个块的引用数，详细说明了单个块的重复计数。 例如，在 4 KB 的块大小下，94%的重复数据删除块被文件系统引用 10 次或更少（等效地，6%的重复数据删除块被引用超过 10 次）。因此，在原始数据中，大部分数据块被重复了少量次，但有一个很长的尾巴，其中一些数据块被重复了很多次。在 4 kb 分布的最高峰时，一些块被复制了 100,000 次以上。这些块中的每一个单独表示存储重复数据所浪费的 400 MB 空间。总的来说，这些数据显示了在 VDI 环境中从重复数据删除中节省空间的潜力。

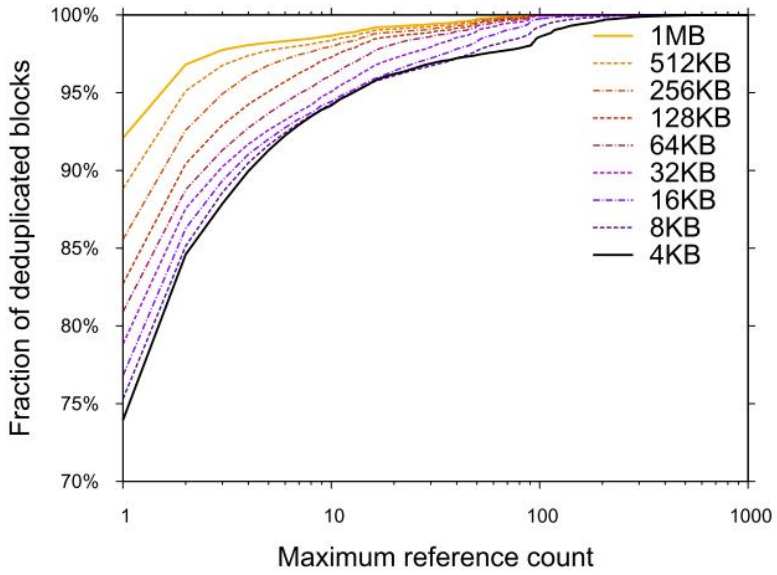


图 6：块重复计数的 CDF。有几个块出现超过 10 万次。

数据来自同一个部署，如图 5 所示。

4.1.1 空间管理开销

虽然 DEDE 减少了文件数据所需的空间，但它需要额外的空间用于索引和混合块大小引入的额外元数据。 对于我们的 VDI 数据集，在 4 KB 的块大小下，这些额外的数据总计为 2.7GB，仅比重复数据删除文件数据增加 1.1%的开销。

索引使用了 1.5GB 的开销，其中 194 MB 是 virtual arena 的文件系统元数据（指针块）。索引的大小与重复数据删除的数据的大小成线性比例，因为每个重复数据删除的块都有一个索引条目。然而，它的相对开销确实随着唯一块与共享块的比例而变化，因为共享块需要 4 个字节来定位加上 virtual arena 元数据，而唯一块需要 12 个字节，超过每个条目的头和哈希平均所需的 18 个字节。然而，即使在最坏的情况下，该指数也只代表 0.73%的开销。

在重复数据删除之前，由于跟踪 VMFS 的 1 MB 文件块的效率，文件元数据（索引节点和指针块）仅表现出 0.0004%的开销。重复数据删除后，每个被划分为子块的 1 MB 块需要一个新的指针块，每个指针块的大小为 1 KB。结果，在重复数据

删除后，元数据开销增加到 0.49%，即总计 1.1GB 的数据。虽然这是一个巨大的增长，但元数据仍然是整个空间的一个非常小的部分。

4.1.2 分区对齐问题

我们将磁盘划分为固定大小的块的方法对这些磁盘上的数据对齐很敏感。不幸的是，由于历史原因，在普通 PC 系统上由 fdisk 之类的实用程序创建的分区表的第一个分区的起始地址比 4 KB 的边界短 512 字节，这反过来会导致所有逻辑文件系统块跨越 4 KB 的磁盘块边界。 众所周知这会带来负面的性能影响[22]，特别是对于存储阵列缓存，它被迫为每个请求的文件系统块提取两个块。我们最初担心这种分区不对齐可能会对重复数据删除机会产生负面影响，所以我们通过将所有虚拟磁盘移动 512 字节来“修复”VDI 数据的对齐。 图 5 比较了重复数据删除的结果，结果表明，分区对齐实际上对已实现的重复数据删除影响很小。虽然这对于成熟的客户文件系统仍然是一个问题，但如果有必要，可以在虚拟化环境中通过填充虚拟磁盘映像文件来将客户文件系统块与主机文件系统块重新对齐来解决这个问题。

4.1.3 重复数据删除与连锁克隆的比较

与重复数据删除相比，链接克隆是一种更简单、节省空间的替代方案，在重复数据删除中，单个用户 VM 最初被构造为金主 VM 的块级 COW 快照。这使用了与 DEDE 相同的 COW 机制，但是所有共享都发生在 VM 创建期间，用户 VM 映像严格地与基磁盘分开，并且随着时间的推移彼此分开。

为了比较链接克隆与完全重复数据删除的有效性，我们在 VDI 数据集上模拟了链接克隆的结构化共享。这种比较必然是不完美的，因为我们既不能访问基础磁盘，也不能访问 VDI VM 的原始信息，但它确实产生了链接克隆所需总空间的下限。该分析使用了我们的常规重复数据删除算法，但将其限制为仅在两个文件中位于相同偏移量的块进行重复数据删除，这与用户磁盘的情况是近似的。

%-Sequential	Baseline			DeDe		
	T (MB/s)	*L* (ms)	CPU	*T* (MB/s)	*L* (ms)	CPU
100%	233	8.6	33%	233	8.6	220%
0%	84	24	16%	84	24	92%

表 1：纯 IO 工作负载上带内写监视的开销。

结果是 Iometer 向 5 GB 虚拟磁盘发送 32 个未完成的 64 KB IOS 的吞吐量(T)和延迟(L)。

CPU 列表示相对于单个核的已利用处理器时间。

图 5 比较了链接克隆与 DEDE 实现的节省，同样是在不同的 COW 块大小下。

链接克隆最大减少了 44%的空间，将 1.3TB 的原始数据减少到 740 GB，存储需求比完全重复数据删除大三倍以上。

4.2 重复数据删除的运行时性能

DEDE 主要在带外运行，对于访问没有从重复数据删除中受益的块不会产生任何影响。它还可以通过减小存储数组缓存的工作集大小来提高某些工作负载下的文件系统性能。然而，对于重复数据删除块的访问，带内写监视以及 COW 块和混合块大小的影响会影响文件系统的正常性能。除非另有说明，我们对重复数据删除的运行时影响的所有测量都是在 EMC Clariion CX3-40 存储阵列的 400 GB 5 磁盘 RAID-5 卷上存储的虚拟机中使用 Iometer[9]进行的。

4.2.1 带内写监视的开销

由于 DEDE 的设计对删除的写日志条目有弹性，如果系统过载，我们可以根据用户指定的策略放弃或推迟带内哈希计算的工作。 但是，如果启用了写监视，DEDE 对每个写 IO 执行的哈希计算可能会带来相当大的开销。

为了了解最坏的情况，我们在一个 5 GB 虚拟磁盘上运行了一个写密集型工作负载，计算量很小。 表 1 显示，这些最坏情况下的影响可能是显著的。例如，对于 100%顺序、100%写工作负载，在相同的吞吐量水平下，CPU 开销是正常情况下的 6.6 倍。 但是，由于 VMware ESX Server 将 IO 发出路径代码的执行（包括哈希计算）卸载到空闲处理器内核上，因此该工作负载的实际 IO 吞吐量没有受到影响。

	Baseline	Error	SHA-1	Error
Operations/Min	29989	1.4%	29719	0.8%
Response Time (ms)	60 ms	0.8%	61ms	1.4%

表 2：运行在线电子商务应用程序的 SQL Server 数据库 VM 的带内写监视开销。对于此工作负载，平均事务率（操作/分钟）和 10 次运行的响应时间都在噪声范围内。报告的“误差”是标准差占平均值的百分比。

我们不认为额外计算的影响会对实际工作负载造成严重限制，与我们的微基准不同，实际工作负载除了 IO 之外还执行计算。为了说明这一点，我们在一个实际的企业工作负载上运行了 Inband SHA-1 计算。我们在运行 Microsoft SQL Server 2005 Enterprise Edition 数据库的 Windows Server 2003 VM 上进行了实验，该数据库配置有 4 个虚拟 CPU、6.4 GB 的 RAM、10 GB 的系统磁盘、250 GB 的数据库磁盘和 50 GB 的日志磁盘。数据库虚拟磁盘托管在一个包含 6 个磁盘的 800 GB

RAID-0 卷上；日志虚拟磁盘被放置在一个 100 GB 的 RAID-0 卷上，该卷有 10 个磁盘。 我们使用 Dell DVD Store(DS2)数据库测试套件[2]，它实现了一个完整的在线电子商务应用程序，来强调 SQL 数据库并测量其事务吞吐量和延迟。DVD 存储工作负载随机发出 8 KB 的 IOS，其写/读比为 0.25，未完成的写 IOS 数量高度可变，峰值约为 28[7]。表 2 报告了使用和不使用带内 SHA-1 写入计算的应用程序总体性能的总结。对于这个工作负载，我们没有观察到应用程序可见的性能损失，尽管其他处理器核心上的额外 CPU 周期被用于哈希计算。

4.2.2 COW 特化的开销

在 VMFS 中写入 COW 块是一个昂贵的操作，尽管当前的实现并没有针对 DEDE 广泛使用的 COW 子块进行很好的优化。在我们的原型中，专门化一个 COW 块需要大约 10 毫秒，因为这需要将其内容复制到新分配的块中以便更新它。因此，任何工作负载相移，其中大量先前重复数据被特化，都将导致显著的性能损失。 然而，一般来说，我们希望 VM 之间相同的块也不太可能被写入，而且与大多数重复数据删除方法不同，我们不会因为写入唯一块而遭受这种损失。正如[8]中所建议的那样，将共享延迟到候选块“稳定”一段时间后的优化可能有助于进一步减轻这种开销。

4.2.3 混合块大小的开销

从虚拟磁盘 IO 转换到物理磁盘上的操作, VMFS 的 1 MB 文件块允许非常低的开销。虽然我们添加到 VMFS 中的混合块大小支持是为了在可以使用 1 MB 块时保持这种效率，但它不可避免地引入了 4 KB 块的开销，因为它遍历了额外的指针块级别，并增加了外部碎片。

为了衡量其效果，我们将 IO 与两个 5 GB 虚拟磁盘进行了比较，一个完全由 1 MB 块支持，另一个完全由 4 KB 块支持。这些配置分别代表了重复数据删除的两个极端：所有唯一块和所有共享块。第一个磁盘需要一个指针块级别，在物理磁盘上分成 3 个独立的区，而第二个磁盘需要两个指针块级别，跨越 163 个独立的区。

% Sequential	IO Type	Throughput (MB/s)		Overhead
		BS=1 MB	BS=4 KB	
100%	Writes	238	150	37%
0%	Writes	66	60	9%
100%	Reads	245	135	45%
0%	Reads	37	32	14%

表 3：混合块碎片的开销。通过 16 个出色的 IOS 实现了 64 KB 顺序和随机工作负载的吞吐量。比较的对象是块大小(BS)分别为 1 MB 和 4 KB 的两个虚拟磁盘。在 4 KB 的情况下，虚

拟磁盘文件由 163 个不相交的片段组成，这意味着连续运行平均为 31 MB。

从这些虚拟磁盘读取的结果汇总在表 3 中。不幸的是，子块为顺序 IO 引入了大量开销。这部分是因为 VMFS 的子块放置和 IO 处理还没有得到很好的优化，因为子块以前没有在 VM IO 关键路径中使用过，而 VMFS 的文件块 IO 已经得到了很大的优化。减少此开销的一种可能方法是防止重复数据删除进程细分文件块，除非它们包含至少 4 KB 的候选共享块。这会影响重复数据删除的空间节省，但会防止 DEDE 为了一两个共享块而细分整个文件块。在子块 IO 性能和块细分方面的改进被认为是未来的工作。

4.2.4　磁盘阵列缓存的益处

对于某些工作负载，重复数据删除实际上可以通过减少工作负载的存储数组缓存占用来提高运行时性能。为了证明这一点，我们选择了一个常见的、关键的、有时间限制的 VDI 工作负载：同时引导许多 VM。VDI 启动风暴可能会发生在每晚关闭 VM 及其主机以节省电力、集体修补客户操作系统、集群故障转移或出于无数其他原因的循环中。

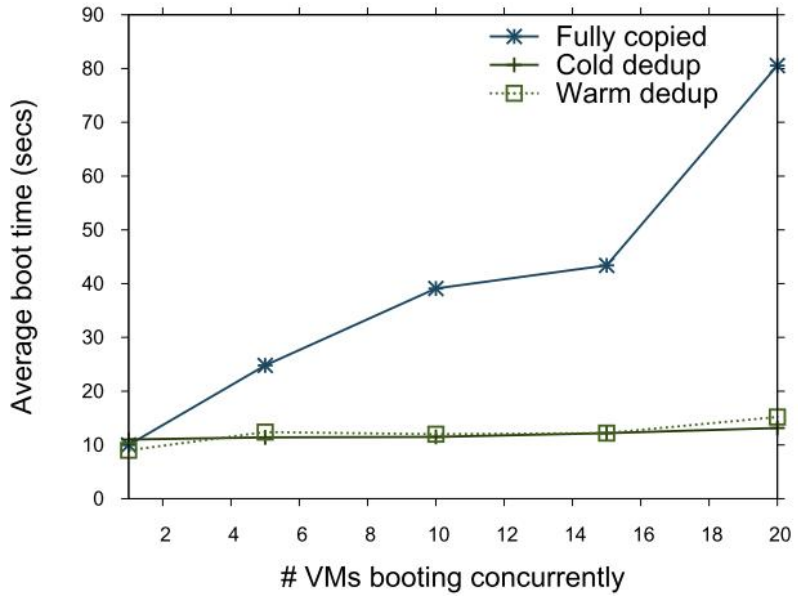


图 7：完全拷贝的 VM 和重复数据删除的 VM 之间的 Windows XP VM 启动时间比较。重复数据删除的 VM 被引导两次，以测量写入重复数据删除块的影响。

为了测试重复数据删除的缓存效果，我们比较了两种配置下同时从一个到二十个 VM 引导所需的平均时间：（1）每个 VM 都是 Golden VM 的完整副本（非常类似于 4.1 节中的 VDI 配置）和（2）VM 都是重复数据删除副本。图 7 所示的结

果显示，由于缓存占用的减少，重复数据删除与完整副本相比有了显著的改进。

为了进一步验证 COW 特化对实际工作负载的开销，我们还在重复数据删除后第二次引导 VM 集。盘面影像第一次“遇冷”；它们完全由 COW 块组成。第二次，写入的任何块都已经特化，可以直接写入。该图显示了这两种情况之间几乎没有区别，表明 COW 特化开销不是这个工作负载的问题。这并不意外，因为在 VM 引导期间只有少量的写操作。

4.3 重复数据删除率

虽然我们原型的索引实现还没有得到优化，但我们测量了它处理修改块的总体速率，以及它执行的三个主要操作的性能：扫描索引、将 1 MB 块细分为 4 KB 块和 COW 共享副本。

索引扫描过程几乎以磁盘的顺序访问速率运行，如第 3.2.2 节所述。在每个索引项 23 字节时，我们的原型每秒可以处理 6.6GB 的数据块。但是，与块细分和 COW 共享所需的时间与新共享的块数量成正比不同，索引扫描所需的时间与文件系统中的块总数成正比，因此快速是至关重要的。一旦索引扫描发现了新的重复项，包含这些重复项的 1 MB 文件块可以以 37.5MB/秒的速度细分为 4 KB 块。 最后，这些新发现的副本可以通过 2.6Mb/sec 的 COW 共享来消除。

COW 共享步骤将我们的原型限制在每小时处理大约 9 GB 的新共享块。唯一块（即最近修改的块，其哈希与索引中的任何内容都不匹配）可以以完整的索引扫描速率进行处理。此外，来自模板（大量重复数据的来源）的供应可以直接作为 COW 副本执行（大约 1 GB/秒），因此我们的重复数据删除速率仅适用于供应操作之外出现的重复数据。尽管如此，我们仍然认为，通过更多的分析和优化工作，我们的奶牛共享率可以显著提高。然而，即使在目前的速率下，原型也可以在一个 VDI 工作负载下以合理的速率消除重复，只需每天几个非高峰时间来执行带外重复数据删除。

5 相关工作

已经有很多工作对集中组件文件系统的重复数据删除做了研究。Venti[16]开创了将内容寻址存储(CAS)应用于文件系统的先河。Venti 是一个块存储系统，其中块通过其内容的抗冲突加密哈希来标识，并存储在磁盘上的仅追加日志中。磁盘上的索引结构从内容散列映射到块位置。Venti 的仅追加结构使其非常适合存档，但不适用于活动文件系统。Venti 还严重依赖于中央服务器来维护块索引。

其他各种系统，特别是 Data Domain 的 Archival System[26]和 Foundation[17]，已经扩展和增强了 Venti 方法，但仍然遵循相同的基本原则。虽然归档的重复数据删除通常很好理解，但实时文件系统中的重复数据删除提出了非常不同的挑战。由于备份系统涉及将数据保存在任意长的时间内，备份重复数据删除可以依赖于相对简单的仅追加数据存储。但是，用于实时重复数据删除的数据结构必须允许动态分配和垃圾收集。此外，与备份系统不同，实时文件系统对读写都很敏感。 因此，实时文件系统重复数据删除必须对这些关键路径产生最小的影响。备份数据也倾向于结构良好并以顺序流的形式呈现给备份系统，而实时文件系统必须处理随机写入。

许多基于 CAS 的存储系统，包括[5,16,20]，都是通过内容哈希来专门寻址数据的。写操作返回用于后续读操作的内容哈希。将这种方法应用于 VM 磁盘存储意味着多级块地址解析，这可能会对性能产生负面影响[10]。此外，由于数据存储在哈希空间中，VM 磁盘数据的空间局部性丢失，这可能导致某些工作负载的性能严重损失。DEDE 通过依赖常规的文件系统布局策略，并通过{文件名, 偏移量}元组而不是内容地址来寻址所有块，从而避免了这两个问题。DEDE 只使用内容散列来标识重复项。

NetApp 的 ASIS[14]和微软的 Single Instance Store[1]都使用带外重复数据删除来检测后台实时文件系统中的重复数据，类似于 DEDE。SIS 构建在 NTFS 之上，并将内容寻址存储应用于整个文件，使用 NTFS 过滤器来实现文件级的 COW 近似的语义。

虽然 SIS 依赖于集中式文件系统和单个主机来执行扫描和索引，但 FarSite 构建在 SIS 之上，以在分布式文件系统中执行重复数据删除[3]。Farsite 根据文件内容的哈希值将每个文件的责任分配给主机。每个主机将文件存储在其本地文件系统中，依靠 SIS 在本地删除它们。但是，由于大多数文件系统操作（包括读操作）需要跨主机通信，并且文件修改至少需要更新分布式内容哈希索引，因此这种方法会带来大量的网络开销。

Hong 的重复数据删除(DDE)系统[8]通过构建 IBM 的 Storage Tank SAN 文件系统[11]来避免 FarSite 的大部分跨主机通信开销。DDE 主机可以直接访问共享磁盘，因此可以直接从文件系统读取。但是，元数据操作（包括对重复数据删除共享块的更新）必须报告给集中式元数据服务器，该服务器只负责检测和合并重复数据。DEdE 在灵魂上是最接近 DDE 的。但是，由于 DEDE 使用一个完全分散的方案，没有元数据服务器，因此它不会出现单点故障或争用。此外，DEDE 通过分区工作和依赖粗粒度文件锁来防止跨主机并发问题，而 DDE 在多主机文件系统中从中央主机删除重复的方法引入了复杂的并发问题。

许多研究已经解决了内容可寻址存储对各种工作负载的有效性。集中于 VM 部署的工作[12,17]已经得出结论，CAS 在减少存储空间和网络带宽方面比传统的数据压缩技术（如压缩）非常有效。

其他工作已经解决了文件系统之外的重复数据删除问题。我们的工作灵感来自 WaldSpurger[25]，他提出了内存内容的重复数据删除，现在在 VMware ESX Server 虚拟机管理程序[23]中实现。在该系统中，来自多个虚拟机的相同内存页由同一页支持，并标记为即写即拷贝。该工作中共享提示的使用类似于我们的合并请求。

6 结论

本文研究了分布式集群文件系统中的重复数据删除问题。我们描述了一个新的软件系统，DEDE，它在没有任何中央协调的情况下提供了一个活动的共享文件系统的块级重复数据删除。此外，DEDE 在现有文件系统之上构建，而不会违反文件系统的抽象，这使得它能够利用常规的文件系统块布局策略和对唯一数据的就地更新。使用我们的原型实现，我们证明了这种方法可以在实际工作负载上以较小的性能开销实现高达 80%的空间减少。

我们相信我们的技术适用于虚拟机存储之外，并计划在未来的其他设置中检查 DEDE。我们还计划探索替代索引方案，以允许对重复数据删除策略进行更大的控制。例如，在产生大量重复数据的操作（例如，大量软件更新）期间，高频重复数据删除可以防止临时文件系统膨胀，推迟合并操作有助于减少文件系统碎片。此外，我们计划进一步探讨本文中提到的权衡，例如块大小与元数据开销、带内散列与带外散列、顺序索引更新与随机索引更新。

DEDE 只是虚拟机环境中重复数据删除的众多应用之一。我们认为，对于重复数据删除而言，下一步应该将其应用集成统一到文件系统、内存压缩、网络带宽优化等方面，实现端到端的空间和性能优化。

鸣谢

我们要感谢 Mike Nelson、Abhishek Rai、Manjunath Rajashekhar、Mayank Rawat、Dan Scales 、 Dragan Stancevic 、 Yuen-Lin Tan 、 Satyam Vaghani 和 Krishna Yadappanavar，他们与两位合著者一起，在未发表的工作中开发了 VMFS 的核心，本文在此基础上构建了 VMFS。我们感谢 Orran Krieger、James Cipar 和 Saman

Amarasinghe 的对话，他们帮助澄清了在线重复数据删除系统的要求。 我们感谢我们的指导者 Andrew Warfield、匿名评审者 John Blumenthal、Mike Brown、Jim Chow、Peng Dai、Ajay Gulati、Jacob Henson、Beng-Hong Lim、Dan Ports、Carl Waldspurger 和 Shioyun Zhu 对我们工作的详细评论以及他们的支持和鼓励。最后，感谢每一个注意到我们项目代号中重复并提请我们注意的人。本材料部分基于国家科学基金会研究生研究奖学金资助的工作。

参考文献

[1] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur. Single instance storage in Windows R 2000. In Proceedings of the 4th USENIX Windows Systems Symposium (WSS ’ 00), Seattle, WA, Aug. 2000. USENIX.

[2] Dell, Inc. DVD Store. http://delltechcenter.com/page/DVD+store.

[3] J. Douceur, A. Adya, W. Bolosky, P. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In Proceedings ofthe 22nd International Conference on Distributed Computing Systems (ICDCS ’02), Vienna, Austria, July 2002. IEEE.

[4] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki. Hydrastor: A scalable secondary storage. In Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST ’09), San Francisco, CA, Feb. 2009. USENIX.

[5] EMC Centera datasheet.http://www.emc.com/products/detail/hardware/centera.htm.

[6] R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong. Extendible hashing—a fast access method for dynamic files. ACM Transactions on Database Systems, 4(3), Sept. 1979.

[7] A. Gulati, C. Kumar, and I. Ahmad. Storage workload characterization and consolidation in virtualized environments. In 2nd International Workshop on Virtualization Performance: Analysis, Characterization, and Tools (VPACT), 2009.

[8] B. Hong, D. Plantenberg, D. D. E. Long, and M. Sivan-Zimet. Duplicate data elimination in a SAN file system. In Proceedings ofthe 21st Symposium on Mass Storage Systems (MSS ’04), Goddard, MD, Apr. 2004. IEEE.

[9] Iometer. http://www.iometer.org/.

[10] A. Liguori and E. V. Hensbergen. Experiences with content addressable storage and virtual disks. In Proceedings of the Workshop on I/O Virtualization (WIOV’08), San Diego, CA, Dec. 2008. USENIX.

[11] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, and B. Hillsberg. IBM storage tank—a

heterogeneous scalable SAN file system. IBM Systems Journal, 42(2), 2003.
[12] P. Nath, M. A. Kozuch, D. R. O'Hallaron, J. Harkes, M. Satyanarayanan, N. Tolia, and M. Toups. Design tradeoffs in applying content addressable storage to enterprise-scale systems based on virtual machines. In Proceedings ofthe USENIXAnnual Technical Conference (ATEC '06), Boston, MA, June 2006. USENIX.
[13] P. Nath, B. Urgaonkar, and A. Sivasubramaniam. Evaluating the usefulness of content addressable storage for high-performance data intensive applications. In Proceedings ofthe 17th High Performance Distributed Computing (HPDC '08), Boston, MA, June 2008. ACM.
[14] Netapp Deduplication (ASIS). http://www.netapp.com/us/products/platform-os/dedupe.html.
[15] K. W. Preslan, A. P. Barry, J. E. Brassow, G. M. Erickson, E. Nygaard, C. J. Sabol, S. R. Soltis, D. C. Teigland, and M. T. O'Keefe. A 64-bit, shared disk file system for Linux. In Proceedings ofthe 16th Symposium on Mass Storage Systems (MSS '99), San Diego, CA, Mar. 1999. IEEE.
[16] S. Quinlan and S. Dorward. Venti: A new approach to archival data storage. In Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST '02) [19].
[17] S. Rhea, R. Cox, and A. Pesterev. Fast, inexpensive content addressed storage in Foundation. In Proceedings ofthe USENIX Annual Technical Conference (ATEC '08), Boston, MA, June 2008. USENIX.
[18] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In Proceedings ofthe 1st USENIX Conference on File and Storage Technologies (FAST '02) [19].
[19] USENIX. The 1st USENIX Conference on File and Storage Technologies (FAST '02),Monterey, CA, Jan. 2002.
[20] M. Vilayannur, P. Nath, and A. Sivasubramaniam. Providing tun able consistency for a parallel file store. In Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST '05), San Francisco, CA, Dec. 2005. USENIX.
[21] VMware, Inc. VMFS datasheet. http://www.vmware.com/pdf/vmfs_datasheet.pdf.
[22] VMware, Inc. Recommendations for aligning VMFS partitions. Technical report, Aug. 2006.
[23] VMware, Inc. Introduction to VMware Infrastructure. 2007.http://www.vmware.com
[24] VMware, Inc. VMware Virtual Desktop Infrastructure (VDI)datasheet, 2008. http://www.vmware.com/files/pdf/vdi_datasheet.pdf.
[25] C. A. Waldspurger. Memory resource management in VMware ESX Server. In Proceedings ofthe 5th USENIXSymposium on Operating Systems Design and Implementation (OSDI '02), Boston, MA, Dec. 2002. USENIX.
[26] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the Data Domain deduplication file system. In Proceedings ofthe 6th USENIX Conference on File and Storage Technologies (FAST

’08), San Jose, CA, Feb. 2008. USENIX.

译文原文出处：CLEMENTS A T, AHMAD I, VILAYANNUR M, et al. Decentralized Deduplication in SAN Cluster File Systems[C]//USENIX annual technical conference. 2009, 9: 101-114.