复合文件文件系统：解耦文件和元数据的一对一映射以获得更好的性能

摘要

传统的文件系统优化通常使用逻辑文件到其物理元数据表示的一对一映射，这个映射导致错过在这样的一类优化的机会耦合被删除。

本文已经设计、实施和评估了一个复合文件文件系统，它允许多对一文件到元数据的映射，本文已经探究过了设计不同制图策略的空间。在网络服务器和软件开发工作负载下，本文的实证评估显示性能提高达27％。这个结果证明了复合文件的预期目的。

# 1简介

文件系统性能优化是一个很好的领域。但是，大多数优化技术（例如，高速缓存，更好的数据布局）保留逻辑文件到其物理元数据表示的一对一映射（即，每个文件与其在UNIX平台上的它自己的i-节点相关联）。这样的映射是可取的，因为元数据结构是根深蒂固的数据结构，以及许多存储组件和机制，比如VFS API 、预取和元数据缓存 -依赖于这样的构造。然而，这种严格的映射也为一类性能优化提供了一个盲点。

本文设计，实现和评估了复合文件文件系统（CFFS），其中许多逻辑文件可以组合在一起并与单个i节点（加上作为扩展属性存储的额外信息）相关联。这样的安排是可能的，因为一起访问的许多文件共享相似的元数据子字段，可以进行重复数据删除。因此，CFFS可以减少对存储的元数据访问，这是访问小文件的一个重要的开销来源，这仍然是现代工作负载的大多数文件引用的主导。

基于Web服务器和软件开发工作量，CFFS可以比ext4高出27％，这表明放宽文件到数据映射的方法是有希望的。

# 2观察

以下观察导致了CFFS设计：

频繁访问小文件：研究表明，小文件接收大部分文件引用。本文对桌面文件系统的内部分析证实访问量的80％以上是小于32个字节文件。此外，访问磁盘上的小文件的访问时间的40％可归因于元数据访问。因此，减少这种访问开销可能会导致较大的性能增益。

冗余元数据信息：传统文件与其自身的物理元数据关联，用于跟踪文件块的位置，访问权限等信息。但是，许多文件具有相似的文件属性，因为文件所有者的数量，权限模式等是有限的。典型工作站的元数据压缩率高达75％。因此，本文看到了很多减少冗余元数据信息的机会。

分组访问的文件：文件往往被一起访问，如[KRO01，LI04，DIN07，和JIA13]， 例如，网络访问通常涉及访问许多关联的文件。但是，利用文件分组的优化可能不会产生自动性能增益，因为识别和分组文件的过程会导致开销。

预取限制：虽然预取是一种有效的优化，但是单独获取每个文件和相关的元数据访问会带来很高的开销。例如，访问32个小文件可能会比访问大小等于32个文件总和的单个文件高50％，即使使用warm caches也是如此。

这个观察引起了本文是否可以通过整合一起访问的小文件来提高性能的问题。 这是通过本文将逻辑文件的一对一映射解耦为元数据的物理表示的方法来实现的。

# 3复合文件文件系统

## 3.1设计概述

CFFS引入了一个称为复合文件的内部物理表示，它保存了经常访问的小文件的内容。复合文件是对最终用户是不可见的，并且与小文件之间共享的单个复合i节点相关联。存储在小文件inode中的原始信息被重复数据删除并存储为复合文件的扩展属性。各个小文件的元数据属性仍然可以被重构，检查和更新，以便传统访问语义（例如，类型，权限，时间戳）保持不变。扩展属性还记录了单个小文件的复合文件内的位置。通过这种表示，CFFS可以将物理复合文件转换为逻辑文件。

哪些文件合并成一个复合文件是一个重要的依赖于工作负载的决策。作为一个例子，CFFS已经被配置了三种方式。第一种方案是基于目录的合并，目录中的所有文件（不包括子目录）形成一个复合文件。第二种方案是嵌入引用合并，其中提取文件内容中的文件引用以识别可以形成合成文件的文件。三是基于频次挖掘的整合，通过设置频次挖掘对文件参考进行分析，使得经常一起访问的文件形成复合文件。

一个复合文件利用传统的VFS预取机制，因为整个复合文件可以以与通过将小数据块组合成更少的较大块而获得的益处FFS类似的方式被预取为单元。

## 3.2数据表示

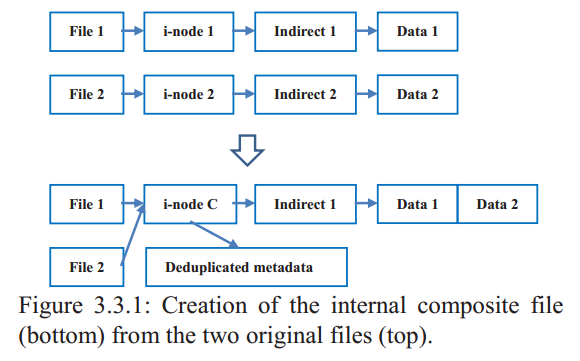
复合文件的内容是通过连接小文件形成的，称为子文件。复合文件中的所有子文件共享相同的i节点以及间接块、重间接块等。复合文件的最大大小限制不是一个问题，因为复合文件是为了组合小文件而设计的。如果子文件大小的总和超过了最大文件大小限制，本文可以使用多个复合文件。

通常，复合文件中的第一个子文件是入口点，其入口将触发预取其余子文件。 例如，当浏览器访问一个html文件时，它会加载一个css文件和flash脚本。该html文件可以作为这个三子文件合成文件的入口点和预取触发器。对于基于频率的合并，子文件的排序反映了它们如何被访问。尽管同一组文件可能具有不同入口点的不同访问模式，但是数据布局基于最普遍的访问模式

## 3.3元数据表示和操作

组合文件创建：创建组合文件时，CFFS将分配一个i节点，并将子文件的内容作为其数据进行复制和拼接。 组合文件将各个子文件的复合文件偏移量和大小以及重复数据删除的i-节点信息记录到其扩展属性中。然后原始子文件被截断，其目录条目被重新映射到组合文件的i节点，以扩展到还包括子文件ID和其原始i节点的重新分配。因此，最终用户仍然感知到名称空间中的单个逻辑文件，而单个子文件仍然可以被定位（图3.3.1）

I节点内容重构：重复删除的子文件i节点在飞行中重建。默认情况下，除非在扩展属性中另行指定，否则子文件的i节点字段会继承复合文件的i节点字段的值。



权限：在文件打开时，首先根据复合i节点检查权限测试。如果失败，则不需要进一步检查。否则，如果子文件具有作为扩展属性存储的不同权限，则将再次检查权限。因此，复合i节点将具有跨所有子文件的最广泛的权限。例如，如果在复合文件中，本文有一个只读子文件A和一个可写子文件B，则复合i节点的权限将是可读写的。但是，打开具有写入权限的子文件A时，扩展属性中的只读权限限制将捕获违规。

时间戳：每个文件操作都会更新单个子文件和复合文件的时间戳。但是，在检查过程中（例如stat系统调用），本文返回子文件的时间戳。

大小：对于数据访问，偏移量通过扩展属性中编码的子文件偏移量和大小进行转换和绑定检查。复合文件的大小是复合文件的长度，可以大于其子文件的总大小。例如，如果复合文件中间的子文件被删除，则该区域将被释放，而不会更改复合文件的大小。

inode命名空间：对于大于阈值X的i节点号，使用较高的零扩展N位用于复合i节点号，较低的M位被保留用于子文件ID。本文将这个范围的i-节点号码称为CFFS唯一ID（CUID）。

子文件查找和重命名：如果目录中的名称映射到CUID，则可以通过子文件ID查找子文件的属性。重命名将继续进行，就像一个CUID是非CFFS系统中的一个i-node编号一样。由于将子文件移入和移出复合文件将改变其CUID，因此本文需要存储后台指针，以更新映射到CUID的所有名称。

CUID中的改变可能会中断通过其i节点号唯一标识文件的应用程序（例如，备份）。然而，今天的文件系统也可能导致不同的文件在不同的时间共享相同的i节点号码;CFFS设计放大了应用程序不应该假设i节点号码是文件的唯一属性的原因

子文件和子文件成员资格更新：将子文件添加到组合文件时，它会附加到组合文件。从复合文件中删除子文件时，复合文件中相应的数据区域在扩展属性中被标记为已释放。

子文件打开/关闭操作：对子文件的打开/关闭调用与对复合文件的打开/关闭调用相同，并转换文件位置指针。

子文件写操作：就地更新的处理方式与传统文件系统中的相同。但是，如果更新涉及在复合文件中间生成子文件，并且在子文件末尾没有可用空间，则将更新的子文件移动到复合文件的末尾。该方案利用潜在的时间局部性，即在不久的将来，不断增长的子文件可能再次增长。

硬链接：目录中的不同名称可以映射到相同的i节点编号或CUID。

空间压缩：当一半分配的大小不包含有用数据时，复合文件压缩其空间。并发更新到复合文件中的子文件：对复合文件中的子文件的并行更新与对普通文件的并发更新具有相同的语义。为避免锁争用，检测到涉及并发更新的文件可能必须提取到多个常规文件中。

锁定和一致性：CFFS不支持群集，但本文相信可以实现子文件锁定子系统。

## 3.4辨别复合成员资格

### 3.4.1基于目录的合并

鉴于遗留文件系统具有围绕目录的根深蒂固的空间局部优化，目录是文件访问模式和用于形成组合文件的良好近似。目前，这个合并计划不包括子目录。

可以在所有目录上执行基于目录的整合，而无需跟踪和分析文件参考。但是，它不会捕获跨目录的文件关系。

### 3.4.2基于嵌入式参考的整合

基于嵌入式参考的合并根据文件中嵌入的文件引用来标识复合文件成员资格。 例如，超链接可能被嵌入在一个html文件中，一个网络爬虫很可能通过这些链接访问每个网页。在这种情况下，本文合并原始的html文件和引用的文件。类似的想法适用于编译。本文可以从Makefile中提取依赖关系规则，并合并导致生成相同二进制文件的源文件。由于文件更新可能会打破依赖关系，CFFS可以定期通过修改的文件进行筛选以协调复合文件成员资格

### 3.4.34基于频率挖掘的整合

在他们对基于频率挖掘的整合的探索中，他们使用了Apriori算法[AGR94]的一个变种。关键的观察是，如果经常访问一组文件，它的子集也必须是（Apriori属性）。图3.4.3.1说明了对文件A，B，C，D和E的访问流的算法。

初始传递：首先，本文计算每个文件的访问次数，然后删除计数小于阈值（如两个）的文件以供进一步分析。

第二遍：对于其余文件，本文对所有可能的双文件引用集进行排列，构建和计数。只要文件A在B之后访问，反之亦然，本文增加文件集{A，B}的计数。 计数小于阈值的集被删除（例如，{B，D}）。

第三遍：本文可以根据剩余的两个文件的引用集生成所有的三个文件的引用集。但是，如果一个三个文件的引用集经常出现，那么所有的两个文件引用集都需要经常出现。 因此，{A，B，D}等文件集被修剪，因为在第二遍中{B，D}被消除了。

终止：由于本文不能再生成四个文件的参考集，算法结束。 现在，如果一个文件可以属于多个文件集，本文将集合{A，B，C}和{A，D}作为两个经常访问的集合返回。诸如{A，B}的集合被删除，因为它们是{A，B，C}的子集。

变化：另一种方法是使用标准化的阈值或支持度，即设置事件的百分比（一组事件的发生次数除以总事件的次数，范围介于0和1之间）。

本文也可以跟踪文件参考序列来确定合成文件的入口点和内容布局，而不是跟踪文件集。本文目前不允许重叠文件集来避免复制的复杂性和保持一致性。 要选择两个复合文件之间的子文件成员资格，决定取决于复合文件是否有更多的子文件，更高的支持和更新的创建时间戳。

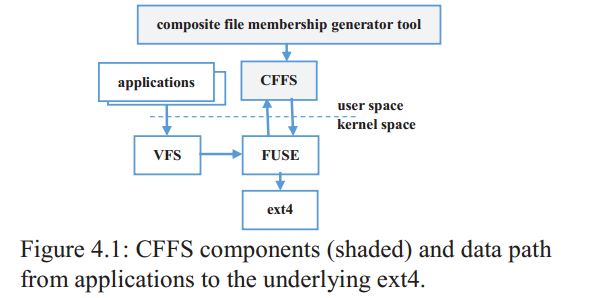
基于频率挖掘的整合可以根据动态文件引用标识复合文件候选项。但是，运行它的成本将其应用限制在更受欢迎的文件参考序列中。

# 4实验

本文的原型的两个主要组件是复合文件成员资格生成器工具和CFFS。

本文通过在Linux 3.16.7上运行的FUSE（v2.9.3）框架（图4.1）在用户空间中创建了CFFS原型。CFFS堆叠在ext4上，以便本文可以利用旧版工具和功能，如持久性引导（例如文件系统创建实用程序），扩展属性和日志记录。

CFFS定期与生成器工具进行协商以创建新的复合文件。 本文利用类似于硬链接的机制来允许将多个文件名映射到相同的复合i节点。 由于需要更新各个子文件的时间戳，本文拦截了所有与文件系统相关的调用。 本文还需要确保各种访问使用正确的权限（例如open和readdir），已翻译的子文件偏移和大小（例如，读和写）以及时间戳（例如，getattr和setattr）。 实际的组合文件，其i节点及其扩展属性由底层的ext4文件系统存储。 CFFS用C ++实现，大约有1600个分号。

对于基于目录的合并，本文使用Perl脚本将目录中的所有文件列为复合文件成员。 对于基于嵌入式参考的方案，

本文专注于两种情况。对于Web服务器工作负载，本文合并HTML文件及其立即引用的文件。 如果复合文件成员资格冲突，则优先考虑index.html和那么首先包含一个文件的html。 另一个是源代码编译。本文使用Makefile作为指导来整合源代码文件。 对于基于频率分配的方案，成员资格生成器工具可以采用http访问日志或strace输出。生成器实现了Apriori算法，支持参数设置为5％。分析批量大小设置为50K参考。参数是根据经验经验选择的，以限制存储量和处理开销。 发生器代码包含~1200个分号。

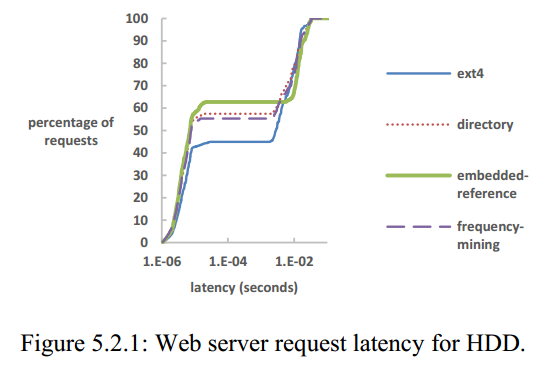
# 5效果评估

本文通过FUSE比较了ext4堆叠的CFFS和基准ext4文件系统（通过一个空的FUSE模块路由请求）的性能。

本文通过重播两条曲线来评估本文的系统。第一条是从部门网络服务器（01/01 / 2015-03 / 18/2015）收集的http日志。跟踪包含14M文件对1.0TB数据的引用;这个，3.1M的文件是独一无二的，拥有76GB的数据。第二个跟踪是通过软件开发工作站（2014年11月20日 - 2014年11月30日）strace收集的。包含超过240M文件系统相关系统的跟踪调用24GB数据;其中，291,133个文件是唯一的，2.9GB字节。在读写操作之间，59％是读取，41％是写入。

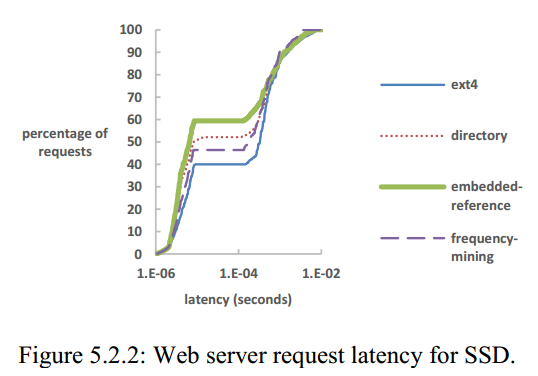
本文在存储设备上进行了多线程，零思考时间的跟踪回放。本文还跳过了没有任何活动的跟踪时间间隔。重放实验在戴尔工作站上进行（表5.1）。每个实验重复5次，结果以90％置信区间呈现。

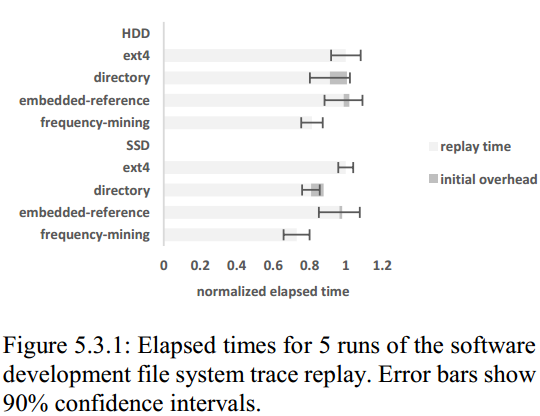
在每个实验之前，本文用虚拟内容重建了文件系统。 对于基于目录和嵌入式引用的方案，复合文件成员资格会不断更新。 对于基于频率挖掘的合并，分析是分批执行的，但是合成文件每天更新



## 5.2Web服务器跟踪重现

硬盘性能：图5.2.1显示了磁盘的Web服务器请求延迟的CDF，从请求发送到请求完成的时间。

本文工作的最初意图是减少经常访问的小文件的IO数量。 但是，对统一元数据访问次数较少的好处是将自身显示为所有子文件的元数据预取，复合文件语义启用跨文件预取，从而实现更高的高速缓存命中率。

基于嵌入式参考的整合效果最好，高速缓存的请求率为62％，比ext4高20％。 因此，基于嵌入式引用创建的复合文件更准确地捕获访问模式。 整体重播时间也减少了20％左右。

基于目录的复合文件也可以将缓存命中率提高15％，反映目录捕获空间位置的有效性。

基于频率挖掘的整合表现比基于目录的更差。 本文检查了踪迹，发现48％的引用是由爬虫做的，其余的则由用户做的。 因此，挖掘算法的分叉流量模式形成较不积极的文件分组，从而降低了收益。

SSD性能：图5.2.2显示了SSD的Web服务器请求延迟的CDF。 与磁盘相比，相对趋势相似，由于SSD的速度，缓存未命中的请求延迟时间减少了两个数量级。 由于主要的性能提升是由较高的缓存命中率和IO回避造成的，因此这20％的收益与底层存储介质相当独立。

## 5.3软件开发文件系统跟踪重播

对于软件开发工作量的重放，捕获各个文件系统调用请求的延迟比较困难，因为许多是异步的（例如写入），像mmap这样的调用不知道发送到底层存储的请求的数量。 因此，本文用总时间总结了本文的结果，包括复合文件操作的所有开销，不包括基于目录和嵌入引用的方案的初始设置成本（图5.3.1）。

硬盘性能：基于嵌入式参考的方案覆盖率很差，因为许多参考文献与编译无关，所以耗费的时间更接近于ext4。 基于目录的整合实现了17％的时间缩短，但基于频率挖掘的方案可以达到27％，因为复合文件跨目录包含文件。

SSD性能：不同合并设置的相对性能趋势与HDD相似。 类似于网迹，增益高达20％。

当比较HDD和SSD实验之间的性能改善差距时，SSD下无法实现高达11％的HDD性能增益，因为SSD不会导致磁盘寻道开销。

## 5.4管理费用

基于目录和嵌入式参考的方案：基于目录和嵌入式参考的方案会产生初始部署成本，以根据目录和嵌入式文件引用来创建复合文件。嵌入式参考方案的初始成本取决于可从中提取文件参考的文件类型的数量。对于本文的工作负载，这个成本在1到14分钟之间（图5.3.1）。

至于更新复合文件成员资格的增量成本，添加成员涉及附加到复合文件。删除成员主要涉及元数据更新。 一个复合文件不压缩，直到其一半的分配空间未使用。由于跟踪重播的数量已经包含这个开销，所以这个成本被好处所抵消。

基于频率挖掘的方案：跟踪收集开销低于0.6％，跟踪分析的内存开销在200MB以内，平均每日日志15M行。

基于频率挖掘的方案涉及从最近的文件参考中学习，并花了几个重播日子来获得这个方案的全部好处。

## 5.5讨论和未来的工作

复合文件可以使用不同的存储介质，从而使读写显性和读写性工作负载都受益，这表明性能提升的主要原因是IO数量减少（〜20％）。 SSD与硬盘之间的性能差距缩小表明由于磁盘寻道减少而导致的性能提升，修改后的数据布局高达10％左右。

总体而言，本文对形成复合文件的方式，合成元数据的性能影响以及复合文件启用的预取之间的关系很感兴趣。 未来的工作将探索形成复合文件的额外方法，并量化它们与性能贡献的不同组件之间的相互作用。另外，未来的工作将更充分地探索元数据压缩，并发性和安全性的后果。

# 6相关工作

小文件优化：虽然本文的研究主要关注逻辑文件和物理元数据的多对一映射，但这项工作与通过减少存储访问次数来优化小文件访问密切相关。在这方面的早期工作涉及将文件的i节点与其第一个数据块和将i节点嵌入到目录中。后来，hFS使用单独的存储区来优化小文件和元数据访问。 Btrfs将元数据和小文件打包到copy-on-write b-trees.TableFS将元数据和小文件打包到表中，并刷新表条目修改的2MB日志，组织为日志结构合并树。CFFS通过整合通常一起访问的文件的i节点来补充现有的许多方法。

并行和分布式计算领域也探索访问子文件区域和合并元数据的想法，在这个领域中，多台计算机上的CPU需要访问相同的大数据文件。Facebook的照片存储利用观察图像的权限大致相同，可以巩固。 但是，这些机制是为非常同类的数据类型量身定制的。通过不同的方式来形成组合文件，CFFS可以处理具有更多不同内容和访问语义的子文件。

预取：尽管可以发现大量的工作来改善预取，但也许C-Miner最接近本文的工作。特别是C-Miner在块级应用频繁序列挖掘，优化了文件和元数据块的布局，提高了预取能力。然而，CFFS减少了频繁访问的元数据块的数量，并且避免了需要大表来映射逻辑块到物理块。另外，本文的文件系统级挖掘处理的对象和关联开销也明显减少。 DiskSeen结合磁盘布局的知识来改进预取，预取可以跨文件和元数据边界。 CFFS主动减少物理元数据项的数量并改变存储布局以促进顺序预取。Soundararajan等人观察到，通过将高级执行上下文（例如，线程，应用程序ID）传递到块层，结果数据挖掘可以在并发工作负载下生成具有较长运行的预取规则。由于CFFS在文件系统级执行数据挖掘，因此可以使用PID和IP地址来解开并发的文件引用。然而，CFFS的重点是改变逻辑文件到其物理表示的映射，并且可以采用各种挖掘算法整合元数据并改善存储布局。

# 7结论

本文介绍了一个复合文件文件系统的设计，实现和评估，它探索了逻辑文件和元数据的多对一映射。 可以对CFFS进行不同的配置以识别经常一起访问的文件，并可以合并其元数据。 结果显示，在两个真实世界的工作负载下，性能提高达27％。 CFFS的经验表明，将文件和元数据的一对一映射解耦的方法是有希望的，并且可能导致许多新的优化机会。

参考文献：

Abd-El-Malek M, Courtright WV, Cranor C,Ganger GR, Hendricks J, CKlosterman AJ, Mesnier M, Prasad M, Salmon B, Sambasivan RR,Sinnamohideen S, Strunk JD, Thereska E, Wachs M, Wylie JJ. Ursa Minor: Versatile Cluster-based Storage. *Proceedings of the 4th USENIX Conference on File and Storage Technology*

Agrawal R, Srikant R. Fast Algorithms for Mining Association Rules, Proceedings of the 20th VLDB Conference, 1994.

Beaver D, Kumar S, Li HC, Vajgel P. Finding a Needle in Haystack: Facebook’s Photo Storage. Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2010.

Bloom B. Space/Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM (CACM), 13(7):422-426, July 1970。

Chandrasekar S, Dakshinamurthy R,Seshakumar PG, Prabavathy B, Chitra B. A Novel Indexing Scheme for Efficient Handling of Small Files in Hadoop Distributed File System.Proceedings of 2013 International Conference on

Computer Communication and Information, 2013.

Chidambaram V, Sharma T, Arpaci-Dusseau AC, Arpaci-Dusseau RH. Consistency without Ordering. Proceedings of the 10th USENIX Conference on File and Storage Technologies,2012.

Ding X, Jiang S, Chen F, Davis K, Zhang X.DiskSeen: Exploiting Disk Layout and Access History to Enhance Prefetch. Proceedings of the 2007 USENIX Annual Technical Conference (ATC),2007.

Dong B, Qiu J. Zheng Q, Zhong X, Li J, Li Y.A Novel Approach to Improving the Efficiency of Storing and Accessing Smaller Files on Hadoop: a Case Study by PowerPoint Files. Proceedings of the 2010 IEEE International Conference on Services Computing, 2010.

Edel NK, Tuteja D, Miller EL, Brandt SA.Proceedings of the IEEE Computer Society’s 12th Annual International Symposium on Modeling,Analysis, and Simulation of Computer and Telecommunications Systems, 2004.

Ganger GR, Kaashoek MF. Embedded Inode and Explicit Grouping: Exploiting Disk Bandwidth for Small Files. Proceedings of the USENIX 1997 Annual Technical Conference (ATC), 1997.

Garrison JA, Reddy ALN. Umbrella File System: Storage Management across Heterogeneous Devices. ACM Transactions on Storage (TOS), 5(1), Article 3, 2009.

Harter T, Dragga C, Vaughn M, ArpaciDusseau AC, Arpaci-Dusseau RH. A File is Not a File: Understanding the I/O Behavior of Apple Desktop Applications. Proceedings of 23rd Symposium on Operating Systems Principles (SOSP), 2011.