**海量存储报告**

|  |
| --- |
| **对海量小文件存储优化方法的研究** |

|  |  |
| --- | --- |
| 院（系）名 称 ： | 计算机学院 |
| 专 业 名 称 ： | 计算机软件与理论 |
| 学 生 姓 名 ： | 罗倩雯 |
| 学号 ： | 2017202110051 |

摘 要

随着科学技术的发展，计算科学技术的飞速提升，越来越多的信息被数据化，比如大量的网络信息，社会媒体数据，工业制造大数据。越来越多数据的产生使得数据存储结构面对前所未有的挑战。不仅需要稳定的存储，还要能够高效的查询，修改等功能，传统的文件系统典型优化使用逻辑文件一对一映射到其物理元数据表示。这种映射导致失去了消除耦合这类优化的方式。

在我所看的文献中撰稿人已经设计，实现和评估了一个允许多个文件对一个元数据的复合文件系统，而且探索了不同映射策略的设计空间。在网络服务和软件开发工作环境中，实验的评估表明达到了27%的性能提升。这个实验结果表明了复合文件很有前景。

**关键词**：海量小文件；存储；元数据；优化

**ABSTRACT**

Traditional file system optimizations typically use a oneto- one mapping of logical files to their physical metadata representations. This mapping results in missed opportunities for a class of optimizations in which such coupling is removed.

We have designed, implemented, and evaluated a composite-file file system, which allows many-to-one mappings of files to metadata, and we have explored the design space of different mapping strategies. Under webserver and software development workloads, our empirical evaluation shows up to a 27% performance improvement. This result demonstrates the promise of composite files.

**Key Words**: small file; storage；metadata ；optimization

**目 录**

[1 引言 1](#_Toc500512926)

[1.1 研究背景 1](#_Toc500512927)

[1.2 相关项目与文献分析 2](#_Toc500512928)

[1.3 课程报告框架和主要内容 2](#_Toc500512929)

[2 问题调研 4](#_Toc500512930)

[3 系统方案设计原理 5](#_Toc500512931)

[3.1 设计概述 5](#_Toc500512932)

[3.2 数据表示 5](#_Toc500512933)

[3.3 元数据表示和操作 6](#_Toc500512934)

[3.4 识别复合文件成员 7](#_Toc500512935)

[4 系统实现 9](#_Toc500512936)

[5 系统性能评估 10](#_Toc500512937)

[5.1 网页服务器跟踪重放 10](#_Toc500512938)

[5.2 软件开发文件系统跟踪重播 11](#_Toc500512939)

[5.3 讨论 11](#_Toc500512940)

[6 总结 13](#_Toc500512941)

[参考文献 14](#_Toc500512942)

# 引言

本章主要介绍所看文献的研究目的与研究意义，这部分的内容也是自己结合在项目实现过程中要解决的问题，因为我所在的实验室项目组目前主要在做制造业背景下的非结构化数据管理系统的设计与实现，该项目的重难点就是海量小文件的存储管理问题，然后介绍相关的研究工作，文献的研究问题，最后介绍本论文的框架和主要内容。

## 研究背景

随着计算科学的飞速发展，越来越多的信息被数据化，生活工作中的越来越多的应用数据呈爆炸式增长[1]，2012年，大数据在中国开始兴起并引起关注，2013年被广泛认为是“大数据时代第一年”。根据ZDNET的数据[2]，2013年中国生产的数据总量已经超过0.8个ZB（相当于8亿TB），是2012年的两倍，等于全球数据总量据估计，到2020年，中国生产的数据总量将是2013年的10倍，超过8.5亿片。大量数据的主要来源为网络信息数据，社会媒体数据，工业制造大数据，例如一个大型的社交网站，仅每周的图片信息就多达60TB[3]。比如Facebook 存储系统中，图片总容量超过1.5PB。我国最大的电子商务平台淘宝，图片服务器存储的图片总容量达到1PB。腾讯用户在相册上传的图片总容量达12PB。但是这些海量的图片文件中，大部分是单个文件容量不超过1MB 的小文件。所以不能仅靠存储单个文件容量很大的传统方式来存储文件。目前常用的分布式存储系统都采用传统的主从结构，包括Google 使用的GFS[6]、Yahoo 的HDFS[7] 及Amazon 的S3[8]。传统分布式存储模式是将数据存储在多个特定的节点上，通过网络使用系统中的各个节点上的磁盘空间，并将这些分散的资源构成一个虚拟的存储设备．分布式共享存储和集群存储是组成传统分布式存储模式的主要方面。它主要是通过运用分布式计算和共享存储等相关技术，将不同节点上的存储节点整合起来，并向外界提供服务，它应用于分布式文件系统。但目前分布式共享还存在一些问题，例如系统的性能提高以及系统的可扩展性等都制约着这门技术的应用范围。对于数据量的急剧增加，服务器承载的压力越来越大，巨量增加的数据存储任务，让服务器的性能显著降低，为了提高服务器的性能，出现了集群存储技术．集群存储就是通过将数据分布在集群中的各个节点的存储方式，它提供单一的使用接口与界面，使用户可以方便地对所有数据进行统一使用与管理．对于存储在集群上的数据会按照一定的负载均衡策略，从多台设备上存储和读取，以获得更高的存储性能。从多个节点上获得过程对于用户来说是透明的，给用户的感觉是仅仅从一个节点获得数据．基于以上的这些优点，集群存储往往被用于非结构化数据为主的应用，目前常见的应用是在图像处理、高性能计算、音视频应用等方面。然而这些传统的分布式文件系统主要是面向大文件数据存储与访问而设计，面对海量小文件存储与访问，存在元数据结构效率低、主服务器性能瓶颈、磁盘I /O效率低、磁盘空间利用率低和网络通信延时高等一系列瓶颈问题，如何对海量小文件进行高效存储与访问支持，已经成为存储领域不可回避的重大课题。

目前我所参与的实验室项目制造业非结构化大数据管理系统需要解决针对工业数据中的海量小文件存储问题来实现高效的查询和修改等功能，像Hadoop这样的数据处理平台是目前广泛采用的方案，因为Hadoop在可扩展性，健壮性，计算性能和成本方面具有不可替代的优势，已经成为当前互联网企业的主要大数据分析平台。于是我们也选择了基于Hadoop的管理海量小文件的存储方案，但是Hadoop分布式文件系统（HDFS）在众多小文件存储方面仍然存在很多弱点。大量小文件的存储将增加整个群体的负担，降低效率，因为Hadoop通过Hadoop分布式文件系统（简称HDFS）进行数据存储。然而，由于Hadoop使用簇中所有数据块的信息单元数据存储在Namenode的主要节点中，因此Namenode节点的操作压力在存储大量小于5MB的小文件时会急剧增加，这就需要更多的内存来存储信息元数据。而且当处理大量的小文件时，需要建立更多的MapReduce任务，而大量的MapReduce任务的交互和通信会增加CPU的开销，这将大大降低整个Hadoop集群的运行效率。为了解决这种小文件存储的缺陷，我们一般会合并小文件，合并后存放大文件。但这种方法没有采用文件的大小分布，也没有进一步提高小文件合并的效果。所以希望能够结合所选论文来提出有效的解决方案。

## 相关项目与文献分析

在我们的项目研究中采用了Sequencefile合并策略，对小文件进行合并，使得Namenode存储中能够充分利用内存不会有索引爆炸的情况产生，如图1所示。这种方法和我所看的论文中的方法有异曲同工之妙，在该论文中是把传统文件系统中的这种逻辑文件到其物理元数据表示的一对一映射去除，而采用许多个逻辑文件与物理元数据表示多对一映射，在网页服务器和软件开发工作中实验评估该系统已经达到27%的性能提升。具体实现为复合文件文件系统（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。

## 课程报告框架和主要内容

1. 第1章 引言

本章节主要是首先介绍本课程论文研究问题的目的和研究意义，然后介绍相关的研究工作，之后简述报告的框架和主要内容。

1. 第2章 问题调研

本章节主要是介绍课程论文解决的问题存在于实际应用中的场景，介绍了该方案提出的必要性。

1. 第3章 系统方案设计原理

本章节主要是介绍了论文中的系统方案设计的基本要求，实现的机制和原理。

1. 第4章 系统实现

本章节主要是介绍系统实现架构，系统工作流程然后对系统的核心部分做了介绍。

1. 第5章 系统性能评估

本章首先介绍系统的进行性能测试的实验环境，然后介绍了开销代价的确定，最后得到不同的实验环境下的性能增益。

1. 第6章 总结思考

总结论文的工作并结合自己所做的项目实际对相关研究工作的思考。

# 问题调研

本章主要介绍研读的论文解决的问题存在于实际应用中的场景，介绍了该方案提出的必要性。

随着科技的发展，各种数据信息急剧增长，其增长速度已经远远地超过了摩尔定律。尤其是大量来自搜索、社交网络等领域的数据，在以一种前所未有的速度占据着我们的存储设备，而且这类数据还有一个明显的特点就是数量多，单个文件小。于是在实际应用存在以下现象：

对小文件的频繁访问：在以往的研究中表明小文件接收大部分文件引用。桌面文件系统的内部分析验证了大部分的访问是对小于32个字节文件的操作。而且访问磁盘上的小文件的访问时间的40%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相关联，其跟踪文件块的位置，访问许可等的信息。然而，许多文件共享相似的文件属性，因为文件所有者的数目，许可模式等是有限的。在典型工作站中，元数据压缩率高达75％。 因此，这里有很多减少冗余元数据信息的机会。

分组访问文件：文件倾向于集中访问。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，但是取出每个文件和相关的元数据访问的单独行为会带来很高的开销。 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％，即使使用温缓存也是如此。

以上的这些问题是我们在实际应用中调研发现的，而且这些问题的出现十分广泛，因为目前大量的数据操作即是如此，所以引起了我们对是否能够通过合并一起访问的小文件来提高性能。首先我们就需要解除逻辑文件和物理元数据代表信息的一对一耦合关系来实现。

# 系统方案设计原理

本章节主要是介绍了论文中的系统方案设计的基本要求，实现的机制和原理。

## 设计概述

复合文件系统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i节点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建，检查和更新，使得传统访问语义（例如类型，权限，时间戳）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。

决定哪些文件要组合成一个复合文件是一个重要的依赖于工作负载的策略决策。在CFFS中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复合文件的文件。 三是基于频率挖掘的合并，通过设置频率挖掘对文件参考进行分析，以便经常一起访问的文件形成复合文件。

一个复合文件利用了传统的VFS预取机制，因为整个复合文件可以被预取为一个单元以与通过将小数据块组合成更少的较大块的与FFS类似的方式

## 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，双重间接块等。复合文件的最大大小限制不是问题，因为复合文件被设计为对小文件进行分组。 如果子文件大小的总和超过了最大文件大小限制，我们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，但是数据布局是基于最普遍的访问模式。

## 元数据表示和操作

复合文件创建：创建复合文件时，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，最终用户仍然可以感知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2）。

I结点内容重构：重复删除的子文件i节点即时重建。 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，首先根据组合i节点检查权限。 如果失败，则不需要进一步检查。 否则，如果一个子文件有一个不同的权限存储为扩展属性，权限将被再次检查。 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，如果在一个复合文件中，我们有一个只读的子文件A和一个可写的子文件B，复合i节点的权限将是读写。但是，当用写权限的打开子文件A时，在扩展属性中的只读权限限制将捕获违规。

时间戳：各个子文件和复合文件的时间戳会随每个文件操作而更新。 但是，在检查期间（例如统计系统调用），我们返回子文件的时间戳。

大小：对于数据访问，偏移量通过在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。

I节点命名空间：对于大于阈值X的i-节点号，将高位零扩展N位用于复合i节点号，并且将低位M位保留用于子文件ID。 我们将这个范围的i-节点号码称为CFFS唯一ID（CUID）。

子文件查询和重命名：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，因此我们需要存储后台指针[CHI12]，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更新：当一个子文件被添加到一个复合文件时，它被追加到复合文件中。 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。

子文件的打开/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，文件位置指针已翻译。

子文件写操作：自身更新按照传统文件系统中相同的方式处理。 但是，如果更新包含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，即不断增长的子文件可能在不久的将来再次增长。

硬链接：目录中 不同的名字可以被映射到相同的I节点或者CUID。

空间压缩：复合文件压缩它的空间，当一半分配的大小不包含有用的数据时。

复合文件内的子文件的并发更新：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。

锁和一致性：CFFS不支持集群，但是我们相信实现子文件锁子系统是可能的。

## 识别复合文件成员

鉴于遗留文件系统具有围绕目录的深根空间局部优化，目录是文件访问模式和用于形成复合文件的良好近似。 目前，这个合并计划不包括子目录。但是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。 而且，它不会捕获跨目录的文件关系。

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，我们合并原始的html文件和引用的文件。 类似的想法适用于编译。 我们可以从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。基于嵌入式参考的方案可以识别跨目录访问的相关文件，但是可能不容易提取超出基于文本的文件格式（例如，html，源代码）的嵌入式文件引用。 另外，它需要知道特定的文件格式。

　在我们对基于频率挖掘的整合的研究中，我们使用了Apriori算法的一个变种。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）。说明了对文件A，B，C，D和E的访问流的算法。首先，我们计算每个文件的访问次数，然后删除计数小于阈值（例如2）的文件以供进一步分析。对于其余文件，我们对所有可能的双文件引用集进行排列，构建和计数。只要在B之后访问文件A，反之亦然，我们增加文件集{A，B}的计数。 计数小于阈值的集被删除（例如，{B，D}）。我们可以根据剩余的两个文件参考集生成所有三个文件的参考集。 但是，如果一个三文件参考集经常出现，那么它所有的两文件参考集也需要频繁出现。 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。由于我们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，我们将集合{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。

另一种方法是使用归一化的阈值或支持度，它是设置出现次数的百分比（出现次数除以总出现次数，介于0和1之间）。我们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。我们目前不允许重叠的文件集，以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。基于频率挖掘的合并可以基于动态文件引用来识别复合文件候选。 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。

# 系统实现

本章节主要是介绍系统实现架构，系统工作流程然后对系统的核心部分做了介绍。

我们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。

我们通过在Linux 3.16.7上运行的FUSE（v2.9.3）框架在用户空间中创建了CFFS原型。 CFFS堆叠在ext4上，所以我们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），扩展属性和日记功能。CFFS定期与生成器工具进行协商以创建新的复合文件。 我们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，我们拦截了所有与文件系统相关的调用。 我们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）和时间戳（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，我们使用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，我们关注两种情况。 对于Web服务器工作负载，我们合并HTML文件及其立即引用的文件。 在复合文件成员资格冲突的情况下，首选index.html，然后是首先包含文件的html。 另一个是源代码编译。 我们使用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。

# 系统性能评估

本章首先介绍系统的进行性能测试的实验环境，然后介绍了开销代价的确定，最后得到不同的实验环境下的性能增益。

我们通过FUSE比较了ext4上堆叠的CFFS与基准ext4文件系统（通过一个空的FUSE模块路由请求）的性能。我们通过重播两条曲线来评估我们的系统。第一条是从部门网络服务器收集的http日志。 跟踪包含14M文件对1.0TB数据的引用; 对此，3.1M文件是独一无二的，持有76GB的数据。 第二个跟踪是通过软件开发工作站收集的。 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。

我们在存储设备上进行了多线程，零思考时间的跟踪回放。 我们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。 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，我们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。

## 网页服务器跟踪重放

HDD性能：我们工作的最初意图是减少经常访问的小文件的I／O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。 我们检查了踪迹，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：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。

## 软件开发文件系统跟踪重播

对于软件开发工作负载重放来说，捕获单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），并且像mmap这样的调用不知道请求发送到底层存储。所以我们用总时间总结了我们的结果，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。

HDD性能：基于嵌入式参考的方案覆盖率较差，因为许多参考文献与编译无关。 因此，经过的时间更接近于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。 类似于网页跟踪，增益高达20％。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。

基于目录和嵌入式引用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。 对于我们的工作负载，这个成本在1到14分钟之间。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，所以这个成本被优势所抵消。

跟踪采集开销低于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。

## 讨论

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数量的减少。 SSD和HDD之间的性能差距缩小表明，由于磁盘搜索量减少而导致的性能提升，修改后的数据布局高达10％。

虽然该论文侧重于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数量优化小文件访问的方式密切相关访问。 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搭配并将i-节点嵌入到目录中。 之后，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b-树上。 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。在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多台电脑需要访问同一个大型数据文件。 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。虽然可以发现大量的工作来提高预取，但是C-Miner也许是最接近我们的工作的。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，我们的文件系统级挖掘处理的对象和关联开销也明显减少。 DiskSeen结合了磁盘布局的知识来改善预取，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）传递给锁层，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，因此我们可以使用PID和IP地址来解开并发文件引用。然而，CFFS的重点是改变逻辑文件到其物理表示的映射可以采用各种挖掘算法来合并元数据，改善存储布局。

# 总结

总的来说，这篇论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，优化其存储。 这种方案在实证实验证明下表明在两个真实的工作负载下，性能提高达27％。 所以说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。因为我目前所在的实验室项目组需要完成一个非结构化数据管理系统其中涉及到大量的工业数据，比如大量的风机的工作数据记录，图片信息，它们每一个的大小很小，但是数量很多，我们采用传统的文件系统方案设计系统时发生了索引爆炸的现象，元数据所在的Namenode节点存储空间已满，但是Datanode的空间还有很多剩余，于是造成了大量的空间浪费，存储效果很差，于是目前在研究小文件合并的方法策略，看了这篇论文后感觉思路更明确了，我觉得这种多对一映射和小文件合并有异曲同工之妙，前者通过可以聚集访问的小文件同时映射到一个物理元数据，而小文件合并也是通过某种有效的算法把选择一些关联的小文件合并起来，这时合成一个大文件便只需要一个元数据了。从最后的实现上看有这些相似但其实本质上还是有差异的，总之在海量小文件存储这个问题上还要继续学习研究下去，比如结合项目实际情况如何去解决存储问题，然后在存储的基础上又如何使查询，删除和修改的性能更好，结合论文中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又是怎样的，更全面地探索元数据压缩，并发和安全的后果等。感谢何老师在这门课程中 的指导与帮助！

参考文献

[1] Zhang S, Catanese H, Wang A I A. The Composite-file File System: Decoupling the One-to-One Mapping of Files and Metadata for Better Performance[C]//FAST. 2016: 15-22.

[2] 吕雪锋, 程承旗, 龚健雅,等. 海量遥感数据存储管理技术综述[J]. 中国科学:技术科学, 2011(12):1561-1573.

[3] 侯建, 帅仁俊, 侯文. 基于云计算的海量数据存储模型[J]. 通信技术, 2011, 44(5):163-165.

[4] 吴明礼, 张宏安. 数据存储技术综述[J]. 北方工业大学学报, 2015, 27(1):30-35.

[5] 周国安, 李强, 陈新,等. 云环境下海量小文件存储技术研究综述[J]. 信息网络安全, 2014(6):11-17.