**课程论文**

**Hops FS:使用New SQL数据库扩展分层文件系统元数据**

****

院（系）名 称 ：计算机学院

专 业 名 称 ：计算机技术

学 生 姓 名 ：代 文 文

学 号 ：2017282110244

课 程 名 称 ：海量存储技术

二○一七年十二月

HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases

HopsFS:使用 NewSQL 数据库扩展分层文件系统元数据

Abstract

最近在无共享，事务性，内存中的NewSQL数据库性能和可伸缩性方面的改进已经重启研究课题，即是否可以使用商品数据库管理分层文件系统的分布式元数据。在本文中，我们将介绍HopsFS，它是Hadoop分布式文件系统(HDFS)的下一代分布，它取代了HDFS的单一节点内存元数据服务，并在一个新的sql数据库上建立了分布式元数据服务。通过删除元数据的瓶颈，HopsFS可以实现比HDFS更大、更高的吞吐量集群。元数据容量已经增加到至少37倍的HDFS容量，并且在基于Spotify的工作负载跟踪的实验中，我们显示HopsFS支持16到37倍的Apache HDFS的吞吐量。对于许多并发客户机来说，HopsFS也有较低的延迟，在故障转移期间也没有停机时间。最后，由于元数据现在存储在商品数据库中，因此可以安全地扩展并轻松导出到外部系统进行在线分析和自由文本搜索。

1 Introduction

分布式文件系统是许多大型数据并行处理系统的重要基础组件，如MapReduce[13]、Dryad[27]、Flink[5]和Spark[77]。到本世纪末，存储多个exabytes（艾字节216）数据的数据中心将不会罕见[12,47]。对于大型分布式分层文件系统，元数据管理服务是可伸缩性瓶颈[62]。许多现有的分布式文件系统将元数据存储在单个节点或共享磁盘文件系统，如存储区域网络(SAN)，两者都具有有限的可伸缩性。著名的例子包括GFS[17]，HDFS[61]，QFS[41]，Farsite[3]，Ursa Minor[2]，GPFS[58]，Frangipani[67]，GlobalFS[50]和UNK as[73]。其他系统通过静态地分出命名空间并将碎片存储在不同的主机上来扩展元数据，例如NFS[44]，AFS[36]，MapR[64]，Locus[49]，Coda[57]，Sprite[40]和XtreemFS[26]。但是，静态切分命名空间会对跨不同碎片的文件系统操作产生负面影响，尤其是移动操作。此外，它还使文件系统的管理变得复杂，因为管理员必须将元数据服务器映射到命名空间的碎片，这些碎片随着时间的推移会改变大小。

最近在无共享，事务性，内存中的NewSQL[42]数据库性能和可伸缩性方面的改进重新开放了在商品数据库中存储分布式文件系统元数据的可能性。迄今为止，传统观点一直是这样的，在分布式数据库中，存储分层文件系统元数据(在吞吐量和延迟方面)过于昂贵[59,33]。

在本文中，我们将介绍如何使用NewSQL数据库构建高吞吐量和低操作延迟分布式文件系统。我们介绍了HopsFS，它是Hadoop分布式文件系统(HDFS)的一个新的发行版，它将文件系统元数据存储和管理服务解耦了。HopsFS将所有的元数据都存储在高度可用的、内存中的、分布式的、关系数据库中，称为网络数据库(NDB)，这是MySQL集群的一个新sql存储引擎[38,54]。HopsFS提供冗余无状态服务器(名称节点)并行地读取和更新存储在数据库中的元数据。

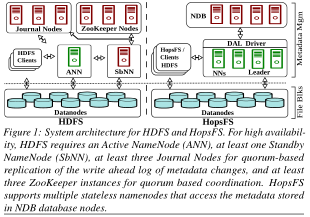
HopsFS将文件系统操作封装到分布式事务中。为了提高文件系统操作的性能，我们利用了经典数据库诸如批处理(批量操作)和在事务中提前写缓存的技术，以及在新sql数据库中常见的分布感知技术。这些分布感知的NewSQL技术包括应用程序定义的分区（我们分区的命名空间，一个目录的所有后裔（子文件/目录）的元数据驻留在同一个数据库碎片作为有效的目录列表）和分布感知事务(我们在数据库碎片上启动一个事务，该事务存储文件系统操作所需的所有/大部分元数据)，以及分区修剪索引扫描(扫描操作本地化为单个数据库碎片[78])。我们还引入了一个索引节点提示缓存，以加快文件路径的解析。在解决深度N的路径时，缓存命中可以减少从N到1的数据库往返次数。

但是，在大目录子树上的一些文件系统操作(例如移动和删除)可能太大，无法在单个数据库事务中执行。例如，由于数据库管理系统所施加的限制，删除包含数百万个文件的文件夹不能用单个数据库事务来执行。

但是，在大目录子树(例如移动和删除)上的一些文件系统操作可能太大，无法在单个数据库事务中执行。例如，由于数据库管理系统对可以包含在单个事务中的操作数的限制，删除包含数百万个文件的文件夹不能执行单个数据库事务。对于这些子树操作，我们引入了一种新的协议，它使用应用级分布式锁定机制来隔离大型子树来执行文件系统操作。在隔离子树后，大型文件系统的操作被分解

为并行执行的较小事务。子树操作协议确保在执行操作失败的命名节点时不会违反名称空间的一致性。

HopsFS是一个HDFS的替代品。HopsFS自2016年4月以来一直在生产运行，为瑞典Lulea一家数据中心的研究人员提供hadoop即服务[63]。在实验中，使用Hadoop / Spark应用程序生成的实际工作负载，我们得出，HopsFS的吞吐量比HDFS高16倍，而HopsFS在故障转移期间没有停机时间。对于更高的写密集型工作负载，HopsFS提供的吞吐量是HDFS的37倍。在我们的知识中，HopsFS是第一个在分布式关系数据库中存储完全标准化的元数据的开源分布式文件系统。



2 Background

本节描述Hadoop分布式文件系统(HDFS)和MySQL集群网络数据库(NDB)存储引擎。

2.1 Hadoop Distributed File System

Hadoop分布式文件系统(HDFS)[61]是谷歌文件系统的一个开源实现[17]。HDFS的元数据存储在名为Active 名称节点(ANN)的单个Java进程的堆上，见图1。这些文件被分割成小的(通常是128 MB)的块，默认情况下，这些块会在数据节点上重复进行三次复制。为了获得元数据管理服务的高可用性，活动的名称节点日志使用基于仲裁的复制将元数据更改到日志服务器。元数据更改日志被异步复制到备用名称节点(SbNN)，它也执行检查点功能。在HDFS中，ZooKeeper的协调服务[25]使这两种机器都能达成一致，即运行主动名称节点(防止一个分裂的大脑场景)，以及协调从活动名称节点到备用名称节点的故障转移。

名称节点为数据节点和客户机的潜在的您提供服务，并通过原子地执行文件系统操作来保持元数据的一致性。名称节点通过在整个文件系统元数据上使用单个全局锁来实现原子操作，提供单作者、多读者并发语义。一些大型的文件系统操作不是原子性的，因为它们会把全局锁保存得太久，让客户感到饥饿。例如，删除大目录是批量执行的，首先删除索引节点，然后在以后的阶段删除块。此外，由于将名称空间更改写入日志节点的仲裁时间可能需要很长时间，在日志记录操作之前，全局文件系统锁将被释放，以防止其他客户机处于饥饿状态。并发客户端可以在之前的操作被记录之前，在名称节点故障转移过程中，以不一致的文件系统操作的代价来阻止对文件系统的锁定。例如，当活动的名称节点失败时，没有登录到日志节点的所有更改将丢失。

数据节点连接到活动的和备用的名称节点。所有数据节点都会定期生成一个包含其自身存储块信息的块报告。名称节点处理块报告，以验证名称节点的块映射与实际存储在数据节点中的块的一致性。

在HDFS中，相对于文件数据，元数据的数量相当低。每一PB字节文件系统数据的大约有1g的元数据[62]。Spotify的HDFS集群有1600个节点，存储了60个PB字节数据，但它的元数据适合于140 GB的Java虚拟机(JVM)堆。额外的堆空间由临时对象、RPC请求队列和维护文件系统所需的辅助元数据组成。然而,目前的趋势是走向更大的HDFS集群(Facebook HDFS集群与超过100 PB的数据[48]),但当前JVM垃圾收集技术不允许非常大的堆大小,当应用程序暂停JVM垃圾收集器会影响操作造成的HDFS[22]。因此，JVM垃圾收集技术和HDFS 名称节点的整体架构现在是Hadoop的可伸缩性瓶颈[62]。这种架构的另一个限制是，数据结构被优化以减少它们的内存占用，结果是元数据很难修改或导出到外部系统。

2.2 Network Database (NDB)

MySQL集群是一个不共享的、复制的、在内存中、自动分片、一致的、新的sql关系数据库[38]。网络数据库(NDB)是MySQL集群的存储引擎。NDB支持数据节点级和集群级故障恢复。使用事务重做和撤消日志执行数据节点级别的故障恢复。NDB 数据节点还会异步地将它们的状态快照到磁盘，以限制日志的大小，并提高数据节点的恢复时间。集群级别的恢复使用一个全局检查点协议来支持，该协议每2秒递增一次全局的划时代id。在集群级别的恢复中，数据节点将所有事务恢复到最新的epoch-id。

NDB水平划分了被称为NDB 数据节点的存储节点之间的表。NDB还支持用于表的应用程序定义分区(ADP)。事务协调器位于所有NDB 数据节点中，在数据碎片之间实现高性能事务，即多分区事务。通过提供一个基于应用程序定义的分区方案的提示，可以在NDB 数据节点上启动包含该事务读取/更新的数据的事务，从而有可能实现分发感知的跨操作(DAT)。特别是，单行读取操作和分区修剪索引扫描(扫描操作，其中一个数据碎片参与)从分布感知事务中获益，因为它们可以在本地读取所有数据[78]。不正确的提示导致额外的网络流量发生，但在其他方面是正确的系统操作。

2.2.1 NDB Data Replication and Failure Handling

NDB 数据节点被组织到节点组中，其中数据复制因子R决定了节点组中的数据节点数量。给定一个集群大小N，就有N / R节点组。NDB将表(默认情况下的散列分区)划分为分布在节点组上的一组固定分区。新节点组可以在线添加，现有数据自动重新平衡到新的节点组。分区是节点组存储和复制的数据的一部分。每个数据节点存储分配给它的节点组的分区的副本(副本)。在NDB中，默认复制度为2，这意味着每个节点组可以容忍一个NDB 数据节点失败，因为节点组中的其他NDB 数据节点包含完整的数据副本。因此，一个12节点的NDB集群有6个节点组可以容忍6个NDB 数据节点失败，只要在每个节点组中有一个幸存的NDB 数据节点。为了在一个节点组中容忍多个故障，可以以较低的吞吐量增加复制度。

2.2.2 Transaction Isolation

NDB只支持已提交的事务隔离，这保证了在读取时读取的任何数据都是提交的。read- committed隔离级别不允许脏读，但幻象和模糊(不可重复的)读取可以在事务中发生[7]。然而，NDB支持行级锁，例如，独占(写)锁、共享(读)锁和可以用来隔离冲突事务的已提交锁。

3 HopsFS Overview

HopsFS是HDFS v2.0.4的一个分支。与HDFS不同，HopsFS通过解耦元数据存储和操作服务提供了一个扩展的元数据层。HopsFS支持使用Java编写的多个无状态名称节点来处理客户端的请求，并处理存储在外部分布式数据库中的元数据，请参见图1。每个名称节点都有一个数据访问层(DAL)驱动程序，类似于JDBC，它封装了所有允许HopsFS在各种新sql数据库中存储元数据的数据库操作。在名称节点中，必须协调内部管理(例如:数据节点故障处理)操作。HopsFS通过选举负责管理的领导名称节点解决了这个问题。HopsFS使用数据库作为共享内存来实现一个领导者选举和会员管理服务。领导者选举协议为每个名称节点分配一个惟一的ID，而名称节点的ID在名称节点重新启动时更改。领导选举协议定义了一个活着的名称节点，它可以在有限的时间内写入数据库，在[56]中可以找到详细信息。

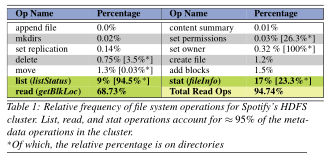
客户端可以选择随机的、循环的和粘性的策略，以选择一个名称节点来执行文件系统操作。HopsFS客户端定期刷新名称节点列表，使新的名称节点可以加入操作集群。HDFS v2.x客户端完全兼容HopsFS，尽管他们不将操作分发给名称节点，因为他们假设有一个活动名称节点。与HDFS一样，数据节点连接到所有名称节点，但是数据节点将块报告发送给只有一个名称节点。领导名称节点负载平衡块报告所有活着的名称节点。

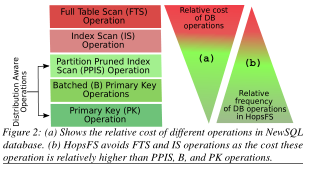
在第4部分中，我们讨论了HopsFS的自动分片方案如何使普通文件系统操作使用低成本的数据库访问查询来读取元数据。第5节讨论如何通过将文件系统操作转换为分布式事务来维护文件系统元数据的一致性，以及如何使用每个事务和名称节点级别缓存减少分布式事务的延迟。然后，在第6节中，引入了一个协议来处理那些太大而不能适合单个数据库事务的文件系统操作。

4 HopsFS Distributed Metadata

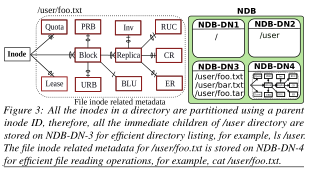
分层分布式文件系统的元数据通常包含关于索引节点、块、副本、配额、租约和映射的信息(目录到文件，文件到块，和块到副本)。当元数据被分发时，需要一个应用程序定义的分区方案来分割元数据，并需要一个协商一致协议，以确保跨切分的操作的元数据完整性。基于仲裁的一致性协议，如Paxos，在单个碎片中提供高性能，但通常与使用两阶段提交协议的事务相结合，用于跨切分的操作，如Megastore[6]和Spanner [ 10 ]。在HopsFS中的文件系统操作主要是使用多分区事务和MySQL集群中的行级锁来提供对元数据操作的可串行化[23]。

分层命名空间的分区方案的选择是分布式元数据架构的一个关键设计决策。我们根据预期的HDFS操作在生产部署中的相对频率，以及可用于实现文件系统





操作的不同数据基操作的成本，建立分区方案。表1显示了Hadoop应用程序生成的工作负载中所选择的HDFS操作的相对频率，例如，Pig、Hive、HBase、MapReduce、Tez、Spark和Spotify中的Giraph。列表,统计和文件读取操作仅占在HDFS集群≈95%的业务。这些统计数据类似于Yahoo[1]、LinkedIn[52]和Facebook[65]的Hadoop集群发布的工作负载。图2a显示了不同数据库操作的相对成本。我们可以看到，全表扫描或索引扫描(所有数据库碎片都参与其中)的成本要远远高于仅一个数据库碎片参与的分区修剪索引扫描。HopsFS元数据设计和元数据分区支持使用低成本数据库操作实现普通文件系统操作，即主键操作、批处理主键操作和分区修剪索引扫描。例如，读取和目录列表操作，只使用(批处理)主键查找和分区修剪索引扫描来实现。在可能的情况下，可以避免索引扫描和全表扫描，因为它们会影响所有的数据库碎片和伸缩性。



4.1 Entity Relation Model

在HopsFS中，文件系统元数据存储在表中，目录索引节点由索引节点表中的一行表示。然而，文件索引节点有更多相关的元数据，例如一组块、块位置和校验和，它们存储在单独的表中。

图3显示了在HopsFS元数据模型中描述关键实体的实体关系模型。文件和目录由索引节点实体表示，该实体包含对文件系统层次结构中的父索引节点(父索引节点 ID)的引用。在索引节点条目中，我们存储路径单个组件，而不是完整路径。每个文件包含存储在块实体中的多个块。每个块副本的位置存储在副本实体中。在它的生命周期中，一个块经过不同的阶段。如果一个数据节点失败，并且这些块存储在被复制的块表中(URB)，那么块可能会被重复使用。复制管理器位于领导上，向数据节点发送命令，以创建被重复复制的块的更多副本。

正在进行复制的块存储在挂起的复制块表(PRB)中。同样地，一个块的复制品在它的生命周期中有不同的状态。当一个副本被损坏时，它被移动到被破坏的副本(CR)表中。每当客户端写到一个新块的副本时，这个副本就被移动到正在构建的副本(RUC)表中。如果有太多的副本存在(例如，由于恢复了包含被重新复制的块的数据节点)，多余的副本存储在多余的副本(ER)表中，而计划删除的副本则存储在失效(Inv)表中。注意，索引节点相关实体的文件还包含索引节点的外键(图3中没有显示)，这也是分区键，使HopsFS可以使用分区修剪索引扫描来读取文件索引节点相关的元数据。

4.2 Metadata Partitioning

除了热点区域(请参阅下面的小节)，HopsFS分区是由他们的父母的索引节点 id进行的，导致索引节点与存储在同一个数据库碎片上的相同的父节点。这具有在所有数据库碎片中均匀地划分元数据的效果，并且能够有效地实现目录列表操作。当在目录中列出文件时，我们使用一个提示机制启动事务协调器的事务，该事务协调程序位于数据库碎片上，该事务协调程序将为该目录保存子节点。然后，我们可以使用修剪索引扫描来检索本地目录的内容。文件索引节点相关的元数据，即块、复制映射和校验和，使用文件的索引节点 ID进行分区，这将导致一个给定文件的元数据被存储在一个数据库碎片中，再次启用高效的文件操作，见图3。

4.2.1 Hotspots

热点是接收高比例文件系统操作的索引节点。可以在热索引节点上执行的文件系统操作的最大数量受到存储索引节点的数据库碎片的吞吐量的限制。目前，HopsFS没有建立在运行时识别热点的机制。

所有文件系统操作都涉及到解决路径组件，以检查用户权限和路径的有效性。在所有文件系统有效路径中共享根索引节点。当然，所有文件系统路径解析操作都读取根索引节点。由于所有文件系统操作都将从相同的数据库碎片中检索根索引节点，因此存储根索引节点的数据库碎片成为一个瓶颈。HopsFS在所有名称节点中缓存根索引节点。在HopsFS中，根索引节点是不可变的，也就是说，我们不允许操作，例如，重命名、删除或修改根索引节点的权限。使根索引节点不可变，可以防止其缓存导致的任何不一致。

在HopsFS中，所有路径解析操作从第二个路径组件开始(也就是顶层目录)。对于顶级目录，我们的分区计划无意中引入了一个热点——所有顶级目录和文件都是根目录的子目录，因此，它们位于同一个数据库碎片上。在这些索引节点上的操作由数据库中的一个碎片处理。为了克服这一瓶颈，HopsFS使用了一个可配置的目录分区方案，其中，顶层目录的直接子项是伪随机分区的，通过哈希这些孩子的名字。默认情况下，HopsFS伪随机划分文件系统层次结构的前两个层次，即根目录及其直系后代。但是，根据文件系统的工作负载，可以将其配置为伪随机分区，以降低移动速度和在文件系统层次结构顶层的ls操作。

5 HopsFS Transactional Operations

HopsFS中的事务性元数据操作属于两个类别中的一个:在单个文件、目录或块上操作的索引节点操作(例如，创建/读取文件、mkdir，以及块状态更改操作)，以及在未知的地方运行的子树操作索引节点的数量，可能有数百万，(例如，递归删除、移动、chmod和chown在非空目录上)。

本节描述了HopsFS如何有效地封装在NDB中的事务中的索引节点操作。NDB提供的最强大的事务隔离级别是read- committed，它不够强大，不能提供至少与HDFS一样强大的一致性，它使用单个全局锁来序列化所有的HDFS操作。为此，我们使用行级锁定来序列化反弹的索引节点操作。也就是说，操作是并行执行的，因为它们不会在相同的索引节点上使用冲突锁。然而，在事务中使用多个锁可能导致大量的死锁和事务超时。原因如下:

循环死锁:在HDFS中，并不是所有的索引节点操作都是在锁定元数据时使用相同的顺序，在我们的情况下会导致循环死锁。为了解决这个问题，我们重新实现了所有的索引节点操作，以使它们以相同的总顺序获取元数据上的锁，从根下遍历文件系统树，使用左排序的深度优先搜索来离开节点。

锁升级:在HDFS中，许多索引节点操作都是在相同的元数据上进行写操作的。当在同一个事务中转换为数据库操作时，这会导致死锁，因为从读到独占锁的升级。我们已经检查了所有的锁都被索引节点操作,并重新实现,这样所有的数据需要以反式-行动开始时只读取一次事务(参照5.2.1节Lock Phase)在最强的锁级别可能需要在事务,从而防止锁升级。

5.1 索引节点 Hint Cache

解决路径和检查权限是目前大多数HDFS工作负载中最常见的操作，见表1。在HDFS中，完整路径被递归分解为单个组件。在HopsFS中，对于深度N的路径，需要N次往返于数据库以检索文件路径组件，从而导致文件系统操作的高延迟。

与AFS[36]和Sprite[40]类似，我们使用提示[30]来加速路径查找。提示是快速检索并行(批处理操作)中的文件路径组件的机制。在我们的分区方案中，索引节点有一个复合主键，其中包含父索引节点的ID和索引节点的名称(即文件或目录名)，而父索引节点的ID作为分区键。每个名称节点只缓存索引节点的主键。给定一个路径名称和对所有路径组件目录的命中，我们可以发现所有路径组件的主键，这些组件用于并行地读取路径组件，它们使用的是只包含主键查找的单个数据库批处理查询。

5.1.1 Cache Consistency

我们使用索引节点提示缓存条目，在一个文件系统操作的事务开始时，在单个批处理查询中读取整个索引节点。如果提示条目无效，则主键读操作失败，路径解析返回到解析文件路径组件的递归方法，然后修复缓存。缓存条目不经常变得过时，因为移动操作更新了索引节点的主键，在典型的Hadoop工作负载中只有不到2%的操作，见表1。此外，典型的文件访问模式遵循一个重尾分布(在Yahoo 3%的文件中，80%的访问[1])，并且使用一个粘性策略为HopsFS客户端改进时间局部性和缓存命中率。

5.2 Inode Operations

HopsFS实现了一个悲观的并发模型，它支持在名称上并行读写操作——速度、序列化冲突的索引节点和子树操作。我们选择了一个悲观的方案，与乐观的并发控制相比，它为中等到高水平的资源利用提供了良好的性能[4]，而许多HDFS集群，如Spotify，在高负载下运行。索引节点操作被封装在单个事务中，由三个不同的阶段组成，即锁定、执行和更新阶段。

5.2.1 Lock Phase

在锁阶段，元数据被锁定并从数据库中读取最强大的锁，这将是事务持续时间所需要的。锁是按照前面定义的全部顺序进行的。索引节点操作是基于路径的，如果它们不是只读操作，它们只修改路径的最后一个组件(例如，rm / etc/conf和chmod +x /bin/ script)。因此，只有文件路径的最后一个组件被锁定为文件系统操作。

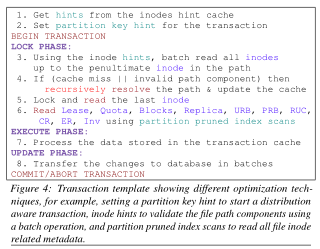


图4显示了HopsFS 索引节点操作的事务模板。使用索引节点提示缓存文件路径组件的主键被发现，第1行。事务开始于保存所有或大部分所需数据的数据库碎片，第2行。一个批处理操作读取所有文件路径组件到倒数路径组件，而没有锁定元数据(read- committed)，第3行。对于深度N的路径，这将移除N - 1次访问数据库。如果索引节点提示无效，那么文件路径将被递归地解析，而索引节点提示缓存将被更新，第4行。

在路径被解析之后，在路径中的最后一个索引节点组件上使用一个共享或一个独占锁，第5行。共享锁用于只读的索引节点操作，而对于修改名称空间的索引节点操作，则采用独占锁。此外，根据操作类型和提供的操作参数，索引节点相关的数据，例如块、副本和PRB，使用分区修剪扫描操作，第6行，从数据库中读取数据。

HopsFS对索引节点操作使用分级锁定[19]，也就是说，如果数据按照树状结构排列，所有数据操作操作从上到下遍历整个层次结构，然后在树/子树的根上锁定一个锁，隐式地锁住树/子树的子树。有关文件索引节点相关数据的实体关系图，见图3，显示实体在树根上有一个索引节点实体的树。也就是说，使用索引节点的锁隐式锁住了文件索引节点相关数据树。与所有操作一样，索引节点首先被读取，然后是相关的元数据。对于一些操作，例如创建文件/目录和清单操作，父目录也被锁定，以防止对文件系统操作的幻象和模糊读取。

5.2.2 Per-Transaction Cache

从数据库读取的所有数据都存储在每个事务缓存(快照)中，该缓存将更新的缓存记录的传播保存到数据库，直到事务结束。缓存节省了许多往返于数据库的往返行程，因为元数据通常在同一事务中多次读取和更新。元数据的行级锁定确保了缓存的一致性，也就是说，没有其他事务可以更新元数据。此外，当锁在完成事务时释放时，缓存就被清除了。

5.2.3 Execute and Update Phases

索引节点操作是通过在每个事务缓存中处理元数据来执行的。在第二个阶段生成的更新和新的元数据存储在缓存中，在最后的更新阶段被批量发送到数据库，之后事务要么提交，要么回滚。

6 Handling Large Operations

包含数百万索引节点的大目录的递归操作太大，无法在单个事务中进行匹配，也就是说，在现有的在线事务处理系统中不支持将数百万行锁在事务中。这些操作包括移动、删除、更改所有者、更改权限和设置配额操作。Move操作改变所有后代索引节点的绝对路径，而delete则删除所有的后代索引节点，而设置的配额操作会影响到所有后代节点如何消耗磁盘空间，以及它们可以创建多少文件/目录。同样，更改目录的权限或所有者可能会使在较低子树中执行的操作失效。

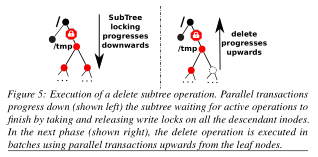
6.1 Subtree Operations Protocol

我们的解决方案是一个在批量事务中逐步实现子树操作的协议。我们的子树操作协议使用应用程序级分布式锁定机制来标记和隔离子树，而不是行级数据库锁。我们通过确保在执行新请求的子树操作之前在子树中进行所有正在进行的索引节点和子树操作来序列化子树操作。我们实现这个序列化属性通过执行以下不变量:(1)在操作完成之前，没有新的操作访问子树，(2)在子树操作开始之前，子树被静默，(3)如果出现故障，则不会出现孤立的索引节点或不一致。

我们的子树操作协议提供了与在HDFS中的子树操作相同的一致性语义。对于删除子树操作，HopsFS提供了更强的一致性语义。在HDFS中失败的删除操作可能会导致最终被阻塞报告子系统(后数小时)所声称的孤立块。HopsFS改进了删除操作的语义，因为失败的操作不会导致任何元数据不一致，见第6.2节。子树操作有以下阶段。

阶段1:在第一个阶段，在子树的根上获得一个独占锁，而子树锁标记(也包含拥有锁的名称节点的ID)被设置并保存在数据库中。该标志指示子树的所有后代都锁定了独占(写)锁。

在设置锁之前，必须在子树的任何较低级别上没有其他活动子树操作。设置子树锁可以阻止在子树的子集上执行的活跃子树操作。我们将所有活动的子树操作存储在一个表中，并查询它以确保在子树的较低级别上没有执行子树操作。在典型的工作负载中，这个表不会变得太大，因为子树操作通常只占所有文件系统操作的很小一部分。需要注意的是，在路径解析过程中，使用子树锁的索引节点和子树操作会自动中止事务并等待直到删除子树锁。



阶段2:对子树进行查询，我们等待所有正在进行的索引节点操作完成，并在子树的所有索引节点上执行和释放数据库写锁，这与锁住索引节点的完全相同。为了有效地做到这一点，并行执行分区的一个线程池对写锁子节点进行了修剪。这是在子树下对叶子进行重复，树的数据结构包含子树中的索引节点，在名称节点的内存中构建，见图5。该树后来被一些子树操作使用，例如移动和删除操作，以处理索引节点。我们通过使用投影只读取索引节点 id来减少在子树中读取所有索引节点的开销。如果子树操作协议由于在子树上并发的文件系统操作而不能对子树进行quiesce，那么它将以指数型后退来尝试。

阶段3:在最后阶段，文件系统操作被分解为并行执行的较小的操作。为了提高性能，在每个事务中都处理了大量的索引节点。

6.2 Handling Failed Subtree Operations

HopsFS采取懒惰的方法来清理失败的名称节点[45]留下的树锁。每个名称节点维护由领导选举服务提供的活动名称节点的列表。如果操作遇到带有子树锁集的索引节点，而子树锁的名称节点 ID属于死名称节点，则可以清除子树锁。然而，重要的是，当执行子树操作的名称节点失败时，它不应该让子树处于不一致的状态。如果名称节点失败，在第二阶段构建的内存树将在保持名称空间一致性方面发挥重要作用。例如，在删除操作的情况下，在使用事务的后序树遍历方式中，删除子树。如果在操作过程中，名称节点失败了一半，那么未删除的索引节点仍然连接到命名空间树。HopsFS客户端将透明地将文件系统操作重新提交给另一个名称节点，以删除子树的其余部分。

其他子树操作(移动、设置配额、chmod和chown)不会导致任何不一致，因为在第三阶段使用只更新子树的根节点和内部索引节点的单个事务来修改元数据的实际操作。在失败的情况下，名称节点可能无法打开子树锁，但是这并不是一个问题，因为其他名称节点在发现子树锁属于死名称节点时可以很容易地删除子树锁。

6.3 INod and Subtree Lock Compatibility

与索引节点操作的锁定机制类似(参见5.2.1节)，子树操作也实现了分层锁定，也就是说，在目录下设置子树标记会隐式锁住目录的内容。无论是索引节点还是子树锁定机制都是兼容的，它们都尊重它们对应的锁。也就是说，不能在由索引节点操作锁定的目录上设置一个subtree标志，而索引节点操作在遇到带有子树锁的目录时自动中止事务。

7 HopsFS Evaluation

当HopsFS解决如何扩展HDFS的元数据层时，我们所有的实验都是为了相对测试HDFS和HopsFS中名称节点(s)在控制条件下的性能和可伸缩性，从而在大的生产集群中近似真实的文件系统负载。

7.1 Experimental Setup

基准:我们扩展了用于测试Quantcast文件系统性能的基准测试(QFS)[41]，它是谷歌文件系统的一个开源c++实现。基准测试实用程序是一个分布式应用程序，它产生了成千上万的HDFS / HopsFS文件系统客户机，分布在许多机器上，它们同时执行名称节点上的文件系统(元数据)操作。基准实用程序可以根据工业负载跟踪测试单个文件系统操作和文件系统工作负载的性能。HopsFS和基准实用工具是开放源码的，鼓励读者进行他们自己的实验来验证我们的发现[21,24]。

HopsFS设置:所有的实验都是基于Dell PowerEdge R730xd服务器((Intel,R)Xeon(R)CPU e5 -2620 v3 @ 2.40GHz,256 GB RAM,4 TB 7200 RPM HDDs)，连接使用一个10 GbE网络适配器。除非另有说明，NDB(版本7.5.3)部署在配置为使用22个线程的12个节点上，并且数据复制度为2。

HDFS设置:在中型到大型Hadoop集群中，需要5到8台服务器为HDFS元数据服务提供高可用性，请参见图1和第2节。5个服务器的设置包括一个活跃的名称节点，一个备用名称节点，至少三个日志节点，至少有三个ZooKeeper节点。在8个服务器设置中，ZooKeeper节点安装在单独的服务器上，以防止在服务器失败时出现多个服务失败。在我们的实验中，Apache HDFS版本2.7.2部署在5个服务器上。基于Spotify的HDFS运行的经验,我们配置了HDFS 名称节点 240客户端处理程序线程(dfs.namenode.handler.count)。

没有一个文件系统客户端与名称节点或数据库节点联合在一起。由于我们只评估元数据的性能，所以所有的测试都创建了0长度的文件(类似于NNThroughputBenchmark[ 62])。对非空文件进行测试需要一个数量级的HDFS / HopsFS数据阳极，并没有提供进一步的见解。

7.2 Industrial Workload Experiments

我们用基于从Spotify上的基于oper的跟踪的工作负载来对HopsFS进行基准测试，它运行的Hadoop集群由1600个节点组成，其中包含60个pb级的数据。该名称空间包含1300万个目录和2.18亿个文件，其中每个文件平均包含1.3个块。Spotify的Hadoop集群平均每天有4万份来自不同应用程序的工作，比如Pig、Hive、HBase、MapReduce、Tez、Spark和Giraph。表1总结了这些应用程序生成的文件系统工作负载，显示了HDFS操作的相对频率。在Spotify中，平均文件路径深度为7，平均索引节点名称长度为34个字符。平均每个目录包含16个文件和2个文件子目录。在数据节点上有2890万个块。我们使用这些统计数据来生成文件系统工作负载，这些工作负载在Spotify的生产中接近HDFS的使用。

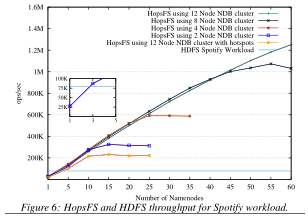
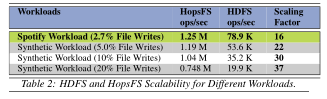


图6显示，对于我们的工业工作负载，使用60个名称节点和12个NDB节点，HopsFS可以每秒执行125万次操作，这是HDFS吞吐量的16倍。如前所述，在中型到大型Hadoop集群中，需要5到8台服务器来提供高可用性的HDFS。与等效硬件(2 NDB节点和3 名称节点),HopsFS提供更高的吞吐量比HDFS≈10%。随着更多的名称节点节点被添加到系统中，HopsFS的性能直线上升。

表2显示了用于编写密集合成工作负载的HopsFS和HDFS的性能。这些合成工作负载来自先前描述的工作负载，但是在这里我们增加了文件创建操作的相对百分比，并减少了文件读取操作的百分比。在这个实验中，HopsFS使用了60个名称节点。由于HopsFS只在索引节点和subtrees上使用锁，而与HDFS的全局锁相比，HopsFS在工作负载中比HDFS高出37倍，其中20%的文件系统操作是文件创建操作。



7.2.1 Hotspots

大数据应用程序在单个目录中创建数以百万计的文件并不少见[51,43]。如第4.2.1节所讨论的，如果文件系统操作在所有数据库碎片之间不均匀地分布，就会影响HopsFS的性能。在这个实验中,所有的文件系统操作路径共享一个共同的祖先,即/ shared-dir ....所有文件系统操作操作使用共同祖先操作的文件和目录，并使用上一节7.2中描述的工作负载生成文件系统操作。这个工作负载受到保存/shared-dir的数据库碎片的性能的限制。尽管目前的HopsFS版本还没有提供一个解决热点问题的解决方案，但是当前的解决方案比HDFS高3倍，见图6。我们没有看到在出现热点时HDFS的性能有任何影响。

7.3 Metadata (Namespace) Scalability

在HDFS中，由于整个名称空间元数据必须适合于单个JVM的堆，所以数据结构高度优化，以减少内存占用[60]。在HDFS中，一个带有两个模块的文件被复制三种方式，需要448 + L字节的元数据，其中L表示文件名长度。如果文件名有10个字符长，那么一个1 GB的JVM堆可以存储230万个文件。实际上，JVM堆大小必须要大得多，才能容纳次级元数据，数千个并发RPC请求，每个数据块的大小都是几十兆字节，以及其他临时对象。

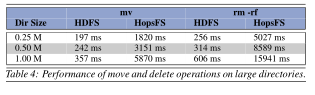
将元数据迁移到数据库会导致存储索引、主/外键和填充的内存容量的增加。在HopsFS中，如果元数据被复制两次，上面描述的相同文件需要1552字节。对于高可用性部署一个HDFS的活动和备用名称节点,您将需要两倍的内存,因此,HopsFS需要比HDFS≈1.5倍的内存来存储高可用的元数据。表3显示了HDFS和HopsFS的元数据可伸缩性。

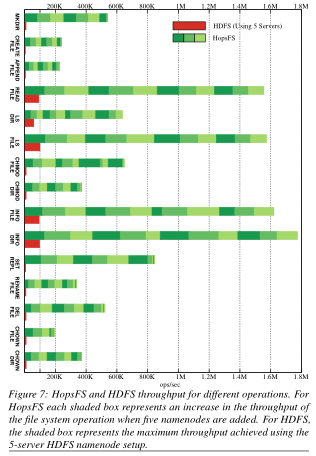
NDB支持最多48个数据节点，它允许在每个NDB 数据节点上有512gb RAM的集群中扩展到24 TB的数据。HopsFS可以存储170亿个文件使用24 TB的元数据,这是(≈37次)高于HDFS。

7.4 FS Operations’Raw Throughput

在这个实验中，对于每个文件系统操作，基准实用程序会使用相同的文件系统操作来淹没名称节点。这个测试特别有助于确定特定文件系统操作的最大吞吐量和可伸缩性。在现实部署，名称节点经常收到相同的文件系统操作类型的泛滥，例如，读取大量数据的大作业将生成大量的请求来读取文件和列出目录。

图7显示了比较不同文件系统操作的吞吐量的结果。对于每个操作，HopsFS的结果显示为堆叠矩形的条形图。当添加5个新的名称节点时，每个矩形代表吞吐量的增加。对于所有的文件系统操作，HopsFS优于HDFS，而且在最常见的文件系统操作中，它的性能明显优于HDFS。





7.4.1 Subtree Operations

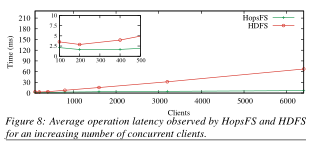
在表4中，我们显示了在包含不同数量文件的目录下移动和删除子树操作的延迟，范围从1 / 4到100万。在这个实验中，测试在HopsFS和HDFS集群上执行了50%负载的Spotify工作负载(在图6中所观察的最大吞吐量的50%)。

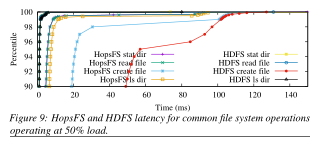
在HopsFS中，通过网络读取大量数据，并在许多小事务批中执行操作。移动操作的执行时间不会快速增长，因为它不会更新子树的所有内部节点或叶子。HDFS优于HopsFS，因为所有数据在内存中都很容易获得。然而，由于这种操作在典型的工业工作负载下的频率较低(见表1)，我们认为这是一个可以接受的折衷方案，以提高HopsFS中公共文件系统操作的性能。

7.5 Operational Latency

在未加载的HDFS 名称节点上，单个文件系统操作的延迟总是低于HopsFS，因为所有元数据在HDFS 名称节点的主内存中都很容易获得，而在HopsFS中的名称节点则是远程的。图8显示了在运行Spotify工作负载时并发客户端观察到的平均文件系统操作延迟。对于这样的工作负载，HopsFS比HDFS的操作延迟更低，因为在HDFS文件系统操作中，更新名称空间会阻塞所有其他文件系统操作。大型的HDFS部署，可能有成千上万的客户端[61]，随着文件系统操作在名称节点[55]中的RPC调用队列中等待，客户机的端到端延迟会增加。相比之下，HopsFS可以处理更多的并发客户，同时保持低的操作延迟。

图9显示了在非重载集群中不同文件系统操作的第99个百分位数延迟。在这个实验中,我们跑HopsFS和Spotify HDFS 50%负荷下工作负载(50%的最大通过——把观察到图6) ，在HopsFS中，百分之九十九常见的文件系统操作,例如touch文件，读取文件,ls dir和stat dir分别为100.8毫秒,8.6毫秒, 11.4和8.5毫秒。在一个类似的HDFS实验中，运行在50%负载下，对于99%的操作touch文件、读取文件、ls dir和stat dir，分别为101.8、1.5、0.9和1.5 ms。



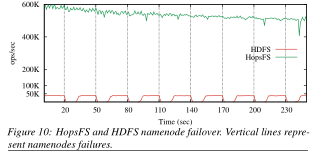


7.6 Failure Handling

现在我们讨论了当名称节点、NDB 数据节点和日志节点失败时，HDFS和HopsFS的性能如何受到影响。

7.6.1 Namenodes failure

图10显示了当名称节点失败于Spotify工作负载的50%时，如何影响文件系统元数据服务的性能。名称节点失败是通过杀死和重新启动名称节点上所有文件系统过程来模拟的。对于HDFS,活跃名称节点周期性地在HopsFS中被杀死，名称节点周期性地以循环方式被杀死。在图10中，垂直线表示名称节点失败。在HDFS中，待机名称节点在检测活动名称节点失败时接管。我们的实验中，在HDFS的故障转移过程，我们观察到了8到10秒的停机时间。在此期间，不能执行任何文件系统元数据操作。我们的故障转移测试对HDFS很有利，因为NNs在实验中存储的元数据量很小。在Spotify，每天有140千兆字节的元数据和4万个工作机会，故障转移至少需要5分钟，通常需要15分钟。虽然我们不确定原因，但可能是因为备用名称节点所扮演的额外的检查点角色。此外，启动名称节点需要几十分钟的时间来构建从磁盘名称空间映像到名称空间的内存表示，并应用优秀的重做日志。相反，在HopsFS中，当一个名称节点失败的客户端在系统上重新执行失败的文件系统操作。在这些实验中，文件系统的客户数量是固定的，在实验过程中没有添加新的客户端。对于HopsFS来说，随着越来越多的名称节点重新启动，吞吐量逐渐下降。这是因为在名称节点失败后，客户端切换到剩余的名称节点。在实验中，HopsFS使用了粘性的名称节点选择策略，由于在实验中没有新客户端启动，因此重新启动的名称节点并没有收到像其他名称节点那样的文件系统操作请求。



7.6.2 Failure of NDB Datanodes or Journal Nodes

对于拥有N个日志节点的HDFS集群,HDFS可以容忍失效的日志节点数最多为。在我们的测试中，有三个日志节点，HDFS只能容忍一个日志节点失败。增加quorum到5的大小使HDFS能够容忍两个日志节点的失败。我们已经测试了带有3、5和7个日志节点的HDFS，并且当日志节点失败只要仲裁配额没有丢失时，HDFS 名称节点的性能不会受到影响。当更多的日志节点失败同时仲裁配额丢失时，HDFS 名称节点关闭。

HopsFS能够容忍的NDB节点失败的数量取决于NDB数据节点的数量和复制度。NDB的设计目的是提供99.999%的可用性[37]。使用默认的NDB复制度2，一个4节点的NDB集群可以容忍最多2个NDB 数据节点故障，而一个12节点NDB集群可以容忍多达6个NDB数据节点在不相交的副本组中失败。我们已经测试了HopsFS在2、4、8和12节点NDB上的集群，当NDB 数据节点在每个节点组中至少有一个剩余的NDB 数据节点存在时，HopsFS的性能不会受到影响。如果在一个节点组中所有的NDB数据都失败，那么HopsFS 名称节点关闭。

针对两阶段提交协议的常见问题是，事务协调器或参与者的阻塞和失败将导致系统阻塞。NDB内部实现了事务协调器故障转移协议，该协议将失败的协调器的事务移交给不同的协调器(默认的1500 ms心跳超时为3个错过的心跳提供了6秒的上限)。事务参与者的失败是通过非常低的非活动超时来确定的(默认是1200毫秒，也用于我们的实验和生产)。在事务参与者失败的情况下，失败的事务将被名称节点自动重试，并将由该复制组中幸存的数据节点s处理。

7.7 Block Report Performance

在HDFS中，每个数据节点都会周期性地向名称节点发送一个块报告，其中包含存储在数据节点上的所有块的IDs。块报告有两个用途:(1)当名称节点重新启动时，它们帮助重建块位置映射，因为HDFS不保存这个信息，(2)它们作为系统中可用块的事实依据。我们在HopsFS中重新实现了HDFS块报告解决方案。尽管该解决方案是完全功能性的，但它并不具有很高的吞吐量，因为通过名称节点数据库从数据库中读取大量元数据来处理一个块报告。

在一个同样设置的实验中，150个数据节点同时提交了包含100,000个块的块报告。有了30个名称节点,HopsFS每秒钟处理30个块报告，而HDFS每秒处理60个块报告。然而，在HopsFS中，完整的块报告不需要像在HDFS中那样频繁，因为我们在数据库中保留了块位置映射。即使没有进一步的优化，有512mb的块大小，以及每六个小时发送块报告的数据节点,HopsFS也可以在一个艾字节集群中处理块报告。

8 Related Work

InversionFS[39]和Windows Future Storage(WinFS)[74]是存储在关系数据库中的元数据的第一个单片文件系统。Gunawi[20]显示了一些文件系统操作，比如fsck，在使用关系数据库实现时可以提高效率。

最近，HBase(如HBase[16,9]，Cassandra[29]，CalvinDB[69]等高性能分布式数据库支持了文件系统中新的分布式元数据管理体系结构的开发，如CalvinFS[68]、CassandraFS[8]和长颈鹿[18]。所有这些文件系统都存储了非规范化的元数据，也就是说，它们以影响子树操作的每个索引节点存储完整的文件路径。长颈鹿只支持同一目录下的文件移动操作。CalvinFS依赖于CalvinDB来执行大型事务。CalvinDB在两个阶段进行了大量的交易。在第一个阶段中，锁定集被识别出来，在第二阶段，所有的锁都被捕获，并且执行操作，前提是锁集没有改变。然而，CalvinFS并没有实验表明这是一种可行的技术，可以在有数百万文件的目录上执行操作。产品级别的联机事务处理系统在事务的数量上有一个上限，在这个事务中，上界值远低于数千万。

IndexFS[52]和ShardFS[75]是针对具有大量小文件的元数据工作负载进行优化的文件系统。IndexFS和ShardFS是中间件文件系统，也就是说，它们构建在现有的分布式文件系统上，如HDFS[61]、光泽[66]、PanFS[73]和PVFS[31]。在IndexFS和ShardFS中，元数据服务器处理元数据以及存储在本地LevelDB[32]实例中的小文件的用户数据，并将大文件的管理委托给底层分布式文件系统。对于持久性，LevelDB的SSTables存储在底层分布式文件系统中。IndexFS在客户端缓存索引节点信息，而ShardFS在元数据服务器上缓存它。不支持包含底层分布式文件系统和IndexFS / ShardFS元数据服务器的原子文件系统操作。例如，原子删除大型文件，其元数据存储在IndexFS / ShardFS元数据服务器中，而文件数据存储在底层分布式文件系统中不受支持。IndexFS[52]使用缓存机制来提高热目录/文件的性能，而HopsFS目前只有负载平衡用户可配置的顶级目录。我们正在研究更动态的HopsFS方法。

PVFS2[31]，OrangeFS[76]，Farsite[14]，Lustre[66]，Vesta[11]，InterMezzo[46]，zFS[53]，和RAMA[35] 碎片索引节点在多个元数据服务器通过随机分区(1)或(2)分区基于散列文件标识符或散列全部/部分文件路径。这种分区方案通常与客户端元数据的缓存相结合，这会导致大型子树操作的缓存失效风暴。Ceph动态地划分了文件系统树，其中热点目录在多个元数据服务器上被散列[71,72]。

最后，我们的体系结构支持可插入的新sql存储引擎。MemSQL和SAP Hana都是候选对象，因为它们支持高吞吐量的跨分区事务、应用程序定义的分区和分区修剪查询[34]。当它序列化交叉分区事务时，VoltDB不是一个候选对象[70]。

9 External Metadata Implications

管理员经常使用自己的工具来分析HDFS名称空间。HopsFS允许在元数据上进行在线广告分析。使用NDB后端，HopsFS元数据可以选择性地、异步地复制到备份集群或MySQL从服务器上，从而支持复杂的分析，而不会影响活动集群的性能。HopsFS元数据也很容易导出到外部系统，并且很容易安全地扩展元数据。也就是说，可以创建额外的表，其中包含对关联的索引节点的外键，从而确保扩展元数据的完整性。使用此方法，我们已经向HopsFS添加了新特性，包括索引节点的扩展属性和删除编码。此外，在类似的想法[28]之后，我们开发了一个最终一致的复制协议，它复制(扩展)HopsFS元数据，以用于自由文本搜索的弹性搜索[15]。这使我们能够搜索整个名称空间，并有次秒延迟。我们认为，在商品数据基础上，分布式元数据是一项重要的新技术，它可以成为建立在分布式文件系统之上的元数据应用程序的可靠的基础事实来源。

10 Summary

在本文中，我们介绍了HopsFS，即在我们的知识中，第一个产品级分布式分层文件系统，它将元数据存储在一个外部的NewSQL数据库中。HopsFS是一个开源的、高度可用的文件系统，它通过添加新的名称节点和数据库节点来扩展容量和吞吐量。HopsFS可以存储比HDFS多37倍的元数据，而对于来自Spotify的工作负载，HopsFS可以处理16倍于HDFS的吞吐量。HopsFS对大量并发客户机的平均延迟时间也较低，在故障转移期间也没有停机时间。我们的架构支持可插拔的数据库存储引擎，可以使用其他的新sql数据库。最后，HopsFS使元数据变得友好，为用户和应用程序打开它，以新的和创造性的方式扩展和分析。

References

[1] C. L. Abad. Big Data Storage Workload Characterization, Modeling and Synthetic Generation. PhD thesis, University of Illinois at Urbana-Champaign, 2014.

[2] M. Abd-El-Malek, W. V. Courtright, II, C. Cranor, G. R. Ganger, J. Hendricks, A. J. Klosterman, M. Mesnier, M. Prasad, B. Salmon, R. R. Sambasivan, S. Sinnamohideen, J. D. Strunk, E. Thereska, M. Wachs, and J. J. Wylie. Ursa Minor: Versatile Cluster-based Storage. In Proceedings of the 4th Conference on USENIX Conference on File and Storage Technologies - Volume 4, FAST’05, pages 5–5, Berkeley, CA, USA, 2005. USENIX Association.

[3] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. Farsite: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. SIGOPS Oper. Syst. Rev., 36(SI):1–14, Dec. 2002.

[4] R. Agrawal, M. J. Carey, and M. Livny. Concurrency control performance modeling: Alternatives and implications. ACM Trans. Database Syst., 12(4):609–654, Nov. 1987. [5] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinländer, M. J. Sax, S. Schelter, M. Höger, K. Tzoumas, and D. Warneke. The Stratosphere Platform for Big Data Analytics. The VLDB Journal, 23(6):939–964, Dec. 2014.

[6] J. Baker, C. Bond, J. C. Corbett, J. Furman, A. Khorlin, J. Larson, J.-M. Leon, Y. Li, A. Lloyd, and V. Yushprakh. Megastore: Pro- viding scalable, highly available storage for interactive services. In Proceedings of the Conference on Innovative Data system Re- search (CIDR), pages 223–234, 2011.

[7] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O’Neil, and P. O’Neil. A critique of ansi sql isolation levels. SIGMOD Rec., 24(2):1–10, May 1995.

[8] Cassandra File System Design. http://www.datastax.com/ dev/blog/cassandra-

file-system-design . [Online; accessed 1-January-2016].

[9] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. ACM Trans. Comput. Syst., 26(2), 2008.

[10] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google’s Globally-distributedDatabase. InProceedingsofthe10thUSENIX Conference on Operating Systems Design and Implementation, OSDI’12, pages 251–264, Berkeley, CA, USA, 2012. USENIX Association.

[11] P. Corbett, D. Feitelson, J.-P. Prost, and S. Baylor. Parallel access to files in the Vesta filesystem. In Supercomputing ’93. Proceed- ings, pages 472–481, Nov 1993. [12] E. Corporation. HADOOP IN THE LIFE SCIENCES: An Introduction. <https://www.emc.com/collateral/software/white-papers/h10574-wp-isilon-hadoop->

in-lifesci.pdf, 2012. [Online; accessed 30-Aug-2015].

[13] Dean, Jeffrey and Ghemawat, Sanjay. Mapreduce: Simplified data processing on large clusters. Commun. ACM, 51(1):107–113, Jan. 2008.

[14] J. R. Douceur and J. Howell. Distributed Directory Service in the Farsite File System. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI ’06, pages 321–334, Berkeley, CA, USA, 2006. USENIX Association.

[15] Elasticsearch. https://www.elastic.co/products/elasticsearch. [Online; accessed 1-January-2016].

[16] L. George. HBase: The Definitive Guide. Definitive Guide Series. O’Reilly Media, Incorporated, 2011.

[17] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. SIGOPS Oper. Syst. Rev., 37(5):29–43, Oct. 2003.

[18]GiraffaFS.https://github.com/giraffafs/giraffa .[Online; accessed 1-January-2016].

[19] J. Gray, R. Lorie, G. Putzolu, and I. Traiger. Granularity of Locks and Degrees of Consistency in a Shared Database. In IFIP Working Conference on Modelling in Data Base Management Systems, pages 365–394. IFIP, 1976.

[20] H. S. Gunawi, A. Rajimwale, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. SQCK: A Declarative File System Checker. In Proc. of OSDI’08, pages 131–146. USENIX Association, 2008.

[21] Hammer-Bench: Distributed Metadata Benchmark to HDFS. https://github.com/smkniazi/hammer-bench . [On- line; accessed 1-January-2016]. [22] Hadoop JIRA: Add thread which detects JVM pauses. https: //issues.apache.org/jira/browse/HADOOP-9618 . [Online; accessed 1-January-2016]. [23] M. P. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. ACM Trans. Program. Lang. Syst., 12(3):463–492, July 1990.

[24] Hadoop Open Platform-as-a-Service (Hops) is a new distribution of Apache Hadoop with scalable, highly available, customizable metadata. https://github.com/smkniazi/ hammer-bench. [Online; accessed 1-January-2016]. [25] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference, USENIXATC’10, pages 11–11, 2010.

[26] F. Hupfeld, T. Cortes, B. Kolbeck, J. Stender, E. Focht, M. Hess, J. Malo, J. Marti, and E. Cesario. The XtreemFS architecture—a case for object-based file systems in Grids. Concurrency and computation: Practice and experience, 20(17):2049–2060, 2008.

[27] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In Proceedings of the 2Nd ACM SIGOPS/EuroSys European Con- ference on Computer Systems 2007, EuroSys ’07, pages 59–72, New York, NY, USA, 2007. ACM.

[28] C. Johnson, K. Keeton, C. B. Morrey, C. A. N. Soules, A. Veitch, S. Bacon, O. Batuner, M. Condotta, H. Coutinho, P. J. Doyle, R. Eichelberger, H. Kiehl, G. Magalhaes, J. McEvoy, P. Nagarajan, P. Osborne, J. Souza, A. Sparkes, M. Spitzer, S. Tandel, L. Thomas, and S. Zangaro. From research to practice: Experiences engineering a production metadata database for a scale out file system. In Proceedings of the 12th USENIX Conference on File and Stor- age Technologies, FAST’14, pages 191–198, Berkeley, CA, USA, 2014. USENIX Association.

[29] A.LakshmanandP.Malik. Cassandra: A Decentralized Structured Storage System. SIGOPS Oper. Syst. Rev., 44(2):35–40, Apr. 2010.

[30] B. W. Lampson. Hints for computer system design. SIGOPS Oper. Syst. Rev., 17(5):33–48, Oct. 1983.

[31] R. Latham, N. Miller, R. B. Ross, and P. H. Carns. A Next- Generation Parallel File System for Linux Clusters. LinuxWorld Magazine, January 2004.

[32] LevelDB. http://leveldb.org/ . [Online; accessed 1- January-2016].

[33] E. Levy and A. Silberschatz. Distributed file systems: Concepts and examples. ACM Computing Surveys, 22:321–374, 1990. [34] MemSQL, The World’s Fastest Database, In Memory Database, ColumnStore Database. http://www.memsql.com/ . [Online; accessed 30-June-2015].

[35] E. L. Miller and R. Katz. RAMA: An Easy-To-Use, High- Performance Parallel File System. Parallel Computing, 23(4):419– 446, July 1997.

[36] J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. Rosenthal, and F. D. Smith. Andrew: A Distributed Personal Computing Environment. Commun. ACM, 29(3):184–201, Mar. 1986.

[37] MySQL Cluster: High Availability. https: //www.mysql.com/products/cluster/ availability.html. [Online; accessed 23-May-2016].

[38] MySQL Cluster CGE. http://www.mysql.com/ products/cluster/. [Online; accessed 30-June-2015].

[39] M. A. Olson and M. A. The Design and Implementation of the Inversion File System, 1993.

[40] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch. The sprite network operating system. IEEE Computer, 21:23–36, 1988.

[41] M. Ovsiannikov, S. Rus, D. Reeves, P. Sutter, S. Rao, and J. Kelly. The Quantcast File System. Proc. VLDB Endow., 6(11):1092– 1101, Aug. 2013.

[42] F. Özcan, N. Tatbul, D. J. Abadi, M. Kornacker, C. Mohan, K. Ra- masamy, and J. Wiener. Are We Experiencing a Big Data Bubble? In Proceedings of the 2014 ACM SIGMOD International Confer- ence on Management of Data, pages 1407–1408, 2014. [43] S. V. Patil, G. A. Gibson, S. Lang, and M. Polte. GIGA+: Scalable Directories for Shared File Systems. In Proceedings of the 2Nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing ’07, PDSW ’07, pages 26–29, New York, NY, USA, 2007. ACM.

[44] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz. NFS Version 3 - Design and Implementation. In In Proceedings of the Summer USENIX Conference, pages 137–152, 1994.

[45] D. Peng and F. Dabek. Large-scale incremental processing us- ing distributed transactions and notifications. In Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, 2010.

[46] Peter Braam Braam and Michael Callahan. The InterMezzo File System. In In Proceedings of the 3rd of the Perl Conference, O’Reilly Open Source Convention, Monterey, CA, USA, 1999.

[47] A. J. Peters and L. Janyst. Exabyte Scale Storage at CERN. Jour- nal of Physics: Conference Series, 331(5):052015, 2011.

[48] I. Polato, R. Ré, A. Goldman, and F. Kon. A comprehensive view of Hadoop research – A systematic literature review. Journal of Network and Computer Applications, 46:1–25, 2014.

[49] G. Popek and B. J. Walker. The LOCUS distributed system architecture. MIT Press, 1985.

[50] K. W. Preslan, A. Barry, J. Brassow, R. Cattelan, A. Manthei, E. Nygaard, S. V. Oort, D. Teigland, M. Tilstra, and et al. Implementing Journaling in a Linux Shared Disk File System, 2000.

[51] K. Ren, Y. Kwon, M. Balazinska, and B. Howe. Hadoop’s ado- lescence: an analysis of hadoop usage in scientific workloads. Proceedings of the VLDB Endowment, 6(10):853–864, 2013.

[52] K. Ren, Q. Zheng, S. Patil, and G. Gibson. IndexFS: Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC ’14, pages 237–248, Piscataway, NJ, USA, 2014. IEEE Press.

[53] O. Rodeh and A. Teperman. zFS - a scalable distributed file system using object disks. In Mass Storage Systems and Technologies, 2003. (MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on, pages 207–218, April 2003.

[54] M. Ronström and J. Oreland. Recovery Principles of MySQL Cluster 5.1. In Proc. of VLDB’05, pages 1108–1115. VLDB Endowment, 2005.

[55] RPC Congestion Control with FairCallQueue. https: //issues.apache.org/ jira/

browse/HADOOP-9640 .[Online; accessed 1-January-2016].

[56] G. B. Salman Niazi, Mahmoud Ismail and J. Dowling. Leader Election using NewSQL Systems. In Proc. of DAIS 2015, pages 158 –172. Springer, 2015.

[57] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, David, and C. Steere. Coda: A Highly available File System for a Distributed Workstation Environment. IEEE Trans- actions on Computers, 39:447–459, 1990.

[58] Schmuck, Frank and Haskin, Roger. GPFS: A Shared-Disk File System for Large Computing Clusters. In Proceedings of the 1st USENIX Conference on File and Storage Technologies, FAST ’02, Berkeley, CA, USA, 2002. USENIX Association.

[59] M. Seltzer and N. Murphy. Hierarchical File Systems Are Dead. In Proceedings of the 12th Conference on Hot Topics in Operat-ing Systems, HotOS’09, pages 1–1, Berkeley, CA, USA, 2009. USENIX Association.

[60] K. Shvachko. Name-node memory size estimates and optimization proposal. https://issues.apache.org/jira/browse/HADOOP-1687 , August 2007. [Online; accessed 11-Nov-2014].

[61] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), MSST ’10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society. [62] K. V. Shvachko. HDFS Scalability: The Limits to Growth. login: The Magazine of USENIX, 35(2):6–16, Apr. 2010.

[63] HOPS, Software-As-A-Service from SICS’S new datacen- ter. https://www.swedishict.se/hops-software-as-a-service-from-sicss-new-datacenter . [On- line; accessed 23-May-2016].

[64] M.Srivas, P.Ravindra, U.Saradhi, A.Pande, C.Sanapala, L.Renu, S. Kavacheri, A. Hadke, and V. Vellanki. Map-Reduce Ready Distributed File System, 2011. US Patent App. 13/162,439.

[65] The Curse of the Singletons! The Vertical Scalability of Hadoop NameNode. http://hadoopblog.blogspot.se/2010/ 04/curse-of-singletons-vertical.html . [Online; accessed 30-Aug-2015].

[66] The Lustre Storage Architecture http://wiki.lustre.org/ manual/

LustreManual20\_HTML/ UnderstandingLustre.html. [Online; accessed 30-Aug- 2015].

[67] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: A Scalable Distributed File System. In Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles, SOSP ’97, pages 224–237, New York, NY, USA, 1997. ACM.

[68] A. Thomson and D. J. Abadi. CalvinFS: Consistent WAN Repli- cation and Scalable Metadata Management for Distributed File Systems. In 13th USENIX Conference on File and Storage Technologies (FAST 15), pages 1–14, Santa Clara, CA, Feb. 2015. USENIX Association.

[69] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi. Calvin: Fast Distributed Transactions for Partitioned Database Systems. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, SIGMOD ’12, pages 1–12, New York, NY, USA, 2012. ACM.

[70] VoltDB Documentation. http://docs.voltdb.com/ReleaseNotes/. [Online; accessed 30-June-2015].

[71] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A Scalable, High-performance Distributed File System. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI ’06, pages 307–320, Berkeley, CA, USA, 2006. USENIX Association.

[72] S. A. Weil, K. T. Pollack, S. A. Brandt, and E. L. Miller. Dynamic Metadata Management for Petabyte-Scale File Systems. In Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, SC ’04, pages 4–, Washington, DC, USA, 2004. IEEE Computer Society.

[73] Welch, Brent and Unangst, Marc and Abbasi, Zainul and Gibson, Garth and Mueller, Brian and Small, Jason and Zelenka, Jim and Zhou, Bin. Scalable Performance of the Panasas Parallel File System. In Proceedings of the 6th USENIX Conference on File and Storage Technologies, FAST’08, Berkeley, CA, USA, 2008. USENIX Association. [74] WinFS: Windows Future Storage. https:// en.wikipedia.org/wiki/WinFS . [Online; accessed 30-June-2015].

[75] L. Xiao, K. Ren, Q. Zheng, and G. A. Gibson. ShardFS vs. IndexFS: Replication vs. Caching Strategies for Distributed Meta- data Management in Cloud Storage Systems. In Proceedings of the Sixth ACM Symposium on Cloud Computing, SoCC ’15, pages 236–249, New York, NY, USA, 2015. ACM.

[76] S. Yang, W. B. Ligon III, and E. C. Quarles. Scalable distributed directory implementation on orange file system. Proc. IEEE Intl. Wrkshp. Storage Network Architecture and Parallel I/Os (SNAPI), 2011.

[77] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud’10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[78] M. Zait and B. Dageville. Method and mechanism for database partitioning, Aug. 16 2005. US Patent 6,931,390.