**分布式文件系统相关问题的研究与思考**

专业：计算机技术

学号：2017282110213

姓名：王安康

## 1 概述

本文的主要内容包括对来自FAST 17的两篇研究分布式系统存在问题与解决方案的论文的介绍，以及自己在通读论文的过程中对论文研究内容的思考。

分布式文件系统是许多大规模数据并行处理系统（如MapReduce，Dryad，Flink和Spark）的重要基础设施组件。到本十年底，存储多个EB数据的数据中心将不会少见。对于目前的大型分布式文件系统来说，仍存在不少的瓶颈与问题。比如，元数据管理服务是其可扩展性的瓶颈，分布式系统利用冗余来增强其安全性和可靠性是否真的可依赖等问题。

## 2 HopsFS：使用NEWsql管理分布式元数据

#### 2.1 简介

对于大型分布式文件系统，元数据管理服务是可扩展性的瓶颈。许多现有的分布式文件系统将其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））上，这两者的可伸缩性都极其有限。众所周知的例子包括GFS，HDFS等。另外有一些分布式系统通过静态分区命名空间并将分片存储在不同的主机（如NFS，AFS，MapR，Locus，Coda，Sprite和XtreemFS）上来扩展其元数据，但静态分区命名空间会对跨越不同分区的文件系统操作产生消极的影响，特别是移动操作，而且由于管理员必须将元数据服务器映射到随时间变化的命名空间碎片上，会使文件系统的管理复杂化。

NewSQL 是对各种新的具有高可扩展性和高性能数据库的简称，这类数据库不仅具有对海量数据的存储管理能力，还保持了传统[数据库](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93)对[ACID](https://baike.baidu.com/item/ACID)和[SQL](https://baike.baidu.com/item/SQL" \t "_blank)等的支持。这类数据库通常工作在一个分布式集群的节点上，其中每个节点拥有一个数据子集，SQL查询在主节点分成查询片段并发送给相应数据所在的数据节点上执行。

NewSQL数据库在性能和可伸缩性方面的改进重新开放了使用商用型数据库对分布式文件系统中的分布式元数据进行管理的可能性。在文章中，主要介绍了HopsFS系统如何使用NewSQL数据库构建高吞吐量和低操作延迟的分布式文件系统。HopsFS是Hadoop分布式文件系统（HDFS）的新一代发行版，它分离了文件系统元数据存储和管理服务，用建立在NewSQL数据库上的分布式元数据服务取代了HDFS的单节点内存上的元数据服务。通过对元数据瓶颈的消除，HopsFS比HDFS实现了更大、更高的吞吐量集群。其中，元数据容量相比HDFS已经至少增加了37倍，在基于Spotify工作负载跟踪的实验中，显示HopsFS吞吐量是Apache HDFS的16到37倍。而且对于许多并发客户端，HopsFS也具有较低的延迟，并且在故障切换期间不会出现停机的状况。最后，分布式元数据存储在商用型数据库中，就可以安全地扩展并轻松导出到外部系统，以便进行在线分析和自由文本搜索。

#### 2.2 论文主要方法介绍

HopsFS是HDFS v2.0.4的一个分支，与HDFS不同，HopsFS通过分离元数据存储和操作服务来提供向外扩展的元数据层。HopsFS在分布式事务中封装了文件系统操作，使用数据库为每个名称节点分配一个唯一的ID，并且当名称节点重新启动时，名称节点的ID改变。

##### 2.2.1 自动分片

HopsFS的自动分片方案通过操作低成本数据库访问查询来读取元数据，使用MySQL Cluster中的多分区事务和行级锁来提供元数据操作的可串行性。分布式文件系统的元数据通常包含有关节点，块，副本和映射（目录到文件，文件到块和块到副本）的信息。在分发元数据时，需要使用应用程序定义的分片方案来对元数据进行分片，并且需要一致的协议来确保跨越分片操作的元数据的完整性。而分布式元数据结构设计的关键在于命名空间分片方案的选择。

通过论文内容可以看出，数据库分片参与的全表扫描或索引扫描的成本远远高于只有单个数据库参与的分区索引扫描。使用HopsFS元数据设计和元数据分区，只需使用低成本数据库操作（即主键操作，批处理主键操作和分区索引扫描）即可实现通用文件系统操作。例如，读取和目录列表操作仅使用主键查找和分区索引扫描来实现。全表扫描则尽可能被避免，因为它们涉及到所有的数据库碎片。

###### 2.2.2 一致性和高速缓存

HopsFS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，以及使用高速缓存来减少分布式事务的延迟。HopsFS中的事务性元数据操作分为两类：一是在单个文件、目录或块上运行的Inode操作（例如，创建/读取文件，创建目录和块状态更改操作）；二是子树操作一个未知数量的Inode。

HopsFS封装了NDB中事务的Inode操作。由NDB直接提供的事务隔离不足以使用单个全局锁，来序列化所有HDFS操作的一致性语义。为此，论文中使用行级分层锁定来对冲突的Inode操作进行序列化。也就是说，只要这些操作不在相同的Inode上发生冲突，这些操作就会并行执行。如果数据以层次结构排列在树中，并且所有数据操作都从上到下遍历层次结构，则会隐式地锁定树/子树的根节点和子节点。与所有操作一样，首先读取Inode，然后读取相关的元数据。对于某些操作，如创建文件/目录和列表操作，父目录也被锁定。

其次，从数据库读取的所有数据都存储在每个事务缓存（快照）中，该缓存保留更新的缓存记录到数据库的传播，直到事务结束。 由于元数据通常在同一事务中被多次读取和更新，所以缓存可以节省许多往返数据库的时间。 元数据的行级锁定确保了缓存的一致性，即没有其他事务可以更新元数据。 而且，当事务完成释放锁时，缓存将被清除。

###### 2.2.3 处理大型操作

目前存在的问题是：对于包含数百个Inode的大型目录的递归操作太大，无法适应单个事务，也就是说，在现有的事务处理系统中无法支持在事务中锁定数百万行。这些操作包括移动，删除，更改所有者以及更改权限操作。移动操作会更改所有后代索引节点的绝对路径，而删除操作会删除所有后代索引节点。同样，更改目录的权限或所有者可能会使在较低子树上执行的操作无效。

论文中提出的解决方案是制订了一个协议，在批量事务中递增执行子树操作。论文提出的子树操作协议不使用行级数据库锁，而是使用应用程序级分布式锁机制来标记和隔离子树。通过序列化子树操作，确保子树中的所有进行中的Inode和子树操作在执行新请求的子树操作之前完成。通过执行以下固定顺序的操作来实现这个序列化：（1）在操作完成之前，没有新的操作访问子树;（2）子树在子树操作开始之前停顿;（3）如果发生故障，没有孤立的节点或者元数据不一致的情况出现。。

例如，在删除操作的情况下，使用事务以后以遍历方式递增地删除子树。如果操作中途发生错误，那么未被删除的Inode将保持连接到命名空间树， HopsFS将重新提交文件系统操作到另一个命名节点，以删除子树的其余部分。其他子树操作则不会导致任何的不一致，因为元数据修改的实际操作是在第三阶段使用单个事务完成的，该事务只更新子树的根节点而内部节点保持不变。

###### 2.3 HopsFS评估

在论文的后半部分基于真实文件系统负载的受控条件构建了大量的实验，比较测试了在大型集群中，HDFS和HopsFS中命名节点的性能和可伸缩性。由于本文篇幅所限，仅举一例来说明HopsFS相比HDFS的优势。

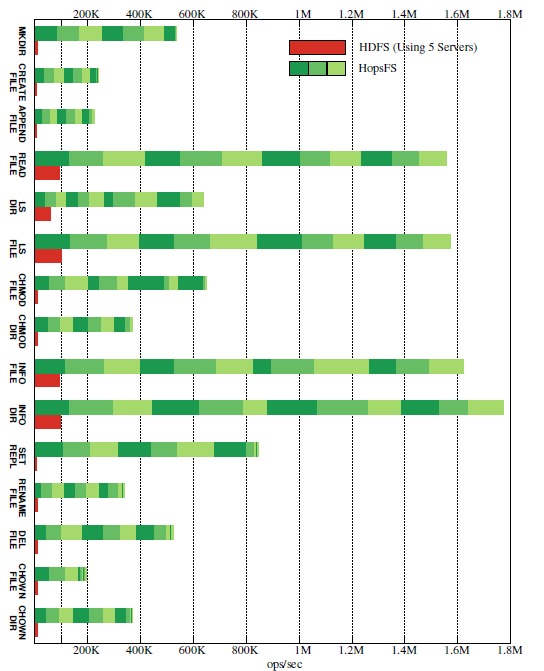


图 1

图1展示了HopsFS和HDFS用于不同的操作时的吞吐量。 对于HopsFS，当添加五个命名节点时，每个阴影框代表文件系统操作吞吐量的增加。对于HDFS，阴影框表示使用五个服务器时命名节点设置实现的最大吞吐量。从图中可以明显而直观的看出，论文实现的HopsFS比HDFS在吞吐量上有着数个数量级的领先。

###### 2.4 总结与思考

该论文主要介绍了第一个生产级的分布式分层文件系统——HopsFS，它把它的元数据存储在一个外部的NewSQL数据库中。 同时，HopsFS是一个开放源代码，高度可用的文件系统，它通过添加新的命名节点和数据库节点，扩展了容量和吞吐量。对于大量的并发客户端，HopsFS也具有较低的平均延迟，并且在故障切换期间不会停机。 并且，该架构支持可插拔的数据库存储引擎，也就是说可以使用其他的NewSQL数据库。 最后，HopsFS为元数据创建了友好的接口，开放给用户和应用程序，以便进行扩展和分析。

分布式文件系统是未来通用文件系统的趋势，在最近几年随着大数据、机器学习等概念的火热，对文件系统提出了更大、更快的要求。传统的分布式文件系统通过一系列管理服务已经实现了存储更多内容的要求，但是由于元数据存储的瓶颈，其在操作大量数据的速率上还处于一个极度受限的状态（受限于内存）。而这篇论文提出的HopsFS，独辟蹊径，将元数据存储在NewSQL数据库中，去除了内存的限制，实现了高吞吐和低延时。而且与此同时，HopsFS源自于的HDFS，在商业上已经较为成熟，利用的媒介NewSQL，也是最近几年流行的兼容传统数据库的分布式数据库，因此HopsFS可以较快的实现传统项目的移植，有利于商业性的传播。

新事物的发展总是两面的。分布式文件系统可以通过实现HopsFS等系统大幅提升其吞吐率，提升操作文件的效率。但是我们也不能忽视分布式文件系统在底层可能存在的问题。在下一章中，将主要介绍另一篇论文，探讨冗余的容错性。

## 3 冗余不意味着容错：分布式存储系统对单一错误的容错性分析

###### 3.1 简介

这篇论文是我们小组所讲主题的主要参考论文，该论文通过对八个流行的分布式存储系统进行分析，发现了许多与文件系统容错有关的漏洞。并通过研究发现，现代的分布式系统并不总是能够利用分布式的冗余特性来从文件系统的故障中恢复：单个文件系统的故障可能导致灾难性后果，比如数据丢失，数据损坏和系统不可用性等。该论文的结论对于下一代容错分布式和云存储系统的设计具有重要意义。

###### 3.2 论文主要内容介绍

###### 3.2.1 提出问题

现代分布式存储系统以复制的方式存储数据（即冗余）以提高其可靠性。在大多数情况下，冗余可以屏蔽故障，例如系统崩溃，电源故障以及磁盘或网络故障。不幸的是，部分文件系统故障会导致其中某些数据块变得不可读或发生读取和写入错误，从而导致数据被默默地破坏。

现代分布式存储系统对于文件系统故障如何响应是至关重要的，并且对基于云的服务有非常的影响。但是现在的情况是，对于现代分布式存储系统如何对文件系统故障作出反应，知之甚少。一个普遍认为的预期是，更高层次（即跨越副本）的冗余能够从本地文件系统故障中重新获取资源。例如，分布式存储系统的一个节点中不可访问的数据块在理想情况下不会导致用户可见的数据丢失，因为相同的数据会被重复地存储在许多节点上。基于这个期望，在该论文中探讨了以下问题：在存在本地文件系统故障的情况下，现代分布式存储系统如何运行？他们能够使用冗余从单个文件系统故障中恢复吗？

###### 3.2.2 构建模型并测试

为研究现代分布式存储系统如何应对本地文件系统故障，在该论文中构建了一个名为CORDS的故障注入模型。该模型有两个重要特征。首先，该模型在单个节点中一次只向单个文件系统块注入单一故障。其次，该模型只将错误注入到应用程序级别的磁盘结构，而不影响文件系统的元数据。

该模型通过插入一些数据项将正在研究的系统初始化为已知状态，并确保它们被安全地复制并保存在磁盘上。然后将应用程序配置为通过指定其挂载点作为应用程序的数据目录在errfs上运行。因此，应用程序执行的所有读写操作都会通过errfs进行，从而可以注入故障。通过多次运行应用程序工作负载，每次通过errfs为单个文件系统块注入单一故障。对于每个注入的故障，CORDS都会自动观察系统的行为。Errfs注入的错误类型包括数据读错误、数据写错误和数据损坏三种类型。

在构建完模型后，该论文在八个目前被广泛使用的分布式系统研究测试了CORDS故障注入模型：Redis，ZooKeeper，Cas-Sandra，Kafka，RethinkDB，MongoDB，LogCabin和CockroachDB。

###### 3.2.1 结论

在理想的情况下，在分布式系统中注入CORDS，应该有如下表现：首先提交的数据不应该丢失，查询也不应该默默返回损坏的数据，在查询失败后重新查询应该能够查询成功，而且分布式集群应该随时可用于读写。

但是，经过论文对八个流行的系统进行测试发现：在大多数现代分布式存储系统中，单个文件系统故障会导致灾难性后果。尽管在分布式存储中普遍存在校验和，冗余和其他弹性方法来保障其可靠性，但单个不合时宜的文件系统故障可能会导致数据丢失、损坏、不可用等问题，并且在某些情况下将损坏扩散到其他完整的副本。

同样由于篇幅所限，本文就不一一列出该论文对于八个文件系统的分别测试结果，只对论文得出结论进行介绍和分析。

首先，论文中发现这些系统经常会检测不到本地系统故障，并静默地将损坏的数据返回给用户，导致丢失数据，或将损坏的数据传播到完整的副本导致整个系统变得不可用，或者在查询中返回意外的错误。例如，在Kafka和RethinkDB中，一个节点上的损坏会导致用户可见的数据丢失。日志初始化期间的单个写入错误可能导致ZooKeeper中的写入不可用。同样，Redis和Cassandra中的一个节点中的损坏数据可以传播到其他完整的副本。

其次，故障和崩溃经常会纠结在一起。系统往往会由于错误引起的故障而导致系统崩溃。而当系统尝试从崩溃的状态重启恢复时，极有可能产生另外的故障，或者恢复时调用的也是一个已经有错误的子系统，错上加错，并最终导致糟糕的结果。

最后，单一的错误会给整个集群带来灾难性的影响。虽然分布式存储系统可以跨多个节点复制数据和功能，但单个节点上的单个文件系统故障可能会导致集群范围内的有害影响。

在上述情况下，大多数分布式存储系统的冗余并不足以使系统健康的恢复，甚至当错误影响到全局的时候，所有的冗余都不能作为恢复的来源。

###### 3.3 总结与思考

该论文通过研究测试发现，在现代分布式存储系统中，对文件系统故障的容忍是不成熟的。这些系统副本的冗余并不足以使其从本地文件系统故障中恢复，因此单个本地文件系统的故障就会导致数据丢失，损坏和不可用性等问题。

通过对该论文的工作发现，随着分布式存储系统成为存储关键用户数据的主要选择，仔细测试所有类型的故障是非常重要的。论文中也提到了，他们已经联系了七个系统的开发人员，其中五个已经意识到他们发现的问题。但是虽然可以通过修复解决一些问题，但许多其他问题容错性需要进行底层的设计更改。因此在未来可见的一段时间内，商业性的分布式存储系统都会存在该论文检测出的安全隐患。在购买分布式系统服务时，自己要做好心理预期，不能完全依赖其尚未完全可靠的冗余特性。

在阅读该论文的过程中，我也一直在思考一个问题，既然冗余不能完全意味着容错,那冗余之外提高分布式存储系统容错性的方法有哪些？其他的方法能否解决这种本地文件系统产生的故障？其实最好的方法是即时更新所有的硬件存储设备，这样就可以减少大部分已知的漏洞漏洞和故障，但是分布式文件系统的大数据存储特性意味着需要大量的设备，因此这在商业上是行不通的。

希望以后可以利用该论文公开的测试框架和测试的错误结果深入研究，以提高下一代云存储系统的弹性，建设下一代可容错易恢复的分布式系统。

## 4 综合思考

通过对上述两篇论文的阅读可以发现，分布式系统不管在商业应用领域还是在科学研究领域都是发展非常迅速的。一方面，之前存在的瓶颈由于人力物力的投入在不断的解决，另一方面，随着研究的不断深入，也逐渐发现了分布式系统潜在的一些问题。这使整个领域的发展逐渐形成一个良性的循环。

另外，在阅读FAST 17这两篇论文的过程中，一直在思考这两篇论文能发在一个A类会议的理由是什么。两篇论文主要内容都没有算法或者底层系统的创新，其中第一篇论文是在已经成熟的HDFS基础上进行了改进，解决了元数据存储瓶颈的问题，而第二篇论文连解决问题都没有，只是提出一个问题并对该问题进行了完整的测试。

但是通过逐渐深入的阅读可以发现，两篇论文首先写作功底非常扎实，其次实验部分完整而且可靠，另外他们的亮点也非常突出。HopsFS的提出解决的是当下一个迫切的问题，不管是在商业还是科研上，并且可以迅速的得到应用。第二篇论文对八个大型分布式系统的完整测试看似只有几页，但做起来难度相当大，人力、物力、实验室的影响力缺一不可，最终才能得到这样一份令人信服的完整测试结果。

希望以后自己能以这样的论文为借鉴，在科研道路上争取走的更远一些。

## 5 参考文献

* [1] Niazi S, Ismail M, Grohsschmiedt S, et al. HopsFS: Scaling Hierarchical File System Metadata Using NewSQL Databases[J]. 2017.
* [2] Ganesan A, Alagappan R, Arpaci-Dusseau A C, et al. Redundancy Does Not Imply Fault Tolerance: Analysis of Distributed Storage Reactions to Single Errors and Corruptions[C]//FAST. 2017: 149-166.