# 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**

Recent improvements in both the performance and scalability of shared-nothing, transactional, in-memory NewSQL databases have reopened the research question of whether distributed metadata for hierarchical file sys- tems can be managed using commodity databases. In this paper, we introduce HopsFS, a next generation distribu- tion of the Hadoop Distributed File System (HDFS) that replaces HDFS’ single node in-memory metadata service, with a distributed metadata service built on a NewSQL database. By removing the metadata bottleneck, HopsFS enables an order of magnitude larger and higher through- put clusters compared to HDFS. Metadata capacity has been increased to at least 37 times HDFS’ capacity, and in experiments based on a workload trace from Spotify, we show that HopsFS supports 16 to 37 times the through- put of Apache HDFS. HopsFS also has lower latency for many concurrent clients, and no downtime during failover. Finally, as metadata is now stored in a commodity database, it can be safely extended and easily exported to external systems for online analysis and free-text search.

对于无共享，事务型的，内存不足的内存NewSQL数据库的性能和可扩展性的最近的改进重新开启了关于分层文件系统的分布式元数据是否可以使用商品数据库进行管理的研究问题。在本文中，我们介绍HopsFS，它是Hadoop分布式文件系统（HDFS）的下一代发行版，用于将HDFS的单节点内存元数据服务替换为基于NewSQL数据库的分布式元数据服务。通过去除元数据的瓶颈，与HDFS相比，HopsFS启用了一个数量级更大和更高的通用集群。元数据容量已经提高到HDFS容量的至少37倍，在基于Spotify（全球最大的正版[流媒体](https://baike.baidu.com/item/%E6%B5%81%E5%AA%92%E4%BD%93)音乐服务平台？）的工作量跟踪的实验中，我们显示HopsFS支持（是）Apache HDFS的16到37倍。 HopsFS对许多并发客户端的延迟更低，在故障转移期间也不会有停机时间。最后，随着元数据现在存储在商品数据库中，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。

## Introduction

Distributed file systems are an important infrastructure component of many large scale data-parallel processing systems, such as MapReduce [13], Dryad [27], Flink [5] and Spark [77]. By the end of this decade, data centers storing multiple exabytes of data will not be uncommon [12, 47]. For large distributed hierarchical file systems, the metadata management service is the scalability bottleneck [62]. Many existing distributed file systems store their metadata on either a single node or a shared-disk file systems, such as storage-area network (SAN), both of which have limited scalability. Well known ex- amples include GFS [17], HDFS [61], QFS [41], Far-site [3], Ursa Minor [2], GPFS [58], Frangipani [67], GlobalFS [50], and Panasas [73]. Other systems scale out their metadata by statically sharding the namespace and storing the shards on different hosts, such as NFS [44], AFS [36], MapR [64], Locus [49], Coda [57], Sprite [40] and XtreemFS [26]. However, statically sharding the namespace negatively affects file system operations that cross different shards, in particular move operation. Also, it complicates the management of the file system, as ad- ministrators have to map metadata servers to namespace shards that change in size over time.

分布式文件系统是许多大规模数据并行处理系统的重要基础设施组件，如MapReduce [13]，Dryad [27]，Flink [5]和Spark [77]。到本十年底，到本十年底，存储多个数据的数据中心并不少见[12,47]。对于大型分布式分层文件系统，元数据管理服务是可扩展性瓶颈[62]。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上，如存储区域网络（SAN），这两者都具有有限的可扩展性。众所周知的例子包括GFS [17]，HDFS [61]，QFS [41]，Far-site [3]，Ursa Minor [2]，GPFS [58]，Frangipani [67]，GlobalFS [50] Panasas [73]。其他系统横向扩展他们的元数据通过静态分隔命名空间并将分片存储在不同的主机上，例如NFS [44]，AFS [36]，MapR [64]，Locus [49]，Coda [57]，Sprite [40]和XtreemFS [26]。但是，静态分配命名空间会对跨越不同分片（碎片？）的文件系统操作产生负面影响，特别是移动（move指文件移动的操作嘛？）操作。此外，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，所以使文件系统的管理变得复杂化。

Recent improvements in both the performance and scalability of shared-nothing, transactional, in-memory NewSQL [42] databases have reopened the possibility of storing distributed file system metadata in a commodity database. To date, the conventional wisdom has been that it is too expensive (in terms of throughput and latency) to store hierarchical file system metadata fully normalized in a distributed database [59, 33].

In this paper we show how to build a high throughput and low operational latency distributed file system using a NewSQL database. We present HopsFS, a new distribution of the Hadoop Distributed File System (HDFS) [61], which decouples file system metadata storage and management services. HopsFS stores all metadata normalized in a highly available, in-memory, distributed, relational database called Network Database (NDB), a NewSQL storage engine for MySQL Cluster [38, 54]. HopsFS provides redundant stateless servers (namenodes) that in parallel, read and update metadata stored in the database.

最新的无共享内存NewSQL [42]数据库的性能和可扩展性的改进已经重新开放了将分布式文件系统元数据存储在商品数据库中的可能性。到目前为止，传统的观点是，在分布式数据库中将分层文件系统元数据完全归一化（标准化？），这是太贵了（在吞吐量和延迟方面）[59,33]。

在本文中，我们将展示如何使用NewSQL数据库构建高吞吐量和低运行时延分布式文件系统。我们提供HopsFS，这是Hadoop分布式文件系统（HDFS）的新发行版[61]，它解耦文件系统元数据存储和管理服务。 HopsFS将所有元数据归一化在高可用性，内存中，分布式的关系数据库中，称为网络数据库（NDB），一种MySQL群集的NewSQL存储引擎[38,54]。 HopsFS提供冗余的无状态服务器（Namenodes）并行读取和更新存储在数据库中的元数据。

HopsFS encapsulates file system operations in distributed transactions. To improve the performance of file system operations, we leverage both classical database techniques such as batching (bulk operations) and write-ahead caches within transactions, as well as distribution aware techniques commonly found in NewSQL databases. These distribution aware NewSQL techniques include application defined partitioning (we partition the namespace such that the metadata for all immediate descendants of a directory (child files/directories) reside on the same database shard for efficient directory listing), and distribution aware transactions (we start a transaction on the data- base shard that stores all/most of the metadata required for the file system operation), and partition pruned index scans (scan operations are localized to a single database shard [78]). We also introduce an inode hints cache for faster resolution of file paths. Cache hits when resolving a path of depth N can reduce the number of database round trips from N to 1.

**！** HopsFS将文件系统操作封装在分布式事务中。为了提高文件系统操作的性能，我们利用经典数据库技术，如批处理（批量操作）和事务中的预先提前缓存，以及NewSQL数据库中使用的分发感知技术。这些分发感知的NewSQL技术包括应用程序定义的分区？（我们对命名空间进行分区，使得目录的所有即时后代的元数据（子文件/目录）驻留在同一数据库分片上，以实现高效的目录列表）和分发感知事务（我们在这样的一个数据库分片上开始事务，这个数据库分片上存储了文件系统操作所需的所有/大部分元数据），以及分区修剪的索引扫描（扫描操作已本地化到单个数据库分片[78]）。我们还引入了一个inode提示缓存来更快地分辨文件路径。解析深度N的路径时的缓存命中可以将数据库往返次数从N减少到1。

However, some file system operations on large directory subtrees (such as move, and delete) may be too large to fit in a single database transaction. For example, deleting a folder containing millions of files cannot be performed using a single database transaction due to the limitations imposed by the database management system on the number of operations that can be included in a single transaction. For these subtree operations, we introduce a novel protocol that uses an application level distributed locking mechanism to isolate large subtrees to perform file system operations. After isolating the subtrees large file system operations are broken down into smaller transactions that execute in parallel for performance. The subtree operations protocol ensures that the consistency of the namespace is not violated if the namenode executing the operation fails.

但是，大型目录子树（如移动和删除）上的一些文件系统操作可能太大，无法适应单个数据库事务。 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量所施加的限制，删除包含数百万个文件的文件夹无法使用单个数据库事务执行。 对于这些子树操作，我们引入了一种新颖的协议，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。 分离子系统之后，大文件系统操作将分解为并行执行的较小事务。子树操作协议确保如果执行操作的namenode失败，则不会违反命名空间的一致性。

HopsFS is a drop-in replacement for HDFS. HopsFS has been running in production since April 2016, providing Hadoop-as-a-Service for researchers at a data center in Luleå, Sweden [63]. In experiments, using a real-world workload generated by Hadoop/Spark applications from Spotify, we show that HopsFS delivers 16 times higher throughput than HDFS, and HopsFS has no downtime during failover. For a more write-intensive workload, HopsFS delivers 37 times the throughput of HDFS. To the best of our knowledge HopsFS is the first open-source dis- tributed file system that stores fully normalized metadata in a distributed relational database.

HopsFS是HDFS的替代品。 HopsFS自2016年4月开始投入生产，为瑞典Luleå数据中心的研究人员提供Hadoop-as-a-Service [63]。 在实验中，使用来自Spotify的Hadoop / Spark应用程序生成的真实工作负载，我们显示HopsFS的吞吐量比HDFS高出16倍，HopsFS在故障转移期间不会有停机时间。 对于更加写密集的工作负载，HopsFS提供了HDFS吞吐量的37倍。 据我们所知，HopsFS是第一个在分布式关系数据库中存储完整归一化元数据的开源分布式文件系统。

1. **Background**

This section describes Hadoop Distributed File System (HDFS) and MySQL Cluster Network Database (NDB) storage engine.

## Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) [61] is an open source implementation of the Google File System [17]. HDFS’ metadata is stored on the heap of single Java process called the Active NameNode (ANN), see Figure 1. The files are split into small (typically 128 MB) blocks that are by default triple replicated across the datanodes. For high availability of the metadata management service, the Active namenode logs changes to the metadata to journal servers using quorum based replication. The metadata change log is replicated asynchronously to a Standby NameNode (SbNN), which also performs checkpointing functionality. In HDFS, the ZooKeeper coordination service [25] enables both agreement on which machine is running the active namenode (preventing a split-brain scenario) as well as coordinating failover from the active to the standby namenode.

Hadoop分布式文件系统（HDFS）[61]是Google文件系统的开源实现[17]。 HDFS'元数据存储在名为Active NameNode（ANN）的单个Java进程的堆中，请参见图1.这些文件被拆分为小的块（每块大小128MB），在datanodes上有3个副本。 为了元数据管理服务的高可用性，Active namenode日志将使用基于副本的仲裁将元数据更改为日志服务器。 元数据更改日志异步复制到备用namenode（SbNN），该备用namenode也执行检查点功能。 在HDFS中，ZooKeeper协调服务[25]使得能够在哪台机器上运行主namenode（防止裂脑情景）以及协调主动到备用节点的故障切换。

The namenode serves requests from potentially thousands of datanodes and clients, and keeps the metadata strongly consistent by executing the file system operations atomically. The namenode implements atomic operations using a single global lock on the entire file system metadata, providing single-writer, multiple-readers concurrency semantics. Some large file system operations are not atomic, as they would hold the global lock for too long, starving clients. For example, deleting large directories is performed in batches, with inodes first being deleted, then the blocks are deleted in later phases. Moreover, as writing namespace changes to the quorum of journal nodes can take long time, the global file system lock is released before the operation is logged to prevent other clients from starving. Concurrent clients can acquire the file system lock before the previous operations are logged, preventing starvation, at the cost of inconsistent file system operations during namenode failover. For example, when the active namenode fails all the changes that are not logged to the journal nodes will be lost.

namenode提供潜在的数千个datanodes和客户端的请求，并通过原子执行文件系统操作来保持元数据的一致性。 namenode使用单个全局锁对整个文件系统元数据实现原子操作，提供单写入器，多读取器并发语义。一些大文件系统操作不是原子操作，因为它们会把全局锁保存得太久，饿死客户端。例如，删除大目录是分批执行的，首先删除索引节点，然后在稍后阶段删除这些块。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，因此在记录操作之前释放全局文件系统锁以防止其他客户端挨饿。并发客户端可以在先前的操作被记录之前（即写到日志里之前）获取文件系统锁，从而避免当主namenode结点宕机，避免客户端饥饿和文件系统操作不一致的代价。例如，当主namenode结点宕机时，所有未记录到日志节点的更改都将丢失。

The datanodes are connected to both active and standby namenodes. All the datanodes periodically generate a block report containing information about its own stored blocks. The namenode processes the block report to vali- date the consistency of the namenode’s blocks map with the blocks actually stored at the datanode.

In HDFS the amount of metadata is quite low relative to file data. There is approximately 1 gigabyte of metadata for every petabyte of file system data [62]. Spotify’s HDFS cluster has 1600+ nodes, storing 60 petabytes of data, but its metadata fits in 140 gigabytes Java Virtual Machine (JVM) heap. The extra heap space is taken by temporary objects, RPC request queues and secondary metadata required for the maintenance of the file system. However, current trends are towards even larger HDFS clusters (Facebook has HDFS clusters with more than 100 petabytes of data [48]), but current JVM garbage collection technology does not permit very large heap sizes, as the application pauses caused by the JVM garbage collector affects the operations of HDFS [22]. As such, JVM garbage collection technology and the monolithic architecture of the HDFS namenode are now the scalability bottlenecks for Hadoop [62]. Another limitation with this architecture is that data structures are optimized to reduce their memory footprint with the result that metadata is difficult to modify or export to external systems.

datanodes连接到主和备namenode节点。所有datanodes周期性地生成包含有关其自己的存储块的信息的块报告。 namenode处理块报告以验证namenode的块映射与实际存储在数据库中的块的一致性。

在HDFS中，相对于文件数据，元数据量相当低。对于每一PB文件系统数据，大约有1 GB的元数据[62]。 Spotify的HDFS集群拥有1600多个节点，存储60 PB的数据，但其元数据适合于140 GB的Java虚拟机（JVM）堆。临时对象，文件系统维护所需的RPC请求队列和辅助元数据占用额外的堆空间。然而，目前的趋势是朝着更大的HDFS集群（Facebook拥有超过100 PB的数据的HDFS集群[48]），但目前的JVM垃圾回收技术不允许非常大的堆大小，因为JVM垃圾收集器引起的应用程序暂停会影响HDFS的操作[22]。因此，JVM垃圾收集技术和HDFS nmenode的单片架构现在是Hadoop的可扩展性瓶颈[62]。该架构的另一个限制是数据结构被优化以减少其内存占用，导致元数据难以修改或导出到外部系统。

## Network Database (NDB)

MySQL Cluster is a shared-nothing, replicated, in- memory, auto-sharding, consistent, NewSQL relational database [38]. Network DataBase (NDB) is the storage engine for MySQL Cluster. NDB supports both datanode-level and cluster-level failure recovery. The datanode-level failure recovery is performed using transaction redo and undo logs. NDB datanodes also asynchronously snapshot their state to disk to bound the size of logs and to improve datanode recovery time. Cluster-level recovery is supported using a global checkpointing protocol that increments a global epoch-ID, by default every 2 seconds. On cluster-level recovery, datanodes recover all transactions to the latest epoch-ID.

MySQL集群是一个无共享，复制，内存，自动分片，一致的NewSQL关系数据库[38]。网络数据库（NDB）是MySQL集群的存储引擎。 NDB支持数据库级和集群级故障恢复。使用事务重做(redo)和撤消日志(undo)来执行数据库级别的故障恢复。 NDB数据库还将其状态异步映射到磁盘以限制日志大小并提高数据恢复时间。使用全局检查点协议支持集群级别恢复，默认情况下每2秒增加一个全局epoch-ID。在集群级恢复时，datanodes将所有事务恢复到最新的epoch-ID。

NDB horizontally partitions the tables among storage nodes called NDB datanodes. NDB also supports application defined partitioning (ADP) for the tables. Transaction coordinators are located at all NDB datanodes, enabling high performance transactions between data shards, that is, multi-partition transactions. Distribution aware transactions (DAT) are possible by providing a hint, based on the application defined partitioning scheme, to start a transaction on the NDB datanode containing the data read/updated by the transaction. In particular, single row read operations and partition pruned index scans (scan operations in which a single data shard participates) benefit from distribution aware transactions as they can read all their data locally [78]. Incorrect hints result in additional network traffic being incurred but otherwise correct system operation.

NDB水平划分存储结点的表，叫做NDB datanodes。 NDB还支持表的应用程序定义分区（ADP）。事务协调器位于所有NDB数据节点上，从而实现数据分片之间的高性能交易，即多分区事务。通过提供基于应用程序定义的分区方案的提示，可以在包含由事务读取/更新的数据的NDB数据结点（datanodes）上启动事务，从而实现分发感知事务（DAT）。特别地，单行读取操作和分区修剪的索引扫描（单个数据分片参与的扫描操作）受益于分布式感知事务，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通信，但是否则系统正常运行。

* + 1. **NDB Data Replication and Failure Handling**

NDB datanodes are organized into node groups, where the data replication factor, R, determines the number of datanodes in a node group. Given a cluster size N, there are N/R node groups. NDB partitions tables (hash partitioning by default) into a fixed set of partitions distributed across the node groups. New node groups can be added online, and existing data is automatically rebalanced to the new node group. A partition is a fragment of data stored and replicated by a node group. Each datanode stores a copy (replica) of the partition assigned to its node group. In NDB, the default replication degree is two, which means that each node group can tolerate one NDB datanode failure as the other NDB datanode in the node group contains a full copy of the data. So, a twelve node NDB cluster has six node groups can tolerate six NDB datanode failures as long as there is one surviving NDB datanode in each of the node groups. To tolerate multiple failures within a node group, the replication degree can be increased at the cost of lower throughput.

NDB数据节点被组织成节点组，其中数据复制因子R确定节点组中的数据节点的数量。给定簇N，有N / R个节点组。 NDB将表（默认为哈希分区）分区为分布在节点组中的一组固定的分区。可以线上添加新节点组，并将现有数据自动重新平衡到新节点组。分区是由节点组存储和复制的数据的片段。每个datanode存储分配给其节点组的分区的副本（副本）。在NDB中，默认的副本为2，这意味着每个节点组可以容忍一个NDB datanode故障，因为节点组中的其他NDB datanode包含数据的完整副本。因此，只要在每个节点组中存在一个幸存的NDB datanode，一个十二节点NDB集群有六个节点组可以容忍六个NDB数据节点故障。为了容忍节点组中的多个故障，可以以较低吞吐量为代价来增加副本的数量。

**2.2.2 Transaction Isolation**

NDB only supports read-committed transaction isolation, which guarantees that any data read is committed at the moment it is read. The read-committed isolation level does not allow dirty reads but phantom and fuzzy (non- repeatable) reads can happen in a transaction [7]. However, NDB supports row level locks, such as, exclusive (write) locks, shared (read) locks, and read-committed locks that can be used to isolate conflicting transactions.

NDB只支持读取提交的事务隔离，这样可以保证在读取时提交任何数据。 读取提交的隔离级别不允许脏读取，但是在事务中可能会发生幻影和模糊（不可重复）读取[7]。 但是，NDB支持行级锁，例如独占（写）锁，共享（读取）锁和可用于隔离冲突事务的读取提交的锁。

**3 HopsFS Overview**

HopsFS is a fork of HDFS v2.0.4. Unlike HDFS, HopsFS provides a scale-out metadata layer by decoupling the metadata storage and manipulation services. HopsFS supports multiple stateless namenodes, written in Java, to handle clients’ requests and process the metadata stored in an external distributed database, see Figure 1. Each namenode has a Data Access Layer (DAL) driver that, similar to JDBC, encapsulates all database operations allowing HopsFS to store the metadata in a variety of NewSQL databases. The internal management (house-keeping) operations, such as datanode failure handling, must be coordinated amongst the namenodes. HopsFS solves this problem by electing a leader namenode that is responsible for the housekeeping. HopsFS uses the database as shared memory to implement a leader election and membership management service. The leader election protocol assigns a unique ID to each namenode, and the ID of the namenode changes when the namenode restarts. The leader election protocol defines an alive namenode as one that can write to the database in bounded time, details for which can be found in [56].

HopsFS是HDFS v2.0.4的分支。与HDFS不同，HopsFS通过解除元数据存储和操作服务来提供横向扩展的元数据层。 HopsFS支持使用Java编写的多个无状态Namenode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namenode都有一个数据访问层（DAL）驱动程序，与JDBC类似，封装了所有数据库允许HopsFS将元数据存储在各种NewSQL数据库中的操作。内部管理（保管）操作，如数据库故障处理，必须在网络中进行协调。 HopsFS通过选择负责家政管理的领导者来解决这个问题。 HopsFS使用数据库作为共享内存来实现领导选举和会员管理服务。领导选举协议为每个namenode分配一个唯一的ID，当namenode重新启动时，namenode的ID会更改。领导选举协议将一个活着的namenode定义为可以在有限时间内写入数据库的细节，详细信息可以在[56]中找到。