一种文件路径和属性分离的元数据组织方法

李春花 杨勇 周可

（华中科技大学武汉光电国家实验室，武汉 430074）

摘 要

分布式元数据管理方案对大数据存储的性能和扩展性有着关键的作用。针对基于子树划分和基于哈希划分的元数据管理方案扩展代价大、重命名产生迁移等问题，提出一种文件路径与属性信息分离的元数据组织方案。将元数据组织成目录索引和属性信息两个部分，通过构建目录索引将元数据属性以目录或小于目录为单位保存到桶（Bucket）内，再将桶根据元数据服务集群负载情况进行分配。目录索引的构建有效地解决了重命名造成的元数据迁移和集群的扩展问题，并且桶结构增强了元数据的访问局部性，提高了管理性能。测试结果表明，本文方法能获得较高的管理性能，特别适合高并发的情况；具有良好的可扩展性和较好的访问局部性，而且可以不限制目录的大小；避免了重命名元数据造成的不必要的迁移。与集中式元数据管理方法对比，本文方法采用单一元数据服务器时，元数据的创建、查询、遍历等操作性能都有了数倍的提升。

引 言

元数据管理的性能和扩展性对于分布式存储系统有着至关重要的影响。研究显示，分布式文件系统中有50%-80%的文件操作关于元数据。随着大数据时代的到来，存储系统需要管理的文件数量不断增加，使得集中式元数据管理系统的负担越来越重，逐渐成为大数据存储的瓶颈。为此，有必要提出分布式元数据管理方案，利用MDS集群管理元数据，提高元数据管理的性能和扩展性。

当前有两种分布式元数据划分方法：目录子树划分和基于哈希的划分。目录子树划分又分成两种：1）静态子树划分，将全局目录树划分成多个子目录树并分配到指定的元数据服务器上；但其缺点也很明显，随着系统中文件和目录的增加，不同的目录子树中包含的文件数量会不平衡，导致集群负载不均衡，需要人工介入，重新划分负载较重的服务节点上的目录子树。2）动态子树划分，根据MDS的负载情况，动态地将目录划分到多个节点中，避免单个节点负载过重；但其缺点包括集群扩展代价大、目录结构变化发生数据迁移、延迟时间长等。基于哈希的划分，支持快速的访问操作，其结构易于实现负载均衡；其主要不足在于目录操作性能低，集群变化需要执行re-hash，并且rename操作会导致元数据迁移。

大数据环境中，元数据管理面临着诸多挑战，需要提高元数据管理的性能和可扩展性，具体而言又包括以下几点：1）有较高的性能；2）有较好的访问局部性；3）可以避免rename操作造成的元数据迁移；4）MDS集群具有平滑的扩展性；5）具备均匀分布大目录下元数据的能力。

针对以上问题，本文提出一种文件路径和属性信息分离的元数据管理方案，实现在大数据环境下元数据高效管理、平滑扩展，同时避免rename操作产生的数据迁移。

整体架构

如图1所示，系统的整体架构包含目录索引模块（Index Service，IS）、元数据服务器集群（Metadata Server，MDS 集群）、定位服务模块（Location Service，LS）和客户端（Client）四个部分。

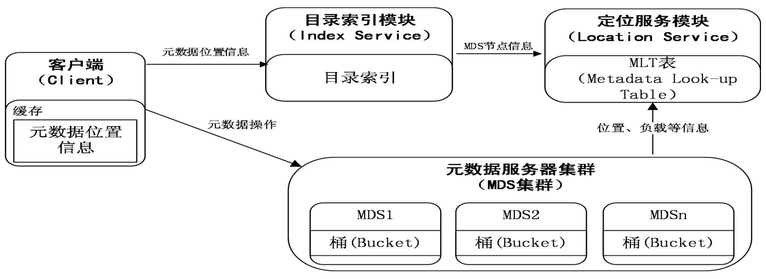

图1 元数据组织方案整体架构

Client是整个方法的入口，通过Client可以获得方法提供的服务，执行相关的元数据操作，Client通过IS得到元数据位置信息，请求MDS集群执行元数据操作。IS（Index Service）维护了全局目录树的名字空间索引信息，索引包含目录到MDS的映射关系，IS请求LS（Location Service）获得元数据的位置。LS维护MLT表，表中包含MDS集群的位置和负载信息，LS根据MLT表计算出集群中负载最低的MDS。MDS集群以桶为单位存储元数据，通过桶号和文件名称能在桶内获取元数据。

IS包含全局目录的结构，负责维护目录索引、计算元数据位置信息。其中的目录索引为键值对结构，如图2所示。键是一个字符串，由父目录全局ID和目录名组合而成，值包含了两个元素：目录的全局ID和分布编码映射关系。目录下元数据根据分布编码映射关系分布到MDS上，并根据映射关系中的分布编码保存到桶内。

键	值	
<父目录全局ID，目录名>	全局ID	映射关系：<分布编码1，MDS编号> <分布编码2，MDS编号>

图2 目录索引结构

元数据属性信息保存在桶内，桶由MDS集群保存，如图3所示。桶号为分布编码。桶内包含多个元数据，采用键值对结构，键为文件或目录名称，值为元数据。桶内的元数据有较好访问局部性，桶内保存的元数据属于同一个目录。

桶号	分布编码	
	文件名	文件元数据
	目录名	目录元数据
	…	…

图3 桶结构

LS负责监控MDS集群的负载、为目录索引模块提供MDS编号和计算MDS地址信息。模块记录和维护的信息保存在改进的MLT（Metadata Look-up Table）表中，结构如图4所示，包含了MDS的编号、MDS的IP地址、端口号、MDS平均负载和处理次数。

编号	MDS IP地址	MDS端口	MDS平均负载	处理次数

图4 MLT表结构

Client使用的元数据位置信息结构，如图5所示。根据位置信息客户端可以找到元数据所在的桶，包含了MDS的IP和端口以及分布编码。Client根据缓存的元数据位置信息，缩短访问流程。缓存的位置信息，如图6所示。采用了键值对结构，键为目录路径，值为元数据位置信息。

MDS IP地址	MDS端口	分布编码

图5 元数据位置信息结构

键	值		
父目录路径	MDS IP地址	MDS端口	分布编码

图6 缓存位置信息结构

元数据划分和分布策略

分布式元数据管理的关键在于如何划分元数据的名字空间，不同的划分策略带来的效果也不相同。但是，名字空间的划分策略不应过于复杂，避免给系统带来较多的开销，而且划分后的元数据应有较好的访问局部性。

在本文方法中，划分策略的基本思想是：将同一目录下的元数据划分到一个或多个桶（Bucket）内。桶内的元数据属于同一目录，拥有较好的访问局部性。桶小于或等于目录的大小，目录大小超过桶的大小才会发生划分，而且划分过程简单不会带来较多的开销。

在分布式元数据管理中，一些元数据操作，如重命名，可能造成多个MDS上元数据的迁移。为了避免这种情况的发生，引入了全局目录索引结构，如图2所示，用于建立目录到MDS的映射关系。因此，本文方法的名字空间通过三级映射机制实现。第一级采用目录路径计算到全局目录索引中的节点，得到目录的分布编码映射关系。第二级根据分布编码映射关系中的分布编码在MLT表中查找，得到保存桶的MDS。第三级根据文件名称在桶内查找到元数据。

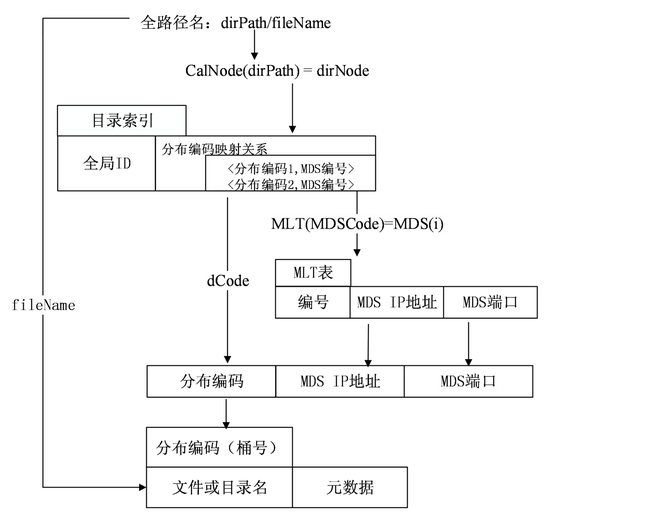

图7 元数据划分策略

从文件的路径到保存文件的元数据位置的算法，如图7所示。对于目录或文件，首先通过其路径名（不包含目录或文件名称，即其父目录的全路径）计算出全局目录索引的节点：$CalNode(dirPath) = dirNode$，得到目录或文件的父目录索引节点；通过索引节点中的分布编码dCode和MDS编号MDSCode，计算 $MLT(MDSCode) = MDS(i)$，得到MDS的位置信息；然后根据dCode和定位到桶，得到元数据位置信息。

MDS集群扩展策略

分布式元数据管理能够对MDS集群扩展，并通过扩展MDS集群提升管理元数据的数量和性能。扩展MDS集群时，应具有平滑的扩展性，不应该影响其他MDS的服务。

本文方法中采用监控机制发现新加入的MDS节点。MDS加入集群时都会向负载监控的Zookeeper集群注册信息，之后定期向监控集群发送信息。LS模块实时监控注册信息并更新MLT表，完成MDS集群的扩展，新的元数据分布请求到来将会考虑新加入的MDS。如图8所示，MDSn节点加入集群时，向Zookeeper集群注册自身信息，包括IP、端口、负载和处理次数（注册时未有任何请求，处理次数为0），LS通过Watcher机制发现MDSn的注册事件，并根据注册信息更新MLT表。

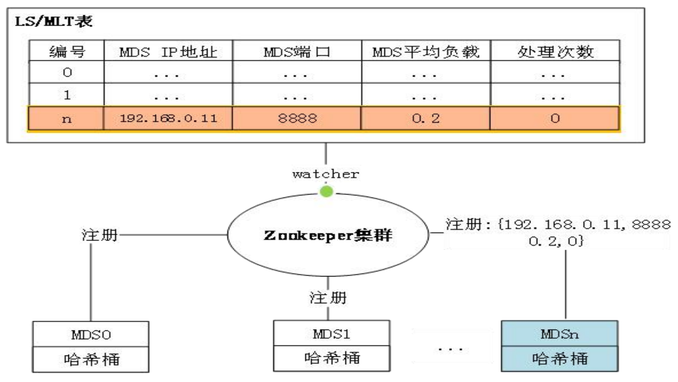

图8 MDS集群扩展策略

本文方法引入的全局目录索引，持久化保存了目录与MDS的映射关系。新加入的MDS节点不会影响之前MLT表中MDS注册的信息，因此不会影响其他MDS的服务，也不会造成元数据迁移。

元数据位置缓存策略

本文采用事件通知和缓存过期相结合的方法，解决缓存数据的同步。同步方法原理如图9所示。

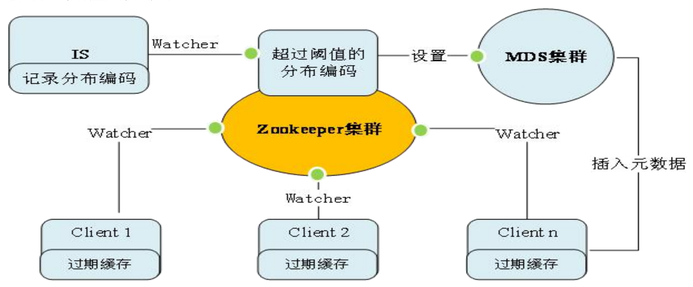

图9 缓存位置同步方法

Client和IS都会在Zookeeper上Watcher一个节点，当MDS集群中出现桶中数据超过阈值时，MDS会向Watcher的节点上添加此桶号。Zookeeper的Watcher机制将此变化通知所有Client和IS，Client执行过期相应的本地缓存操作，IS记录相应的分布编码。Client因本地无缓存而向IS请求目录的位置信息，此时IS会为目录添加新的分布编码映射关系，并向Client返回最新的元数据位置信息，Client缓存最新位置后缓存数据实现同步。

缓存数据同步的方式具有很高精度和较低资源占用率。Client 会启动后端监控线程执行事件监听和缓存过期操作，线程一直阻塞到Watcher的节点发生变化，线程被唤醒后执行过期相应缓存的操作，不影响缓存的其他位置信息。

综上所述，本方案适合大数据环境下元数据的管理，能够兼顾小目录和大目录的情况。利用Client 缓存元数据位置能够满足大部分小目录的元数据管理需求，而提出的缓存同步方法可以有效应对出现大目录的情况。

实验测试与分析

对原型系统的性能、扩展性、访问局部性等方面进行测试，并与集中式元数据管理方法和基于哈希的划分方法进行对比分析。限于篇幅，只展示了部分实验结果。

原型系统部署在4台服务器上。其中，一台服务器用于部署IS和LS模块，其他三台服务器部署MDS集群和Zookeeper集群。选择HDFS代表集中式元数据管理方法，HDFS部署在四台服务器上，其中一台为主服务器，三台数据服务器。选择基于目录哈希的元数据管理方法代表分布式元数据管理方法，部署在3台MDS上，客户端直接访问MDS提供的元数据管理服务。

测试使用的服务器配置相同，4核CPU(Intel Xeon CPU E5606 @ 2.13GHz)，8G内存和2T硬盘，百兆网卡，操作系统为Linux master 2.6.32-45-generic-pae。

（1）性能与扩展性分析

（a）创建文件元数据

测试了客户端数量从1到64的变化情况。每个客户端分别在100个目录中创建1000个文件，桶的阈值设为5000，测试结果如图10所示。

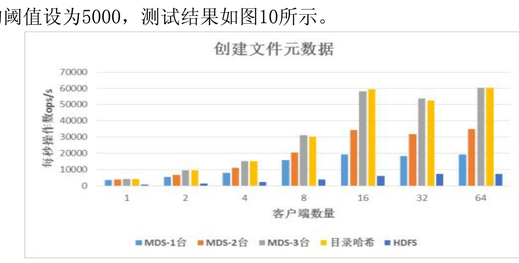

图10 创建文件元数据

（b）重命名目录元数据

为每个客户端准备1000个用于重命名的目录，测试结果如图11所示。从图可以看出，本文方法的重命名目录元数据操作具有很好的扩展性，性能随着MDS的增加而提升。客户端数量为16时，1-3台MDS性能分别提升了10%和23%。

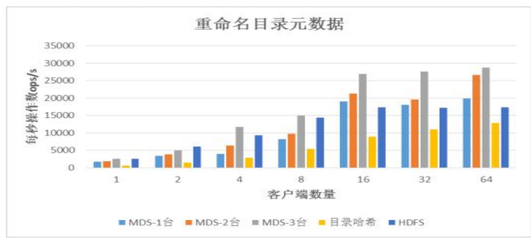

图11 重命名目录元数据

与HDFS相比，本文方法表现了较好的高并发支持。客户端为2时，HDFS性能超过了本文方法使用3台MDS的性能，但客户端的增加到16时，1台MDS的性能也超过了HDFS。这是因为，本文方法的磁盘和网络开销被大量的请求均分，并发度越高，性能越好。此外，重命名目录操作是客户端发起，由IS执行，IS完成目录索引更新后，利用线程池启动的后台线程去执行MDS上的元数据更新操作，比客户端重命名文件操作过程简单，资源利用率也更高。所以性能高于重命名文件的性能。

与目录哈希方法对比，本文方法采用3台MDS时的性能是其两倍左右。这是因为重命名目录操作本文方法只需要更新目录索引中的键值以及MDS中目录元数据中名称属性，并不会导致元数据迁移，而目录哈希方法则需要重新计算路径哈希值并将目录下所有元数据迁移到新计算出的MDS上，并且其性能随着目录中元数据数量的增加而下降。

（c）删除目录元数据

为每个客户端准备10层目录，每层目录中包括10个子目录和1000个文件。每个客户端从顶层目录开始执行递归删除，由于测试结果数量级不同，为了便于显示，将测试结果分别展示（如图12所示）。

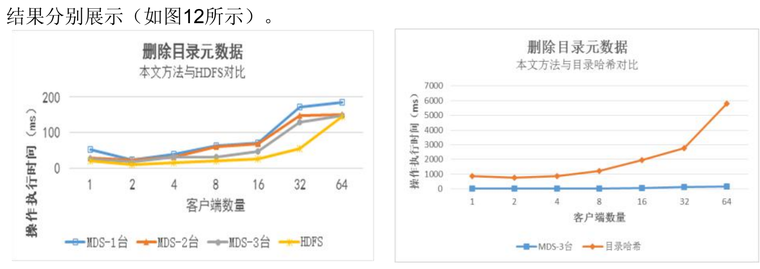

图12 删除目录元数据

从图12可以看出，本文方法具有较高的元数据删除性能和元数据访问局部性。HDFS在内存中完成删除操作，有很高的性能。随着客户端数量的增加，本文方法和HDFS花费的时间都有所增加，当客户端为64时，HDFS随着要删除的数据的增加，性能出现下降，而本文方法对高并发有较好的支持，两者的性能持平。

相比目录哈希方法，本文方法延时更少，当客户端数量为8时，本文方法延时为32ms，目录哈希延时为1211ms。随着客户端数量的增加两种方法延时都在增加，但是本文方法增加的幅度远小于目录哈希方法。这是因为目录哈希方法必须在客户端判断删除目录中是否包含子目录，并对子目录执行递归删除，整个过程有多次网络传输，而本文方法利用构建的目录索引直接由IS模块获得待删目录的子目录结构，并对元数据执行批量删除，从而减少网络传输次数，降低延迟。

（2）访问局部性测试

为了测试访问局部性，将相同目录下的元数据聚集在一起，以便执行目录的遍历或列表时，具有较低的响应延迟。每个客户端遍历10层目录，每层目录中包括10个子目录和1000个文件，阈值设为1000，测试结果如图13所示。

从图可以看出，本文方法具有较好的元数据访问局部性。HDFS遍历操作是在内存中完成，效率非常高，具有很好的元数据访问局部性。随着客户端数量的增加，本文方法表现出较高的并发支持度，性能超过HDFS。目录哈希方法遍历目录时间和本文方法相近。

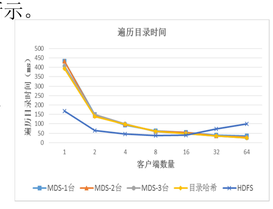

图13遍历目录测试结果

（3）典型组织方法对比

分别从重命名和扩展MDS是否造成迁移、是否限制目录大小和列表目录的时间复杂度等方面，将本文方法与典型元数据组织方法进行了对比，如表下所示。

方法名	重命名造成元数据迁移	不限制目录大小	扩展MDS造成元数据迁移	列表目录的时间复杂度
集中式（HDFS）	no	no	no	O(1)
基于哈希划分	大量迁移	yes	重哈希，大量迁移	low
动态子树划分	no	no	大量迁移	O(1)
本方法	no	yes	no	O(1)

由表可知，与集中式、基于哈希和动态子树划分对比，重命名时本文方法不会造成数据迁移，但哈希会会造成迁移；本文方法和哈希方法都可以不限制目录大小；扩展MDS时不会造成迁移，但哈希方法和动态子树方法都会有迁移，而集中式方法不能扩展；此外，本文方法与HDFS和动态子树相同，具有较高的访问局部性能，能在O(1)的时间内得到结果，而哈希划分方法需要跨多个节点，有较大延迟。

结束语

分布式系统中的元数据受限于其自身的目录结构，扩展性远不如文件数据。本文提出一种路径和属性分离的组织方案，将元数据分成目录索引和属性信息，通过构建索引时生成的三层映射关系，减少了元数据路径和属性之间的耦合关系，有效地提升了MDS的扩展性和元数据的访问局部性，并在执行重命名操作时不会产生数据迁移。