ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.2, February 2007, pp.236−245
DOI: 10.1360/jos180236

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于目录路径的元数据管理方法*

刘 仲+, 周兴铭

(国防科学技术大学 计算机学院,湖南 长沙 410073)

A Metadata Management Method Based on Directory Path

LIU Zhong+, ZHOU Xing-Ming

(School of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: zhongliu@nudt.edu.cn, http://www.nudt.edu.cn

Liu Z, Zhou XM. A metadata management method based on directory path. *Journal of Software*, 2007,18(2): 236−245. http://www.jos.org.cn/1000-9825/18/236.htm

Abstract: A metadata management method dividing directory path attribute from directory object is proposed, which extends the present object storage architecture. This method avoids efficiently the large-scale metadata migration according to the updating directory attributes, improves the cache utilization and hit rate by reducing the overlap cache of prefix directory, reduces the disks I/O demands by reducing the overhead of traversing the directory path and exploiting directory locality, and avoids overloading a single metadata server by dynamic load balancing. Experimental results demonstrate that this method has obvious advantages in improving the throughput, scalability, balancing metadata distribution, and in reducing the metadata migration.
Key words: object-based file system; metadata management; directory path; scalable

摘 要: 提出目录路径属性与目录对象分离的元数据管理方法,扩展了现有的对象存储结构.该方法能够有效避免因为目录属性修改而导致的大量元数据更新与迁移;通过减少前缀目录的重迭缓存提高了元数据服务器 Cache 的利用率和命中率;通过减少遍历目录路径的开销和充分开发目录的存储局部性,减少了磁盘 I/O 次数;通过元数据服务器的动态负载均衡避免单个服务器过载.实验结果表明,该方法在提高系统性能、均衡元数据分布以及减少元数据迁移等方面具有明显的优势.
关键词: 对象文件系统;元数据管理;目录路径;可伸缩
中图法分类号: TP311 文献标识码: A

新兴的对象存储结构利用现有的处理技术、网络技术和存储组件提供空前的可伸缩性和聚合吞吐量,为构建 PB(10^{15} 字节)规模的并行存储系统提供了基础[1,2].这种基于对象存储的大规模存储系统最重要的一个特点是将文件的元数据与数据访问分离,由独立的元数据服务器集群承担元数据访问服务.高效的元数据管理对实现存储系统的高性能和高可伸缩性至关重要.元数据的分布管理问题正成为一个重要的研究热点[3,4],目前还没有一种很好的解决方法.

* Supported by the National Natural Science Foundation of China under Grant Nos.60503042, 60573135 (国家自然科学基金); the National Grand Fundamental Research 973 Program of China under Grant No.2003CB317008 (国家重点基础研究发展规划(973))
Received 2005-10-09; Accepted 2005-12-31

静态子树分割(static subtree partitioning)方法根据目录子树组织元数据,优点是元数据的分布简单,便于开发目录的存储局部性;缺点是不能有效地平衡元数据服务器之间的工作负载,遍历目录路径开销大.静态散列(static hashing)方法根据文件的散列值分布元数据.其优点是通过散列计算能够直接定位元数据所分布的元数据服务器,元数据均匀地分布到各个元数据服务器中.其缺点是需要复制前缀目录,减少了元数据可用的 Cache 容量,降低了 Cache 的利用率和命中率;散列值的改变导致大量的元数据迁移,不便于开发目录的存储局部性.延迟混合(lazy hybrid)方法[3]基于文件全路径名的散列值定位元数据服务器,将目录的访问权限合并到每一个文件的元数据当中,减少了遍历目录路径的开销;其缺点是重新计算散列值导致大量的元数据迁移和更新.动态子树分割(dynamic subtree partitioning)方法[4]将文件系统目录层次结构中的不同子树委托授权到不同的元数据服务器中,优点是分割的粒度更小,方法更灵活,能够根据元数据服务器的负载情况实现动态的负载均衡;其缺点是遍历目录层次树开销较大,重迭缓存前缀目录信息降低了 Cache 的利用率和命中率,当重新委托子树时,需要迁移大量的元数据.

我们在元数据管理中引入目录路径标识,提出将元数据分离为目录路径属性和目录对象分离管理的元数据管理方法,扩展了现有的对象存储结构.它通过消除目录属性修改对元数据的影响,有效地避免了元数据的更新与迁移;通过减少前缀目录的重迭缓存,提高了元数据服务器 Cache 的利用率和命中率;通过减少遍历目录路径的开销,充分开发目录的存储局部性,使磁盘 I/O 次数大为减少.实验结果表明,与现有的元数据管理方法相比,该方法在提高系统性能、均衡元数据分布以及减少元数据迁移等方面具有明显的优势.

1 基于目录路径的元数据管理方法

1.1 基本思想

在传统的文件系统中,文件的元数据用于记录文件的访问属性与数据块的存储位置,文件的数据块保存实际的数据内容.目录文件是包含文件列表信息的文件,目录路径用于实现文件系统的目录层次和权限管理.从存储管理的角度来看,数据块的访问受控于文件属性,文件的访问受控于目录路径属性,因为文件的访问不仅仅与父目录的元数据信息相关,还与目录路径中的目录层次和访问权限有关.在对象存储结构中,通过文件属性和数据块分开管理将文件系统的元数据信息与实际的数据读写分离,将元数据中的 inode 部分的工作负载分布到各个智能化的 OSD(object-based storage device),从而极大地减轻了系统的元数据工作负载,提高了系统的整体性能.在系统的元数据管理中,我们进一步将目录路径属性与文件的元数据分开管理,目录路径属性包括文件所在的目录路径名称及相应的路径访问控制属性,文件的元数据包括文件名、文件长度、创建时间、修改时间等访问属性.这种分割符合面向对象的管理方法,是对象存储结构的进一步延伸.一方面消除了目录路径属性的修改对元数据的影响,有效避免了元数据更新与迁移;另一方面可以根据目录路径与文件元数据的特点分别采用优化的管理方法,如减少遍历目录路径的开销、开发目录的存储局部性、减少磁盘 I/O 次数等.而且,目录路径的分离管理能够有效解决元数据服务系统的动态负载平衡与自动可伸缩管理问题.

1.2 系统结构

如图 1 所示,在我们实现的基于对象存储的集群文件系统(object-based cluster file system,简称 OCFS)原型中,主要由客户端文件系统(client file system,简称 CFS)、目录路径索引服务器(directory path index server,简称 DPIS)、元数据服务器(metadata server,简称 MDS)和对象存储服务器(object storage target,简称 OST)4 类子系统组成.

- OST 负责底层的数据块分配、布局,对客户的数据请求进行认证、响应,并提供基于对象接口的数据存储服务.
- DPIS 负责目录路径属性的存储管理,包括查询、创建、更新和删除.
- MDS 负责目录对象的存储管理,包括文件元数据的查询、创建、更新和删除;MDS 和 DPIS 协作为整个存储系统提供统一、一致的命名空间,负责存储系统的目录层次和权限管理,控制客户端对存储服务器的授权访问,协调不同客户对元数据 Cache 的一致性.

- CFS 给上层应用提供树形目录文件结构和 POSIX 兼容的文件系统接口,支持上层应用执行标准的文件操作,如 Open,Read,Write,Close 等操作.

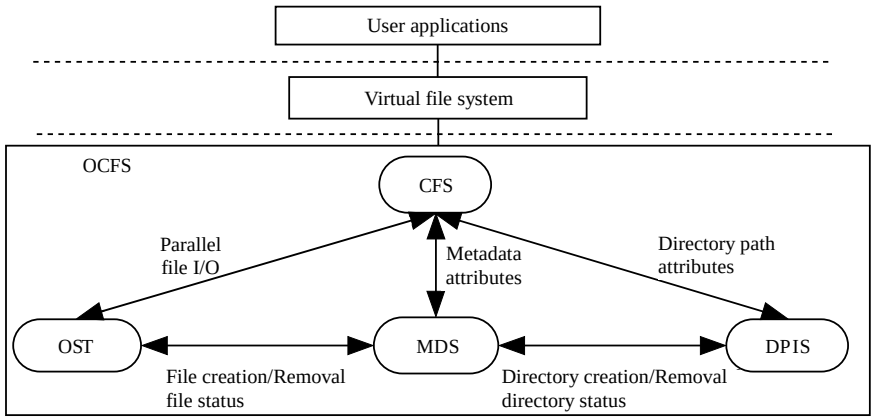

Fig.1 OCFS system architecture

图 1 OCFS 的系统结构

1.2.1 元数据分布

在元数据服务器集群中,建立独立的目录路径索引服务器,将目录的层次、权限管理与目录的文件内容管理分离,目录路径索引服务器承担目录遍历与权限检查任务,减少了元数据服务器的 Cache 重迭,将更多的 Cache 留作文件元数据的缓存,减少元数据读操作引起的磁盘 I/O 次数,简化元数据服务器中的文件元数据管理任务,以便专注于目录文件内容的管理.

1.2.2 数据对象分布

在传统的文件系统中,通过文件元数据中的磁盘地址明细表标识出文件数据块在存储介质的物理分布.读取文件数据前必须先获得数据块的物理分布,增加了磁盘 I/O 开销,并且在文件长度发生变化时(表现为数据块的增加或减少),必须同步更新元数据中的磁盘地址明细表.在对象存储结构中,现有的元数据管理方法通过一个数据对象列表记录该文件包含的所有数据对象所分布的 OSD.我们采取一种完全不同的文件数据对象定位策略,文件的元数据不再需要跟踪记录该文件的数据对象所分布的 OSD,而是通过一种确定性的数据对象布局算法[5−7]快速地计算出每一个数据对象所分布的 OSD.该数据对象布局算法能够将所有数据对象均匀地分布到所有 OSD 中,数据对象的分布是由客户端根据数据对象的对象 ID 自主计算确定的,并且能够动态地适应 OSD 数量的变化.

1.2.3 元数据存储

我们的策略是:充分利用对象存储的优点,由 OSD 构成的集群实现元数据服务器共享的后端存储.由于 OSD 提供基于对象的访问接口,并且 OSD 具有一定的智能,能够优化数据对象的存储管理,因此,在充分利用共享存储的数据管理和可靠性方面的优点的同时,克服了其容易导致 I/O 瓶颈的缺点;另一方面,MDS 集群专门提供元数据访问服务,由于访问最多的元数据能够在 MDS 的 Cache 中命中,所以 MDS 的磁盘 I/O 主要由元数据写操作占据,对于数据量小的元数据写操作,采用日志写、组提交、Write anywhere 策略能够有效地提高写性能,而这些策略能够在 OSD 的对象操作中得到充分体现.

1.3 元数据表示

1.3.1 目录路径索引项

区别于现有的元数据管理方法,在我们的元数据表示中,将目录路径信息与文件的元数据分开管理.由目录路径索引服务器单独管理目录路径信息,能够为系统中的所有目录路径分配一个全局唯一的目录路径 ID.目录路径索引服务器根据存储规模由一台服务器或多台服务器构成的集群提供服务,保证目录路径 ID 是全局唯一

的.目录路径索引服务器中的索引项包括:

- DPID,表示全局唯一的目录路径 ID;
- DirectoryPath,表示该目录路径名字;
- ACp,表示该目录路径的访问控制属性.

索引项由 DPID 唯一确定,该目录路径下的文件元数据分布根据对 DPID 的散列值分布到不同的元数据服务器.这种设计使得文件路径中目录名和访问权限可以任意修改,而 DPID 始终保持不变,所以不会因为目录名和访问权限的修改而导致该目录路径下文件的元数据更新,从而避免了大量元数据的迁移.特别约定根目录"/"的 DPID 为 0.

在访问某个具体的文件时,首先根据文件的目录路径名称向 DPIS 查询该目录路径的 AC,根据 AC 确定用户是否具备访问该目录路径下文件的权限:若没有,则拒绝访问;若有,则返回 AC 以及 DPID.用户根据 DPID 的散列值确定文件所分布的 MDS,向 MDS 获取文件的元数据.用户将获得的目录路径索引项缓存在本地 Cache 中,下一次访问时首先查找本地 Cache,若命中,则不再需要访问 DPIS.根据局部性原理,用户接着访问同一目录路径下文件的概率很高,所以访问 DPIS 的开销被平摊,极大地提高了元数据的访问效率.在两种情况下,用户缓存的目录路径索引项失效:一是用户检查到 Cache 中的缓存项超过了有效期,则主动将它从 Cache 中移去;二是 DPIS 端根据其他用户的请求更改了目录路径中的目录名称或访问权限,从而更新目录路径索引项,但 DPID 保持不变,DPIS 将索引项失效的消息发送到由该 DPID 确定的 MDS,当用户请求该 DPID 下的文件元数据时,MDS 通知该索引项失效.

下面以访问文件为例,对比目录路径方法与采用静态子树分割方法的 NFS 中的访问过程.假定客户端主机为 sun,服务器主机为 bsdi,访问服务器上的文件为/usr/src/test/hello.c.

NFS 中访问文件的网络访问过程包括:(1) getattr (usr);(2) lookup (src);(3) getattr (src);(4) lookup (test);(5) getattr (test);(6) lookup (hello.c);(7) getattr (hello.c);(8) read (hello.c).其中:前面 7 次过程可看作获取元数据;过程(8)读取文件的实际数据.

目录路径方法中访问文件的网络访问过程包括:(1) getattr (/usr/src/test);(2) lookup (hello.c);(3) getattr (hello.c);(4) read (hello.c).其中前面 3 次过程可看作获取元数据,过程(4)读取文件的实际数据.

更一般的情况,假定文件 hello.c 前面的目录层数为 n,对 NFS,需要 $2n+1$ 次过程获取元数据,1 次过程读取实际数据;对目录路径方法,与目录层数 n 无关,仍然只需要 3 次过程获取元数据,1 次过程读取实际数据;很显然,当 $n>1$ 时,目录路径方法需要的网络访问消息数量明显少于 NFS.

1.3.2 目录路径对象

在传统的文件系统中,目录文件的内容以及文件的元数据(inode 层)是分散的,为了获得文件的元数据,需要多次磁盘 I/O.将文件的元数据嵌入到目录文件中能够提高元数据的访问效率[8].目录路径对象类似于传统文件系统中的目录文件,包含该目录路径下的文件元数据列表.所不同的是,传统目录文件中的登记项只包含文件名及其索引节点号,读取文件元数据需要先访问目录文件获得文件索引节点号,再根据索引节点号读取该文件的元数据;而目录路径对象的登记项包含文件的全部元数据,因此,读取目录路径对象时直接获得文件的元数据.

目录路径对象的内部结构如图 2 所示.

目录路径对象作为目录路径的数据对象由 MDS 管理,目录路径对象包含一个或多个固定大小的桶(bucket)对象,每一个 Bucket 对象包含固定数量的文件或目录文件元数据的登记项入口(entry),在目录路径对象的起始位置包含描述该目录路径对象属性的元数据.

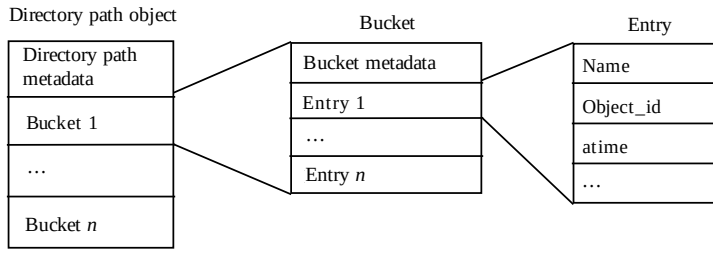

Fig.2 Directory path object structures

图 2 目录路径对象的组成结构

目录路径对象的元数据主要包括:

- DPID:等于目录路径索引服务器中的对应索引项的 DPID,是标识目录路径对象的全局唯一的对象 ID,用一个 48 位的整数标识,48 位的标识符允许系统支持近 3 万亿个目录,对 PB 规模的存储系统已经足够;
- Count:Bucket 对象计数器,表示当前目录路径对象包含的 Bucket 对象个数;
- Split_level 和 *p*:用于线性散列方法所需的记录分裂级和下一个分裂 Bucket 对象指针;
- dirpath_flag:标识目录路径对象当前的状态.

Bucket 对象主要包括:

- Bucket_id:标识目录路径对象内部的 Bucket 对象 ID,用一个 16 位的整数标识,16 位的标识符允许每个目录路径对象最多支持 65 536 个 Bucket 对象;
- bucket_flag:标识 Bucket 对象当前的状态;
- Entry:用于登记文件或子目录的元数据项.

Entry 主要包括:

- name:文件名;
- Object_id:标识文件对象的全局唯一的对象 ID;
- type:标记元数据项类型;
- file_flag:标记元数据项的当前状态;
- 基本属性:如 mode,uid,gid,size,atime,ctime,mtime 等.

在传统的文件系统中,通过逐级遍历目录文件内容查找文件的索引节点来定位文件的元数据.由于受限于文件大小和低效的线性查找方法,单个目录支持的文件数量受到限制.在目录路径对象内部的元数据管理中,初始时,目录路径对象只包含一个 Bucket 对象,随着所包含的文件数量的增加或删除,采用线性散列(linear hashing)方法[9]对目录路径对象进行扩展和收缩,增加新的 Bucket 对象或者合并已有的 Bucket 对象,新申请的 Bucket 对象通常是由连续的数据块组成,能够通过一次磁盘 I/O 读取.由于采用线性散列方法动态扩展目录的容量,查找一个元素的平均时间为 $O(1)$,所以,这种管理策略有效地解决了目录的大容量和访问效率问题,理论上能够支持任意的文件数量.并且,在访问文件的元数据项时,当目录路径对象的元数据已经在内存时,只需要一次磁盘 I/O(大多数情况);当目录路径对象的元数据不在内存时,最多只需要两次磁盘 I/O,极大地提高了文件元数据的访问效率.

1.4 元数据管理

1.4.1 元数据定位

元数据的定位方法是将字典表与散列方法相结合,在 DPIS 上建立一个 DPID 与 MDS 的映射表.MDS 映射表包括主 MDS 映射表(表 1)和从 MDS 映射表(表 2).主 MDS 映射表的入口是按照 DPID 的散列值计算得到的散列值与 MDS 的映射表;从 MDS 映射表的入口是直接将 DPID 的值与 MDS 建立映射关系.对任意的目录路径,根据它的 DPID 在主 MDS 映射表中查找到的 MDS 称为该目录路径的主 MDS,在从 MDS 映射表中查找的 MDS

称为该目录路径的从 MDS.在查找时,首先查找从 MDS 映射表的入口,若没有相应入口,则查找主 MDS 映射表.

初始时,MDS 映射表中只包含主 MDS 映射表,任意的文件元数据所分布的 MDS 由其所属目录路径的 DPID 的散列值确定,即在正常访问负载情况下,文件的元数据访问由所属目录路径的主 MDS 响应.根据 DPID 的全局唯一且保持不变的特点,任何目录路径的主 MDS 都是确定的,不会随着目录属性的修改而变化.与现有的根据全路径名计算散列值的方法相比,不仅具有在平均访问负载情况下能够实现 MDS 集群负载均衡的优点,而且避免了由于目录属性修改而导致大量元数据迁移的缺点;另一个重要好处是开发目录的存储局部性,根据 DPID 计算散列值的方法,散列分布的粒度是目录,使得同一目录下的文件元数据分布到同一台 MDS 中,易于开发目录的存储局部性.为了充分利用各个 MDS 的处理资源,可根据各个 MDS 的处理权重在映射表中设置不同的入口数目,如表 1 中 IP 地址为 X.X.X.1 的 MDS 的处理能力是 IP 地址为 X.X.X.3 的 MDS 的 3 倍,所以它的映射入口多了两个,即它负责处理的元数据任务更多.

Table 1 Master MDS mapping table

表 1 主 MDS 映射表

Value	IP address of MDS
0	X.X.X.1
1	X.X.X.2
2	X.X.X.3
3	X.X.X.1
4	X.X.X.2
5	X.X.X.1

Table 2 Slave MDS mapping table

表 2 从 MDS 映射表

Value	IP address of MDS
89	X.X.X.1
200	X.X.X.2
1001	X.X.X.3
2008	X.X.X.4
3008	X.X.X.5
…	…

针对极端访问负载的情况,我们采取负载迁移的方法避免出现 MDS“过载”.具体的方法是:MDS 在目录路径对象中设立计数器,用以记录该目录路径下文件被访问的“忙”程度,MDS 通过心跳消息定期将计数达到某个阈值的目录路径信息以及总的负载信息传递给 DPIS.DPIS 接收到计数达到某个阈值的目录路径信息时,选择一个负载最轻的 MDS 作为该目录路径的从 MDS,并且将该映射信息插入到从 MDS 映射表中.若某个目录路径在从 MDS 中的计数也达到某个阈值,则继续增加该目录路径的从 MDS.若目录路径有多个从 MDS,则依据 MDS“忙”程度确定对应的从 MDS.若“忙”目录路径的计数下降到某个阈值,则 MDS 也通过心跳消息汇报到 DPIS,DPIS 将对应的映射信息从映射表中删除.为了保持元数据的一致性,从 MDS 只负责热点元数据的读取,若需要更新元数据则转发给主 MDS 处理,有效地保证了元数据的一致性.

MDS 映射表设立一个版本号,当 DPIS 更新 MDS 映射表以后,通过广播消息将更新的 MDS 映射表传递给各个 MDS.当客户端连接到 DPIS 或 MDS 时,若客户端的 MDS 映射表版本号与 DPIS 或 MDS 的不一致,则将获得新版本号的 MDS 映射表.

LazyHybrid 方法也采用了 MDS 映射表的方法.所不同的是,在该方法中,根据全路径名的散列值确定 MDS,由于目录名或文件名的修改导致 MDS 的变化,尤其是目录的修改导致该目录下所有文件和子目录下文件的 MDS 修改,从而需要迁移大量的元数据;另一方面,它没有涉及处理热点访问等极端访问负载的策略.

1.4.2 访问控制

在传统的基于目录的访问控制方法中,需要遍历文件所在路径的所有目录才能确定用户对该文件的访问权限.在目录路径方法中,将文件本身的访问控制属性与所在的目录路径访问控制属性分开管理,其中:目录路径访问控制属性的构造与现有的遍历文件所在路径中的所有目录来确定该文件的访问权限的方法相同;不同的是每次创建目录时,将获得的路径访问控制属性记录在当前索引项中,并且在该目录下创建新的子目录时,不再需要遍历前面的各层目录,而是递归地使用当前的路径访问控制属性与新建目录的访问控制属性构建新的目录路径访问控制属性.这样带来的好处是,只需要通过 DPIS 查询文件的目录路径访问控制属性,无须遍历文件所在路径的所有目录.其实质是将遍历目录的结果记录下来,将多次查询减少为一次查询操作,极大地提高了元数据访问效率.图 3 给出了目录路径访问控制属性的构造过程.

用户若在目录/home/cpp 下新建目录 test,那么构造目录路径/home/cpp/test 的访问控制属性过程如下:

- 用户查询目录路径/home/cpp 的访问控制属性,获得/home/cpp 的访问控制属性为 rwxr-xr-x,即用户有新

建目录的权限;

- 用户设定新建目录 test 的访问权限属性为 rwxr--r--;
- 根据目录路径/home/cpp 的访问控制属性以及新建目录 test 的访问权限属性,构造目录路径/home/cpp/test 的访问控制属性为 rwxr--r--.

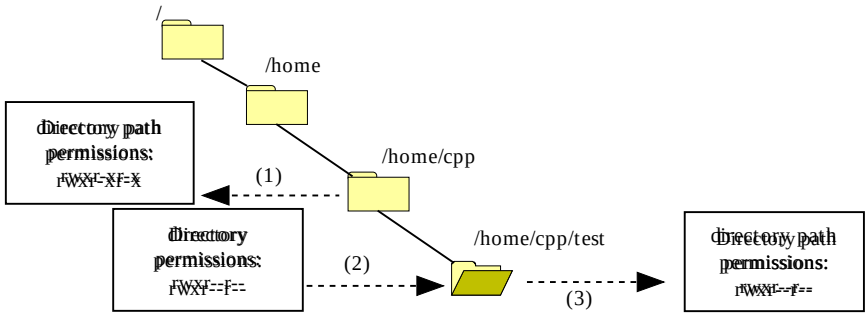

Fig.3 An example of creating directory path permissions

图 3 构造目录路径访问权限的示例

LazyHybrid 方法采用与我们类似的方法.所不同的是,在 LazyHybrid 方法中,路径访问权限保存在所有文件(包括目录文件与普通文件)当中,并且文件的元数据根据全路径名进行散列计算分布到各个 MDS.如果更改某一个目录的访问权限,那么该目录下的所有子目录和文件的元数据信息必须更新,导致大量的元数据更新,涉及到所有的 MDS,既增加了系统的开销又引发元数据的一致性问题.在我们的方法中,任何目录权限的修改,只需要在 DPIS 中更新相关目录路径的访问控制属性,可通过 SoftUpdate[10]方法保证一致性,与 MDS 中的文件元数据无关,修改简单而且快速.

2 实验与性能分析

2.1 实验方法

本节在原型系统 OCFS 中对现有的几种典型的元数据管理方法进行评估与比较.根据相关研究[11],在分布式文件系统中,在读取文件元数据所涉及的元数据操作中,lookup,getattr 和 read 操作所占的比重排在前 3 位,合起来约占 80%,其中 lookup 操作查找文件的目录项,getattr 操作读取文件的属性,read 操作从磁盘读取数据到内存.我们的测试实验主要针对这 3 种元数据操作进行测试,测试的方法是从 Linux 文件服务器中读取测试数据,将所有的文件元数据写入到数据库中,实验数据包括 9 215 个文件,分布在 1 525 个目录中.实验中测试的几种元数据管理方法分别为:

- DirPath:本文提出的目录路径方法.
- Subtree:通过人工方法将不同的目录子树分布到各个 MDS,以目录遍历的方式访问元数据.
- LazyHybrid:根据文件的全路径名散列值确定所分布的 MDS,并且将前缀目录的访问权限保存到文件的元数据中,确定文件的访问权限不需要遍历目录子树.
- FileHash:根据文件的全路径名散列值确定所分布的 MDS,查找目录项的方式与 Subtree 方法相同.
- DirHash:根据文件的目录名散列值确定所分布的 MDS,查找目录项的方式与 Subtree 方法相同.

2.2 实验分析

2.2.1 性 能

在固定 MDS 的 Cache 大小和客户端访问文件数量、顺序的情况下,图 4 显示了不同元数据管理方法在不同的 MDS 配置下总的 read 操作次数.从图中可以看到:DirPath 和 Subtree 方法所需要的 read 操作次数最少;DirHash 和 LazyHybrid 方法所需要的 read 操作次数中等;而 FileHash 方法所需要的 read 次数最多.出现这种

性能差别的原因在于 Cache 的命中率不同,其中,DirPath,Subtree,DirHash 方法都是根据文件所在目录的不同,将元数据分布到不同的 MDS 中,在同一个目录下的文件元数据分布到相同的 MDS 中.一方面,开发了目录的存储局部性,提高了 Cache 的命中率,减少了 read 操作次数;另一方面,减少了前缀目录在不同的 MDS 中的重叠缓存,使得有更多的 Cache 空间可用来缓存文件元数据项,提高了 Cache 的命中率,也减少了 read 操作次数.而 LazyHybrid 和 FileHash 方法根据全路径文件名的散列值确定元数据分布的 MDS,使得同一目录下的文件元数据分布到不同的 MDS 中,忽略了目录的存储局部性,并且目录在多个 MDS 中的重叠缓存减少了用于文件元数据缓存的 Cache 空间,降低了 Cache 的命中率.由于 LazyHybrid 方法将目录访问的权限同时保存在文件元数据中,避免了目录遍历所需要的开销,减少了 read 操作,所以与 FileHash 方法相比,LazyHybrid 方法所需要的 read 操作次数明显减少.

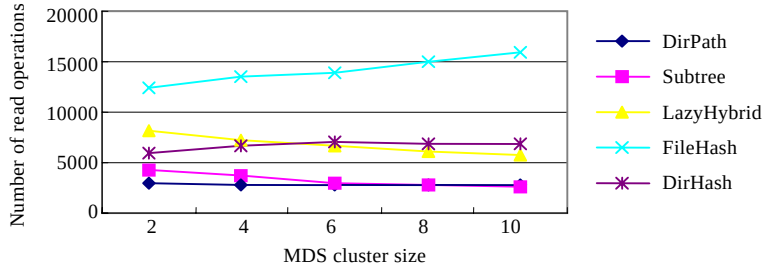

Fig.4 Number of read operations vs. the MDS cluster size scales

图 4 Read 操作次数对比 MDS 集群规模的变化

2.2.2 负载分布

在固定 MDS 的数量、Cache 大小和客户端访问文件数量、顺序的情况下,图 5 显示了不同元数据管理方法中各个 MDS 所分布的元数据数量情况.从图中可以看出:基于文件全路径名的散列值进行 MDS 分配的 LazyHybrid 和 FileHash 方法在元数据分布均匀方面是最好的,文件元数据能够均匀地分布到各个 MDS 中;基于 DPID 散列值的 DirPath 方法以及基于目录名散列值的 DirHash 方法具有较好的负载均衡性,文件元数据能够比较均匀地分布到各个 MDS 中;而 Subtree 方法中各个 MDS 所分布的元数据数量差异非常明显.所以,在平均工作负载情况下,LazyHybrid,FileHash,DirPath 和 DirHash 方法能够实现各个 MDS 的负载均衡,而静态 Subtree 方法不能有效地实现各个 MDS 的负载均衡,动态 Subtree 方法根据负载均衡策略动态复制元数据实现 MDS 的负载均衡,但增加了复杂性和复制元数据的开销.

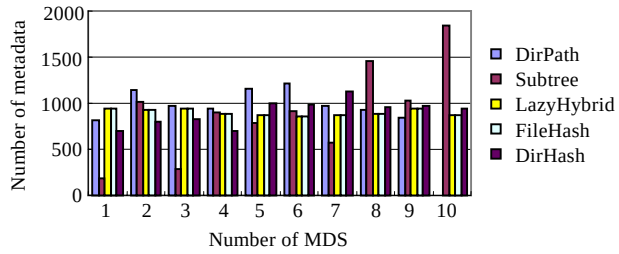

Fig.5 The rate of directory updated vs. the rate of metadata migrated scales

图 5 修改目录的比例对比需要迁移的元数据变化的比例

2.2.3 元数据迁移

在固定 MDS 数量的情况下,用户修改目录(包括修改目录名称、访问权限)和文件,随机地抽取全部目录的 1%~5%,修改其名称,图 6 显示了需要迁移的元数据数量占全部文件元数据的比例.从图中可以看出,对于 LazyHybrid 和 FileHash 方法,尽管修改的目录比例较小(只占目录的 1%~5%),但需要迁移的文件元数据数量很大(占全部文件元数据的 2.47%~12.75%),对于 DirHash 方法,需要迁移的文件元数据量明显减少(占全部文件元

数据的 1.17%~6.15%).相反地,对于 DirPath 和 Subtree 方法,目录的修改不会导致元数据的迁移,Subtree 方法只需要修改单个目录,而 DirPath 方法只需要在 DPIS 中修改涉及到的目录路径数据项即可,但不影响文件元数据的迁移.

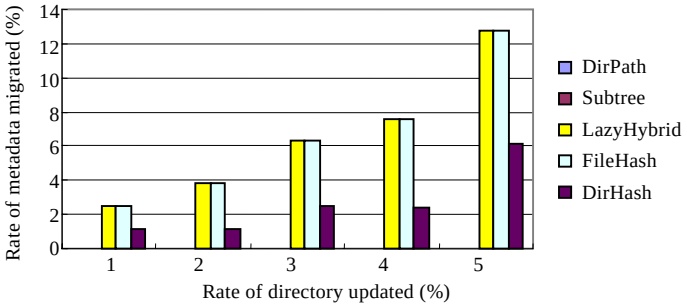

Fig.6 The rate of directory updated vs. the rate of metadata migrated scales

图 6 修改目录的比例对比需要迁移的元数据变化的比例

2.2.4 综合比较

根据前面的讨论和测试实验分析,表 3 对不同元数据管理方法在各个方面进行了综合比较.

Table 3 Comparison of different metadata management methods

表 **3** 不同元数据管理方法的比较

	Exploiting directory locality	Cache utilization	Workload distribution	Hot-Spots	Migrated metadata (directory updates)	Migrated metadata (MDS cluster scales)
DirPath	yes	high	more balanced	yes	none	less
Static subtree	yes	high	unbalanced	no	none	less
Dynamic subtree	yes	high	unbalanced	yes	none	less
Lazy hybrid	no	more high	balanced	no	large	large
File hash	no	low	balanced	no	large	large
Dir hash	yes	medium	more balanced	no	medium	large

从表 3 中可以看出,与其他元数据管理方法相比,DirPath 方法在提高系统吞吐率、实现负载均衡以及减少元数据迁移等方面具有明显的优势.

3 结 论

高效的元数据管理对实现大规模集群文件系统的高性能和高可伸缩性至关重要.本文的主要贡献是进一步扩展了现有的对象存储结构,在元数据管理中引入独立的目录路径标识,提出将元数据分离为目录路径属性和文件元数据的元数据管理方法.实验结果表明,该方法在提高系统吞吐率、实现负载均衡以及减少元数据迁移等方面具有明显的优势.

References:

[1] Nagle D, Serenyi D, Matthews A. The Panasas ActiveScale storage cluster delivering scalable high bandwidth storage. In: Benton V, ed. Proc. of the ACM/IEEE SC 2004 Conf. Washington: IEEE Computer Society, 2004. 53−62.

[2] Lustre SP. Building a file system for 1000 node clusters. In: John WL, ed. Proc. of the 2003 Ottawa Linux Symp. Ottawa: Red Hat, Inc., 2003. 401−407.

[3] Brandt SA, Xue L, Miller EL, Long DDE. Efficient metadata management in large distributed file systems. In: Miller E, Meter RV, eds. Proc. of the 20th IEEE/11th NASA Goddard Conf. on Mass Storage Systems and Technologies. San Diego: IEEE Computer Society, 2003. 290−298.

[4] Weil SA, Pollack KT, Brandt SA, Miller EL. Dynamic metadata management for Petabyte-scale file systems. In: Benton V, ed. Proc. of the ACM/IEEE SC 2004 Conf. Washington: IEEE Computer Society, 2004. 4−15.

[5] Liu Z, Zhou XM. A data object placement algorithm based on dynamic interval mapping. Journal of Software, 2005,16(11):

1886−1893 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/1886.htm

[6] Liu Z, Zhou XM. An adaptive data objects placement algorithm for non-uniform capacities. In: Jin H, Pan Y, Xiao N, Sun JH, eds. Proc. of the 3rd Int'l Conf. on Grid and Cooperative Computing. LNCS 3251, Berlin, Heidelberg: Springer-Verlag, 2004. 423−430.

[7] Liu Z, Xiao N, Zhou XM. RDIM: A self-adaptive and balanced distribution for replicated data in scalable storage clusters. In: Pan Y, Chen DX, Guo MY, Cao JN, Dongarra J, eds. Proc. of the 3rd Int'l Symp. on Parallel and Distributed Processing and Applications. LNCS 3758, Berlin, Heidelberg: Springer-Verlag, 2005. 21−32.

[8] Ganger GR, Kaashoek MF. Embedded inodes and explicit groupings: Exploiting disk bandwidth for small files. In: Proc. of the 1997 USENIX Annual Technical Conf. Anaheim: USENIX, 1997. 1−17.

[9] Litwin W, Neimat MA, Schneider DA. LH*—A scalable, distributed data structure. ACM Trans. on Database Systems, 1996,21(4):480−525.

[10] McKusick MK, Ganger GR. Soft updates: A technique for eliminating most synchronous writes in the fast file system. In: Proc. of the FREENIX Track: 1999 USENIX Annual Technical Conf. Monterey: USENIX, 1999. 1−18.

[11] Duchamp D. Optimistic lookup of whole NFS paths in a single operation. In: Proc. of the USENIX Summer 1994 Technical Conf. Boston: USENIX, 1994. 161−169.

附中文参考文献:

[5] 刘仲,周兴铭.基于动态区间映射的数据对象布局算法.软件学报,2005,16(11):1886−1893. http://www.jos.org.cn/1000-9825/16/1886.htm

刘仲(1971−),男,湖南邵东人,博士,副教授,主要研究领域为网络存储,并行与分布处理.

周兴铭(1938−),男,教授,博士生导师,中国科学院院士,CCF 高级会员,主要研究领域为高性能体系结构,并行与分布处理.

全国第 2 届语义 Web 与本体论学术研讨会（SWON 2007）

征 文 通 知

全国语义 Web 与本体论学术研讨会(SWON)是中国计算机学会暨电子政务与办公自动化专委会主办的系列会议。SWON 2007 会议将于 2007 年 9 月 14 日~16 日在中国人民大学召开。会议目的是为语义 Web 学术界和工业界提供一个交流平台，反映国际国内关于语义 Web 的最新研究成果和进展。会议录用论文将由《东南大学学报》（英文版）（EI 源刊）正刊专辑出版。会议期间除了进行会议论文交流以外，还将邀请著名学者作特邀报告。本次会议仍将评选大会优秀学生论文。

一、征文范围（包括但不限于）

语义 Web 语言与工具，语义 Web 知识表示，语义 Web 知识管理，语义 Web 推理，语义 Web 服务，语义 Web 安全，语义 Web 挖掘，语义信息标注，语义检索和查询，本体学习与元数据生成，本体存储与管理，语义集成和映射。

二、投稿要求

1．本次会议只接受 E-mail 投稿。

2．本次会议只接受英文稿，一般不超过 6000 字，为了便于出版论文集，来稿必须附中英文摘要、关键词、资助基金与主要参考文献，注明作者及主要联系人姓名、工作单位、详细通信地址（包括 E-mail 地址）与作者简介。稿件要求采用 Word 或 PDF 格式，按照《东南大学学报》（英文版）格式编排。

三、重要日期

征文截止日期：2007 年 4 月 1 日　　录用通知发出日期：2007 年 4 月 20 日　　正式论文提交日期：2007 年 5 月 10 日

四、联系方式

投稿地址：华中科技大学计算机学院　李瑞轩，文坤梅（swon2007@hust.edu.cn）

会务情况：中国人民大学　杨楠，姜芳艽（wisa2007@gmail.com）

大会网站：http://www.ruc.edu.cn/wisa2007/，http://www.neu.edu.cn/wisa2007