基于 DBMS 的元数据管理策略 *

吴 婷, 鞠时光, 蔡 涛
(江苏大学 计算机科学与通信工程学院, 江苏 镇江 212013)

摘 要: 海量存储系统的元数据一般采用层次结构或哈希法来管理,存在元数据修改和查询目录等操作所需时间和空间开销大等问题,严重影响了系统的性能。通过引入二维表保存元数据信息,提出了一种基于 DBMS 的新型元数据管理策略。分析了将基于 DBMS 元数据管理策略用于管理海量存储系统中的元数据信息时,所需的时空开销以及管理元数据的灵活性。验证了基于 DBMS 元数据管理策略能有效地减少查询和更新所需的时空开销,实现高效、灵活的元数据管理功能,从而有效地提高海量存储系统的性能。

关键词: 海量存储系统; 元数据管理; 数据库管理技术
中图分类号: TP315 **文献标志码**: A **文章编号**: 1001-3695(2010)04-1297-04
doi:10.3969/j.issn.1001-3695.2010.04.025

Metadata management strategy based on DBMS

WU Ting, JU Shi-guang, CAI Tao
(*School of Computer Science & Telecommunication Engineering, Jiangsu University, Zhenjiang Jiangsu* 212013, *China*)

Abstract: The mass storage systems generally exploit the hierarchy architecture or hashing scheme to manage the metadata, which requires more time and memory consumption for some operations, such as modifying metadata and querying directory, thus, it seriously affects the performance of the system. This paper introduced the two-dimension table to preserve the metadata and proposed a new management strategy for metadata which based on DBMS. At the same time, analyzed the time and memory consumption and the flexibility of managing the metadata, when using the new strategy to manage the metadata information in the mass storage system. Finally, validated that the new strategy could reduce the time and memory consumption for query and update effectively, and could manage the metadata efficiently and flexibly. So improved the performance of the mass storage system effectively.

Key words: mass storage system; metadata management; database management technology

0 引言

海量存储系统中需要保存 Terabyte、Petabyte 级别甚至更大规模的数据。它们的元数据,如文件的名字、属性和访问授权等信息集中由元数据服务器(metadata server, MDS)进行管理。访问存储系统中的数据时[1],首先需要访问 MDS,利用文件名等信息查询,获得数据属性和访问授权等信息后,才能读取相应的数据。海量存储系统需要处理大量的访问请求,元数据的管理性能对海量存储系统的 I/O 性能有着很大的影响。现有的海量存储系统[2]一般采用目录子树分区算法和文件 hash 算法两类元数据管理算法。由于采用层次结构或 hash 方法保存元数据,这些算法在执行改名、更改权限和目录内容列表等操作时,需要迁移或修改大量的元数据,给海量存储系统的性能带来很大影响。如何针对传统元数据管理算法存在的弱点,研究新型元数据管理策略具有重要的意义。

本文在分析海量存储系统中传统元数据管理算法的基础上,针对其存在的问题,引入 DBMS 技术来管理元数据,用二维表保存海量存储系统中的元数据,设计基于 DBMS 的元数据管理策略;分析了传统元数据管理算法,给出了元数据的保存方法和在执行不同类型命令时操作元数据的方法,并分析了基于 DBMS 元数据管理策略的性能。

1 相关研究

元数据管理算法对海量存储区域网系统的性能影响很大,如何设计高性能的元数据管理算法是提高存储区域网性能的重要手段。常用的元数据管理算法包括目录子树分区法和文件哈希算法。

1986 年美国麻省理工学院的 Popek 等人[3~5]提出的目录子树分区(directory subtree partitioning)算法,将层次式的目录结构划分为若干子树,把不同的目录子树分布到不同的元数据服务器中,从而提高元数据管理的性能。但层次式的目录结构存在遍历开销大的问题,在执行目录改名和更改访问授权等操作时需要移动大量的元数据,影响了海量存储系统的性能。

1996 年 IBM 的 Corbett 等人[6]提出的 hash 算法,通过设计相应的 hash 函数,可以将同一目录下的文件均匀地分布到不同的元数据服务器中,能减少元数据操作中的瓶颈。但该算法

收稿日期: 2009-09-08; **修回日期**: 2009-10-20 **基金项目**: 江苏省自然科学基金资助项目(BK2007086);江苏大学高级人才启动基金资助项目(09JDG038);江苏省高校自然科学基金资助项目(09KJB520001)

作者简介:吴婷(1983-),女,江苏常州人,硕士研究生,主要研究方向为存储系统(wt10061008@163.com);鞠时光(1955-),男,江苏海安人,教授,博士,主要研究方向为信息安全、存储系统;蔡涛(1976-),男,江苏海安人,副教授,博士,主要研究方向为安全存储系统.

破坏了目录的层次结构,在执行与目录有关的目录列表和访问授权管理等指令时,需要遍历所有的元数据信息;同样存在执行目录改名和更改访问授权等操作时所需时间和空间开销较大的问题。

2003 年美国加利福尼亚圣克鲁兹大学存储系统研究中心的 Brandt 等人[5]提出的 LH 算法,在保持元数据的层次目录结构的同时,使用 hash 函数计算路径名的 hash 值来确定存放文件元数据的位置;在文件的权限访问方面,将文件的访问授权和目录的访问授权进行区分管理,提高了更改访问授权等操作的性能。但由于文件的访问授权未分散保存于访问路径的所有目录和文件中,在更改某一个目录的访问权限后,需更新该目录下的所有子目录和文件的访问权限,此外还会引发元数据一致性的问题。

2007 年华中科技大学的苏勇等人[7]提出元数据共享存储池管理算法。由网络存储器构建共享存储池,元数据采用 hash 函数进行分布,可以避免当某个目录成为访问热点时所存在的瓶颈问题,但存在 hash 函数确定困难、适应性差等问题。

目录子树和 hash 算法是目前海量存储系统中保存元数据的两大类方法,在执行改名、更改权限和列表目录内容等操作时,会引发时间与空间开销过大等问题。本文在元数据管理中引入 DBMS 技术,提出使用二维表保存文件和目录的元数据及它们之间的层次结构,并设计相应的元数据操作管理算法,消除目录属性修改对元数据的影响,有效避免了元数据的更新和迁移。

2 基于二维表的元数据结构

海量存储系统中[8]通常将文件的元数据与数据分开存储,数据直接保存在存储设备中。元数据包括文件的属性、访问授权和数据的存储位置等信息,一般由专门的元数据服务器进行管理。主机在接收到用户访问数据的请求后,首先向元数据服务器发送请求,根据元数据服务器返回的数据地址信息访问相应的存储设备。此时,元数据服务器需要完成元数据的查找、比较访问授权信息和提取数据地址等一系列操作。此外,还有不少用户的访问请求只需对元数据进行处理,如文件的改名、移动、修改属性(文件大小、所有者、创建时间和修改时间等属性)的操作。

为了解决文件重名的问题,仅使用文件名查找元数据是不够的,还需要文件的访问路径等信息。另外,在访问文件的元数据时,需要查找的不仅是该文件的元数据,还有该文件访问路径上所有目录的元数据。因此,在设计基于二维表的元数据结构时,将文件名与访问路径相结合用于查找文件。

采用层次结构的元数据组织方式在访问某一文件之前,需要依次、逐层地访问该文件访问路径上的所有目录,很难快速、便捷地查找到某个文件。而本文设计多个二维表用于保存元数据,建立索引,可以减少查找文件所需的时间和空间开销。

不同的文件和目录需要保存的属性类型和个数各不相同。传统元数据管理算法中,使用单一格式保存文件和目录的元数据,存在冗余、灵活性差等问题。本文设计动态属性结构,使得能根据各文件和目录的不同需求,灵活地保存不同数量和类型的属性。

目录作为海量存储系统中一类特殊的文件[9],与一般的文件有所不同,它保存了目录所属的文件和子目录信息,还保存了目录与文件之间的层次信息。传统文件系统中采用文本文件保存。本文设计的基于二维表的元数据结构中,目录与文件之间的层次信息由元组之间的关联来表示。此外,由于目录和文件访问授权管理的需求各不相同,文件的访问授权仅需针对单个文件,而目录的访问授权则针对该目录下的所有文件,修改目录的访问授权时需要逐个修改该目录下所有文件的访问授权信息,需要较大的时间和空间开销。本文设计的基于二维表的元数据结构中,保存了目录和文件的访问授权信息。

根据上述对海量存储系统中元数据管理需求的分析,建立五张二维表,即 File 表、File-Attribute 表、Allocation 表、Authorization 表和 Affiliation 表。

File 表(表 1)保存文件和目录的基本信息。其中,ID 是文件或目录的惟一标志;PathName 为文件或目录的访问路径和文件名;DPID 为该文件上层目录的标志。

表 1 File 表

字段	类型
ID	int
PathName	char
DPID	int

File-Attribute 表(表 2)保存文件和目录的属性。其中,ID 是文件或目录的惟一标志;AttributeName 是文件所对应属性的名字;AttributeValue 为文件各属性所对应的值;AttributeType 为文件属性的类型。此时文件所有属性均以字符串的形式保存,使用时再转换为其对应的类型。如 ID 是 1 的文件,包含 Size、Type、Owner 和 ModifyTime 属性,此时 File-Attribute 表中包含(1,Size,int,‘100’)、(1,Type,char,‘normal’)、(1, Owner,char,‘root’)和(1, ModifyTime,time,‘20090626’)等元组。

表 2 File-Attribute 表

字段	类型
ID	int
AttributeName	char
AttributeType	char
AttributeValue	char

Allocation 表(表 3)保存文件中数据所在的数据块信息。其中,ID 是文件的惟一标志;SegmentID 为文件中数据所在数据块的标志。

表 3 Allocation 表

字段	类型
ID	int
SegmentID	int

Authorization 表(表 4)保存文件和目录访问授权的信息。其中,ID 是文件或目录的惟一标志;AC 是文件或目录的访问授权。

表 4 Authorization 表

字段	类型
ID	int
AC	char

Affiliation 表(表 5)保存文件和目录与所有上层目录之间的所属关系。其中,ID 是文件和目录的惟一标志;PID 为文件或目录的所有上层目录标志。

表 5　Affiliation 表

字段	类型
ID	int
PID	int

3　元数据管理算法

本文使用多个二维表保存文件的元数据信息。在这一章中设计一系列对这些二维表的操作序列,用于完成用户访问请求中的元数据操作要求。将其分成对目录的操作、文件的创建与删除、文件属性的更改、文件的访问授权和文件的移动等几个方面,并给出具体的算法。

3.1　目录操作

由于使用层次结构保存文件和目录的结构信息,传统元数据管理算法只能依次打开访问路径中的各目录,逐级访问后才能找到对应文件的元数据。若访问路径的层次较多,则使得访问文件元数据需要较多的时间和空间开销。而文件 hash 算法会破坏目录树的层次结构,使得处理目录列表等操作命令较困难;同一目录所属的各文件与子目录之间缺乏排序机制,因此查找文件较困难。

在本文设计的基于二维表的元数据结构中,文件与目录之间的从属关系蕴涵在 File 表中元组之间的关联中,对这个表进行查找和操作,即能完成处理目录的一系列操作命令。先给出操作命令—目录列表的处理算法:File 表中包含了所有文件和目录的访问路径以及父目录的信息,因此处理目录列表时仅需要查询 File 表。首先根据用户需要将目录内容列表的目录访问路径设为 dp_name,查找该目录所对应的标志 ID,再依据获得的目录标志 ID 查找 File 表中 DPID 与其相同的元组,即可获得该目录的内容列表。对应的 SQL 查询语句如下:

```
select PathName
  from File
    where DPID = (select ID FROM File where PathName = dp_name)
```

当处理修改目录属性的操作命令时,需要查询 File 表和 File-Attribute 表。首先依据访问路径查询 File 表获得目录标志 ID,再更新 File-Attribute 表中该目录的属性。例如,更新目录访问路径为 dp_name 的修改时间为现在的时间(由 now 函数获得),则对应的 SQL 语句如下:

```
update File-Attribute
set AttributeValue = now()
where AttributeName = 'ModifyTime' and ID = (select ID FROM File where PathName = dp_name)
```

3.2　文件操作

对文件的操作包括创建、删除、查询、移动和复制等,分别给出相应的操作算法。创建文件时,首先在 File 表中插入一条元组,记录文件的访问路径、标志和父目录的标志等信息;再将文件的每个属性作为一个元组插入到 File-Attribute 表中;依据文件的访问路径,将文件的每个上层目录作为一个元组插入到 Affiliation 表中;最后将文件数据所在的数据块信息作为一个元组插入到 Allocation 表中。删除文件时所做的工作与创建文件相反。

快速查找文件是海量存储系统重要的要求之一,元数据以层次方式组织,一般以逐层遍历目录内容的方式查找文件,使得查找文件所需的时间和空间开销较大。本文在使用二维表保存文件元数据的基础上,建立索引,并利用文件访问路径缩小查找的范围,减少查找文件所需的时间和空间开销。当已知文件的完整访问路径时,可将访问路径作为关键字查询 File 表,利用索引快速获得文件的标志。当在某一目录下搜索文件时,可首先查找 File 表获得该目录的标志,再查找 Affiliation 表中 PID 值与其相同的元组,从而快速获得所需文件的标志。

移动文件时,只需要修改 File 表中 PathName 的值,用新的访问路径代替。复制文件时,则首先在 File 表、File-Attribute表、Affiliation 表和 Allocation 表中查找需复制文件所对应的元组,用新的文件标志和访问路径替代后再添加到相应的表中。

3.3　文件属性的操作

不同类型文件需保存的属性个数、类型各不相同,File-Attribute 表中每个元组表示某个文件的一个属性,不限制文件属性的个数和类型,从而满足灵活保存文件属性的要求。查询文件的属性时,首先查询 File 表获得文件的标志,然后使用文件标志查询 File-Attribute 表,获得文件的属性;添加文件属性时,将文件标志、属性名、属性值和属性类型作为一个元组添加到 File-Attribute 表中;修改文件属性值时,在 File-Attribute 表中查找相应的元组,将新的属性值写入 AttributeValue 字段中;删除文件的属性时,查找 File-Attribute 表中 ID 值与文件标志相同,且 AttributeName 与要删除的属性名相同的元组。

3.4　访问授权的操作

传统元数据管理算法中,在更改访问授权后,一般采用更新所属的所有文件和子目录或不更新而在访问文件时遍历访问路径上所有目录访问授权两种方法。前一种方法在每次修改目录的访问授权后,需要大量的时间和空间开销用于逐个更新所有文件和子目录的访问授权;第二种方法则在访问文件时,需要大量的时间和空间开销,遍历文件访问路径上的各级目录,逐个比较它们的访问授权。这两种方法均需要大量的时间和空间开销。

通常系统中目录的数量远少于文件的数量,因此更新目录访问授权所需的时间和空间开销占总开销的较少部分;遍历层次型的目录树需要较多的时间和空间开销,因此应避免对文件访问路径中各级目录的遍历。

当修改目录的访问授权后,查找该目录下的所有子目录,用新访问授权更新 Authorization 表中对应的元组。在访问文件时,查找 Authorization 表中该文件和父目录的访问授权并进行比较,若不相同,则将父目录的访问授权作为新的访问授权值,更新该文件元组中 AC 字段的值。

4 性能分析

本文设计了多个二维表用于保存海量存储系统中的元数据,建立索引,并给出了对应不同操作命令的元数据操作方法,实现了高效、灵活的元数据管理算法。下面分别从元数据管理所需时间、空间开销和灵活性两个方面分析基于 DBMS 元数据管理算法的性能。

4.1 元数据管理所需的时间和空间开销

传统的元数据管理算法中,以层次结构保存元数据信息,需要遍历访问路径中的各级目录才能获得文件的元数据,所需的时间和空间开销大。基于 DBMS 元数据管理算法中,使用多个二维表保存元数据信息,原来层次结构的文件与目录之间从属关系转换为元组之间的联系。此外针对 File 表、File-Attribute 表、Allocation 表、Authorization 表和 Affiliation 表分别建立了多个索引,改变了需遍历访问路径中各级目录的方法,有效地减少了查找元数据所需的时间和空间开销。

使用 hash 算法后,由于破坏了文件与目录之间的层次结构,使得执行目录列表等操作命令较困难。基于 DBMS 元数据管理算法中,未破坏文件与目录之间的层次结构,同时代之以查询更为方便的二维表结构保存文件与目录之间的关系,在执行目录列表等操作命令时,只需对 File 表进行两次查询,所需的时间和空间开销小,能有效地改变执行目录列表等操作命令所需时间和空间开销过大的问题。

传统元数据管理算法在修改目录的访问授权后,需要大量的时间和空间开销修改所有的子目录及文件的访问授权,或者在访问文件时需要对遍历访问路径中所有目录提取访问授权进行比较。基于 DBMS 元数据管理算法中,在修改目录的访问授权后,只修改所有子目录的访问授权信息,由于系统中目录数量远少于文件的数量,可减少大量的时间和空间开销;在访问文件时,只需要检查文件及父目录的访问授权,避免了遍历访问路径中各目录所需的大量时间和空间开销。

4.2 元数据管理的灵活性

不同类型文件需保存的属性个数、类型各不相同,传统单一文件属性结构无法适应不同文件和应用的要求。基于 DBMS 元数据管理算法中,由 File-Attribute 表保存文件的属性信息,文件的每个属性对应表中一个元组,不同文件的属性个数、类型等可各不相同,从而实现了灵活保存和管理文件的属性。

在海量存储系统中移动文件时,使用传统元数据管理算法需要查找和修改多个目录项的内容,所需的时间和空间开销较大。而使用基于 DBMS 元数据管理算法时,文件与目录之间的从属关系由元组之间的关联进行表示,修改二维表中的少量数据项即可,所需的时间和空间开销小。

5 结束语

高效的元数据管理对实现海量存储系统中文件存储的高性能和高可伸缩性至关重要。本文的主要贡献是改变了无结构或半结构化的元数据结构,使用二维表保存海量存储系统中的元数据,并建立相应的索引,设计了一种新型的基于 DBMS 的元数据管理方法。通过理论分析,验证了该方法相比现有算法能有效地减少查找和更新元数据所需的时间和空间开销,提高元数据管理的灵活性。

目前在所设计的基于 DBMS 的新型元数据管理策略中使用通用 B 树建立索引,易造成元数据查找效率不高的问题。下一步笔者将根据元数据的特点,设计新型的索引算法,提高元数据的查找效率。

参考文献:

[1] WILLIAM J B, JOHN R D, JACOB R L, *et al*. A five-year study of file-system metadata[J]. ACM Trans on Storage, 2007, 3(3): 31-45.

[2] ZHU Yi-feng, JIANG Hong, WANG Jun, *et al*. HBA: distributed metadata management for large cluster-based storage systems[J]. IEEE Trans on Parallel and Distributed Systems, 2008, 19(6): 750-763.

[3] POPEK G J, RUDISIN G, STOUGHTON A, *et al*. Detection of mutual inconsistency in distributed systems[J]. IEEE Trans on Software Engineering, 1986, 12(11): 1067-1075.

[4] SATYANARAY M, KISTLER J J, KUMAR P, *et al*. Coda: a highly available file system for distributed workstation environments[J]. IEEE Trans on Computers, 1990, 39(4): 184-201.

[5] BRANDT S A, MILLER E L, LONG D D E, *et al*. Efficient metadata management in large distributed file systems[C]//Proc of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage System and Technologies. San Diego: [s. n.], 2003: 290-298.

[6] CORBETT P F, FEITELSO D G. The vesta parallel file system[J]. ACM Trans on Computer System, 1996, 14(3): 225-264.

[7] 苏勇, 周敬利, 余胜生, 等. 基于共享存储池的元数据服务器机群的设计研究[J]. 小型微型计算机系统, 2007, 28(4): 734-737.

[8] FARLEY M. SAN 存储区域网络[M]. 孙功星, 蒋文保, 范勇, 译. 北京: 机械工业出版社, 2002.

[9] 刘仲, 周兴铭. 基于目录路径的元数据管理方法[J]. 软件学报, 2007, 18(2): 236-245.

(上接第 1296 页)

[3] 阎峰, 安晓东. 基于粒子群优化算法的智能抽题策略研究[J]. 中北大学学报: 自然科学版, 2008, 29(4): 333-337.

[4] 俞洋, 殷志锋, 田亚菲. 基于自适应人工鱼群算法的多用户检测器[J]. 电子与信息学报, 2007, 29(1): 121-124.

[5] 黄光球, 姚玉霞, 任燕. 用鱼群算法求解多级递阶物流中转运输系统优化问题[J]. 计算机应用, 2007, 27(7): 1732-1736, 1743.

[6] 高玉芳, 张展羽. 混沌人工鱼群算法及其在灌区优化配水中的应用[J]. 农业工程学报, 2007, 23(6): 7-11.

[7] 李娟, 田胜利. 基于人工鱼群模型的自动组卷抽题算法研究[J]. 许昌学院学报, 2008, 27(5): 92-97.

[8] 张祖平, 袁鑫攀. 智能组卷算法及试卷评价系统的研究[J]. 中国科技论文在线, 2007, 2(10): 730-734.

[9] 王琛, 何琬, 程六满. 基于 AHP/DEA 的自然灾害损失评估模型初探[J]. 中国管理科学, 2007, 15(专辑): 25-28.