# 海量存储系统中高可扩展性元数据服务器集群设计*)

吴 伟[1] 谢长生[1] 韩德志[2] 黄建忠[1]
(华中科技大学计算机学院外存储国家重点实验室 武汉 430074)[1]
(暨南大学计算机系 广州 510632)[2]

**摘 要** 海量存储系统都采用元数据服务器机群的方式来处理文件系统的元数据信息。很多存储系统采用 Hash 算法来实现文件元数据在元数据机群内的分布,但是这些算法都是针对文件进行 Hash。本文提出了一种目录哈希的新算法,针对目录进行 Hash,并把一个目录内的元数据集中存储。本算法克服了文件 Hash 的不足,改善了存储系统的性能,并极大地提高了存储系统的可扩展性。

**关键词** 海量存储系统,元数据服务器机群,高可扩展性,目录哈希

## A Design of High Scalable Metadata Cluster in Mass Storage System

WU Wei[1] XIE Chang-Sheng[1] HAN De-Zhi[2] HUANG Jian-Zhong[1]
(National Storage System Laboratory, School of Computer Science, Huazhong University of Science and Technology, Wuhan 430074)[1]
(Computer Science Depatrment of Jinan University, Guangzhou 510632)[2]

**Abstract** Most large scale mass storage systems exploit metadata server clusters to handle the metadata of file system. Many storage systems use Hash algorithms to distribute the metadata among metadata server cluster. But the Hash key value of these algorithms is files. This paper proposes a new directory Hash algorithm. It treats directory as Hash key value, and implements a concentrated storage of metadata. This algorithm overcomes the disadvantages of file Hash algorithm. It can improve the performance of the storage system, and enhances the system's scalability evidently.

**Keywords** Mass storage system, Metadata server cluster, High scalability, Directory hash

## 1 引言

随着科学计算和各种网络应用的快速发展,人类产生的信息量越来越多。这使得数据的存储越来越被人们所关注,从而使得存储部件在整个计算机系统结构中所处的地位也越来越重要。存储由单一的磁盘、磁带转向磁盘阵列,进而发展到今天日益流行的存储网络,如 NAS、SAN 和 iSCSI 等。大规模的数据应用需求不断涌现,海量数据存储及其应用也成为一个新的发展方向。数据存储已经对人们的工作和生活产生了巨大的影响。

存储系统的规模正不断增大,Terabyte 级别、Petabyte 级别甚至更大规模的海量存储系统不断出现。在这样的系统内,往往存在成百万、上千万个文件。为提高效率,海量存储系统都采用文件本身与其元数据分开存取的组织方式,来对系统内的文件进行处理。这就是存储系统的元数据服务器(Metadata Server),简称 MDS。如果系统规模太大,一台 MDS 已经无法满足元数据处理的需求,于是人们就把多台 MDS 器组成一个机群,提高处理能力,这就是元数据服务器机群。在 MDS 机群中,要想快速、准确、高效地管理元数据,就需要一种良好的元数据分布算法。现有的 MDS 机群中有两种基本的元数据分布方法:子树分区算法和哈希算法。两者各有所长,又都存在着一些缺陷。本文吸取了这两种方法的优点,提出了"目录哈希(Directory Hash)"的新方法,克服了前面两者的不足,提高了存储系统的性能,并且具有很高的可扩展性。

## 2 相关研究

元数据的分布是元数据机群设计中最为关键的问题之一。合理的数据分布算法可以带来更高的性能和更好的可扩展性。常用的设计方法分为两类:一类是目录子树分区算法,另一类是哈希算法。两者都有各自的优点与不足。

目录子树分区算法是以子目录为单位,把根目录下面的各个子目录或者下层的某些子目录分配到不同的元数据服务器上进行存储。该算法可以维持一个完整的传统文件系统层次目录结构。用户对不同子目录的访问,将被引向不同的 MDS,从而达到分流的目的。这种结构便于执行与层次目录结构相关性较大的操作,如"ls"。但是,当某一子目录的内容变成热点数据时,会有很多用户同时访问该子目录,这就会使得存放该子目录的服务器负载加重,成为系统的性能瓶颈。NFS[1]、Coda[2]、Sprite[3]都属于这一类文件系统。

哈希(Hash)算法是通过对文件的某一特定键值(如文件名、路径名等)进行哈希,根据哈希的结果把文件分布到各个服务器上面去。Hash 算法可以把同一目录下的文件均匀分布到各个服务器上,具有更好的负载平衡性,也可以避免目录热点问题。但 Hash 算法也有不足。首先,这种算法不能提供标准的目录层次结构,处理类似"ls"之类的指令比较复杂。其次,Hash 算法的可扩展性比较差。Hash 算法的关键是其 Hash 函数。一旦 Hash 函数确定,其输出范围也就确定了。

*)本文国家"973"重大基础研究项目(编号:2004CB318203)和国家自然科学基金项目(编号:60303031)资助。吴 伟 博士生,研究方向为大规模网络存储系统;谢长生 教授,博士生导师,研究方向为基于网络的存储系统、采用新原理的超高密度、超高速存储技术;韩德志 博士,研究方向为海量网络存储系统;黄建忠 博士,研究方向为网络存储安全。

此时如果系统进行扩展，Hash 函数就要增大输出范围。这样就必须修改 Hash 函数。新的 Hash 函数建立后，又会导致很多元数据的分布和修改之前不一致，于是又要在不同的 MDS 之间进行大量的数据迁移。这些繁琐的操作使得在扩展以 Hash 算法进行元数据分布的存储系统时显得非常困难。Vesta[4]、zFS[5]、Lustre[6]都属于这一类文件系统。

随着对大规模存储系统的需求逐渐递增，海量存储系统中的元数据分布算法正逐渐成为研究热点，而经过改进的哈希算法则成为新的研究趋势。国外一些大学和研究机构在这方面已开展了很多研究工作，比较知名的有懒惰混合(LH)算法[7]、动态元数据管理算法[8]和哈希分区算法(HAP)[9]等。

LH 算法是美国加利福尼亚圣克鲁兹大学存储系统研究中心提出的。它采用路径名哈希来存放文件的元数据，采用懒惰的元数据更新策略，对某些耗时较多的元数据更新操作进行延迟更新，提高了系统的整体运行速度。

动态元数据管理系统也是该校提出来的，它采用目录子树分区的策略，对元数据进行动态的管理。该算法充分利用了文件系统缓存等本地性特点，并结合日志的形式来提高元数据更新操作的响应速度。

新加坡的数据存储中心提出了一种哈希分区(HAP)的元数据分布算法。该算法将元数据的存储和管理分成两层，所有的 MDS 节点共用一个独立的大型的公共存储空间来存取元数据。该空间分为多个逻辑分区，每个 MDS 可以管理一个或多个分区。该算法通过哈希算法和安装/卸载分区的方法，极大地减少了 MDS 机群内部的元数据迁移。另外，还通过动态的负载调节机制，保证系统的负载均衡。

## 3 目录哈希元数据分布算法

本文提出了一种“目录哈希(Directory Hash)”的元数据分布算法，简称 DH 算法。该算法与其它哈希算法有很大的区别。其它的哈希算法都是对文件进行哈希，以文件粒度来分布元数据。为便于说明，我们在下面把这种算法称为“文件哈希算法”。而 DH 算法则是对目录进行哈希，以目录粒度集中分布元数据。该算法可以克服文件哈希算法的许多不足，为存储系统提供更好的性能和更高的可扩展性。

### 3.1 文件与目录的比例关系

DH 算法基于这样一个规律：文件系统的每一个目录下面平均有 10 个以上的文件。或者说，在一个文件系统中，目录的数目平均仅为文件数目的 1/10 以下。我们对很多台计算机的文件系统进行了统计，表 1 是统计结果的一部分。这些计算机中有个人使用的 PC 机，有实验用机，也有文件服务器；操作系统有 Windows 和 Linux 的。统计结果论证了这个规律。基于这点，如果把对文件的操作变成对目录的操作，就可以极大地降低操作的复杂度。

表 1 文件与目录比例关系统计表

| 计算机序号 | 操作系统 | 硬盘上的文件数 | 硬盘上的目录数 | 文件与目录之比 |
|---|---|---|---|---|
| 1 | Windows 2000 | 88676 | 5825 | 15.2 |
| 2 | Windows 2000 | 216628 | 12365 | 17.5 |
| 3 | Windows 2003 | 211666 | 17459 | 12.1 |
| 4 | Linux RedHat | 591193 | 15874 | 37.2 |
| 5 | Linux RedHat | 1504072 | 72826 | 20.7 |
| 6 | Linux RedHat | 1072517 | 46827 | 22.9 |

### 3.2 目录存储单元(DSU)

在 DH 算法中，我们是以目录为单位来集中存储文件系统元数据的。我们把一个目录下的本层文件及本层子目录称为一个目录存储单元(Directory Storage Unit)，下面简称为 DSU。在这里要注意的是，一个 DSU 的元数据信息只包含该目录下面本层文件及子目录的元数据信息，而不包括该目录本身的元数据信息，也不包括子目录下面第二层及更深层次的文件及目录的元数据信息。一个 DSU 的标识为该目录的路径名。图 1 举例说明了 DSU 的划分。

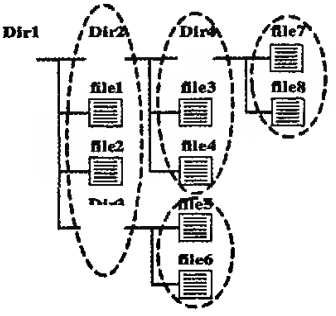


图 1 目录存储单元例图

在图 1 的目录树中，file1、file2、Dir2 和 Dir3 属于目录存储单元 Dir1；file3、file4 和 Dir4 属于目录存储单元 Dir2；file5 和 file6 属于目录存储单元 Dir3；file7 和 file8 属于目录存储单元 Dir4。一个 DSU 的元数据集中存储在某一个 MDS 上，它的索引信息则放置在目录存储单元索引管理服务器上面。

### 3.3 元数据机群体系结构

我们把元数据机群分为 3 个部分：客户端(Client)、目录存储单元索引管理服务器(DIMS)和元数据服务器(MDS)。三者构成了一个类似于面向对象存储系统中常用的三方传输的框架结构。DH 算法中的三方传输和面向对象存储系统中的三方传输的研究对象不同，后者研究的是文件的存放和元数据的管理，前者研究的是元数据的存放和元数据索引信息的管理。图 2 是存储系统元数据机群的体系结构图。


图 2 存储系统元数据机群的体系结构图

图 2 中的 DIMS 就是目录存储单元的索引管理服务器(DSU Index Management Server)，在它上面存放着各个目录存储单元的索引信息。这些索引信息包括该目录下面的文件数目，该目录的访问权限，存放该目录下文件元数据信息的 MDS 的 ID 等。MDS 是元数据服务器，在它上面存放的是各个目录存储单元的元数据信息。

### 3.4 元数据的分布

DH 算法采用路径名哈希的机制，但是哈希的对象不是文件，而是目录。当 Client 端有新的内容要写入存储系统时，对应就会有新的元数据要写入 MDS 机群。如果新建的是文件，则只需要做一件事情，就是找到存放该文件所属 DSU 元数据的 MDS，把文件的元数据写进去。如果新建的是目录，系统就要做两件事情：一是通过 DIMS 找到存放该目录所属 DSU 元数据的 MDS，把该目录的元数据写入；二是在 DIMS 上为该目录新建一个 DSU(不同于该目录所属的 DSU)，并确定存放这一新 DSU 元数据的 MDS。

具体说来，对于文件，该文件属于其上层目录对应的 DSU。Client 先对文件的上层目录进行路径名哈希，通过哈希的结果找到存放该 DSU 索引信息的 DIMS，并向它发送查询请求。DIMS 找到该索引，把索引信息中的 MDS 的 ID 返回给 Client。Client 再向对应的 MDS 发送写请求。MDS 收到请求后，把该文件的元数据添加到对应的 DSU 的元数据集合中去。最后向 Client 返回操作结果。

对于目录，Client 要做两个 Hash：通过对该目录的路径名进行哈希，找到为该目录新建的 DSU 所对应的 DIMS，并向这个 DIMS 发送创建新的 DSU 索引的请求；还要通过对该目录的上层目录进行路径名哈希，找到该目录所属的 DSU 对应的 DIMS，并向该 DIMS 发送查询请求。当 DIMS 接收到创建 DSU 索引请求时，就会为其新建一个 DSU 索引，并确定该 DSU 的元数据所存放的 MDS。在这里，DIMS 采用循环分布的算法来确定 DSU 到 MDS 的映射算法，根据 MDS 的数目，把新建的目录存储单元循环分布到整个 MDS 的存储空间。确定了对应的 MDS 之后，DIMS 就把该 MDS 的 ID 填入 DSU 的索引中，并向 Client 返回操作结果。当 DIMS 接收到查询请求时，其操作和文件的对应操作相同，也是找到对应的 DSU 索引，把 DSU 索引信息中 MDS 的 ID 返回给 Client。接下来，当 Client 收到 DIMS 发回来的 MDS 的 ID 后，就向对应的 MDS 发送写请求。MDS 收到请求后，把该目录的元数据添加到对应的 DSU 的元数据集合中去。最后向 Client 返回操作结果。

在 DH 算法中，DIMS 可以独立确定 DSU 到 MDS 的映射分布算法，并且可以单独替换，而不影响整个系统的其它环节，具有很大的灵活性。这也是我们的元数据存储系统的一个特点。

### 3.5 元数据的访问

当客户端访问某个文件或目录的元数据时，先对其所在目录的路径名进行哈希，由哈希结果访问相应的 DIMS，找到对应的 DSU 的索引信息。然后判断该目录的访问权限。如果目录访问权限为不许可，就向 Client 返回相应的错误信息。如果权限允许，就取得存放该目录存储单元元数据的 MDS 的 ID，并返回给 Client。Client 再访问对应的 MDS，获取该文件的元数据。

### 3.6 访问权限

DH 算法中有两种访问权限：一种是初始访问权限，一种是最终访问权限。初始访问权限就是通常文件或目录自身设定的访问权限，每个文件和目录都有，存在于文件或者目录的元数据中，就相当于 Fat 文件系统下的访问权限。最终访问权限只有目录才有，它是目录的自身访问权限与其上面各层目录的访问权限的交集，由于在 DH 算法中，目录和 DSU 是一一对应的，因此也可以叫做 DSU 的访问权限。最终访问权限存在于 DIMS 上 DSU 的索引信息中，它是在该索引信息建立的时候，由该目录的初始访问权限和上层目录的最终访问权限进行逻辑与操作而确定下来的。如果上层目录的访问权限为真时，该目录的最终访问权限就等于其自身的初始访问权限。如果上层目录的访问权限为假，则该目录的最终访问权限也为假。

在文件哈希算法中，由于没有层次目录的概念，要想确定一个文件或目录的访问权限，必须遍历路径中的各上层目录，这些目录的访问权限与该文件或目录本身的访问权限的交集才是最终的访问权限。相比之下，DH 算法不用遍历其路径上面各层目录去查看它们的权限，减少了很多操作步骤。

在 DH 算法中，确定 Client 对文件或目录的访问权限这一过程其实被分为了两步。第一步，Client 对所访问内容的上层目录的路径名进行哈希，由哈希结果到对应 DIMS 上找到对应目录存储单元的索引信息。该信息中含有该目录的最终访问权限，如果该权限允许，则进行下一步，否则向 Client 返回非法访问权限。第二步，Client 由索引信息中 MDS 的 ID，发送读写请求给对应的 MDS，找到对应的目录存储单元，进而找到所要访问内容的元数据，并判断其初始访问权限。该权限允许则可以访问，不允许就返回非法访问权限。这两个步骤完美地融合进了正常的文件访问流程之中，因而确定文件的访问权限并不会对文件的访问增加额外开销。

## 4 可扩展性

DH 元数据分布算法最大的特点就是其在可扩展性方面的优势。由于引入了 DIMS，使得整个系统的元数据管理的灵活性大大提高。传统数据分布算法中的很多扩展性方面的问题得到了极大的改善。如目录的改名，权限的变更，MDS 的增加与删除等瓶颈操作的复杂度降低，所引起的额外负载减小，操作的速度得到了很大提高。

### 4.1 目录改名

文件哈希由于缺乏层次目录的概念，因此处理目录相关的一些操作非常麻烦，目录改名就是其中之一。目录改名后，该目录及其下面各层子目录与文件的路径名都会改变，因而以路径名为关键值的哈希值也会改变，导致它们的元数据需要从一台 MDS 迁移到其它 MDS 上面。另外，每层目录下面的几十甚至上百个文件可能会分布到所有的服务器上面，每个文件都要单独计算哈希值重新进行定位，然后进行更改，系统开销非常大。此外，在缺乏层次目录结构支持的文件哈希算法中，要想遍历该目录下面所有的子目录和文件本身也是一件不太容易的事情。

DH 算法在目录改名操作时，目录的路径名哈希值会发生改变，从而与 DIMS 的对应关系也会发生变化，因此也会导致部分 DSU 的索引信息在 DIMS 之间的数据迁移。但是相比文件哈希，操作的复杂性就要低得多，速度也要快得多。首先，DH 算法通过目录存储单元的形式把同层的文件及子目录的元数据信息放在一起，便于查找和修改，要遍历某一目录下面的所有文件和子目录也较文件哈希方便得多。其次文件哈希更新和迁移的是每个文件和目录的元数据信息，数量非常大，而 DH 算法中，位于 MDS 上的文件和目录的元数据不需要进行迁移，需要迁移的仅是位于 DIMS 上面的目录存储单元的索引信息。由于文件和目录的比例关系，这个数量仅是前者的 1/10 甚至更少，而且每个索引非常小。这些因素使得 DH 算法中目录改名造成的额外负载比文件哈希少得多，速度也大大加快。

DIMS 上的索引信息更新或者迁移之后，每个索引对应

的 MDS 的 ID 不会改变，还指向原来存储该 DSU 的 MDS，因而，MDS 上面的元数据信息不需要进行任何迁移或改变。

### 4.2 目录更改权限

同目录改名的操作相似，传统的哈希算法在更改某一目录的访问权限时，其下面各层所有的子目录及文件都要相应的更改其权限。更新量非常巨大。而在 DH 算法中，需要修改的只是目录的最终访问权限，即只需要修改 DIMS 上面目录存储单元的索引信息。这个数量只是前者的 1/10 甚至更少，所以更新速度要快得多。

### 4.3 增加、删除 MDS

随着存储系统中文件数目的不断增多，MDS 上的元数据数量也不断扩大。当 MDS 空间不足时，就需要增加 MDS 来对存储系统的容量进行扩展。在传统的 Hash 分布算法中，当增加或者删除 MDS 时，由于 Hash 函数的输出范围会有所变化，因此需要重建 Hash 函数，才能使新增的 MDS 投入使用。但是，当使用新的 Hash 函数时，系统中很多元数据的分布位置会和以前不一致，于是就会产生大量的数据迁移，把和新的 Hash 算法分布不一致的元数据迁移到适当的 MDS 上去。同理，删除 MDS 也会产生相同的情况。

在 DH 分布算法中，只有 DIMS 上的目录索引的分布是由 Hash 函数决定的。增加一台 MDS 并不会导致目录索引管理服务器(DIMS)的增加，所以 Hash 函数不用重建，也就不会发生任何数据迁移。在我们的存储系统中，增加一台 MDS 时，原有的目录索引并不会受到影响。系统会通知各个 DIMS 关于 MDS 数量变化的信息。DIMS 只需要在系统中分布新的元数据时，考虑到 MDS 的增减而对原有平均分布的算法稍做调整即可。我们也可以使用负载平衡机制在系统空闲期间对数据的分布进行调整，通过适当的数据迁移使得系统中各台 MDS 的负载达到新的平衡。

### 4.4 DIMS 的扩展

然而，随着系统规模的不断增大，大到现有的那些 DIMS 的处理能力达到极限时，就需要扩展 DIMS 的数量。这时候，就不可避免地要调整哈希函数，从而导致 DIMS 上某些现有的索引信息的分布规则发生变化，引起某些索引信息在各个 DIMS 之间的数据迁移。尽管如此，DH 算法还是比文件哈希算法具有很大的优越性。这主要体现在以下两点。

首先，由于目录存储单元索引的使用，极大地减慢了系统扩容所造成数据重构的步伐。由前面的论述可知，存储系统中的目录索引和元数据的数量之比，就相当于目录和文件的数量之比。一般情况下文件系统中文件的数目都要比目录的数目多 10 倍以上，而存储系统的规模又主要是由文件的多少来决定的，因此，存储系统规模的扩大对 DIMS 扩展造成的影响相对很小，只是传统 Hash 算法中对 MDS 扩展所造成影响的 1/10 以下。也就是说，目录索引的使用，极大地减慢了系统升级所造成数据重构及迁移的步伐。

其次，由于目录存储单元索引的使用，极大地减小了系统扩容所造成数据迁移的规模。文件哈希算法中增删 MDS 造成数据重构时需要进行迁移的是各个 MDS 上的元数据，而 DH 算法中数据重构时需要进行迁移的数据是目录存储单元的索引信息。在 DH 算法中，由于迁移前后 MDS 上的元数据和 DIMS 上的索引信息中 MDS ID 的对应关系没有变化，因此元数据不需要进行迁移。另外目录索引的数据量仅是元数据的数据量的 1/10 以下，所以即使需要发生数据迁移，其速度也会比 MDS 快上 10 倍以上。

通过以上分析可以看出，DH 算法在可扩展性方面，比传统 Hash 算法相比具有巨大的改进。

## 5 性能影响分析

在 DH 的数据分布算法中，在传统的 Client 和 MDS 之间，加入了 DIMS 这个中间环节。从表面看，一次元数据存取过程被分成了两步来完成，延长了单个请求的响应时间。但实际上，这种体系结构的设计类似于当今很多面向对象存储系统采用的三方传输架构，系统在很短的时间内可能会接收很多的读写请求，由于这两个步骤是分别在 DIMS 和 MDS 两个服务器上面进行的，我们可以通过对这两个步骤进行流水线操作，来产生宏观上的并行效果。因此这一算法不但不会降低系统正常读写操作的性能，反而还会有一定程度的提高。

DH 算法还可以通过使用预取技术来极大地提高存储系统的性能。人们对计算机文件系统的访问模式通常具有一定的空间局部性。也就是说，当人们访问一个文件时，下一次访问这个目录内的其它文件的概率是比较大的。正因为这样，预取技术才能派上用场。文件哈希算法是对文件进行哈希，但同一目录下的文件的元数据哈希后会被分布到不同的 MDS 上面进行存储，这样，要想对该目录下的其它文件的元数据进行预取就变得非常困难了。在 DH 算法的体系结构中，每个目录下的本层文件及目录的元数据是集中存放在一起的，而且元数据通常都比较小，这样就可以一次把一个或多个 DSU 的元数据全部预取到客户端，从而大大提高系统的性能。应该说，DH 算法比文件哈希算法在元数据的存放方式上更接近于人们对计算机的访问方式，所以可以非常简单而又卓有成效地利用预取技术来提高存储系统的性能。

在可扩展性方面，DH 算法通过使用 DIMS 和 DSU，以及对一个 DSU 中文件与目录元数据的集中存放，对系统的可扩展性带来了巨大的提升。它有效地降低了系统在目录改名、权限变更和 MDS 扩展时的系统开销。

**结论** 本文提出了目录哈希(DH)的元数据分布算法来实现大规模存储系统中文件系统元数据的分布。DH 算法引入了目录存储单元(DSU)和目录存储单元索引管理服务器(DIMS)的概念，以目录为单位集中分布元数据。DH 算法在不降低系统性能的情况下，有效地解决了传统 Hash 算法中存在的目录的改名，权限的变更，MDS 的增加与删除等许多难题，极大地提高了存储系统的可扩展性。

## 参考文献


1 Pawlowski B, Juszczak C, Staubach P, et al. NFS version 3: Design and implementation. In: Proceedings of the Summer 1994 USENIX Technical Conference, 1994. 137～151

2 Satyanarayanan M, Kistler J J, Kumar P, et al. Coda: A highly available file system for a distributed workstation environment. IEEE Transactions on Computers, 1990, 39(4): 447～459

3 Ousterhout J K, Cherenson A R, Douglis F, et al. The Sprite network operating system. IEEE Computer, 1998, 21(2): 23～36

4 Corbett P F, Feitelson D G. The Vesta parallel file system. ACM Transactions on Computer Systems, 1996, 14(3): 225～264

5 Rodeh O, Teperman A. zFS - a scalable distributed file system using object disks. In: Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies, April 2003. 207～218

6 Schwan P. Lustre: Building a file system for 1000-node clusters. In: Proceedings of the 2003 Linux Symposium, July 2003

7 Brandt S A, Xue Lan, Miller E L, et al. Efficient metadata management in large distributed file systems. In: Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, April 2003

8 Weil S A, Pollack K T, Brandt S A, et al. Dynamic Metadata Management for Petabyte-scale File Systems. In: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, Nov. 2004

9 Yan Jie, Zhu Yao-Long. A design of metadata server cluster in large distributed object-based storage. In: Proceedings of the 21th IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies, 2004

# 海量存储系统中高可扩展性元数据服务器集群设计



作者： 吴伟， 谢长生， 韩德志， 黄建忠， WU Wei， XIE Chang-Sheng， HAN De-Zhi， HUANG Jian-Zhong

作者单位： 吴伟,谢长生,黄建忠,WU Wei,XIE Chang-Sheng,HUANG Jian-Zhong(华中科技大学计算机学院外存储国家重点实验室,武汉430074)， 韩德志,HAN De-Zhi(暨南大学计算机系,广州 510632)

刊名： 计算机科学 ISTIC PKU

英文刊名： COMPUTER SCIENCE

年，卷(期)： 2007，34(7)



## 参考文献(9条)

1. Pawlowski B. Juszczak C. Staubach P NFS version 3:Design and implementation 1994

2. Satyanarayanan M. Kistler J J. Kumar P Coda:A highly available file system for a distributed workstation environment 1990(04)

3. Ousterhout J K. Cherenson A R. Douglis F The Sprite network operating system 1998(02)

4. Corbett P F. Feitelson D G The Vesta parallel file system 1996(03)

5. Rodeh O. Teperman A zFS-a scalable distributed file system using object disks 2003

6. Schwan P Lustre:Building a file system for 1000-node clusters 2003

7. Brandt S A. Xue Lan. Miller E L Efficient metadata management in large distributed file systems 2003

8. Weil S A. Pollack K T. Brandt S A Dynamic Metadata Management for Petabyte-scale File Systems 2004

9. Yan Jie. Zhu Yao-Long A design of metadata server cluster in large distributed object-based storage 2004

## 引证文献(1条)

1. 于志敏. 谢丽聪. 韩晓芸 基于Web服务的元数据服务体系研究[期刊论文]-微计算机信息 2008(12)