大规模分布式文件系统元数据管理综述

王 洋[1] 刘 星[1] 须成忠[1,2] 江 松[1,2] 王 刚[1] 文 韬[3] 范小朋[1] 陆 平[3]

[1](中国科学院深圳先进技术研究院云计算研究中心 深圳 518055)
[2](美国韦恩州立大学电子与计算机工程系 底特律 48202)
[3](中兴通讯股份有限公司 南京 210012)

摘 要 文件系统的元数据主要是用来描述它的命名空间，访问权限和数据定位等信息的数据。由于50%～80% 的文件系统访问要涉及到元数据，元数据服务的性能将极大地影响整个分布式文件系统的性能。为此，文章重点讨论元数据管理面临的问题，从元数据服务的高可扩展技术、高性能技术和高可用技术三个主要方向进行综述，重点分析了各自的主要问题以及目前发展起来的一些主流技术，同时对未来分布式文件系统的元数据管理一些值得关注的问题进行了梳理和展望，为相关研究提供一定的参考。

关键词 元数据；分布式文件系统；工作负载；高可扩展；高性能；高可用
中图分类号 TP 309 **文献标志码** A

A Review on Metadata Management in Large-Scale Distributed File Systems

WANG Yang[1] LIU Xing[1] XU Chengzhong[1,2] JIANG Song[1,2] WANG Gang[1]
WEN Tao[3] FAN Xiaopeng[1] LU Ping[3]

[1](*Cloud Computing Technology Research Center*, *Shenzhen Institutes of Advanced Technology*, *Chinese Academy of Sciences*, *Shenzhen* 518055, *China*)
[2](*Department of Electrical and Computer Engineering*, *Wayne State University*, *Detroit*, *MI* 48202, *USA*)
[3](*Zhongxing Telecommunication Equipment Corporation*, *Nanjing* 210012, *China*)

Abstract Metadata of file systems is the data in charge of maintaining namespace, permission semantics and location of file data blocks. Metadata operations can account for up to 80% of total file system operations. As such, the performance of metadata services substantially impacts the overall performance of distributed file systems, especially with the advent of big data era, posing great pressure on the underlying storage systems. This paper reports the state-of-the-art research on the metadata services in large-scale distributed file systems. The study was conducted from three perspectives that are always used to characterize the file systems: high-

收稿日期：2015-11-12 **修回日期**：2016-01-26
基金项目：科技部 973 计划(2015CB352400)；国家自然科学基金(U1401258, 61550110250)
作者简介：王洋(通讯作者)，博士，研究员，研究方向为云存储、云计算，E-mail：yang.wang1@siat.ac.cn；刘星，硕士，研究方向为云存储；须成忠，博士，研究员，研究方向为并行与分布式系统；江松，博士，研究员，研究方向为存储系统；王刚，博士，高级工程师，研究方向为机器人与云计算技术；文韬，博士，高级工程师，研究方向为云计算；范小朋，副研究员，研究方向为移动云计算和大数据；陆平，博士，高级工程师，研究方向为云计算。

scalability, high-performance and high-availability, with focus squarely on their respective major challenges as well as their developed mainstream technologies. Additionally, some existing problems in this research were also identified and analyzed, which could be used as a reference for related studies.

Keywords metadata management; large-scale distributed file system; workload; high-scalability; high-performance; high-availability

1 引 言

随着大数据应用的普及与深入，基础计算架构对存储系统在规模和性能要求等方面提出了新的挑战。例如，对健康大数据，交通大数据和金融大数据等这些应用来说，数据量通常都在TB、PB 甚至 EB 数量级，因此需要大量的存储资源来存储和管理这些数据。此外，大量的数据分析任务需要从不同存储地址快速地访问数据，这对于存储系统的读写速度也具有很高的要求。因此，要支持海量大规模数据存储和计算，除了系统的硬件特性之外，高效的数据组织和管理也是必不可少的关键技术之一。目前，分布式文件系统(如 Google File System(GFS)[1])由于其内在设计的简洁性和通用性已成为解决大数据存储管理的有效技术途径。

1.1 元数据工作负载

文件系统元数据是描述文件系统结构特征的系统数据，如文件系统类型、大小、状态信息(超级块)以及针对每个文件或目录的访问权限、拥有者、建立/修改时间等，其中特别重要的是文件数据块的分布信息。对文件系统的操作均要涉及到对元数据的操作，如对一个文件的读写，首先要获得它的元数据，进而定位读写数据块。尽管元数据只占了整个数据空间 0.1%～1% 相对较小的部分[2]，但文件系统中 50%～80% 的访问是针对元数据进行的[3]。

作为文件系统的设计指导，针对元数据的研究工作一直在持续之中。如 Outsterhout 等[3]通过记录用户级的活动分析了对 UNIX4.2BSD 文件系统访问的特点，发现所有文件访问中超过 50% 的访问是针对元数据的访问。此后 Agrawal 等[4]也做过类似的研究，他们从 Windows Desktop 上收集和分析了 2000—2004 年的超过 1 万个文件系统的元数据访问快照，并发现某些文件类型在流行度、名字空间的使用方式以及文件大小变化上都具有很强的时间趋向性。类似的现象在 Leung 等[5]的工作中也有所发现，借助从 NapApp 得来的跟踪数据，Leung 等从两个角度分析了元数据在大规模存储系统中被搜索的模式。从用户角度上看，对元数据搜索的行为主要体现在以下几个方面：第一，为了结果的精细化，95% 以上的搜索是针对多个元数据属性的；第二，大约 33% 的搜索局限在名字空间的某一个相关区域，体现了文件通常是按语义组织存储的特点；第三，用户认为很重要的大约 25% 的搜索涉及元数据的多个版本，这主要来源于用户为了解文件访问趋势而进行的“back-in-time”搜索。从被访问的元数据角度上看，其特点也反映了用户搜索行为的特征：

(1)空间局部性：指文件的属性值是在名字空间中聚集的。例如 John 的文件大多数位于/home/john 子目录树下，而不是散落在整个名字空间中。

(2)偏斜性：元数据的值具有高度的偏斜性，即这些值的分布不是对称的，一些非常热门的元数据值占据了值空间的大部分。例如，80% 的文件具有最普遍的 20 个 ext 和 size，这些现象

在其他文献中也有记载[4,6]。

2015 年，Xiao 等[7]在包括云存储在内的更广泛的存储平台上，对文件系统名字空间的结构和操作分布的偏斜性进行了更精细的分析和刻画。他们发现文件系统名字空间针对目录的大小显示出一种重尾分布，表现为众多小目录中掺杂着少数相对大的目录。例如， 在 60 个考察的文件系统中，含有目录项数目少于 128 的目录占 90%。此外，另一个有价值的发现是大目录随存储系统容量的增长而持续增长。除目录文件外，普通文件大小也显示出类似的分布。在所考察的系统中，有 64% 中位文件的大小小于 64 KB，而最大的文件却可以达到几个太字节，这说明系统中占大多数的是小于几百千字节的小文件。这个观察甚至在针对大数据处理的大规模集群文件系统中也是成立的[8]。至于目录树的深度，与目录的大小情况不同，它并不随系统容量的增长而增长。在考察的系统中，绝大多数(约 90%)目录树的深度不超过 16 级。这个现象在其他研究中也有类似发现[9,10]。与 Leung 等[5]所关心的元数据值的偏斜性不同，Xiao 等[7]研究了针对元数据操作的偏斜性，他们发现在所有考察的 trace 文件中，只有一两个操作占据了主导地位：open 操作第一，下一个是 readdir 操作。

由于元数据访问的性能是整个文件系统设计和实现的关键所在[11]，因此，对元数据及其访问特点的探索，一方面加深了对元数据管理的理解，另一方面也驱动了与元数据服务相关的研究。

1.2　元数据管理架构

传统上，元数据和数据被置于同一个文件系统管理之下。从访问效率上考虑，元数据通常在物理上与其描述的数据相近存储[12]。与传统结构不同，当下主流的大规模分布文件系统基本上都是采用将元数据与数据的管理相分离的结构(如图 1)。该结构主要包括三个部分：客户端文件系统、元数据服务系统(Metadata Server，MDS)和数据存储设备系统(Data Storage Device，DSD)。元数据服务系统一般采用单一元数据服务器或元服务器集群这两种架构来维护文件系统全局名字空间以及文件数据存储的物理视图，这些元数据信息以带外方式向客户端提供服务。同时 MDS 系统还兼管理数据存储设备。访问数据的客户

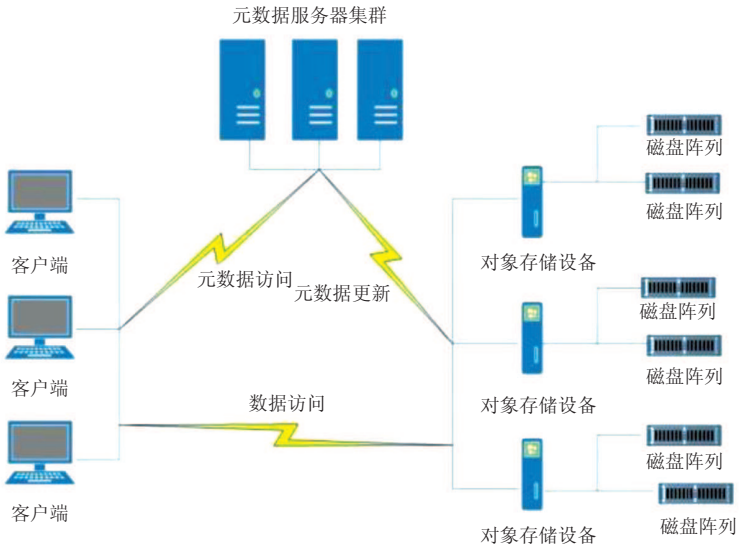

图 1　存储系统架构：MDS、DSD 与客户端之间的交互

Fig. 1　Storage system framework: MDS, DSD and interaction with clients

端首先与 MDS 通信获得欲访问数据的元数据，然后据此与相应的存有欲访问数据块的 DSD 通信，数据的传输只发生在客户端和数据存储设备之间。此架构有效地减轻了 MDS 的访问压力，通过横向扩展可以充分利用 DSD 集群系统的聚合 I/O 带宽，提高 I/O 系统的整体性能。

1.3 元数据管理的挑战

早期的分布式文件系统包括 GFS[1]、Hadoop Distribute File System（HDFS）[13]、Lustre[14]和 Blue Whale Cluster File System（BWFS）[15]等均采用了单一 MDS 架构。在数据规模较小或特定的应用环境下，单一的 MDS 在减小元数据访问的通信代价以及维护元数据的一致性开销等方面显示出了优势，但随着云计算和大数据应用对存储系统的规模、性能以及可用性方面带来的压力不断增大，单一 MDS 架构显然不能满足大数据处理的诸多要求。如在 PB 级的海量存储系统中，元数据的数据量级可以达到 TB 级，这要求元数据服务系统具有很高的可扩展性。另外，元数据往往是高度共享、高并发访问的，对频发的元数据请求的及时响应在很多应用场合是至关重要的[16]，这又要求元数据服务系统具有很高的性能。最后，作为文件访问必经环节，元数据的访问处于关键路径之上，单一 MDS 带来的单点失效问题使服务的可用性大打折扣，而可用性是提供数据服务的基本保证。考虑到这些因素，采用 MDS 集群技术是保证元数据服务高可扩展、高性能以及高可用的一个自然选择。业内众多企业和研究机构均基于 MDS 集群架构，针对元数据服务对高扩展、高性能和高可用等需求展开了广泛而深入的研究。本文将对近年来这方面的进展进行一个概述和梳理，并分析存在的问题以及对未来的发展做一些展望，希望能对相关研究提供一个有价值的参考。值得指出的是，本文并不是穷尽相关技术的一个总结，有些热点话题，如针对新型存储介质（如固态硬盘和变相存储器[17]等）优化元数据管理的技术并未在讨论范围之内。因为这方面的内容本身可以单独进行总结，有兴趣的读者可以参阅相关文献[18-20]。

2 元数据服务的高可扩展技术

高可扩展性是采用 MDS 集群架构的一个主要目的，也是近年来分布式文件系统的一个热点问题，其主要的挑战在于如何将文件系统的整个名字空间划分成片并在多个元数据服务器中均匀分布，同时能够有效应对访问负载的变化和元服务器的弹性增减等问题，最终实现元数据服务的高可扩展性。元数据的高可扩展技术可以从很多角度来分类，本文从元数据的组织查询的角度将现有的技术分为静态空间划分方法和动态空间划分方法分别加以概述和比较。

2.1 静态空间划分法

所谓静态空间划分法指的是对元数据空间的划分只依赖其结构相关的一些信息，如路径名、子树大小等，而不考虑实际负载的变化。子树划分法[21-23]和散列函数映射法[14]是两个常用的基本方法。针对不同的环境和目的，在这两个方法的基础上还产生了一些变形[24-26]。

2.1.1 静态子划分树法

静态子树法是一种比较简单的划分方法，常出现在早期的分布式文件系统中，如 NFS、AFS[27]和 Coda[28]等，以及近来的一些大规模并行和分布式文件系统中，如 Hadoop Federated HDFS[29]、PVFS[30,31]和 Lustre[14]等。该方法一般需要系统管理员来决定如何将目录的层次结构进行划分并将每一个分片（通常是一棵子树）赋给指定的元数据服务器。静态子树法的优点是文件的元数据通常只需要访问相对较少的服务器即可获得。如果一个目录被反复地访问，这种方式的效率还是比较高的。但这种方式的一个明显缺点是不能动态地平衡不均衡的元数据工作负载，也即

不能很好地处理不均匀的访问引起的“热点”问题。例如，当一个目录子树下的一组文件被密集访问的时候，其所在的服务器就会异常忙碌，有可能成为整个系统的性能瓶颈。为了解决这个问题，子树复制在某些情况下会起到一定的缓解作用，但也同时增加了存储成本和维护副本一致性的代价。此外，这种划分方法对元数据服务器的增减处理也比较麻烦，因为对名字空间划分要重新规划，有些子树要从一个元服务器迁移到另一个，会引起较高的性能损失。

2.1.2 散列函数映射法

基于散列函数映射的方法是将散列函数应用于文件的路径名或文件名去定位文件的元数据服务器。这种方式不但可以减轻元数据服务器间工作负载不平衡，将服务请求均匀地分布在 MDS 集群中，而且可以使客户端直接获得所需的元数据。如 Vesta[32]和 zFS[33]利用了对文件路径名的散列来定位数据和服务器。这种方法的主要问题是元数据访问的局部性无法得以很好地保持。另外，路径名的修改会导致大量的文件元数据在 MDS 集群中迁移，增大了网络负载。沿路径查询的访问控制也会由于被查询文件的前缀目录存储在不同的服务器上而引起大量的网络开销。还有，与静态散列方法类似，当服务器数量发生变化时，散列函数可能跟着变化，由此也导致了大量元数据在网络中移动。

目前，针对这些问题，文献[34,35]提出了一些改进措施。与传统基于文件路径名散列的方法不同，Xu 等[34]建议使用能够保持局部性的散列函数以便将一个嵌套点下的整个目录子树分配到同一个 MDS 服务器上。为此，他们利用 simhash[36,37]设计了一个新的保持局部性的散列函数，基本想法是针对路径上每个目录子项进行散列(2 字节编码)，然后同文件 inode 的散列值连接起来，形成一个具有 48 字节的编码作为最终的散列值，而不是散列整个路径名。除保持局部性外，这种方式也可以很方便地命名新文件和目录，对于从一个目录迁移到另一个目录的文件，其散列值可以快速地改变以反映这种迁移。

为有效避免因为目录属性修改而导致的大量数据更新与迁移，刘仲等[35]采取了另外一种策略，提出了将目录路径属性与目录对象分离的元数据管理方法。这种管理办法的优势在于不仅可以减少前缀目录的重叠缓存，提高元数据服务器 Cache 的利用率和命中率，而且可以开发目录的存储局部性，减少磁盘 I/O 次数。这种方式的问题是增加了访问目录索引服务器的额外通信开销，也影响了整个系统的可用性。

2.1.3 子树散列混合法

子树划分法保持了对元数据访问局部性的支持，但不能有效地处理存储负载的平衡问题。与此相反，散列函数映射法能达到较好的负载平衡，但却以牺牲局部性支持为代价。将这两者集成起来取各自的优势是子树散列混合法的一个共同特征。混合法通常利用散列函数实现元数据在 MDS 服务器间的均匀分布，同时基于层次目录结构支持目录和文件的语义查询与控制。

Brandt 等[24]基于这个想法提出了一种名为懒惰更新的混合策略(Lazy Hybrid，LH)。该方法一方面利用元数据查询表结构将路径的散列值与元数据的位置解耦合，实现服务器资源的弹性伸缩；另一方面利用一个双重入口的访问控制表避免了针对许可权限的操作。此外，LH 的一个独到地方是借助混合机制，实现了将元数据的迁移一直推迟到其被修改后的第一次访问，从而避免了 MDS 服务器间的突发网络流量。尽管 LH 有诸多优点，但还是不能减少或消除大量的元数据迁移带来的网络开销。

Xu 等[34]针对 EB 级规模存储系统的元数据管理也提出了一个类似的系统 DROP(Dynamic Ring Online Partitioning)，一方面 DROP 用散列函数去分布元数据，另一方面 DROP 通过维护全

局目录信息以完成依赖于目录属性的操作，如读写许可权的判断。如上文所述，为了实现这些优势，他们采用了一种称为局部保持的散列函数，同时实现了一个基于直方图的动态负载平衡策略以克服散列函数引起的负载不平衡。其思想很简单，就是每个服务器保存一个记录本地访问范围密度的直方图，然后以心跳协议的方式在系统范围内交换直方图信息。通过承载若干虚拟节点，每个 MDS 服务器会负责散列值的某个范围区间。重负载的 MDS 通过迁移虚拟节点到轻负载的 MDS 实现全局负载平衡。但是相比单纯的元数据迁移，DROP 的邻接节点探测和虚拟节点迁移使负载平衡的网络代价过高。

2.2 动态空间划分法

顾名思义，相对于静态空间的划分方法，动态空间划分法就是将全局名字空间分配到不同的元数据服务器上，并且当工作负载发生变化的时候，利用动态负载平衡机制在 MDS 集群间以不同的粒度重新分布元数据。与前述方法相比，动态空间划分法在维持访问局部性、自适应负载变化以及灵活利用资源上来说，都展现出了相对优势，因而在业界和学术界都引起了广泛兴趣。现在主流的大规模分布式文件系统，诸如 Ceph[38]、GFSII[39]、GlusterFS[1][40]以及 Luster 2.0[41]等，均以各种形式支持动态元数据的管理。针对动态子树划分法，研究的重点主要集中在负载平衡技术以及相关的资源弹性化等问题上。

2.2.1 动态子树法

负载平衡要解决的问题在于如何把重负载节点的工作负荷迁移到轻负载的节点。在这个过程中，为保持访问的空间局部性，通常以子树为最小单位进行负载平衡。Ceph 就是利用动态子树划分来实现负载平衡的一个典型。每一个 MDS 服务器使用指数时间衰减的计数器测量目录结构中元数据的访问的频度，任何针对某个元数据的操作都会增加从根到该元数据节点路径上所有目录节点的访问计数，从而为每一个 MDS 服务器提供了一个加权目录树来刻画最近的元数据负载分布。MDS 服务器间负载值的比较是通过周期性地护送心跳信息来实现的，重负载的节点可以将识别出的某个子树负载迁移到轻负载的节点，同时将新节点变为该子树的管理职权节点。在 Ceph 中，某个目录项的职权节点是由目录项的路径名或其 inode 号的散列值定义的，随着目录的增大或访问频度的增高，可以将其散列到其他节点。相反，随着目录的收缩或访问频度的降低，也可以将一个目录子树的不同部分从多个服务器聚合到一个服务器，为实现资源的弹性管理提供支持。

蓝鲸元数据管理系统（Blue Whale Metadata Management System，BWMMS）[21]是中国科学院计算技术研究所面向 EB 级存储而开发的 MDS 集群技术，在集成 pNFS[42]和蓝鲸块设备文件系统 EXFS[43]的基础上，实现了以目录为粒度的元数据分布策略。与 Ceph 中的加权子树法有所不同，BWMMS 将全局命名空间进行分区，并为每个分区规定了存储目录的总数，一旦分区被填满，该分区的新建目录都将分布到其他元数据子卷上。元数据子卷的选择以 Round-Robin 的方式进行，以此将元数据分散到各个 MDS 上。这个思想与 GPFS 处理大目录的可扩散列函数的方法有些类似，区别是对满分区里的内容并没有拆分。BWMMS 没有采用散列路径名的方法定位 MDS 服务器，而是在内存中维护一个 MDS Map 的数据结构存储元数据子卷与 MDS 的映射关系，这种方式虽做到了定位与路径无关，但也引入了对 MDS Map 集中控制所带来的问题。BWMMS 的另一个问题是名字空间的划分只考虑了目录项大小，没有考虑目录项的访问频度以及由此引起的访问热点问题。

2.2.2 目录项子集法

与上文提及的系统不同，针对元数据访问

特点，IndexFS[25]对 MDS 集群采用了层次化的结构设计，将元数据与小文件进行统一处理，并考虑了它们核外的存储优化。每一个目录建立在一个随机初选服务器上，其下所有文件的目录项也将存储在同一个服务器上。对于不断增长的大目录，当它的大小超过了预定的阈值，就会以增量的方式不断地分裂，分裂出的目录子集随机地存储在一个备选元服务器中。为了保证负载平衡，备选元服务器的选择是依据“Power of Two Choices”的原理[44]完成的，即通过探测两个随机选择的服务器，把目录子集放到存有相对较少目录的服务器中。由此可知，与基于子树划分的系统不同，IndexFS 系统的元数据是以一个个目录项子集的形式在元服务器中进行分布式存储。对于后台的存储优化，IndexFS 利用目前 Key-Value Store 系统常采用的 Log-Structured Merge 树[45]技术实现元数据和小文件数据的表示和存储，每一个服务器存储和管理文件系统元数据的一部分，并利用 LevelDB 把元数据和小文件数据打包成一个扁平大文件，然后以 SSTable[46]的形式写入磁盘，存储在共享群集文件系统中。LevelDB 提供了一个简单的 Key-Value Store 接口支持对元数据点与范围的查询。尽管 IndexFS 提高了元数据访问的可扩展性，但研究表明这种方式太过依赖于利用 Cache 的局部性去减少路径查询的代价[7]，因此，在 Cache 更新时会引起负载的不平衡，不太适合于对路径访问比较发散的情况。

2.2.3 动态散列法

GPFS 是 IBM 为共享存储而设计的一个并行文件系统[47]，其利用可扩展散列函数的方法在块级实现了目录组织的动态扩展。目录中的目录项存储在磁盘块上，磁盘块由目录名字的散列值低端 n bit 位来寻址，n 的大小取决于目录的大小到底占多少个磁盘块。具体来说，当对一个目录磁盘块建立新的目录项等到它满了的时候，该磁盘块一分为二，新磁盘块的逻辑块号通过在老块号的第 $n+1$ bit 位置“1”来实现，而其目录项则是由老磁盘块中那些第 $n+1$ bit 位的散列值为“1”的目录项迁移而得，此时老盘块的逻辑块号的第 $n+1$ bit 位为“0”，其目录项也正好由剩下的那些第 $n+1$ bit 位散列值为“0”的目录项组成。这样，无论目录文件的大小和结构怎么样，一次查询通常只要求对一个目录块查询。由于是共享存储，所有节点均可参加包括名字空间在内的元数据管理，元数据的一致性是由节点间令牌控制实现的。黄斌等[48]基于同样的想法，将可扩散列函数应用于元数据集群服务器的编号上现负载在弹性服务器资源中的平衡。但该方式由于负载分裂的随机性，不能很好地支持元数据访问的局部性。在 Hua 等[49]提出的以组概念为基础的层次型元数据管理方法中，组的分裂与合并也采取了类似方法，但只考虑了存储空间的平衡，对访问热点问题考虑不够。

2.3　小结

表 1 针对静态空间法和动态空间法在访问热

表 1　静态空间法和动态空间法对比表

Table 1　Comparison between static namespace method and dynamic namespace method

评价指标	静态空间			动态空间		
	静态子树	散列函数	子树散列	动态子树	目录项子树	动态散列
访问热点	有	无	无	有	有	有
负载均衡	不好	好	好	好	好	好
局部性	好	不好	好	好	好	不好
资源弹性	代价高	代价高	代价高	好	代价高	代价高
名字空间变化	代价高	代价高	代价高	代价高	代价高	代价高

点、负载均衡、对访问局部性的支持、资源弹性变化以及名字空间改变的代价等五个方面进行了一个对比总结。从这个表中可以看到目前的技术在适应资源弹性和名字空间变化等方面，开销均比较大，而对其他方面，动态空间法普遍要好于静态空间法。

3 元数据服务的高性能技术

我们把元数据服务的高性能技术大体上分为三类：一是利用元数据服务的可扩展性技术，如负载平衡、热点消除等，优化元数据的服务，从而达到提高整个文件系统的性能；二是在现有元数据的属性集之上，针对各种查询应用，建立新的索引机制，提高查询应用的效率；第三种技术更进一步，我们称为元数据的增值技术，其核心思想是针对不同的应用，扩展元数据集，扩展的元数据集称为增值元数据集，通过利用面向应用的增值元数据服务，提升特定的应用程序性能。

3.1 缓存与复制技术

缓存与复制技术是实现元数据服务高可扩展性的两个关键技术，其对于提高整个文件系统的性能也起着关键作用。如在 Ceph 中，为了消除访问热点，实现负载平衡，研究采取了职权节点和合作缓存的手段。职权节点是为每一元数据项定义的用以实现在某些时候对元数据的串行访问、记录变化(写入外存)，并且当存在多个副本的时候维护 Cache 的一致性(consistency 和 coherence)。合作缓存则要求当一个元数据项在另一个 MDS 的 Cache 中存有副本时，职权节点有权利与其通信用以维持 Cache 的(同址)一致性(coherence)。如果一个非职权 MDS 的 Cache 副本被移除，也要通知职权节点移除它的本地副本，以实现 Cache 的(异址)一致性(consistency)。职权节点的概念将元数据控制分散化，是实现副本技术的基础。而合作缓存则维持了副本的一致性。

正如在 Ceph 中显示的那样，缓存和复制技术是实现负载平衡、降低延迟提高系统性能的两个基本手段。在分布式的元数据管理中，对路径名查找的性能通常被认为是限制系统扩展性和影响整个系统性能的一个主要因素。Xiao 等[7]通过比较 ShardFS 和上文提到的 IndexFS 对该问题进行了深入的量化研究。为了提高元数据的访问性能，IndexFS 采用了在客户端对路径名的各个组成部分和访问权限维护一个一致性 Lookup 缓存，并通过为每一个目录项设置一个租约(lease)的方法来减少由于路径名的更改而产生的大量失效信息，从而提高元数据查询服务的性能。与此不同，ShardFS 利用在 MDS 集群中复制路径的查询状态的办法，确保每一个文件操作只作用于一个节点，不再需要基于锁的针对多个服务器的访问。此外，通过对元数据操作进行分类，特异化分布式事务的实现，进一步减少复制的代价。这两种方式除了各自具有的优点之外，彼此也存在着各自的问题。在 IndexFS 中，由于访问的某些目录项不在 Cache 中而引起 Cache 失效，客户端因此会有较高的查询延迟；而在 ShardFS 中，客户端的高延迟主要是由目录元数据的更改操作而引起的多副本一致性开销。

Xiao 等[7]的研究发现，如果改变目录查询状态的操作在元数据的所有操作中占有一个固定的份额，ShardFS 的服务器端复制方式没有 IndexFS 客户端的缓存方式在可扩展性和性能等方面来得有效。但如果改变目录查询状态的操作正比于作业的个数，则服务器端复制方式较客户端的缓存方式具有更好的线性可扩展性。如何根据工作负载特征的变化，将两种方式结合起来取得最优的性能是一个值得进一步研究的问题。

3.2 元数据检索技术

在许多情况下，元数据的操作不仅仅是通过目录查询来定位文件并进行访问控制，而是多种

多样的，不但包括对点、范围以及 top-k 这样的检索，还包括像文件属性集上的聚合检索，用来满足一些特定的查询要求，如某个文件在系统中有多少个副本、哪些文件消耗了最多的空间等。对于这些问题，前述技术由于缺少元数据属性集上的索引只能实现蛮力搜索，均不能有效地满足这些要求。高效的元数据索引通常围绕我们对元数据访问的特点及元数据本身的特点这两方面知识的了解程度而设计。Spyglass 针对前述有关元数据以及其访问特点提出了两个主要技术用以实现快速、可扩展的元数据检索服务：层次分割法和分区版本控制法。

首先，利用层次分割法将文件系统的目录空间按子树进行分区，并以 K-D 树[50]的形式给出每个分区负责索引的一个或多个目录子树。分区索引存储了分区中文件的元数据，而分区自身元数据则保留了分区信息以及指向子分区的指针。其中分区信息包括了用来判断该分区是否有与检索条件相关的信息以及支持分区版本控制的信息。前者是通过比较代表文件内容的文件指纹信息来实现的。由此可知，索引后的分区构成了一棵树，称为 Spyglass 索引树，每一个索引分区顺序地存储在磁盘上(如图 2 所示)，这种分区及其存储方式保留了对文件系统访问具有空间局部性的支持，但同时也存在一个问题——如何适应分区里文件元数据增、删、改的变化。为此，Spyglass 采用了版本控制的方法而不是现场修改有关元数据信息的方法来实现。具体来说，Sypglass 每隔一段时间(文件系统参数)批量处理对分区中文件元数据的修改，修改后的分区作为一个新的索引版本并以相对老版本的增量形式进行存储。同时老版本也保留下来作为对用户“back-in-time”检索和趋势查询等的支持。

与传统的基于数据库管理系统建立文件元数据索引的方法相比，Sypglass 提出的方法不但检索的速度提高了 8～44 倍，而且存储空间也减少了 5～8 倍[51]。但无疑，这种索引技术也对元数据的内存容量和组织机构带来了一定的负担和开销，如何与前述的负载平衡技术有机地结合可能是一个复杂但值得探索的问题。

针对元数据的检索，Hua 等[49]也提出一个类似的基于组间层次划分的方法，称作 G-HBA(Group-based Hierarchical Bloom Filter Array)。与 Sypglass 针对目录空间进行子树分区不同，G-HBA 将 MDS 服务器组织成一个个逻辑组，组中每一个 MSD 服务器上维护一组 Bloom

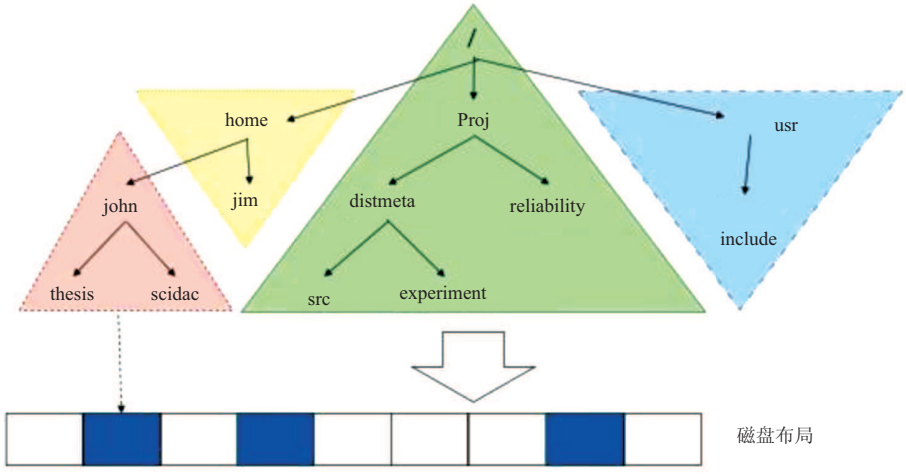

图 2　基于子树的名字空间分解和存储

Fig. 2　Sub-tree based Namespace decom-position on and storage

滤波器，代表了其元数据存于此服务器的所有文件，该服务器也称为这些文件的主服务器。每一个维护的 Bloom 滤波器组定期地向所有其他服务器传递复制信息。这样，每一个服务器维护了系统中一部分文件到其主服务器的映射。而 G-HBA 要求每一个组大致维护了系统中所有文件到其主服务器的映射。针对 G-HBA 的元数据查询与传统的基于路径名散列的方法不同，查询可以从任意一个 MDS 服务器针对 Bloom 滤波器检测开始，经过本地、组内，最后到组间三个层次逐级进行。组间的组织形式与 Sypglass 中沿目录树划分分区的层次方式不同，它没有固定的结构。这种方式很容易实现存储空间的负载平衡，但组内、组间查询不得不使用多播或组播的方法，增加了网络通信负载。另外，如前所述，访问的偏斜性引起的热点问题也是一个值得考虑的因素。

Xu 等[52]针对大数据高性能计算环境下的元数据检索给出了一个较详细的讨论，并从索引的可管理性、可扩展性、性能以及 POSEX 接口支持等几个方面，对包括前述技术在内的现有元数据索引结果进行了归纳，提出了附加模型和数据库模型的分类，同时对它们进行了比较。在此基础上结合文件系统和数据库的优势，提出了一个文件系统级别的元数据管理系统 VSFS（Versatile Searchable File System）。VSFS 支持透明的名字空间查询和灵活多样的文件索引。然而提出的系统的单主节点结构并不具有大规模可扩展性。

3.3 元数据的增值技术

所谓元数据的增值技术就是在现有分布式文件系统元数据管理的基础之上，集成面向各类应用的特定元数据，并与前述索引技术相结合，有针对性地提高某类应用的计算功能和性能。元数据增值技术的覆盖范围比较广，目的各异，本节只专注于其对高性能应用的几个影响。

xStor[53]是这方面的一个尝试，它是中国科学院深圳先进技术研究院设计和开发的一个支持超算与交互服务的混合云存储系统，打破了传统的“One Size Fits All”的文件系统的使用模式，将若干个主流分布式文件系统通过统一的界面整合在一起，对不同的应用提供个性化的支持。因此，其元数据的管理不是针对单一的文件系统，而是针对多个共生分布式文件系统，如 HDFS[13]、Ceph[38]和 GlusterFS[40]，以适应不同应用的不同工作负载的特点，充分发挥了各自文件系统的优势，从整体上提高了应用程序的性能。另外，在不同的文件系统间 xStor 还实现了存储资源的弹性分配，最大化了资源利用率。xStor 虽然通过系统集成的方式提供了一定的灵活性，但其对工作负载的自适应性做得还不充分，所以目前只能针对具体的应用选择一个文件系统来使用，这在某种程度上抵消了其带来的灵活性优势。

为了挖掘科学计算中一些将要废弃的中间结果的价值，Wang 等[54]以现有的分布式文件系统中元数据管理为基础，引进了对计算的中间结果数据文件在结构、语义等方面的描述信息，以增值元数据的形式，加以存储和管理，最终实现了对原本废弃的文件的跟踪和挖掘，使其针对后续的计算过程能够再生重用，最大程度地删除了后续计算过程中的一些冗余计算，相应地提高了复杂科学计算的性能。但这种单纯的删冗只体现了增值元数据功效的一个方面，其实质是实现了工作流计算在存储空间和性能上的一个折衷。另一个方面是如何利用增值元数据记录和挖掘工作流的并发度。

为解决这个问题，以上述研究为基础，Wang 等[55,56]进一步设计和实现了一个面向科学工作感知的文件系统——工作流感知的文件系统（Workflow-aware File System，WaFS），将计算过程中文件之间的读写依赖关系以数据流的形式记录于内存数据库中，这部分增值的元数据克

服了现有文件系统中元数据对数据文件之间依赖关系记录的缺失。工作流调度器对这种依赖关系的挖掘和利用是提高工作流任务的并发度，最大化资源利用率的一个有效手段。尽管有此优势，WaFS 仍局限于对工作流计算的支持，并只对以控制流刻画的工作流具有效果，因为这种情况下要挖掘的数据流隐含在控制流之中了。

这方面的其他工作还包括 Zhao等[57]在 FusionFS 文件系统中针对元数据密集型操作提出的分布式存储中间层技术和 Dong 等[58]提出的“rich metadata”概念，其不仅记录了高性能计算中相关文件实体的一些属性，而且通过提出的通用属性图集成进了它们之间诸如世代关系之类的联系，丰富了高性能计算中针对元数据的各种查询。在概念上“rich metadata”与 WaFS 中提出的数据流依赖关系可互为补充，文件的世代关系与数据依赖关系在纵横两个纬度刻画了计算过程中文件间的逻辑关系，这是传统文件系统所做不到的。

4　元数据服务的高可用技术

由于规模逐渐增大，MDS 一直面临着潜在发生错误的风险。为保证 MDS 提供不间断的元数据服务，要求 MDS 集群具有高可用性。在元数据服务的高可扩展技术中提到过，在共享集群文件系统中，会将分解出来目录子集，根据探测，随机选出数个备选 MDS 服务器，再将目录子集存储在若干目录较少的 MDS 上，以防止意外故障而使得元数据丢失，从而加强元数据服务的可用性。

4.1　基于副本的高可用元数据服务

分布式管理元数据通过多台 MDS 协同管理元数据，克服了集中式管理元数据单点故障的缺点，将元数据的可用性提高。虽然少量 MDS 出现故障时，不会导致整个系统不可用，但还是一定程度上会影响系统的正常访问。为实现 MDS 集群的更高可用性，可以通过元数据冗余来实现。根据集群的多节点的优势，将每个 MDS 上的元数据备份到其他的 MDS 上。Hadoop 和 GFS 均采用了基于副本的元数据管理。考虑到元数据恢复时间，可以将元数据备份分为重启恢复元数据方式和启动热备份节点方式。重启恢复元数据方式主要是定期将该 MDS 上元数据备份到其他的 MDS 上，并设置检查点。当该 MDS 出现故障时，可以启用备用 MDS 加载最新检查点下的元数据备份，从而恢复该点的元数据服务。该方式的优点在于开发简单方便，但存在的最大隐患是元数据的不一致性，在检查点之后的元数据更新信息并没有恢复。为解决这个隐患，通常会结合日志方法，使得备用 MDS 恢复的元数据为最新版本。但是重启和恢复仍需要一定的切换时间，无法做到实时恢复。

重启恢复元数据方式类似，启动热备份节点方式也是为 MDS 选用一个备用 MDS，两个元数据服务器均可访问共享存储设备上的目录，该目录存储了对命名空间等更新的日志。备用元数据服务器将会检测该日志，将所有日志记录更新到自己服务器上，并且数据存储服务器的数据定位信息和心跳检测信息发送给两个元数据服务器，从而可以实时更新相应的信息。一旦有 MDS 出现故障，就会有备用 MDS 代替。Hadoop 的元数据管理就是基于这种高可用性原理。而在 GFS 中，备用 MDS 在一些情况下，可以为一些文件提供只读访问。如果在 MDS 上引入高可用模块，向上为系统提供元数据服务，向下屏蔽故障信息，使得 MDS 出现故障的时候元数据请求会自动切换到其他备选元数据服务器上，从而实现对客户高透明和高可用。这种方式可以保证较好的数据一致性和较短的切换时间，但是增加了一定的复杂度，并且给系统的维护上带来了不小的压力。

Farsite[59]和 Archipelago[60]均使用了基于副本的元数据管理。不过 Archipelago 是将集群中的节点分成若干小岛，将元数据复制到各个小岛上，通过这种冗余方式来提供不间断的元数据服务。而 Farsite 将元数据复制到若干节点上，为了减少复制所带来的开销，将元数据的修改操作记录到日志中，并定期将它们复制到其他 MDS 上。稍近一些的采用类似技术的工作还包括 Chen 等[61]针对曙光集群文件系统 DCFS3 设计的高可用 MDS 服务器，其核心思想是利用提出的 Packed Multi-Paxos 降低延迟，减小由多副本引起的一致性协议的开销。

4.2 基于日志的高可用元数据服务

基于副本的元数据服务不可避免的一个问题就是延迟，而且为了减少元数据的磁盘 I/O，通常使用 Cache 来缓存部分元数据操作，一旦 MDS 宕机，那么缓存的元数据操作将会丢失。为提高元数据服务器的可用性，需要文件系统能够持久化所有的元数据修改操作，这就需要借助日志方法，记录每个 MDS 的元数据修改操作，即使在缓存 Cache 中的数据丢失的情况下，依然能够根据日志恢复元数据。

在共享式集群中，元数据和日志都存储在基于磁盘阵列（如存储区域网）的共享存储上。为维护元数据的一致性，GPFS 使用了分布式锁实现了多用户对元数据的同步访问，并指出分布式锁比集中管理有更好的并发性[47]。GPFS 元数据的更新操作通过预写日志机制（Write-Ahead Logging）来实现文件系统的一致性更新。虽然共享式集群在维持元数据的一致性上比较简单，但是同样会面临“热点”问题，并且无法应对跨数据中心的广域网（Wide Area Network）环境下不稳定且高延迟的网络传输情况[62]。

在无共享集群中，元数据分布在多个独立 MDS 上，拥有多个副本，修改操作很容易涉及到多个 MDS 上的元数据，元数据的一致性实现变得更为复杂。为设计一个能够应对跨数据中心的广域网环境下的元数据集群，根据 CalvinFS[62]的思想，元数据的修改加入了分布式事务逻辑，并引入调度器进行锁管理，通过 Paxos 协议[63]同步 MDS 集群的元数据日志。普通的 Paxos 协议每同步一条元数据需要经历两个阶段，而中国科学院大学在研究基于副本的高可用元数据管理中，使用 Multi-Paxos 协议[64]同步日志，减少第一阶段的次数，从而减少了网络开销。虽然 Google 的 Chubby 采用了这种基于 Paxos 协议同步日志的元数据管理方法，但众所周知，Pasox 协议很复杂，实现很困难。相比之下，Raft[65]一致性协议更为容易理解，实现也更为简单，在 Cloudera 开发的大型开源存储引擎 Kudu[66]中，Tablet 副本管理就是通过 Raft 协议实现的。

虽然元数据和数据的分离使得读写逻辑更为清晰，但是增加了需要访问磁盘的次数。由于每条元数据占用的存储空间很小，将数据块中的数据进行适当压缩，节省小部分空间，并把和数据块相关的文件系统元数据嵌入到节省出来的那部分空间，从而可以一次写入数据和元数据，减少磁盘 I/O，Selfie[67]正是利用了该原理。在这种情况下，即使整个 MDS 集群崩溃了，也可以通过扫描数据块，快速重构出文件系统的元数据，为系统提供服务。相比于将元数据复制到其他 MDS 上，这种方式可以减少更多的网络开销和额外的同步操作。

5 未来发展趋势

元数据的应用并不仅仅限于诸如文件的有效定位和权限控制等系统级别的应用，其应用的方式可以是多视角、全方位的。如多重属性查询，历史纪录查询以及多版本查询等。为了提高对元数据各种检索查询服务的性能，高效可扩展的元数据索引技术的研究仍然是一个值得关注的方向。

此外，在实际应用中，元数据管理的内容也不应仅仅限于系统级别的信息，元数据的增值技术针对特定的应用仍然具有广阔的发展空间。如具有压缩、删冗功能的文件系统，具有文件依赖关系纪录的文件系统，以及具有加密安全功能的文件系统等，其元数据的管理在保证预期的功能条件下，如何实现高效可扩展性是其大规模部署应用的一个重要前提，值得深入研究和探索。

最后，随着大数据时代的到来，数据量呈爆炸性增长，PB 级的文件系统已逐渐不能满足在存储容量上的需求。目前大规模的文件系统主要针对 EB 级的数据量而设计。在这种情况下，元数据量的管理也要相应地达到 PB 量级，这是以前数据量的级别。这个级别的元数据量无疑对元数据的管理技术带来了巨大挑战。除了以上列举的技术外，无元数据服务模型值得进一步探索。理想情况下，这种模型可以显著提高系统的扩展性，使得系统在并发性和性能等方面实现线性扩展和增长。遗憾的是这方面的研究还鲜有成效，GlusterFS[40]是一个典型的代表，但目前也仅仅限于无元服务器下的文件定位功能，远未达到完全移除元服务器而有效实现所有针对文件系统操作的目的。我们认为主要的障碍可能在于复杂的数据一致性问题和诸如低效的文件目录遍历等操作。另外，无元服务器模型也使系统对全局监管的能力大打折扣，同时相应地增加了客户端的工作负载(如文件定位要求的计算)。如何有效地解决这些问题都是值得进一步研究的内容。

6　总　结

本文针对大规模分布式系统中的元数据管理，从高扩展、高性能和高可用三个方面对当前的研究进展进行了归纳和梳理，综合比较了各种技术的优劣，并对未来的发展方向进行了探讨和评述。从文中可以看出，相对于高性能和高可用，高可扩展性仍是最重要的一个方面，也是研究最多的一个方面。研究工作主要集中在如何平衡名字空间访问负载以及如何实现资源弹性等方面，相应的技术在一定程度上可扩展到其他两个方面。现有的技术虽然从某种程度上实现了可扩展性，但面对大数据的急剧增长，相对于需求仍存在巨大的鸿沟和挑战，这迫使我们对元数据的管理进行更深层次、更多视角的考虑，从而发展出适应时代需求技术架构。

参 考 文 献

[1] Ghemawat S, Gobioff H, Leung ST. The Google file system [C] // Proceedings of the 19th ACM Symposium on Operating Systems Principles, 2003: 29-43.

[2] Miller E, Greenan K, Leung A, et al. Reliable, efficient metadata storage and indexing using NVRAM [EB/OL]. [2015-04-21]. http://dcslab.hanyang.ac.kr/nvramos08/EthanMiller.pdf.

[3] Ousterhout JK, Costa HD, Harrison D, et al. A trace-driven analysis of the UNIX 4.2 BSD file system [C] // Proceedings of the 10th ACM Symposium on Operating Systems Principles, 1985: 15-24.

[4] Agrawal N, Bolosky WJ, Douceur JR, et al. A five-year study of file-system metadata [J]. ACM Transactions on Storage, 2007, 3(3): 9.

[5] Leung AW, Shao ML, Bisson T, et al. Spyglass: fast, scalable metadata search for large-scale storage systems [C] // Proccedings of the 7th Conference on File and Storage Technologies, 2009: 153-166.

[6] Douceur JR, Bolosky WJ. A large-scala study of file-system contents [C] // Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, 1999: 59-70.

[7] Xiao L, Ren K, Zheng Q, et al. ShardFS vs. IndexFS: replication vs. caching strategies for distributed metadata management in cloud storage systems [C] // Proceedings of the 6th ACM Symposium on Cloud Computing, 2015: 236-249.

[8] Welch B, Noer G. Optimizing a hybrid SSD/HDD HPC storage system based on file size distributions [C] // Mass Storage Systems and Technologies, 2013: 1-12.

[9] Ellard D, Ledlie J, Malkani P, et al. Passive NFS tracing of email and research workloads [C] // Proceedings of the 2nd USENIX Conference on File and Storage Technologies, 2003: 203-216.

[10] Kavalanekar S, Worthington B, Zhang Q, et al. Characterization of storage workload traces from production windows servers [C] // IEEE International Symposium on Workload Characterization, 2008: 119-128.

[11] Alam SR, El-Harake HN, Howard K, et al. Parallel I/O and the metadata wall [C] // Proceedings of the 6th Workshop on Parallel Data Storage, 2011: 13-18.

[12] McKusick MK, Joy WN, Leffler SJ, et al. A fast file system for UNIX [J]. ACM Transactions on Computer Systems, 1984, 2(3): 181-197.

[13] Apache. HDFS [EB/OL]. [2015-04-21]. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html.

[14] OpenSFS and EOFS. Lustre [EB/OL]. [2015-04-21]. http://www.lustre.org.

[15] Liu JL, Zhang YL, Yang L, et al. Sac: exploiting stable set model to enhance cachefiles [J]. Journal of Computer Science and Technology, 2014, 29(2): 293-302.

[16] Roselli D, Lorch JR, Anderson TE. A comparison of file system workloads [C] // Proceedings of the Annual Conference on USENIX Annual Technical Conference, 2000.

[17] 冒伟, 刘景宁, 童薇, 等. 基于相变存储器的存储技术研究综述 [J]. 计算机学报, 2015, 38(5): 944-960.

[18] Chen JX, Wei QS, Chen C, et al. FSMAC: a file system metadata accelerator with non-volatile memory [C] // IEEE 29th Symposium on Mass Storage Systems and Technologies, 2013: 1-11.

[19] He SB, Sun Xh, Wang Y, et al. A heterogeneity-aware region-level data layout scheme for hybrid parallel file systems [C] // Proceedings of the 44th International Conference on Parallel Processing, 2015: 340-349.

[20] 陈卓, 熊劲, 马灿. 基于 SSD 的机群文件系统元数据存储系统 [J]. 计算机研究与发展, 2012(Z1): 269-275.

[21] 刘健, 张军伟, 邵冰清, 等. 支持 EB 级存储的元数据服务器集群系统 [J]. 中国科学: 信息科学, 2015, 45(6): 721-738.

[22] Nagle D, Serenyi D, Matthews A. The panasas activescale storage cluster: delivering scalable high bandwidth storage [C] // Proceedings of the ACM/IEEE Conference on Supercomputing, 2004.

[23] Weil SA, Pollack KT, Brandt SA, et al. Dynamic metadata management for petabyte-scale file systems [C] // Proceedings of the ACM/IEEE Conference on Supercomputing, 2004.

[24] Brandt SA, Miller EL, Long DDE, et al. Efficient metadata management in large distributed storage systems [C] // Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003: 290-298.

[25] Ren K, Zheng Q, Patil S, et al. IndexFS: scaling file system metadata performance with stateless caching and bulk insertion [C] // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2014: 237-248.

[26] Xu H, Li BH. Dynamic cloud pricing for revenue maximization [J]. IEEE Transactions on Cloud Computing, 2013, 1(2): 158-171.

[27] Morris JH, Satyanarayanan M, Conner MJ, et al. Andrew: a distributed personal computing environment [J]. Communications of the ACM, 1986, 29(3): 184-201.

[28] Satyanarayanan M, Kistler JJ, Kumar KP, et al. Coda: a highly available file system for a distributed workstation environment [J]. IEEE Transactions on Computers, 1990, 39(4): 447-459.

[29] Apache. HDFS Federation [EB/OL]. [2015-04-21]. http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/Federation.html .

[30] Clemson University. PVFS [EB/OL]. [2015-04-21]. http://www.pvfs.org/.

[31] Ross R, Latham R. Pvfs: a parallel file system [C] // Proceedings of the ACM/IEEE Conference on Supercomputing, 2006: 34.

[32] Corbett PF, Feitelson. The Vesta parallel file system [J]. ACM Transactions on Computer Systems, 1996, 14(3): 225-264.

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

[34] Xu QQ, Arumugam RV, Yong KL, el at. Efficient and scalable metadata management in eb-scale file systems [J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(11): 2840-2850.

[35] 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.

[36] Charikar MS. Similarity estimation techniques from rounding algorithms [C] // Proceedings of the 34th Annual ACM Symposium on Theory of Computing, 2002: 380-388.

[37] Manku GS, Jain A, Sarma AD. Detecting near-duplicates for web crawling [C] // Proceedings of the 16th International Conference on World Wide Web, 2007: 141-150.

[38] Weil SA, Brandt SA, Miller EL, et al. Ceph: a scalable, high-performance distributed file system [C] // Proceedings of the 7th Symposium on Operating Systems Design and Implementation, 2006: 307-320.

[39] McKusick MK, Quinlan S. GFS: evolution on fast-forward [J]. ACM Queue-File Systems, 2009, 7(7): 10.

[40] The Gluster community. Gluster Inc. [EB/OL]. [2015-04-21]. http://www.gluster.org.

[41] Palencia J, Budden R, Sullivan K. Kerberized Lustre 2.0 over the WAN [C] // Proceedings of the TeraGrid Conference, 2010: 15.

[42] Taejin K, Sam H. pNFS for everyone: an empirical study of a low-cost, highly scalable networked storage [J]. International Journal of Computer Science and Network Security, 2014, 14(3): 52-59.

[43] Yang D, Huang H, Zhang J, et al. BWFS: a distributed file system with large capacity, high throughput and high scalability [J]. Journal of Computer Research and Development, 2005, 42(6): 1028-1033.

[44] Mitzenmacher M, Richa AW, Sitaraman R. The power of two random choices: a survey of techniques and results [C] // Handbook of Randomized Computing, 2000: 255-312.

[45] Neil PQ, Cheng E, Gawlick D, et al. The log-structured merge-tree (lsm-tree) [J]. Acta Information, 1996, 33(4): 351-385.

[46] Chang F, Dean J, Ghemawat S, et al. Bigtable: a distributed storage system for structured data [C] // Proceedings of the 7th USENIX Symposium on Operating Systems Desing and Implementation, 2006.

[47] Schmuck F, Haskin R. GPFS: a shared-disk file system for large computing clusters [C] // Proceedings of the 1st USENIX Conference on File and Storage Technologies, 2002, 2: 19.

[48] 黄斌, 彭宇行, 彭小宁. 云计算环境中高效可扩展的元数据管理方法 [J]. 计算机工程与设计, 2014, 35(9): 2991-2994.

[49] Hua Y, Zhu YF, Jiang H, et al. Supporting scalable and adaptive metadata management in ultralarge-scale file systems [J]. IEEE Transactions on Parallel Distributed Systems, 2011, 22(4): 580-593.

[50] Bentley JL. Multidimensional binary search trees used for associative searching [J]. Communications of the ACM, 1975, 18(9): 509-517.

[51] Leung AW, Shao M, Bisson T, et al. High-performance metadata indexing and search in petascale data storage systems [J]. Journal of Physics: Conference Series, 2008, 125(1): 012069.

[52] Xu L, Huang ZL, Jiang H, et al. VSFS: a searchable distributed file system [C] // Proceedings of the 9th Parallel Data Storage Workshop, 2014: 25-30.

[53] 黄伟, 文高进, 洪爵, 等. Xstor: 支持超算与交互服务的混合云存储系统 [J]. 先进技术研究通报, 2011, 5(9): 60-64.

[54] Wang Y, Li H, Hu ML. Reusing garbage data for efficient computation of workflow-based workloads [J]. The Computer Journal, 2016, 58(1): 110-125.

[55] Wang Y, Lu P. Dataflow detection and applications to workflow scheduling [J]. Concurrency and Computation: Practice and Experience, 2011, 23(11): 1261-1283.
[56] Wang Y, Lu P, Kent KB. WaFS: a workflow-aware file system for effective storage utilization in the cloud [J]. IEEE Transactions on Computers, 2015, 64(9): 2716-2729.
[57] Zhao DF, Zhang Z, Zhou XB, et al. FusionFS: toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems [C] // IEEE International Conference on Big Data, 2014: 61-70.
[58] Dong D, Ross RB, Carns P, et al. Using property graphs for rich metadata management in HPC systems [C] // The 9th Parallel Data Storage Workshop, 2014: 7-12.
[59] Douceur JR, Howell J. Distributed directory service in the Farsite file system [C] // Proceedings of the 7th Symposium on Operating Systems Design and Implementation, 2006: 321-334.
[60] Ji M，Felten EW, Wang R, et al. Archipelago: an island-based file system for highly available and scalable internet services [C] // Usenix Windows Symposium, 2000.
[61] Chen Z, Xiong J, Meng D. Replication-based highly available metadata management for cluster file systems [C] // IEEE International Conference on Cluster Computing, 2010: 292-301.
[62] Thomson A, Abadi DJ. Calvinfs: consistent wan replication and scalable metadata management for distributed file systems [C] // Proceedings of the 13th USENIX Conference on File and Storage Technologies, 2015: 1-14.
[63] Lamport L. Paxos made simple [J]. ACM Sigact News, 2001, 32(4): 51-58.
[64] Lamport L. The part-time parliament [J]. ACM Transactions on Computer Systems, 1998, 16(2): 133-169.
[65] Ongaro D, Ousterhout J. In search of an understandable consensus algorithm [C] // Proceedings of the Conference on USENIX Annual Technical Conference, 2014: 305-320.
[66] Cloudera. Kudu [EB/OL]. [2015-04-21]. https://github.com/projectkudu/kudu.
[67] Wu XB, Shao ZL, Jiang S. Selfie: co-locating metadata and data to enable fast virtual block devices [C] // Proceedings of the 8th ACM International Systems and Storage conference, 2015: 1-11.

勘误说明

因校对原因，本刊 2016 年（第 5 卷）第 1 期第 5 页右栏标题 3.1，原文是“3.1 多根 PCIe 交换机”，现补充更正为“3.1 多根外设部件高速互连接口（Peripheral Component Interface Express，PCIe）交换机”。特此说明。

《集成技术》编辑部