GCFS 系统名字空间的构建①

The Structure of Namespace in GCFS

王霄军　潘　清　许俊奎　田　昊　(装备指挥技术学院 信息装备系 北京 101416)

摘　要： 简述了 GCFS(General Cluster File System，通用集群文件系统)的研究背景，阐述了 GCFS 系统名字空间的构建方法。分析了平衡二叉树(AVL)名字空间的设计，对改进后的系统进行了 chunk 节点插入、查找性能测试。

关键词： GCFS 集群文件系统 名字空间 平衡二叉树 chunk 节点

随着互联网的迅速发展，对互联网海量数据的存储和读取成为诸多网络应用的首要任务。当文件个数、读取需求急剧增加时，容易导致后台服务器负载过大而成为整个系统的性能瓶颈，传统的文件系统很难满足海量数据存储和读取的性能要求。随着 PC 集群逐渐成为了网络服务器的主角，如何在集群上构建文件系统满足网络应用的要求成为一个迫切需要解决的问题。名字空间是整个文件系统中元数据，像目录名、文件名、文件数据存储位置、文件按名查找方式等的组合，名字空间的构建是文件系统操作的基础。

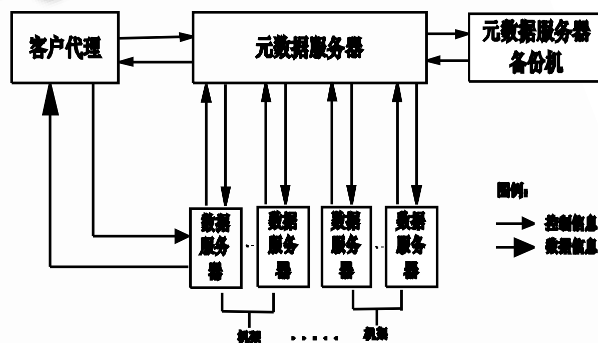

图 1　系统总体结构

1　引言

1.1 GCFS[1-3]系统总体结构

GCFS 集群文件系统由一台元数据服务器(Metadata Server)及其若干台备份机、大量的数据服务器(Data Server)和许多客户端代理(Client Proxy)三部分构成，其中相同网段的数据服务器构成一个机架(Rack)，如图 1 所示。元数据服务器和数据服务器都是运行用户层服务进程的 Linux 机器。

2　常见名字空间的构建方法

尽管集群中可能有许多磁盘可用，但对于一个节点的处理器来说，可能只有与本地节点相连接的磁盘对它可见。为了解决这个问题，许多分布式文件系统采用了 Unix 的安装(Mount)概念来提高系统的可见性，如 NFS[4](Network File System)、CODA[5,6](Constant Data Availability)、Sprite[7]文件系统等。这种方法的思想是像本地系统一样来管理远地文件系统。系统管理员将远地文件系统安装到本地系统可存取的目录上。一旦存取安装后的目录的内容，请求就被发给远地拥有文件系统的节点。远地节点执行操作，并将结果返回给请求节点。所有这些操作对用户都是透明的，用户感觉就像在使用一个单个的文件系统。在这种结构中，目录树都是将本地和远地文件系统组合起来建立的，因此存在一个名字解析问题，即如何根据名字找到对应的文件和目录。一般有集中式和分布式两种方法。

在集中式方法中，有一个节点负责维护映射表。当有文件或目录被创建时，就通知该服务器，它记录

① 收稿时间:2009-01-19

© 中国科学院软件研究所　http://www.c-s-a.org.cn

下新创建对象的物理位置。当某个应用要存取该文件或目录时，系统就向该集中式服务器发一个请求来获得目标的物理位置。这种方式的优点是名字空间数据比较集中，便于管理和保持名字空间的一致性，不足是容易造成性能瓶颈和单点失效。

分布式方法又可分为独立名字空间和全局名字空间。独立名字空间是指每个系统都拥有自己独立的名字空间，如 Sun NFS[4]。每个系统都知道本地安装的远地文件系统，也知道该远地文件系统实际是在哪个节点上。因此给定任何文件名或目录，每个系统都有足够的信息找到该文件或目录在网络中的位置。这种方法的缺点是：它不是位置独立的，如果一个磁盘移到另一个节点上，要重新安装目录树。全局名字空间是指所有节点都使用统一的全局名字空间。在这种情况下，目录树被划分成域(Domain)，每个域有一个名字服务器负责名字解析。名字服务器知道数据在哪个磁盘上。如图 2 是将目录树划分为域，通过名字服务器来定位一个文件的例子。这种方法的优点是元数据是分布的，不存在单点失效和性能瓶颈，缺点是名字空间比较分散，不利于元数据一致性操作(如加锁操作)和崩溃后的恢复。

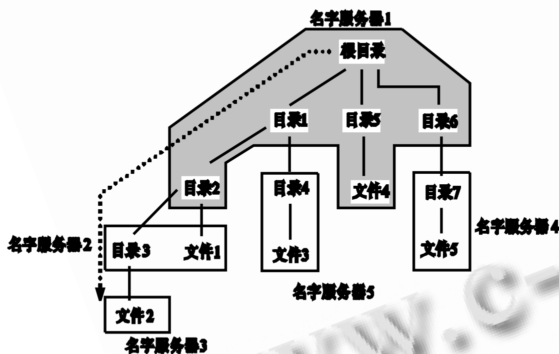

图 2　目录树划分为域的例子

3　系统名字空间

在 GCFS 系统中，Metadata Server 主要负责名字空间的构建维护并完成对元数据的有关操作，如图 3。Metadata Server 主运行程序主要包括五个线程。某些线程还涉及负载平衡、锁管理等方面的工作，线程之间还存在相互间通信。

名字空间的构建采用何种数据结构关系到空间利用率和元数据操作的速度。在 GCFS 中，利用树状结构构建名字空间的 chunk 层，提高了操作效率。

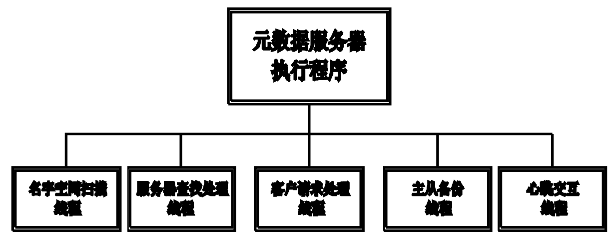

图 3　元数据服务器体系结构图

GCFS 将元数据的名字空间划分为目录(Dentry)空间和机架(Rack)空间。其中 Dentry 空间是指由目录和文件链接构成的一部分空间，Rack 空间是指由 Rack、Data Server 和 chunk 链接构成的一部分空间，二者通过共享 chunk 结构而联系在一起，共同构成了系统的名字空间。每一级采用双向循环链表构建，如图 4 所示。

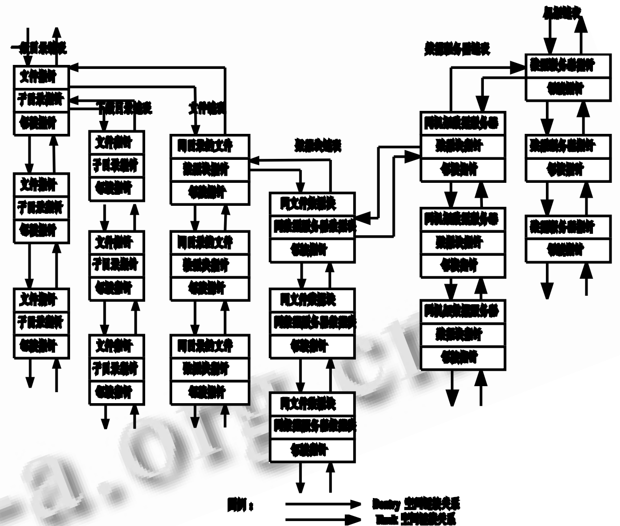

图 4　Metadata Server 原名字空间结构示意图

3.1 树形分域名字空间结构

为保持名字空间的易用性，我们采用了集中式和分布式相结合的树形分域目录结构。如上图所示，存在一个传统的目录树结构(如图 5(a)所示)，而系统所有的文件则根据 hash 算法(或其他的算法)分布于各个文件域中。在目录树结构中存在着一个文件名到域名和文件标志符的映射，每个文件标志符和该域中的一个文件相对应，在该域中是唯一的，以一个 64 位的整数来表示，而在每个域内，存在一个以文件标志符为下标的可扩展数组(如图 5(b)所示)，对文件位置信息的存取就可以利用文件标志符来随机存取。

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn

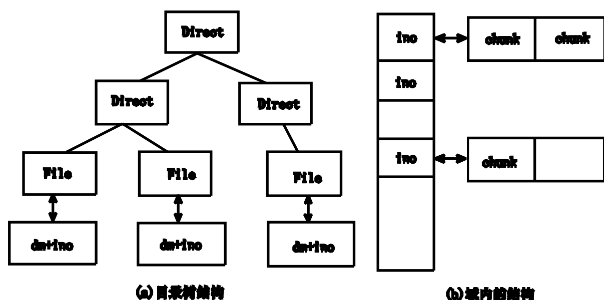

图 5　树形分域名字空间

3.2 平衡二叉树名字空间的设计

由于系统中的文件不同于普通文件，它是若干文件的集合体，也就是一个大文件。在某一个目录下文件数目不会太大，目录数也相对有限。一个 Rack 下也不会有太多的 Data Server,且数量也较为稳定。因此，这几层采用线性结构，对搜索效率影响较小。但是随着数据量的增加，在 Data Server 的数目没有动态增长的情况下，单台机器存储的 chunk 数目必然大量增加；或者在某些应用环境下采用较小的 chunk，chunk 的数目也会有较大的增长。这样在 chunk 层采用双向循环链表这一线形结构，chunk 数目变化必然会对搜索和查找效率产生很大影响。对此，需对 chunk 层构造进行改进。

如果要使搜索和查找效率受 chunk 数目变化的影响尽可能小，应该采用非线性结构。从效率和空间利用等方面综合考虑，采用二叉排序树[8,9]比较合适。正常情况下，它的平均查找长度和 logN(N 为树中结点个数)等数量级，受 chunk 数目变化影响较小；另外，单个节点创建时除自身信息外只需再分配包含左右孩子节点指针大小的空间，不会像非二叉树结构，单个节点可能造成较大的空间浪费。

一般情况下，任意一台 Data Server 上新建 chunk 的 handle 都是增序产生。若使用普通的二叉排序树，单台 Data Server 上所有 chunk 将会构成线性结构，没有任何意义。因此采用经过平衡化处理的平衡二叉树(AVL)[8,9]。

采用递归方式定义平衡二叉树，平衡二叉树或者是一棵空树，或者是具有下列性质的二叉树，左子树和右子树都是平衡二叉树，且左子树和右子树的深度之差绝对值不超过 1。由于平衡二叉树在插入或删除节点的同时通过对初始二叉排序树进行旋转处理来对左右子树的深度差进行调节，所以可有效避免左右子树深度失衡的情况，并能形成一种和折半查找的判定树相似的二叉树结构。在任何情况下，它的深度和 logN 是等数量级的，平均查找长度也和 logN 等数量级。这样，chunk 数目的增加对搜索和查找效率的影响就能得到很大程度的降低。

3.3 chunk 层节点平衡化旋转操作

平衡二叉树(AVL)比较于一般的二叉排序树在操作上主要增加了对左右子树深度差的调节，这依赖于平衡化旋转操作[8]。AVL 平衡化旋转有两类：单旋转(左旋和右旋)和双旋转(先左后右和先右后左)。

(1) 左单旋转(RotateLeft)

原 AVL 树的形状如图 6(a)所示。图中大写字母指明节点，矩形框表示节点的子树，其中字母 h 给出子树的高度。我们在子树 E 中插入一个新节点，如图 6(b)，致使以 A 为根的子树失去平衡。沿插入路径检查三个节点 A、C 和 E，处于一条方向为“\”的直线上，需要进行一次向左的逆时针旋转操作：以节点 C 为旋转轴，节点 A 反时针旋转成为 C 的左孩子，C 代替原来 A，原来 C 的左孩子 D 转为 A 的右孩子，旋转后的形状如图 6(c)所示。

(2) 右单旋转(RotateRight)

原 AVL 树如图 7(a)所示。在节点 B 的左子树 D 上插入新节点，如图 7(b)，致使以 A 为根的子树失去平衡。沿插入路径检查三个节点 A、B 和 D，处于一条方向为“/”的直线上，需要进行一次向右的顺时针旋转操作：以节点 B 为旋转轴，将节点 A 顺时针向下旋转成为 B 的右孩子，B 代替原来 A，原来 B 的右孩子转为 A 的左孩子，旋转后的形状如图 7(c)所示。

(3) 先左后右双旋转(RotationLeftRight)

一棵 AVL 树，如图 8(a)所示。假设在子树 F 或 G 中插入一个新节点，如图 8(b)将新节点插入到子树 F 中，致使以 A 为根的节点失去了平衡。沿插入路径检查三个节点 A、B 和 E，处于一条方向为“<”的折线上，需要进行先左后右的双旋转：以节点 E 为旋转轴，将节点 B 逆时针旋转，进行前面的左单旋转，如图 8(c)。再以 E 为旋转轴，将节点 A 顺时针旋转，进行前面的右单旋转，如图 8(d)所示。

(4) 先右后左双旋转(RotationRightLeft)

一棵 AVL 树，如图 9(a)所示。假设在子树 F 或 G 中插入一个新节点，如图 9(b)将新节点插入到子树 G

© 中国科学院软件研究所 http://www.c-s-a.org.cn

中，致使以 A 为根的节点失去了平衡。沿插入路径检查三个节点 A、C 和 D，处于一条方向为“>”的折线上，需要进行先右后左的双旋转：以节点 D 为旋转轴，将节点 C 顺时针旋转，进行前面的右单旋转，如图 9(c)。再以 D 为旋转轴，将节点 A 逆时针旋转，进行前面的左单旋转，如图 9(d)所示。

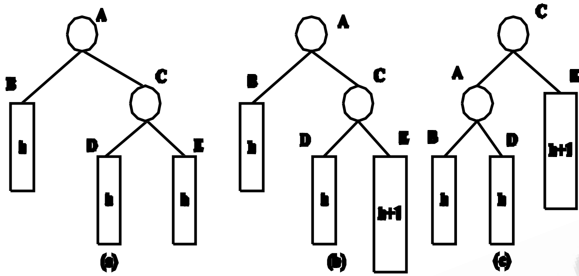

图 6　左单旋转前后树的变化

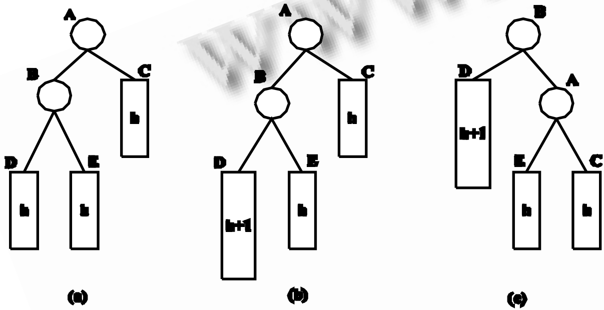

图 7　右单旋转前后树的变化

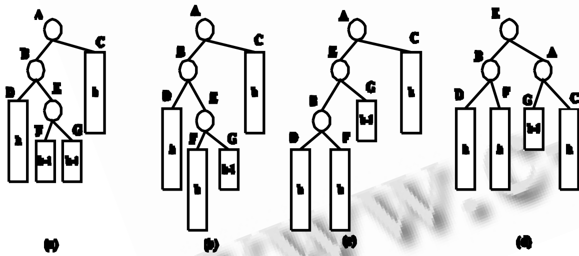

图 8　先左后右双旋转

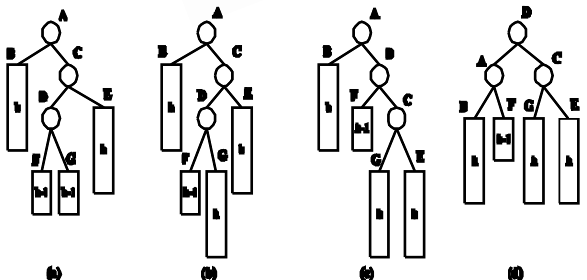

图 9　先右后左双旋转

4　性能测试

4.1　测试环境

一共集中了八台客户机，八台数据服务器，一台元数据服务器和一台备份元数据服务器，所有机器都通过 D-Link 交换机连接成以太网。连接情况如图 10 所示。

由于目前只能将所有机器配置在一个局域网中，无法从规模和硬件等级上体现出 Rack 这一层次，因此采用从逻辑上实现的办法。通过设置一个 Rack 上限值 4，在前四台 Data Server 加入系统时，Metadata Server 创建 Rack 节点并将它们分别作为四个 Rack 的首 Data Server 节点，若再有 Data Server 加入，依次使用每个 Rack 来容纳新的 Data Server 节点。这样最终形成每个 Rack 各有两台 Data Server 的分布格局。

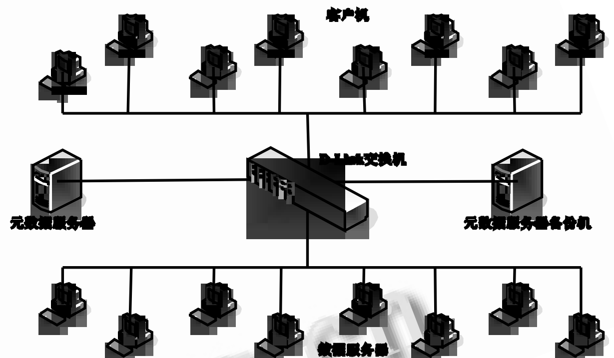

图 10　系统测试环境

4.2 AVL 效率测试

这里对 chunk 层采用 AVL 结构后，在节点搜索查找效率方面与原线形结构进行对比。为了避免网络数据传输对性能比较的影响，采用在单机上直接运行插入节点、查找节点等程序，调用相应数据结构类的操作函数的方式来实现。

(1) 节点插入性能测试

采用将所有 chunk 节点依次链接到各个 Data Server 节点的办法，来近似模拟负载平衡的效果。向系统中插入一定量 chunk 节点，chunk 层采用线性结构和 AVL 结构性能对比如图 11 所示，时间值是十次测试取平均。

可以看出名字空间 chunk 节点插入操作，随着要插入节点数的增加，线性结构操作时间较 AVL 结构出

© 中国科学院软件研究所 http://www.c-s-a.org.cn

现更大幅度的增长。因此采用 AVL 结构较线性结构能较明显地提高操作效率，尤其是在插入节点数较多的情况下，优势更为明显。

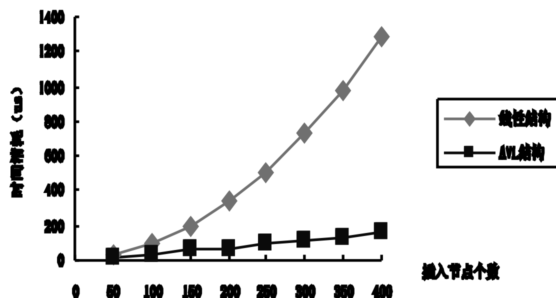

图 11　chunk 节点插入消耗时间对比

(2) 节点查找性能测试

先后按照线形结构和 AVL 结构在 Metadata Server 名字空间的某个 Data Server 节点上插入 1000 个 chunk 节点(实际节点数应该大于此值)，然后按照 handle 从 1 至 1000 分别进行查找，测出它们的查找时间，进行比较，性能对比如图 12 所示。图中标明的时间值是对对应 handle 范围内的每个 chunk 进行查找后得到的一组时间值的均值，在每一组中各个时间值之间的差都不大于 1us。

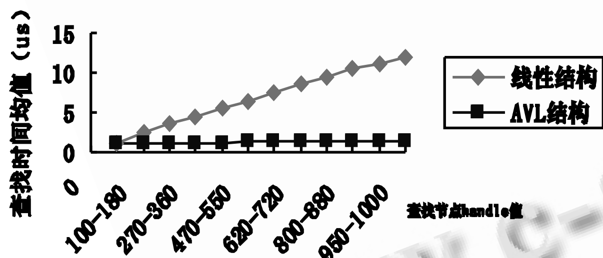

图 12　chunk 节点查找性能对比

从测试结果可以看出线性结构随着节点 handle 值的增加,查找时间也以几乎线性的方式增长,而 AVL 结构查找时间受节点 handle 值影响很小，查找时间在 us 级上非常稳定。可见，在 Data Server 节点上以 AVL 的方式链接 chunk 节点对于查找效率的提高是很有帮助的。

参考文献

1 许俊奎.集群文件系统元数据服务器研究.北京:装备指挥技术学院, 2005.

2 曹培发.集群文件系统数据服务器研究.北京:装备指挥技术学院, 2005.

3 田昊.GCFS2.0 集群文件系统的设计与实现.北京:装备指挥技术学院, 2008.

4 Sun microsystems, Inc,2550 Garcia Ave, Mountain view,CA 94043. Networking on the Sun Workstation,revision B edition February 1986.

5 Braam PJ. The Coda Distributed File System. Linux Journal, June 1998.

6 Satyanarayanan M. Coda: A Highly Available File System for a Distributed Workstation Environment. Proceedings of the Second IEEE Workshop on Workstation Operating Systems, Sep 1989, Pacific Grove, CA.

7 Ousterhout JK, Cherenson AR, Douglis F, Nelson MN, Welch BB. The Sprite network operating system. Computer, February 1988.

8 殷人昆,陶永雷,谢若阳,等.数据结构(用面向对象方法与 C++描述).北京:清华大学出版社, 1999.

9 严蔚敏,吴伟民.数据结构(C 语言版).北京:清华大学出版社, 1997.

© 中国科学院软件研究所　http://www.c-s-a.org.cn