CN43-1258/TP
ISSN 1007-130X

计算机工程与科学
COMPUTER ENGINEERING & SCIENCE

2005 年第 27 卷第 2 期
Vol. 27, No. 2, 2005

文章编号:1007-130X(2005)02-0078-04

基于对象存储的集群存储系统设计*
Design of a Cluster Storage System Based on OSD

刘 仲,章文嵩,王召福,周兴铭
LIU Zhong, ZHANG Wen-song, WANG Zhao-fu, ZHOU Xing-ming
(国防科技大学计算机学院,湖南 长沙 410073)
(School of Computer Science, National University of Defense Technology, Changsha 410073, China)

摘 要:集群存储是解决大规模数据存储的重要方法。本文提出一种基于对象存储的集群存储系统结构,将文件分为目录路径元数据、文件元数据与数据对象三部分并独立管理。性能比较与分析表明,该方法能够支持超大规模的文件及超大容量的目录,明显地减少网络访问消息数量,提高访问性能,并且解决了因为修改目录而导致的大量元数据迁移问题。

Abstract: Cluster storage is an important solution to large-scale data storage systems. In this paper, we present the architecture of a cluster storage system based on OSD in which the file is divided into directory path metadata, file metadata and data objects that are managed separately. Performance comparison and analysis results show that this method supports files with extremely high variance in size and directories containing a very large number of files, reduces obviously the number of network access messages, improves access performance, and resolves the problem of large metadata migration because of modifying the directory.

关键词:集群存储;对象存储;元数据管理;文件系统
Key words: cluster storage; OSD; metadata management; file system
中图分类号:TP333 **文献标识码**:A

1 引言

随着因特网技术的飞速发展和网络用户数量的快速增长,由此产生的各种数据呈几何级数爆炸式增长,促使数据存储容量以每年 3~5 倍的速度急剧增加[1];出现大量数据密集型的应用,如数字图书馆、数据仓库、数据挖掘、气象数据处理、卫星数据处理、医药视频图像数据处理、生命科学研究、多媒体点播、在线数据处理等,这些应用对存储系统的 I/O 性能(带宽、吞吐率、响应时间等)提出了更高的要求。因此,如何构建一个高性能、高可伸缩、高可用、可管理、安全的分布存储系统就成为企业存储所面临的最重要问题。

2 对象存储

目前有三种主要的存储结构[2],即直接附加存储(Direct Attached Storage,简称 DAS)、网络附加存储(Network Attached Storage,简称 NAS)和存储区域网络(Storage Area Network,简称 SAN)。

DAS 是传统的基本存储模式,由单个或多个块存储设备构成,并直接连接到主机系统的扩展接口上。它的优点是连接简单,性能高,易于使用。缺点是难以大容量扩展,适合于有限共享的应用,如小型的数据库或文件服务器。

NAS 是一个专用的文件服务器,通过 IP 网络协议给连接到网络上的用户提供基于文件级的数据共享服务。NAS 的好处在于跨平台文件共享,支持不同平台的用户通过标准的接口(如 CIFS 或 NFS)共享服务器上的存储文件,适合于需要提供跨平台共享文件的应用,如 WEB 服务器、局域网上的文件服务器等。缺点是不适合于块级数据应用(如数据库系统),并且服务器可能成为性能瓶颈。

SAN 是以数据存储为中心,采用可伸缩的交换网络结构替代传统的总线结构,所有的主机系统和存储设备之间都是通过高速的网络相连,提供内部任意节点之间的多路可选择的数据交换。它具有可伸缩、高带宽、高可靠性、高容错能力的优点,适合于性能要求高、对存储设备可伸缩性

* 收稿日期:2003-10-22;修订日期:2003-12-12
基金项目:国家 863 计划资助(2001AA111012)
作者简介:刘仲(1971-),男,湖南邵东人,博士生,助理研究员,研究方向为网络存储、并行与分布处理;章文嵩,博士,副教授,研究方向为集群技术;王召福,博士,讲师;周兴铭,中科院院士,教授,博士生导师,研究方向为高性能体系结构、并行与分布处理。
通讯地址:410073 湖南省长沙市国防科技大学计算机学院;Tel:(0731)4573670,13973177836
Address: School of Computer Science, National University of Defense Technology, Changsha, Hunan 410073, P. R. China

要求很高的应用，如运行在数据中心的、基于集群的大型分布式数据库系统。缺点是只能提供设备共享，不能提供数据共享。

NAS 由于高层次（文件级）的抽象，所以能够实现跨平台数据共享，而服务器的转发降低了系统的性能，并且可能成为系统性能瓶颈；SAN 由于低层次（块级）的抽象，所以只能提供设备共享，不能实现跨平台数据共享，但直接访问存储设备能实现高性能的数据传输。

对象存储（Object-based Storage Device，简称 OSD）[2,3] 正成为当前存储研究中的热点，国际信息技术标准委员会（INCITS）下属的 T10 技术委员会专门成立一个技术工作小组负责 OSD 的标准化制订工作；Peter Braam 博士领导的一个小组正在开发基于 OSD 的集群文件系统的开放源项目 Lustre；Gibson 教授在 Carnegie Mellon 大学组织 NASD 项目；Scott Brandt 教授领导的一个研究小组在 California 大学存储系统研究中心（Storage Systems Research Center）正致力于基于 OSD 的大规模可伸缩的存储系统研究；另外，如 IBM、Intel、HP、EMC 等大公司都启动了相关的对象存储研究。

OSD 通过对象层次的抽象设计，将数据存储的访问、控制、管理等功能重新划分，将文件系统的逻辑结构与物理存储的映射关系隐藏到对象一层，数据的访问操作通过对象的接口实现。相对于 SAN 结构，OSD 能够将与设备相关的特性（如块分配）从设备一层中分离出来，隐藏于对象层中，从而实现跨平台的能力；相对于 NAS 结构，OSD 能够直接与网络中的主机系统进行数据交换，而不需要服务器的数据转发，从而实现高性能的数据传输。因此，基于 OSD 可设计一种新的高性能、高可伸缩、跨平台共享的大规模分布存储服务系统。

传统块存储结构与对象存储结构中各组成部分的逻辑位置对应关系如图 1 所示。

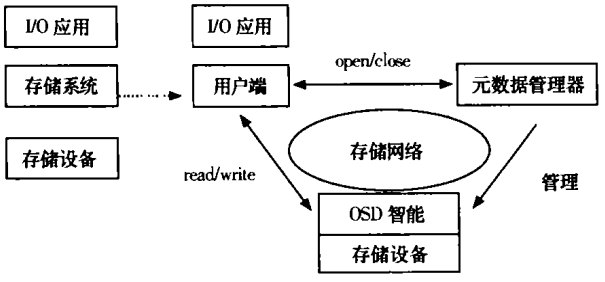

图 1　OSD 将存储系统的功能重新划分

3　集群存储系统的设计

基于对象存储我们提出一种集群存储系统（OSD-based Cluster File System，简称 OCFS）的设计与实现方案。OCFS 包括四个部分，即客户端文件系统（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（Medadata Server，简称 MDS）和对象存储体（Object Storage Target，简称 OST），各个部分之间通过高速网络互连。OCFS 的体系结构如图 2 所示。

在 OCFS 中，任何文件的数据分为三部分：目录路径元数据、文件本身元数据与数据对象。目录路径元数据包括文件所在的目录路径名和路径访问控制属性，保存在 DPIS 中；文件本身元数据包括文件的属性如所有者、文件大小、最近访问时间、数据对象的位置等，保存在 MDS 中；数据对象包含文件的实际数据，保存在 OST 中。由于实际的数据块分配隐藏到 OST 中，并且文件可以包含若干个数据对象。因此，OCFS 的文件大小不再受限于传统设计的数据块个数，能够支持超大规模的文件以及超大容量的目录。用户访问文件首先通过 CFS 访问 DPIS 与 MDS，确定对文件的访问权限和获取文件的位置信息，然后通过网络与 OST 直接交换数据对象。数据对象的访问绕开了服务器的中转，能够实现高性能的并行数据传输。

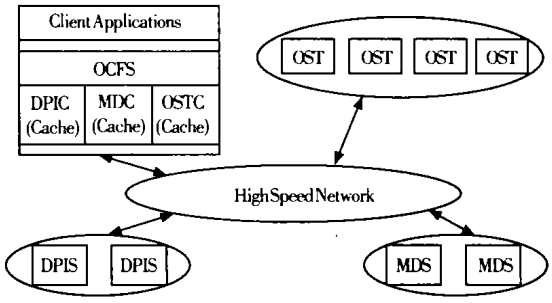

图 2　OCFS 体系结构

3.1　客户端文件系统

客户端的 CFS 在 Linux 内核中实现，通过 Linux 的 VFS（Virtual File System，简称 VFS）接口挂接上。上层的应用程序或用户通过 CFS 看到的是一个兼容标准的 POSIX 文件系统语义的共享文件系统，该文件系统通过文件集的形式提供给用户和上层应用一致的全局命名空间。

3.2　目录路径索引服务器

现有大规模分布存储系统中导致元数据大量更新与迁移的主要原因是文件所在的目录路径的变化（目录名修改、目录权限修改、删除目录）[4]。因此，OCFS 的设计中将文件的目录路径从元数据中抽象出来，由 DPIS 独立管理。这种设计能够避免或减少因为目录修改而导致的大量元数据迁移，提高元数据的访问性能。

DPIS 为系统中的所有目录文件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。若采用集群，则通过集群软件 LVS 统一管理目录路径 ID 的分配，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如下：

$$INDEX=<DPID, DirectoryPath, AC_p>$$

其中，DPID 表示全局唯一的目录路径 ID，DirectoryPath 为目录文件的全路径名，AC_p 表示该目录的路径访问控制属性。路径访问控制属性的构造同现有的遍历文件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，每一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目录文件时，不再需要遍历前面的目录，可以递归使用当前的路径访问控制属性与新建目录文件的访问控制属性构建新的路径访问控制属性。索引项由 DPID 唯一确定。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，只需要修改 DPIS 中相应的索引项，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导致该目录下所有文件的元数据重新分配，避免了大量的元数据迁移。特别约定根目录"/"的 DPID 为 0。

目录索引项在磁盘上的存储采取 B^+ 树算法，能够提供标准的树型层次结构，同一目录下的子目录索引项存储位置尽可能靠近，提高检索和存储效率。

3.3 元数据服务器

MDS 管理文件本身的访问属性和数据对象的位置信息。文件的位置信息由保存文件的数据对象的 OST 编号及该 OST 分配给文件的数据对象 ID 确定。定义文件的位置信息如下：

$$LOCATION=<OST\_NO,OID>$$

其中，OST_NO 表示保存文件数据对象的 OST 编号，OID 表示该 OST 分配给文件的数据对象 ID。

定义所有文件的元数据集合为 D，每一个文件的元数据项 $d\in D$，d 定义如下：

$$ITEM=(DPID,Name,TYPE,AC,LOCATION,OTHER)$$

其中，DPID 表示文件的目录路径 ID，Name 表示全路径名中的最后一个分项名，TYPE 表示文件的类型，AC 表示文件的访问控制属性，LOCATION 表示文件的位置信息，OTHER 表示其他的元数据信息(文件大小、最近访问时间等)。DPID 在新建文件时获得。

在元数据的分配上，OCFS 通过动态线性散列计算实现元数据在不同元数据服务器之间均匀分配，散列计算需要唯一标识一个文件的关键字。由于 DPID 是全局唯一的，而同一个目录下文件名是唯一的，因此 OCFS 中依据文件的 DPID 与文件名作为标识该文件的关键字，对计算出来的关键字 KEY 进行散列计算，分配到不同的元数据服务器。特别约定，根目录"/"的元数据分配到编号为 0 的元数据服务器。

元数据项在磁盘上的存储采取 B+ 树算法，能够提供标准的树型层次结构，同一目录下的元数据项存储位置尽可能靠近，以提高检索和存储效率。

3.4 对象存储体

OST 负责存储文件的数据对象。不同的数据对象有不同的对象标识(OID)，OST 负责管理和分配 OID，根据请求的 OID 读写文件的数据对象。大规模存储系统中的可靠性与容错性非常重要，由于 RAID 技术在系统重建时需要的时间较长，并且不同 OST 的容量、性能不一定一致，所以可将 RAID 技术中的一些算法如 erasure coding、replication 等应用在数据对象一级上，以保障存储数据的可靠性和负载平衡。

4 性能比较与分析

OCFS 的设计中，DPIS 和 MDS 用于管理存储系统的元数据，OST 用于存储系统实际的数据。相对于现有的分布式文件系统，OCFS 在元数据的获取和实际数据的访问上，都可取得更好的性能。下面以访问文件为例，对比 OCFS 与网络文件系统 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/usr/src/test/hello. c。

对于 NFS，执行命令：sun # mount -t nfs bsdi:/usr /nfs/usr，将服务器 bsdi 上的/usr 目录安装成为本地文件系统/nfs/usr。访问文件：sun # cat /nfs/usr/src/test/hello. c，将文件复制到终端。

访问文件经历的网络访问过程包括：(1)getattr(usr)；(2)lookup(src)；(3)lookup(test)；(4)lookup(hello. c)；(5)getattr(hello. c)；(6)read(hello. c)。其中，前五次过程可看作获取元数据，(6)是读取实际数据。

对于 OCFS，执行命令：sun # mount -t cfs bsdi:/usr /cfs/usr，将服务器 bsdi 上的/usr 目录安装成为本地文件系统/cfs/usr。访问文件：sun # cat /cfs/usr/src/test/hello. c，将文件复制到终端。

访问文件经历的网络访问过程包括：(1)getattr(/usr/src/test)；(2)getattr(hello. c)；(3)read(hello. c)。其中，前两次过程可看作获取元数据，(3)是读取实际数据。

更一般的情况，假定文件 hello. c 前面的目录个数为 n，对 NFS，需要 $n+2$ 次过程获取元数据，一次过程读取实际数据；对 OCFS，仍然只需要两次过程获取元数据，一次过程读取实际数据。很显然，当 $n\geq 1$ 时，OCFS 需要的网络访问消息数量明显少于 NFS，并且 OCFS 中读取数据时由 CFS 直接与 OST 进行数据交换，不再需要服务器中转，而 NFS 仍然需要服务器转发数据。当大量用户并发访问时，OCFS 具有明显的性能优势。

OCFS 比 NFS 具有性能优势的原因在于，现有的元数据管理方法必须遍历文件路径中所有目录才能获取文件的元数据以及确定文件的访问权限。在 OCFS 中，将文件的元数据分为目录路径元数据和文件本身的元数据，文件的访问权限通过路径访问控制属性与文件本身的访问控制属性共同确定。而目录路径元数据和文件本身的元数据都只需要一次散列计算即可获取，同时通过路径访问控制属性与文件本身的访问控制属性确定文件的访问权限，不再需要遍历全部目录，大大提高了访问性能(若考虑缓存机制则更明显)。

OCFS 的另一个更重要优点在于，相比较目前的大规模存储系统中的元数据管理方法，OCFS 中引入的 DPIS 能够有效避免或减少因为修改目录而导致的大量元数据迁移。目前的元数据管理方法中典型的做法是，根据文件的全路径文件名进行散列计算，分配元数据到不同元数据服务器中。其优点是能够直接定位文件的元数据信息，但在修改目录(修改目录名、修改目录权限、删除目录)时将导致大量的元数据迁移，最多可能需要 $m\times(n-1)/n$ 条消息(m 为该目录下的文件数目，n 为元数据服务器数目)[4]。OCFS 设计能够避免或大大减少元数据的迁移，最多只需要一条消息：(1)修改目录名，OCFS 中散列计算的关键字只与文件名及 DPID 有关，与目录名、权限无关，因此只需要更新目录路径索引服务器中的索引项以及更新和迁移一条元数据信息(修改的当前目录文件)；(2)修改目录权限，只需要更新目录路径索引服务器中的索引项，不需要更新和迁移元数据信息；(3)删除目录，只需一条消息(多播)即

可，不需要元数据迁移。

5 结束语

本文提出一种基于对象存储的集群存储系统结构，将文件分为目录路径元数据、文件元数据与数据对象三部分并独立管理。由独立的目录路径索引服务器管理文件的目录路径名与路径访问控制属性，由元数据服务器管理文件本身的元数据，由对象存储体管理实际数据对象。性能比较与分析表明，与现有的分布存储系统相比，该方法能够支持超大规模的文件及超大容量的目录，能够明显地减少网络访问消息数量。与 NFS 相比，获取元数据需要的网络操作过程从 $n+2$ 减少到 2 次（n 为目录路径中的目录层数），提高了访问性能，并且解决了因为修改目录而导致的大量元数据迁移问题；与文献[4]中的方法相比，修改目录导致的元数据更新消息从 $m\times(n-1)/n$ 条消息（m 为该目录下的文件数目，n 为元数据服务器数目）减少到 1 条。下一步的研究工作着重于更细致的目录索引项与元数据项的分配、存储、定位等算法的研究以及元数据的可靠性与容错性研究。

参考文献：

[1] Robert M Montague, Steven L Denegri, Thomas H Curlin, et al. System Area Networks: The Next Generation of Scale in the Data Center[Z]. Dain Rauscher Incorporated, 2001.

[2] http://www.intel.com/labs/storage/osd/tech.htm,2003-03.

[3] Peter J Braam. The Lustre Storage Architecture[EB/OL]. http://www.lustre.org/docs/lustre.pdf, 2003-03

[4] S A Brandt, E L Miller, D D E Long, et al. Efficient Metadata Management in Large Distributed Storage Systems[A]. Procs of the 20th IEEE/11th NASA Goddard Conf on Mass Storage System and Technologies[C]. 2003. 290-298.

（上接第 61 页）

如果有连接，找到连接链表项，关闭连接上本地端口，清除 TCP select 操作注册的信息，去掉对会调函数的引用，删除表项，然后发送本进程的监听端口号给本地协调者进程。

（3）在出错进程检查点上恢复的进程首先清除占用的与连接相关的资源，然后在本地协调进程上注册，并且另外打开一个监听端口把相应信息发送给本地协调进程，该消息需被特殊标识，而进程本身进入等待进程通道表填充状态。

（4）协调进程（与出错进程不在同一主机）在收到全部注册的本地进程的监听端口信息后，确定本地相关进程已经进入等待同步更新进程通道表状态，发送收集的端口信息给出错进程所在主机的协调进程，自己进入等待同步更新通信子表项状态。

（5）新生成进程的协调进程在收到所有进程的监听端口信息之后，用新生成进程作为替换进程占用出错前对应的指针数组序列位置，然后发送合并的进程通道表（只含进程标识和监听端口）给替换进程，使其进入 WAIT_RUN 状态，发送替换进程表项信息给本地其它注册进程，使其进入 WAIT_RUN 状态，发送替换进程表项信息给其它协调进程，然后本地协调进程自身进入 WAIT_SYNC 状态，等待其它协调进程的确认。

（6）其它协调进程收到出错进程的通道信息后发送给本地注册进程，替换各进程通信子副本数组中对应位置的通道数据结构记录，使其进入 WAIT_RUN 状态。协调进程发送 UPATA_SYNC 消息给替换进程的协调进程，等待同步运行的确认。

（7）替换进程端的协调进程收到所有协调进程的确认后，唤醒本地注册进程继续向前同步执行，广播 RUN_CONTINUME 消息给其它处于等待同步运行的协调进程，相应地唤醒本地注册进程继续向前同步执行。

5 性能分析

在节点数目一定的情形下，单一节点内进程出错恢复时间消耗如图 4 所示。恢复时间包含出错进程检查点上重启动、两次主机间协调进程之间的广播通信和主机内进程两次多播通信以及一次通道信息报告通信。这里面通道信息报告可以通过程序启动在协调进程上直接注册的形式变成只要一次出错进程另外注册所替代。出错进程检查点上重启动所需要的时间趋于一个常量，如图 4 所示。占用恢复时间的主要是几次广播操作，开销随着进程数目的增加而增加。在节点数目增加的时候，协调进程间的通信代价也会提高。

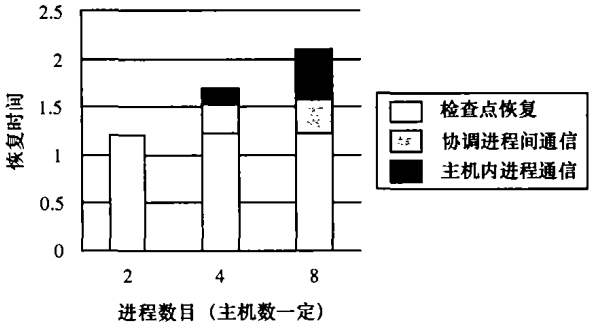

图 4 恢复时间代价

6 结束语

本文提出了一种 MPI 通信子动态重构方法，实现了对 MPI 进程的动态管理。因为重构过程涉及多次广播同步，目前的实现只适合集群环境。协调者进程在同步通道信息表时起到关键作用，如何实现在主机失效时竞争选举一个协调者进程发起更新过程，替换协调进程，以及处理好 P4 通信库信号处理过程中可能出现的竞争，是要进一步解决的问题。

参考文献：

[1] R Butler, E Lusk. Monitors, Message, and Clusters: The p4 Parallel Program System[J]. Parallel Computing, 1994, 20: 547-564.

[2] S Rao, L Alvisi, H Vin. Egida: An Extensible Toolkit for Low-Overhead Fault Tolerance[A]. Proc of the 29th Int'l Symp on Fault-Tolerant Computing[C]. 1999.