CN43-1258/TP
ISSN 1007-130X
计算机工程与科学
COMPUTER ENGINEERING & SCIENCE
2008 年第 30 卷第 11 期
Vol 30, No 11, 2008

文章编号: 1007-130X(2008)11-0140-04

基于面向对象存储系统的对象及数据副本布局方法设计*

A Layout Algorithm for Objects and Data Replicas Based on the OSD Systems

陈平增, 齐　路, 刘晓光, 王　刚
CHEN Ping-zeng, QI Lu, LIU Xiao-guang, WANG Gang
(南开大学计算机科学与技术系, 天津 300071)
(Department of Computer Science, Nankai University, Tianjin 300071, China)

摘　要: 面向对象存储是近年来存储系统领域的研究热点之一。与传统存储系统相比, 面向对象存储系统具有高性能、可伸缩、更安全、更可靠的特点。针对面向对象存储的特点, 本文给出了一种对象和数据副本的布局方法, 并在原型系统上验证了实现。实验测试表明, 这一布局方法能够很好地实现系统的负载均衡和可伸缩性。

Abstract: The object-based storage device (OSD) technology has become a hot topic in the field of storage systems these years. Compared with the traditional storage technologies, the OSD system has better performance, scalability, security, and reliability. Based on the characteristics of OSD, a layout algorithm for objects and their copies is presented in this paper. The experimental results show that this algorithm can satisfy the requirements of system load balancing and scalability.

关键词: 面向对象存储; 数据布局; 副本

Key words: OSD; data layout; replica

中图分类号: TP391　　文献标识码: A

1　引言

目前, 存储与网络相结合的网络存储技术已成为 IT 行业第三次浪潮的先锋。一方面, 网络存储技术被广泛地应用于数字图书馆、电子商务、搜索引擎等新技术中, 潜移默化地影响着人们的生活; 另一方面, 各种应用所表现出的新特点也对网络存储提出了严峻的挑战。现有的存储系统(如 DAS、NAS、SAN[1,2])在设备级的基本单元是块, 设备本身并不能对数据进行智能的管理。这项工作是由更高层的软件(比如文件系统)来完成的。在某些情况下, 设备层软件就显得比较笨拙。

在面向对象的设备(Object-based Storage Devices, 简称 OSD)[3]中, 对象是数据存储的基本单元。对象是一系列有序字节的数据集合, 能够存储任意类型的数据。对象包括数据和属性, 其属性可以根据应用需求进行设置。对象独立维护自己的属性, 从而简化了存储系统的管理任务, 增加了系统的灵活性。

与基于块设备的存储系统相比, OSD 存储具有高性能、可伸缩、易管理、更安全、更可靠的特点。面向对象存储的思想源于 CMU(Carnegie Mellon University, 简称 CMU)的 NASD(Network Attached Secure Disks, 简称 NASD)[4]项目。目前, SNIA(Storage Networking Industry Association, 简称 SNIA)的对象存储设备工作组已经发布了 ANSI 关于对象存储的 X3T10[5]标准。该领域主要工作包括 Panasas 公司的对象文件系统 PanFS[6]、Cluster File System 公司的 Lustre[7]、加州大学圣克鲁斯分校开发的 Ceph[8]。

针对面向对象存储系统的负载均衡和数据可靠性问题, 本文给出了一种对象及其数据副本的布局方法。在我们设计的面向对象存储系统 NLOV(Network Logical Object Volume, 简称 NLOV)上, 通过实验验证了上述布局方法的有效性。

2　NLOV 系统结构

现有的 OSD 系统, 例如 Lustre、Ceph 等, 大多是实现

* 收稿日期: 2008-04-13; 修订日期: 2008-07-10
基金项目: 国家自然科学基金资助项目(90612001); 教育部博士点基金资助项目(20070055054); 天津市科技发展计划资助项目(08JCYBJC13000))
作者简介: 陈平增(1983-), 男, 广西南宁人, 硕士生, 研究方向为网络存储系统。
通讯地址: 300071 天津市南开大学计算机系; Tel: (022) 23504780; E-mail: wgzwp@163.com
Address: Department of Computer Science, Nankai University, Tianjin 300071, P. R. China

于文件系统层面上的。这些系统在内部使用了对象作为基本的存储和访问单元,同时还考虑了数据的共享访问、名字空间等分布式系统的问题。OSD 存储系统也可以在设备层实现。设备级 OSD 系统只需考虑对象的存储、散布、容错等问题,屏蔽了底层的存储细节。同时,设备级 OSD 系统提供相关的访问接口,由上层的分布式文件系统处理分布式访问方面的问题。NLOV 就是这样一个实现在设备层的面向对象存储系统。

NLOV 由存储节点、元数据服务器和客户端三个子系统组成,各子系统之间通过对象存储协议进行通信。三个子系统可以分别运行在不同的节点机上,也可以运行在同一个节点机上。NLOV 的系统结构如图 1 所示。

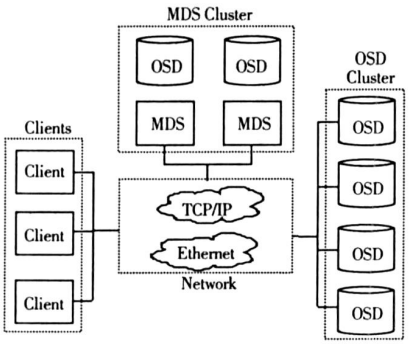

图 1　NLOV 系统结构

存储节点是存储实际数据的容器,每个存储节点都包含一个 OSD,用于存储和访问数据对象。从用户的角度看,所有的存储节点组成一个大的虚拟存储池,每个用户都可以使用其中的一部分。而具体的存储空间分配情况,则由元数据服务器中的对象定位逻辑决定。

客户端为用户提供了访问存储资源的接口,它以逻辑卷设备形式显示。所谓逻辑卷,是指整个大的虚拟存储池的一个逻辑视图。通过它,客户端将数据对象请求发向对应的存储节点。

元数据服务器(Meta Data Servers,简称 MDS)在系统中起着承上启下的作用。承上是指处理客户端的创建、删除、激活逻辑卷等操作;启下是指管理着各存储节点的信息和状态,包括加入、离开、信息更新等。将客户端与存储节点连接起来的是一张存储节点映射表,客户端通过它进行对象的定位,存储节点通过它进行自动的负载均衡。元数据也存储在 OSD 上,这样可以使用统一的对象访问接口,充分利用 OSD 的智能性,提高元数据的管理和访问效率。

NLOV 采用了控制流和数据流分离的系统结构,数据对象的实际 I/O 过程中,客户端和存储节点进行直接通信,不需要服务器的转发,从而消除了数据访问的瓶颈,充分利用了网络带宽。同时,通过对象散布策略使得客户端并发地访问各存储节点,从而得到较高的聚合带宽和吞吐率。

3　对象和数据副本布局算法

能否充分利用存储系统的处理能力和实现负载均衡,是评价存储系统的一项重要指标。通过将对象均匀散布到整个存储空间中,能够显著提高 OSD 系统的负载均衡。此外,数据可靠性也是存储系统的一项重要评价指标。RAID[9]是目前工业界使用最广泛的存储可靠性技术。但是,随着存储设备单位制造成本近年来的大幅降低,数据冗余率已经不再是存储系统设计考虑的关键因素。在这种情况下,副本策略成为兼顾存储系统可靠性和性能的最优选择。

在 NLOV 系统中,我们综合考虑了上述两个因素,设计了一种类似动态区间映射方法[10]的对象和数据副本布局方法。NLOV 中,数据存储和访问的基本单元是对象。对象类似于文件,是能够自我描述的一系列有序字节的集合。每个对象由三部分组成:ID、属性、数据。ID 是对象在存储系统中的唯一标识,NLOV 的用户通过对象的 ID 实现对对象的访问。在这里,对象 ID 是一个 128 位的整数,分别标识对象的类型、是否副本、数据的逻辑卷号和分片号,如表 1 所示。

表 1　对象 ID 各字段含义

对象类型	是否副本	msb	lsb
0 数据对象	0 是 1 否	数据所在的逻辑卷号	在卷中的分片号
1 逻辑卷对象	0	逻辑卷号	
2 映射表对象	0	存储节点映射表的版本号	
3 节点对象	0	节点编号	

这里,我们假定只有数据对象存在副本。根据对象是否是数据副本,对象的映射被分为两部分:源数据的映射和副本的映射。

3.1　源数据的映射

NLOV 中使用哈希函数将对象均匀映射到一个 32 位的哈希空间,$\vec{\Phi}=\{x \mid 0 \leqslant x < 2^{32}, x \in N\}$。根据各存储节点的权重,将对象映射空间分割为若干个不交叉的独立区间,即对任意节点 S_i,其权重为 W_{S_i},其区间集合为 $\overrightarrow{I_{S_i}}=\{\vec{\varphi} \mid \vec{\varphi} \subseteq \vec{\Phi}\}$,并且对任意的节点 S_i、S_j 有以下关系:

$$\frac{|\overrightarrow{I_{S_i}}|}{|\overrightarrow{I_{S_j}}|}=\frac{W_{S_i}}{W_{S_j}} \quad (1)$$

对于任意的对象,我们通过一个哈希算法将其 ID 转换为哈希空间上的一个值,根据这个值所属的区间决定对象所处的存储节点,如图 2 所示。

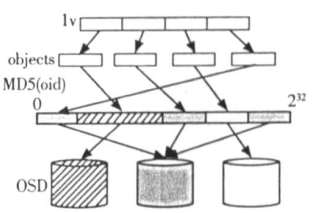

图 2　对象定位示意图

在这里,哈希函数是基于 MD5 算法实现的。主要利用 MD5 均匀散布的特性实现对象的均匀散布。对象映射算法的描述如下:

```
Procedure map_object(object_id)
    x = MD5(object_id) & 0xffffffff;
    i = find_interval(x);
    return find_node(i);
```

其中,*find_interval* 函数根据映射值找到对应的区间,而 *find_node* 函数根据区间找到区间所处的节点。在算法的实现中,所有的区间组织成索引树的结构,使得查找的复杂度为 $O(\log n)$,而通过每个区间的结构中保存着的其所属

存储节点的索引值,可以直接得到对象映射到的节点。

3.2 副本的映射

在NLOV中,副本只针对数据对象。在当前的实现版本中,暂时只支持单副本。每个存储节点维护一个本节点数据对象的副本映射表。副本布局方法的核心思想是再哈希,即在由去除数据对象所在节点的其它节点所构成的哈希空间中进行再哈希,以确定副本的存储位置。在这里,关键因素是如何构造再哈希空间。我们的方法如下:

假设源数据存储在第 i 个节点上,标记为 $Node_i$。以该节点的存储容量作为再哈希空间。然后,以剩余的所有节点作为节点集合,采用与源数据映射相同的方法,根据节点的权重将它们映射到该再哈希空间中。最后,利用前述的对象映射算法根据副本的对象ID确定其存储位置。

3.3 系统规模调整的影响

一个好的存储系统应该具有可伸缩性,即它的规模可以根据需要动态调整。随着存储系统规模的变化,为了达到对象均匀散布的目标,对象和数据副本的映射也必须随之调整。下面以为OSD系统增加一个节点为例进行说明,添加存储节点时,因为添加前对象散布是均衡的,要保证加入节点后系统依然负载均衡,需要将现有的每个节点的区间按新的权重比例划出一部分给新加的节点。算法描述如下:

```
Procedure add_node(p)
  for S_i in S do
    pivot = W_{S_i}/(W_S + W_p) * 0xffffffff;
      u = 0;
      for I in I_{S_i} do
      if pivot >= u + length_I then
            u += length_I
      else if pivot > u then
            I' = split_interval(I, pivot);
            add_interval(p, I';)
              u = pivot;
          else
              add_interval(p, I');
          end if
        end for
      end for
end Procedure
```

上述调整过程分别在对象所对应的哈希空间和副本所对应的再哈希空间中完成。存储系统规模缩小(或者某个节点失效)的映射关系调整方法也可以采用类似方法完成。

4 实验测试

基于PC机和千兆以太网,我们构造了实验测试环境。其中,PC机处理器是Intel P42.6GHZ,内存512M,操作系统为Redhat Linux AS4。

4.1 散布均匀性测试

对象在存储系统中的散布是否均匀,决定了存储系统的性能和可靠性。设系统中有10个节点,各节点的权重从1至100中随机均匀选取。通过我们编写的仿真程序,模拟了32 768个请求,请求所属对象ID为0~32 767。

仿真结果显示,节点的权重在整个系统中所占的比例与节点的负载占全部负载的比例基本持平,即节点的负载情况与节点的权重成正比。所以,系统在静态情况下能够达到负载均衡。仿真结果如表2所示。

表2 均匀性仿真结果

节点	权重	权重比例(%)	请求数	请求比例(%)
1	98	16.07	5 271	16.09
2	31	5.08	1 635	5.00
3	64	10.49	3 467	10.58
4	100	16.39	5 359	16.35
5	50	8.20	2 630	8.03
6	55	9.02	2 920	8.91
7	14	2.30	806	2.46
8	30	4.92	1 611	4.91
9	85	13.93	4 622	14.11
10	83	13.61	4 447	13.57

4.2 系统扩展后的均匀性

现实中的系统并非是完全静态的。当存储容量不能满足当前需求时,就需要对存储系统进行扩展。所以,接下来,本文针对系统扩展后的负载散布情况进行仿真测试。设初始系统中只有一个节点,随后逐一向系统中添加九个节点,各节点权重从1至100中随机均匀选取。扩展完成后,仿真程序模拟了32 768个请求,请求所属对象ID为0~32 767。仿真结果如表3所示。

表3 扩展后均匀性仿真结果

节点	权重	权重比例(%)	请求数	请求比例(%)
1	4	0.80	256	0.78
2	9	1.79	626	1.91
3	71	14.14	4 613	14.08
4	77	15.34	5 047	15.40
5	66	13.15	4 273	13.04
6	87	17.33	5 678	17.33
7	37	7.37	2 394	7.31
8	23	4.58	1 575	4.81
9	64	12.75	4 155	12.68
10	64	12.75	4 151	12.67

仿真结果显示,扩展后节点的权重在整个系统中所占的比例与节点的负载占全部负载的比例基本持平。所以,系统在动态扩展后依然能够达到负载均衡。

5 结束语

OSD存储技术是存储系统领域近年的研究热点之一。与传统存储系统相比,OSD系统具有高性能、可伸缩、易管理、更安全、更可靠的特点。本文详细介绍了我们实现的面向对象存储系统NLOV的对象和数据副本的布局方法,并通过实验测试验证了该方法的有效性。目前,我们只实现了单副本散布方法。在下一步的工作中,我们将解决多副本的数据散布问题,以进一步提高数据的可靠性。

参考文献:

[1] Gibson G A, van Meter R. Network Attached Storage Architecture[J]. Communications of the ACM, 2000, 43(11): 37-45.

[2] Liao Heng. Storage Area Network Architectures[R]. Technology White Paper, PMC-Sierra, Inc, 2003, 41: 84-90.

[3] Mesnier M, Ganger G R, Riedel E. Object-Based Storage

[J]. IEEE Communications Magazine, 2003, 41: 84-90.

[4] Gibson G, Nagle D, Amiri K, et al. A Cost-Effective, High-Bandwidth Storage Architecture[C]// Proc of the ACM 8th Int'l Conf on Architectural Support for Programming Languages and Operating Systems, 1998: 92-103.

[5] T10 Work Group. SCSI object-based storage device command revision 1.0 [S]. T10/1355-D Working Draft, 2004.

[6] Panasas. Panfs: Object-Based Architecture[R]. Panasas Technology Whitepaper, 2004.

[7] Braam P J. The Lustre Storage Architecture[R]. Lustre Technical Report, 2002.

[8] Weil S, Brandt S A, Miller E L, et al. Long, Carlos Maltzahn, Ceph: A Scalable, High-Performance Distributed File System[C]// Proc of the 7th Conf on Operating Systems Design and Implementation, 2006.

[9] Patterson D A, Gibson G, Katz R H. A Case for Redundant Arrays of Inexpensive Disks (RAID)[C]// Proc of ACM SIGMOD, 1988: 109-116.

[10] 刘仲. 基于对象存储结构的可伸缩集群存储系统研究[博士学位论文][D]. 长沙: 国防科学技术大学, 2005.

(上接第 125 页)

FCMC 除有一个极小值对应 *Err* 的一个峰值外, 其他三个极小值均对应了 *Err* 的三个极小值, 其中包含最小值。

总体而言, *FCMC* 较 *KTA* 能更好地反映 *Err* 的变化, 进而能作为较好的核参数以及核矩阵组合系数模型选择标准。

5 结束语

针对组合核矩阵缺乏完善的模型选择标准问题, 本文提出了一种特征距离模型选择标准。该标准兼顾了特征的类别内距离和类别间距离, 将 FSM 标准扩展到组合核矩阵研究, 克服了核目标匹配标准的冗余性, 并在实验中表现出了良好的性能, 为选择最优矩阵组合择最优矩阵组合系数提供了依据, 也为进一步研究核模型选择组合方法奠定了基础。进一步工作包括: 设计式(3)的高效求解算法, 研究大规模标准数据集上组合模型选择问题新标准的应用方法。

参考文献:

[1] Vapnik V N. The Nature of Statistical Learning Theory [M]. New York: Springer, 1995.

[2] Amari S, Wu S. Improving Support Vector Machine Classifier by Modifying Kernel Function [J]. Neural Networks, 1999, 12(66): 783-789.

[3] Crammer J K, Elisseeff A, Shawe-Taylor J. Kernel Design Using Boosting [C]// Advances in Neural Information Processing Systems 14, 2002: 537-544.

[4] Charles A M, Pontil M. Learning the Kernel Function via Regularization[J]. Journal of Machine Learning Research, 2005, 6: 1099-1125.

[5] Lanckriet G, et al. Learning the Kernel Matrix with Semi-definite Programming[J]. Journal of Machine Learning Research, 2004, 5: 27-72.

[6] Cheng S O, Smola A J, Williamson R C. Learning the Kernel with Hyperkernels[J]. Journal of Machine Learning Research, 2005, 6: 1043-1071.

[7] Lewis D, Jebara T, Noble W. Nonstationarly Kernel Combination [C]// Proc of the 23rd Int'l Conf on Machine Learning, 2006: 553-560.

[8] Nguyen C H, Tu B H. Kernel Matrix Evaluation [C]// Proc of the 20th Int'l Joint Conf on Artifficial Intelligence, 2007: 987-992.

[9] Cristianini N, et al. On Kernel-Target Alignment [J]. Journal of Machine Learning Research, 2002, 1: 1-31.

[10] Kandola J, Shawe-Taylor J, Cristianini N. Optimizing Kernel Alignment over Combinations of Kernel [R]. Technical Report NC-TR-2002-121, 2002.

(上接第 139 页)

注: $ PBS_NODEFILE Torque 分配的计算节点名称, Gaussian 需要节点文件来做并行;

输出文件部分如下:

```
* * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * *
Gaussian 03: IA32L-G03RevC.02 12-Jun-2004 12-Sep-2007
* * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * *
%NprocLinda= 8(采用 Linda 方式)
Will use up to 8 processors via Linda.
JHJP TEST 3-21G SCFDM
-------------
```

该作业在这套高性能集群中提交后, 申请四个节点, 每个节点两个 CPU, 一共八个进程, 因此 *NprocLinda*= 8, 作业顺利完成业。这说明, 利用 Rocks 搭建的集群并行计算效率得到提高。

7 结束语

本文提出的利用 Rocks 搭建高性能集群平台, 为今后在此集群上更多的高性能计算应用打下基础, 同时也提供了一种快捷、可靠的构建集群的方法。

参考文献:

[1] 车静光. 微机集群组建、优化和管理[M]. 北京: 机械工业出版社, 2004.

[2] http://www.scyld.com/home.html.

[3] Sacerdoti F D, Chandra S, Bhatia K. Grid Systems Deployment & Management Using Rocks[C]// Proc of the IEEE Int'l Conf on Cluster Computing, 2004.

[4] Bruno G, Katz M J, Sacerdoti F D, et al. Rolls: Modifying a Standard System Installer to Support User-Customizable Cluster Frontend Appliances[C]// Proc of the IEEE Int'l Conf on Cluster Computing, 2004.

[5] Rocks Cluster Distribution: Users Guide[M]. UC Regents, 2004.

[6] Sacerdoti F D, Katz M J, Papadopoulos P M. 411 on Scalable Password Service[C]// Proc of the IEEE High Performance Distributed Computing Conf, 2005.