基于网络附属对象设备的集群存储体系结构*

张悠慧+, 郑纬民

(清华大学 计算机科学与技术系 高性能计算技术研究所,北京 100084)

A Cluster Storage Architecture on Network-Attached Object-Based Devices

ZHANG You-Hui, ZHENG Wei-Min

(Institute of High Performance Computing Technologies, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

+ Corresponding author: Phn: 86-10-62783505, Fax :86-10-62771138, E-mail: zyh02@tsinghua.edu.cn

http://www.cs.tsinghua.edu.cn

Received 2002-05-13; Accepted 2002-08-22

Zhang YH, Zheng WM. A cluster storage architecture on network-attached object-based devices. *Journal of Software*, 2003,14(2):293~299.

Abstract: With the ever-increasing development of Internet, today's data-centric applications require storage platforms not only to possess some properties including high capacity and high scalability, but also to support structured data directly, which is beyond the abilities of current file systems and DBMS. In this paper, a model of network-attached object-based storage devices is presented. It provides storage and index interfaces for structured data employing the power of embedded processors, which eliminates the bottleneck of conventional storage systems. Then the architecture of cluster storage based on the model OStorage is proposed. It introduces the uniform storage of data and meta-data as well as query-locating mechanism to adapt storage hierarchies to structured data. These two methods improve the scalability and availability of the storage system. The prototype of Ostorage is implemented. The experimental results show that the throughput of it increases with the system scale linearly.

Key words: structured data; object-based device; cluster storage

摘 要: 随着 Internet 的发展,应用的数据存储量与其增长速度都相当高,同时数据具有结构化特点,当前的(分布式)文件系统与数据库系统都无法较好地满足这一类需求.提出了一种网络附属对象存储设备模型,利用自身处理器的能力,提供结构化数据的存储/检索接口,消除了传统存储系统的服务器瓶颈问题.同时提出了基于该对象设备的集群存储体系——OStorage.它利用集群网络方式,实现了数据/元数据统一存储与查询式数据访问机制.其在系统的可扩展性、可用性与对结构化数据的支持上,均较符合当前存储应用的特点.实现了该体系的原型系统.测试结果表明,其吞吐率随规模的扩大呈线性增长.

关键词: 结构化数据;对象设备;集群存储

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2001AA111010 (国家高技术研究发展计划); the National Grand Fundamental Research 973 Program of China under Grant No.G1999032702 (国家重点基础研究发展规划(973))

第一作者简介: 张悠慧(1974－),男,浙江宁波人,博士,讲师,主要研究领域为网络存储,并行处理.

中图法分类号: TP393 文献标识码: A

随着 Internet 的发展,电子邮件、个性化网站、数字图书馆、远程教育、远程医疗等 Internet 信息服务日益成为人们工作、生活中必不可少的部分.这些服务所管理的数据具有一定的规则结构,往往进行比较简单的保存、读取、查询小数据集等操作.这些结构化数据的存储量是极大的,而且增长速度非常快.但目前最常用的存储系统,如文件系统(包括分布式文件系统,如 GFS[1], Archipelago[2]等)与数据库,却不能满足这类要求.

文件系统是操作系统提供的基本服务之一,但其本身只提供无结构数据流存储接口,没有完整的操作原子性与意外恢复的能力.这样,当应用设计者在集群环境下搭建应用时,就会面临分布式数据命名、存储、数据一致性维护等问题.在整个系统实现中,真正与应用逻辑相关的工作量并不多,这种开发方式是低效的.

关系数据库是很成熟的技术,而且提供严格的 ACID 语义保证,关系模型能在一定程序上满足结构化数据的要求.但关系数据库的结构过于复杂,处理查询语言的开销也很大,对于某些应用来说提供了很多没有必要的复杂功能,扩展性和管理性受到很大的限制[3].面向对象数据库也存在功能复杂、可扩展性低等体系结构方面的问题.

为了解决此类问题,一些机构进行了相关研究,提出结合文件系统与数据库优点,实现结构化数据/对象数据存储平台这一思想,包括以下几个典型的研究项目:

Lore[4]是 Stanford 大学研发的,最初是依照对象交换模型研究此类数据的管理系统,后来(1998 年)全面转向支持 XML,主要的研发重点是为 XML 定义数据模型与查询语言.但只支持 XML 数据的存储与处理,其应用面过于狭窄.

Porcupine 中的 Transactional Record Store(TRS)[3].Porcupine 是华盛顿大学研发的超大规模的集群 E-mail 系统,TRS 是其中的单结点存储子模块,提供了面向记录的数据存储功能.与 Lore 一样,是针对特定数据类型与应用的.

DDS[5]是加州大学 Berkeley 分校的 Ninja 计划的一部分,是一个面向 Internet 服务的基于集群的存储层.DDS 实现了集群环境下对应用透明的数据存储和复制功能,但仅支持哈希表这一种数据结构.

较早期的项目还有 Shore[6]等.但这些存储系统基本上都是以文件系统作为存储底层,整个存储体系结构仍是传统的块设备-文件系统-应用接口这一模式.这样,底层数据结构与应用数据因不匹配而带来的"服务器瓶颈问题[7]"依然存在,制约了系统性能与扩展性的提高.

我们遵循设备智能化与存储网络化这两个发展趋势,提出了基于对象设备的集群存储——OStorage,从存储体系结构角度来研究与解决这类问题.与一般的方案相比较,它具有以下特点:

(1) 以网络附属对象存储设备作为存储底层,消除了传统存储系统的服务器瓶颈问题.

(2) 设计了数据与元数据的统一存储与查询式定位机制,使得存储体系中各层次的接口与结构化数据相适应,扩大了传统的数据命名与定位的涵义,更符合应用逻辑.

(3) 采用了 Peer-Client 方式与分层结构,存储容量、网络带宽与处理性能的扩展相互独立.(4) 以对象数据作为存储单位,构造统一的对象空间,方便地实现面向对象的应用存储接口.

我们实现了 OStorage 的原型系统.该系统向开发者提供透明的面向对象存储接口,显著地减少了存储应用的开发难度,能够较好地适用于 IO 密集型的网络存储应用.测试表明,其吞吐率随着规模的扩大呈线性增长.

本文第 1 节介绍网络附属对象存储设备的设计思想与可行性分析.第 2 节描述了 OStorage 的整体结构、数据管理与分布式操作机制.第 3 节介绍了原型系统的实现与特点.第 4 节给出了原型系统的扩展性/吞吐率测试.最后是对下一步工作的介绍.

1 网络附属对象存储设备

1.1 设计思想

在传统的结构化数据/对象数据存储平台中,使用的存储底层一般都是块设备与(分布式)文件系统.这种模

式带来了“服务器瓶颈问题”,即服务器上集中进行的数据定位、访问控制与数据解析成为性能瓶颈.

近年来,用于 IO 处理的芯片性能突飞猛进,为设计功能更强的智能存储设备奠定了硬件基础.此类存储设备与传统块设备的最大差别在于,前者提供了更为高级的数据抽象与访问接口.CMU 的 NASD[8],HP 的 Attribute Based Storage[9]是典型的研究,这些研究项目与其他相关研究一起提出了对象存储(object-based storage)的概念.这里的对象指的是在存储设备内部实现的类似于文件系统中 Inode 那样的结构,从而将文件服务器上的部分功能转移到存储设备上,以提高系统性能与扩展性.

我们借鉴了这些设计思想,更进一步地提出了在存储设备内部直接支持结构化数据的想法,从而给出了网络附属对象存储设备的概念.这里的对象指的是非定长的结构化类型数据.

1.2 设备结构

网络附属对象存储设备(简称对象设备)充分利用存储设备处理器能力,提供结构化数据存储接口,在内部实现非定长结构化数据的物理定位与存取,并支持数据检索功能.实际上是将传统的对象存储服务器上的部分负载移至存储设备,即所谓的“将计算向数据移近”对象设备的内部结构设计如图 1 所示.

目前许多数据处理应用的特点是处理的数据量远小于最终结果的数据量,而且需要对数据进行分析.如果在存储服务器直接处理原始数据之前,先在各个设备上进行数据预处理以降低数据传输量,则有可能会带来整体性能的提高.因为其大大降低了数据传输开销,同时将数据处理并行化.

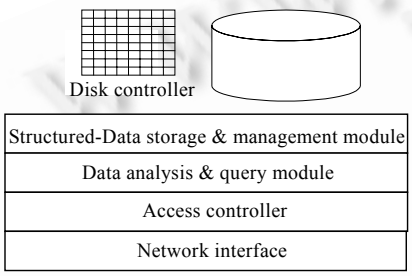

Fig.1 Network attached object-based storage device architecture

图 1 网络附属对象存储设备的内部结构

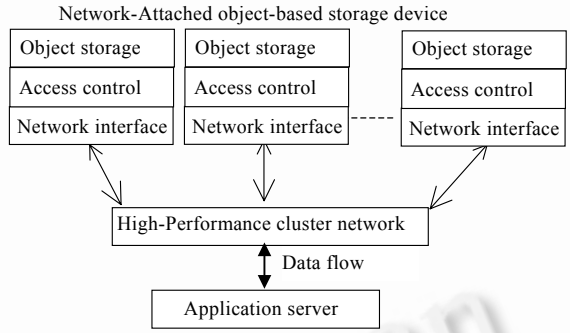

Fig.2 Distributed data access model on object-based devices

图 2 基于对象设备的的分布式数据访问模型

1.3 数据访问模型

该模型的结构如图 2 所示.图中的应用服务器是运行 IO 密集型应用的结点,其接收特定的请求,从存储设备读取相关数据进行分析处理,将最终结果传回请求端.

我们认为,整个存储系统所处理的原始数据量要远小于最终结果数据量.如果存储设备提供的是块数据接口或文件系统接口,那么几乎所有的数据都必须先被读取,集中到服务器后再进行处理以得出最终结果.如果能够利用对象存储设备对数据进行预处理,就能够减轻服务器与网络的负载.本节引入了几组参数来描述系统的数据处理模型.

应用参数:

原始数据量:D_{in};结果数据量:D_{out};平均单字节处理所需的 CPU 周期:d;利用面向对象设备的总运行时间与吞吐率:T_i, $THROUGHPUT_i$;块设备的运行时间吞吐率:T, $THROUGHPUT$.

系统参数:

服务器的主频:P;存储设备处理器主频:P_i;存储设备的数据读取速率:R;单一设备的网络接口速率:N;设备数:I.

辅助参数:

$a=D_{in}/D_{out}>1$; $b=P/P_i>1$.

假设数据处理的复杂度较小(也即 d 很小),同时所需的系统资源也较小,这样对于数据的处理可以分布到对象设备上进行,而服务器只负责任务的分发与结果后处理,又假设这两项任务的时间消耗可以忽略不计(这不包括结果收集时的网络传输开销).

这样,可以由以下公式给出 T_i 与 T:

$$T_i=\max\left(\frac{D_{\text{in}}/I}{R},\frac{D_{\text{out}}}{N},\frac{(D_{\text{in}}/I)*d}{P_i}\right),$$

$$T=\max\left(\frac{D_{\text{in}}/I}{R},\frac{D_{\text{in}}}{N},\frac{D_{\text{in}}*d}{P}\right),$$

即在应用程序流水化非常理想的情况下,处理时间取决于硬盘数据读取速率、传输数据速率与处理速度的最小值.而系统的吞吐率为

$$THROUGHPUT_i=\frac{D_{\text{in}}}{T_i}=\min\left(R*I,a*N,\frac{P*I}{d*b}\right),$$

$$THROUGHPUT=\frac{D_{in}}{T}=\min\left(R*I,N,\frac{P}{b}\right).$$

下面针对各种情况进行讨论.

(1) 应用服务器在使用非对象设备时是 Disk-Bound(磁盘读取慢于网络传输与 CPU 处理)的,即 $R*I$ 值最小.在这种情况下,

$$\frac{THROUGHPUT_i}{THROUGHPUT}=I/b,$$

即最终结果取决于各个对象设备的处理能力之和是否能够超过服务器的处理能力.

(2) 应用服务器在使用非对象设备时是 CPU-Bound,即 P/d 最小.在这种情况下,

$$\frac{THROUGHPUT_i}{THROUGHPUT}=\frac{I}{b},$$

与上述(1)一样,取决于各个对象设备处理能力之和是否能够超过服务器的处理能力.

(3) 应用服务器在使用非对象设备时是 Network-Bound,即 N 最小.在这种情况下,

$$\frac{THROUGHPUT_i}{THROUGHPUT}=\min\left(\frac{R*I}{N},a,\frac{P*}{d*b*N}\right).$$

显然,$\frac{R*I}{N}>1$,$a>1$,$\frac{P*I}{d*b*N}>I/b$.最终结果还是取决于各个对象设备的处理能力之和是否能够超过服务器的处理能力.而当前许多较大规模数据存储系统的 I/b 均大于 1.

2 集群存储体系结构——Ostorage

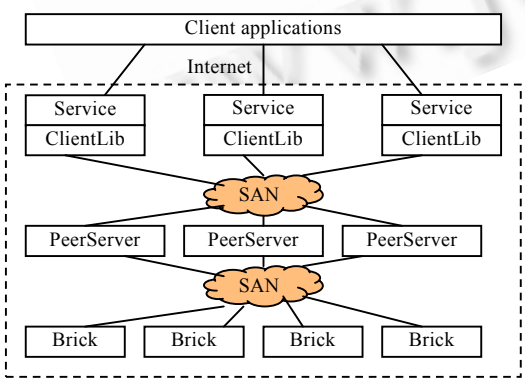

Fig.3 OStorage system architecture

图 3 OStorage 的系统结构

OStorage 是我们提出的基于对象设备的集群存储体系,由多层运行于系统域网(system area network)中的若干个服务结点与网络对象存储设备构成,最小存储单位是数据对象,称之为 OItem,可以在系统中被存取、查询、复制以及删除等.整个存储体系如图 3 所示.

- 若干个由高速网络相连的 Brick(即网络对象存储设备)构成 OStorage 中的存储结点层,向上层提供了基本 OItem 的读写、创建、删除、查询等功能.
- 由高速网络相连的多个 PeerServer 构成了服务入口层,每个 PeerServer 都对外部的客户端(即图中的 Service)提供了所有内部存储数据的系统一致性镜像(single image).

ClientLib 是提供给开发者的 API 库,用于访问 OStorage

提供的对象存储功能.

2.1 对象数据组织

OItem 由对象标识 OID、对象本身的数据 OValue 与对象类型 OType 这样一个三元组来表示.任意数据对象都可以通过全局惟一的 OID 来存取.OID 包括 OItem 所存储的 Brick 的标识符——BID;同时包括 OType 标识符——CLID,是 OStorage 对于所支持的对象数据类型的惟一标识.最后 48 位是 Brick 用于标识本地数据对象的序列号.

OStorage 中的各层都直接支持对于 OItem 的存取,OStorage 支持 Integer, Float, Double, Byte, Char, Short, Long, String, Date 等数据类型,并支持数组类型,均由系统预先设定的 CLID 标识.

OStorage 提供动态增加对象类型的功能——上层应用可以按照 OStorage 给出的对象类型描述规范,生成新的与类型描述相关的 OValue,以插入一个普通对象的方式将此 OValue 及相关的 OID 写入各个 Brick.同时支持对象类型的继承关系.

OValue 本身的结构则是依照类型中定义的域顺序,将各个域值顺次存储下去——如果是基本类型或基本类型的数组,则直接存值;否则将创建一个新的对象,而本身存储的只是此对象的 OID,也可以认为是存储了另一对象的引用.

利用对象设备的处理能力,Brick 能够根据对象的类型描述数据来解析对象内容本身,取得各个域的值,并进行比较等操作.

2.2 对象元数据与数据的统一存储与查询式数据定位

OStorage 利用底层对象设备实现了各层对于结构化数据的直接支持,而无须像传统的存储体系那样,需要在服务器上实现对象映射机制.利用这一点,可以实现对象元数据与数据的统一存储.这里的元数据包括两类,一是数据名称、属主、创建与访问时间以及存放位置等.二是对象数据的类型描述,即 OType.系统处理类型描述的方式与处理数据本身一样.而对于第 1 类元数据,OStorage 利用了对象设备的能力,将其归结为对象的属性.

我们认为,这种管理方式适合集群系统各个结点对等的特点——每个 Brick 可以完全独立地处理本地数据,不需要与其他结点交互.这样带来了两个优点,既可以增加了系统扩展性,又增强了系统可用性,任何一个 Brick 的故障只会影响到本地数据.

与对象数据统一存储相适应的是查询式数据定位机制,PeerServer 为 Service 提供了根据 CLID 的名字,或者根据相应的数据类型内部某个(些)域的条件,通过查询操作取得数据对象集合的功能.查询条件由多个最基本的数据过滤条件,以“与或”关系符连接而成.将这种查询操作分布到各个 Brick 上,可减少结果数据的传输量.

3 系统原型

我们实现了该结构的原型——TODS(Tsinghua object data storage)系统,它利用对象设备的模拟实现作为存储底层,实现了第 2 节所述的技术要点.TODS 的硬件平台是运行 Linux 系统的由百兆以太网与 Myrinet[10]互连的高性能 PC 服务器集群,除了模拟对象存储设备外全部由 Java 实现.

TODS 对外提供海量的结构化数据存储能力以及事务处理功能,实现了透明的数据分布、缓存、检索、意外恢复等机制.其结构特点如下:

- 采用了 Peer-Client 方式,即 PeerServer,Brick,与基于 TODSLib 的应用服务程序可以运行在一个结点上,而且 PeerServer 与 Brick 可以运行于同一进程空间内.
- 在网络结构上采用交换式结构,即 PeerServer 与 Brick 通过存储子网形成了一种交换式结构,任何一个 PeerServer 可以根据应用的需求管理任意多个 Brick,能够将硬件性能发挥到极限.

TODS 的 ClientLib 层实现了透明的面向对象开发接口,符合 SUN 提出的 Java 对象存储接口标准——JDO[11].JDO 向开发者提供了完全透明的对象持久化机制——开发者的程序源代码不必作修改;并引入了可替代 SQL 的查询机制,使用者能够用类 Java 风格的代码实现数据查询.使用 JDO 接口,开发者可以将大部分的工作集中于与应用逻辑相关的部分,而不必去实现数据存储、查询、事务处理等常用功能.根据我们在 TODS 上

的开发经验,利用 JDO 接口实现对象存取与关系表现的代码量,只占一般使用 JDBC 接口的 1/10.

4 性能测试

本节给出了 TODS 的可扩展性与性能测试,测试环境由 36 台高性能 PC 服务器构成,每台服务器配置有 4 个 PIII 700Mhz 处理器,1G 内存与两个 40GSCSI 硬盘(10000 RPM),由百兆以太网互连.操作系统为 Red Hat Linux 7.2,核心版本 2.4.7.TODS 的 Java 部分使用的是 SUN 公司的 JDK1.4.0-b92.

4.1 理想测试

此项测试的目的是给出 TODS 的通信性能与协议开销,在没有引入磁盘操作的情况下,测试其理想的可扩展性,即 Brick 收到读请求后立即返回相应长度的对象数据,而并不进行真正的磁盘读取.在每一台结点上运行了 Brick,PeerServer 与客户端测试程序,测试程序均与本地的 PeerServer 建立连接,而 PeerServer 则向所有的 Brick 注册自身.读取的对象数据大小为 1K 字节,我们记录了使用不同个数结点时的 TODS 吞吐量,如图 4 所示.明显地,TODS 具有良好的扩展性,其吞吐量随着系统结点个数的增长呈线性增长.另外进行了对象读取性能测试,使用了所有 36 个结点,测试结果如图 5 所示.

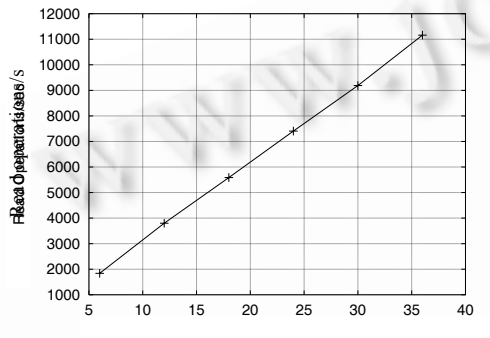

Fig.4 Ideal read scalability test

图 4 理想的读取可扩展性测试

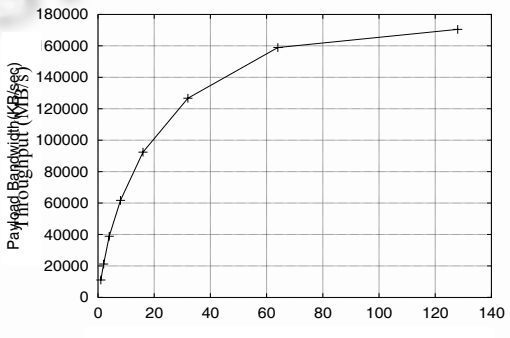

Fig.5 Ideal read throughput test

图 5 理想的读取吞吐量测试

4.2 磁盘读取测试

我们首先在每个 Brick 上分别写入大小不同的对象,每类的个数为 5 000,然后通过客户端测试程序随机地读取这些对象;同时关闭了 TODS 中的缓存功能以显示磁盘操作对性能的影响.测试结果如图 6、图 7 所示.实际的情况应该更好,因为数据访问具有一定的本地性,而且缓存的引入可以较大幅度地提高系统吞吐率.

5 下一步工作

在下一步的工作中,我们将研究对象存储设备的具体实现——如何依据当前硬件的性能、结构与接口标准,实现嵌入式对象存储设备.同时将研究基于 Web 的大跨度的数据存储,随着高速 Internet 的进一步发展与信息交流的日益频繁,这将成为一个有意义的课题.

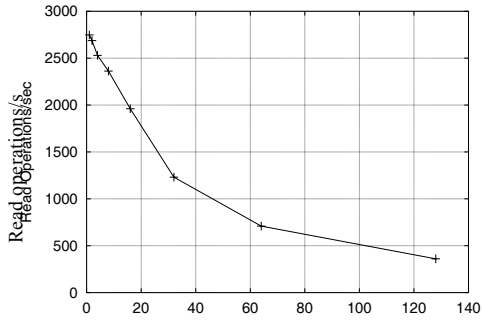

Fig.6 On-Disk read throughput test

图 6 磁盘读取的吞吐量测试

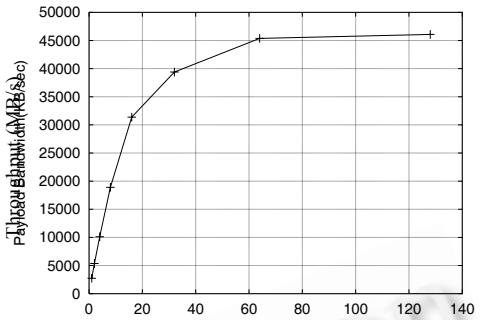

Fig.7 On-Disk read payload test

图 7 磁盘读取的网络流量测试

References:

[1] Preslan KW, Barry AP, Brassow J. Scalability and failure recovery in a Linux cluster file system. In: Salus P, ed. Proceedings of the 4th Annual Linux Showcase and Conference. Atlanta: ACM Press, 2000. 8~10.

[2] Ji M, Felten E, Wang R, Singh JP. Archipelago: an island-based file system for highly available and scalable internet services. In: Chen JB, Draves R, eds. Proceedings of the 4th USENIX Windows Systems Symposium. Washington: USENIX Press, 2000. 37~48.

[3] Grimm R, Swift MM, Levy HM. Revisiting structured storage: a transactional record store. Technical Report, UW-CSE-00-04-01, Department of Computer Science and Engineering, University of Washington. 2000. http://www.cs.washington.edu/homes/rgrimm/papers/tr00-04-01.pdf.

[4] Goldman R, McHugh J, Widom J. From semistructured data to XML: migrating the lore data model and query language. In: Maedche A, Sattler K-U, eds. Proceedings of the 2nd International Workshop on the Web and Databases. Philadelphia, Pennsylvania: USENIX Press, 1999. 12~20.

[5] Gribble SD, Brewer EA, Hellerstein JM. Scalable, distributed data structures for Internet service construction. In: Jones MB, Kaashoek F, eds. Proceedings of the 4th Symposium on Operating Systems Design and Implementation. San Diego: USENIX Press, 2000. 68~76.

[6] Carey MJ, DeWitt DJ, Franklin MJ. Shoring up persistent applications. In: Snodgrass RT, Winslett M, eds. Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data. Minnesota: ACM Press. 1994. 383~394.

[7] Gibson GA, Nagle DF, William, CII. NASD scalable storage systems. In: Rubin A, Antonelli C, eds. Proceedings of the USENIX'99, Extreme Linux Workshop. Monterey: USENIX Press, 1999. 121~130.

[8] Gibson G. File server scaling with network-attached secure disks. In: Leutenegger S, ed. Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems. Washington: ACM Press, 1997. 272~284.

[9] Shriver E. A formalization of the attribute mapping problem. Technical Report, HPL-1999-127, HP Labs, 1999. http://www.hpl.hp.com/techreports/1999/HPL-1999-127.pdf.

[10] GM: the low-level message-passing system for Myrinet networks. Technical Reports, Myricom Company. 2001. http://www.myricom.com/scs/index.html.

[11] Russell C. JavaTM data objects specification overview. In: Jordan MJ, ed. Proceedings of the Sun's 2000 Worldwide Java Developer Conference. San Francisco, CA: Sun Press, 2000. 34~41.