HANDY: 一种具有动态扩展性的集群文件系统

程斌* 金海 李胜利 邵志远
华中科技大学计算机科学与技术学院 武汉 430074
Email: showersky@hust.edu.cn

摘要：针对当前集群文件系统只支持静态扩展、不提供高可用性等问题，我们实现一种支持节点动态扩展的集群文件系统 HANDY。本文提出一种基于全局哈西表的动态扩展协议——GHT，给出协议的设计、相应算法和分析。最后的实验数据证明，采用 GHT 的集群文件系统 HANDY 具有良好的动态扩展性和可用性，减小系统的管理和维护开销。

关键字：集群，文件系统，动态扩展，高可用
中图分类号：TP393

HANDY: A Cluster File System with Dynamic Scalability

CHENG Bin, JIN Hai, LI Sheng-Li and SHAO Zi-Yuan
(College of Computer Science & Technology, University of HUST, Wuhan 430074)

Abstract: Many existing cluster file systems suffer from static scalability and bad availability, so we implement a new cluster file system with dynamic scalability, called HANDY. In this paper, we proposal a dynamic scalability protocol based on global hash table and describe its design, algorithm and the scaling procedure. The final results show that HANDY has good dynamic scalability and high availability, alleviating the management and maintenance consuming significantly.

***Key words*:** Cluster, File System, Dynamic Scalability, High Availability

1. 引言

当前数据资源的极度膨胀使得服务器存储系统面临越来越严峻的挑战，如何高效、快速、可靠地存储日益增长的用户数据成为我们关注的重点。然而根据硬件的发展趋势，网络传输速度与磁盘存取速度之间的鸿沟会愈来愈大，服务器的本地存储瓶颈也会越来越严重，于是研究基于高速网络的集群文件系统具有重要意义。

可扩展性历来是集群文件系统的一个重要评价指标。根据扩展过程中是否需要停止原系统，本文将文件系统的扩展性分为两种：静态扩展和动态扩展。静态扩展是指系统增减节点时，需要先停止当前运行的系统，然后根据节点增减的情况进行重新配置，最后重新启动整个系统，使之得以正常工作。动态扩展则是一种在线扩展，系统增减节点时能够自动适应这种变化，自动完成资源的迁移和重新配置，在无人干涉的情况下进行自动管理和自我维护。

根据调查发现，当前的多数集群文件系统都只实现静态扩展，如 PVFS，DCFS。LUSTRE 只实现数据节点的自动增加，但不能适应节点减少的变化情况。静态扩展不需要考虑数据容错、分布式的元数据管理、资源动态分配等问题，因而在实现上相对容易，但由此也带来一些的问题，如系统扩展过程繁琐、可用性不高、管理和维护困难等。当节点数增加到几十或几百时，静态扩展的繁琐程度将使管理员难以忍受，因为中间任何环节出错都将导致系统扩展失败。另外，在此期间整个系统不能对外提供存储服务，这对于某些可用性要求较高的集群服务器来说是致命的。

项目来源：国家高技术研究发展计划“863”项目 (No. 2002AA1Z2102)

针对上述问题本文提出一种基于逻辑矢量环的集群文件系统 HANDY，实现系统的动态扩展。首先介绍 HANDY 的系统结构，随后重点讨论 HANDY 中基于全局哈西表（GHT）的动态扩展协议的设计与实现，并在此基础上分析 HANDY 系统的正反向扩展过程和相关的优化策略，最后给出系统原型的实验数据以及相关的结果分析。

2. HANDY 集群文件系统

2.1. 系统结构

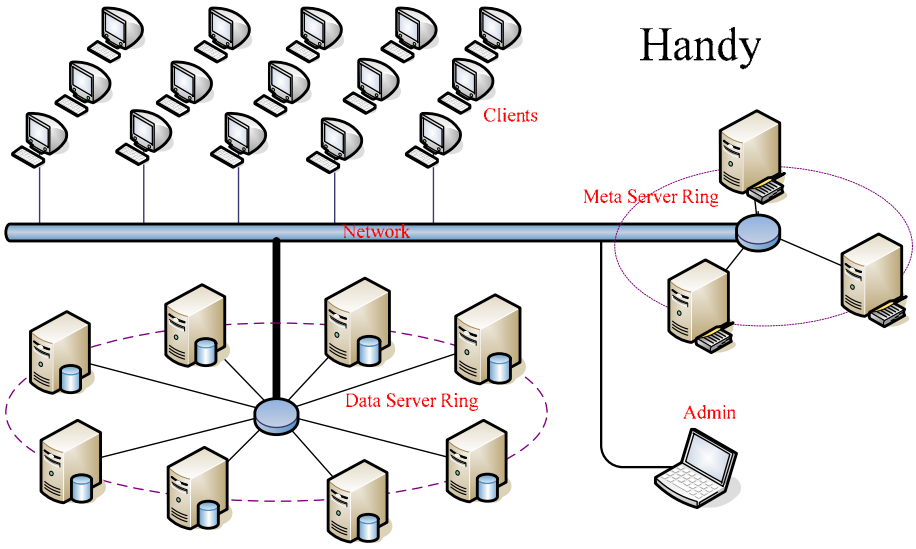


图 1　系统结构图

如图 1 所示，整个系统分为四大组成部分：元数据服务器池、数据服务器池、客户端、管理控制台，各部分之间通过高速网络互联，如 1000Mb 以太网、Myrinet 等。其中元数据服务器、数据服务器通过动态扩展协议各自维护一个逻辑矢量环，以表示节点之间的逻辑关系。每个元数据服务器通过 Berkeley DB 存放元数据信息，包括文件和目录的访问权限、创建修改时间、目录项、存放策略、分片文件的 Handle 列表等；每个数据服务器主要存放文件的数据分片；客户端为上层应用程序提供集群文件系统 3 种形式的访问接口：VFS 内核虚拟文件系统的访问接口、库函数访问、ROMIO 并行程序访问接口；管理控制台部分提供 HANDY 文件系统的监控管理工具，查看各个服务器的资源利用情况、节点逻辑关系、统计信息以及数据和元数据的读写带宽等。

与其它系统的不同之处在于，HANDY 集群文件系统在元数据和数据服务器的管理上采用基于逻辑环的分布式管理策略。它利用逻辑矢量环实现系统的动态扩展和资源的动态管理，采用分布式的元数据管理策略，引入邻接复制技术实现元数据的容错，通过可定制的数据存放策略满足多种应用的数据存储需求。本文重点在 HANDY 动态扩展性的设计与实现算法。

3. 动态扩展

3.1. 动态扩展

根据分布式系统应用环境的不同，可以将动态扩展分为基于广域网的动态扩展和基于局域网的动态扩展。随着 P2P 研究的热化，目前广域网中涌现出各种分布式的动态扩展协议，如 Chord[+]、CAN、PASTRY 等。Chord 协议是一种基于环状结构的 DHT，主要用来实现分布式资源定位，可以用来存放分布式文件系统的元数据信息，如合作式文件系统（CFS）就是基于 Chord 实现。CAN 基于虚拟的 d 维笛卡儿坐标空间实现其数据组织和查找功能，整个坐标空间动态地分配给系统的所有结点，每个结点拥有独立的互不相交的一块区域。Pastry 也是一种用于 p2p 存储的分布式动态扩展协议，每个 Pastry 结点记录结点空间中和它直接相邻的邻居结点。

上述系统实现的均是基于广域网的动态扩展，而集群文件系统需要的是一种基于内部局域网的动态扩展协议，如果直接采用并不能获得好的效率和性能，受这些背景知识的启发，本文根据集群文件系统自身的环境特点，设计了一种适合于内部局域网的动态扩展协议——GHT。（动态扩展的效果，分为数据服务器和元数据服务器）。虽然它是针对集群存储系统而设计，但其思想和方法也可以扩展到集群的其它系统。

3.2. 协议设计

相关定义

Handle

Handle是HANDY系统最为重要的一个概念，是一个 64 位的数值。为了便于实现对象的寻址，我们将Handle的 64 个bit位进行分段，用于表示不同意思。如图 2 所示，前 1 位表示管理该Handle的服务器节点类型，0 表示元数据服务器，1 表示数据服务器；接下来的 9 位表示节点在逻辑环的ID号；最后的 54 位表示节点内部的Handle地址空间，也就是说每个节点可以管理分配的Handle数目大小为 2^{54}。系统中存在如下几种存储对象：目录和文件的元数据信息、目录项信息、分片文件，每个对象均由一个Handle唯一标识。

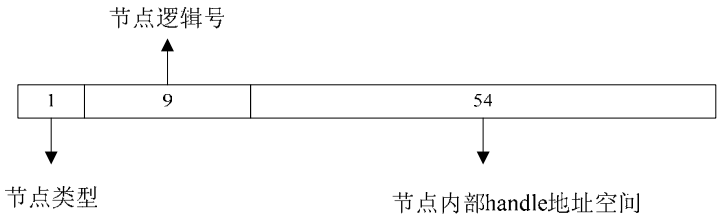


图 2　Handle 的位段表示

ServerMap

该表是GHT协议的核心数据结构，由有序链表构成，记录每个服务器节点负责的Handle范围。系统中所有节点都维护一个ServerMap表，用于定位某个Handle对应的服务器节点。如图 3，有Node1、Node2、Node3 三个节点，Handle空间为[0, 2^3-1]，每个节点负责的Handle范围如图中ServerMap所示。当某个节点需要读取 6 所对应的存储对象时，它首先去查找ServerMap表，查出 6 由Node3 负责，然后才向Node6 发送请求。

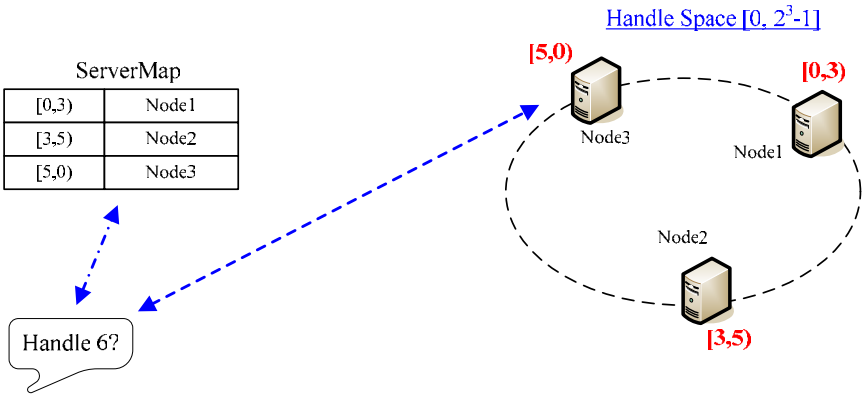


图 3　客户端访问 ServerMap 的过程图

GHT 资源定位的准确性取决于系统中 ServerMap 表和 Handle 实际分布的一致性。HANDY 系统将元数据节点和数据节点组织成两个单独的逻辑矢量环，每个逻辑矢量环负责一段连续的 Handle 值空间，环上节点将一个连续的 Handle 值空间 U 分成 n 个首尾连续的 Handle Range，每个节点有一个唯一的 ID 号，它决定该节点在环上的位置 P，节点之间按照位置由小到大的排列顺序构成一个首尾相连的逻辑环，顺着排列顺序的方向为环的正方向。(说明元数据服务器和数据服务器在 Handle 分配上的区别)

为了便于描述系统资源的一致性，本文给出如下定义：

定义 1 整个HANDY系统的Handle值空间U＝{0, 1, 2, …, $2^{64}-1$}。

定义 2 元数据的矢量环为VR_m，表示的Handle空间为U_m，元数据服务器M_i的ID号为$MN_i \in \{0,1,2,3,\cdots\cdots,2^9-1\}$，在整个$VR_m$中的位置为$MP_i = MN_i << 54 + 2^{54}-1 \in U_m$，负责的Handle段为$MCH_i=(MP_{(i-1)}, MP_i]$，可分配的Handle段为$MAH_i=[MP_i-2^{54}+1, MP_i]$，存放在主目录上的Handle集合为$MCH_i$，备份目录的Handle集合为$BCH_i$，当前保存所有对象的handle集合为$MH_i$。

定义 3 数据的矢量环为VR_d，表示的Handle空间为U_d，数据服务器D的ID号为$DN_j \in \{2^9, 2^9+1, 2^9+2, 2^9+3,\cdots\cdots,2^{10}-1\}$，在整个$VR_d$的位置$DP_j = DN_j << 54 + 2^{54}-1 \in U_d$，负责的Handle段为$DCH_i=[DP_j-2^{54}+1, DP_i]$，可分配的Handle段为$DAH_i= [DP_j-2^{54}+1, DP_j]$。

根据 HANDY 逻辑矢量环的构成特征，有如下结论：

结论 1 $U = U_m \cap U_d$。

结论 2 元数据服务器M_i的ID满足条件：$MN_i \ \& \ 0x1FF = 0$。

结论 3 数据服务器D_j的ID满足条件：$DN_j \ \& \ 0x1FF \neq 0$。

结论 4 元数据服务器环上的 Handle 空间$U_m = \sum_{i=1}^{n} MCH_i$，n 为当前元数据服务器的数目。

结论 5 对于一个元数据服务器 i 而言，它负责的 Handle Range 要大于或等于它可分配的 Handle Range，即$MCH_i \supseteq MAH_i$。

结论 6 对于一个数据服务器 j 而言，它负责的 Handle Range 等于自己可分配的 Handle Range，即$DCH_j = DAH_j$。

结论 7 只要每个节点的 ID 号不一样，那么节点可分配的 Handle Range 就不会有重复，即

$$MAH_{i1} \cap MAH_{i2} = \phi, i1 \neq i2 \text{ ； } DAH_{j1} \cap DAH_{j2} = \phi, j1 \neq j2$$

结论 8 设在元数据服务器构成的环中，当前元数据服务器为 i2, 前续服务器为 i1，后续服务器为 i3，则 i1 负责分配出去的每个 Handle 可以在 i2 的备份目录找到副本，i2 负责的每个 Handle 可以在 i3 的备份目录找到副本，即

$$MCH_{i1} = BCH_{i2} \text{ ； } MCH_{i2} = BCH_{i3} \text{ ； } MH_{i2} = BCH_{i1} \cap MCH_{i2}$$

当上述结论均成立时，称 HANDY 系统的资源分布达到稳定状态，客户端通过 GHT 定位算法能够正确地找到对应的资源，此时称 ServerMap 表和资源分布之间达到一致性。当有新的服务器加入、某个现有节点离开或失效时，系统的稳定性和一致性将受到破坏，需要根据动态变化的情况对资源进行动态管理。

协议格式

消息类型	节点标识**ID**	节点类型	节点**IP**	端口号	**root handle**	负载信息

图 4 协议消息格式

根据动态扩展协议的功能需求，HANDY 系统中 GHT 协议采用的消息格式如图 4 所示，其中前 4 个字节表示消息的类型，主要定义有如下 8 种类型的消息：SERVER_JOIN、SERVER_LEAVE、HEARTBEAT、SERVER_REQUEST、SERVER_REPLY、SERVER_CRASH、CLIENT_REQUEST、ROOT_HANDLE。节点 ID 是发送消息节点的标识号，表明消息的来源。节点类型分为 3 种：Client 客户端，Meta Server 元数据服务器，Data Server 数据服务器。后面的节点 IP 和端口号在加入时用，root handle 字段在服务器询问根目录时用到，最后的负载信息位用来实现系统优化和负载平衡。

消息类型

表 1 消息列表

描述符	**说明**
SERVER_JOIN	服务器加入时发送的通告消息，利用消息后面的字段表明自己的类型、IP 地址、端口号
SERVER_LEAVE	服务器正常离开或程序正常退出时发送的消息
HEARTBEAT	服务器给后续节点发送的心跳消息
SERVER_REQUEST	服务器加入前发送的组播请求信息，用来获取环上当前处于 active 状态的服务器列表
CLIENT_REQUEST	客户端加入前发送的组播请求消息，用来获取元数据和数据服务器的列表，并构建 ServerMap
SERVER_REPLY	服务器发送给客户端或者其它新加入的服务器的回应消息，后面会附加自己的 IP、PORT、类型信息
ROOT_HANDLE	ROOT META SERVER 发送此类消息将根的 handle 告诉新加入的节点
SERVER_CRASH	当前节点检测到前续节点失效时发送的组播消息，用来触发其它节点更新 ServerMap 表

算法描述

服务器节点$node_i$的GHT算法流程如下：

(1) 从消息队列取出一个消息 pMsg，如果消息队列为空则 Sleep；

(2) 获取消息发送者的相关信息，$node_j$ ＝ pMsg->Sender, type = pMsg->Type；

(3) 如果消息类型为SERVER_REQUEST，给$node_j$回应自己的IP、Port、标识ID、Handle范围等信息；如果$node_j$为元数据服务器，则启动主目录元数据的迁移任务，依次执行图 5(a)中 1、2 两步，使得：

$$MCH_i = MCH_i^{'} \cap MCH_j \ ; \ MCH_i^{'} = (P_j, P_i] \ ; \ MCH_j = (P_{(i-1)}, P_j] \ ;$$

另外，如果$node_i$为ROOT META SERVER，则给$node_j$发送ROOT_HANDLE类型的消息，通告根的handle值。

(4) 如果消息类型为SERVER_JOIN，将该节点加入到相应的逻辑环。如果$node_j$是自己的前续，则更新心跳接收时间LastTime；如果$node_j$同时也是元数据服务器节点，则完成备份元数据的迁移任务，依次执行图 5(a)所示的 3、4 两步，使得：

$$BCH_j = MCH_{(i-1)} \ ; \ BCH_i^{'} = MCH_j \ ; \ BCH_{(i+1)}^{'} = MCH_i^{'}$$

(5) 如果消息类型为SERVER_CRASH或SERVER_LEAVE，则从逻辑环中删除该节点的信息。如果$node_j$是自己的前续节点，则更新心跳接收时间LastTime，如果$node_j$同时也是元数据服务器节点，则完成节点失效时元数据的迁移任务，依次执行图 5(b)所示的步骤 1、2、3，使得：

$$MCH_{(i+1)}^{'} = (P_{(i-1)}, P_{(i+1)}] \ ; \ BCH_{(i+1)}^{'} = MCH_{(i-1)} \ ; \ BCH_{(i+2)}^{'} = MCH_{(i+1)}^{'}$$

(6) 如果消息类型为 HEARTBEAT，更新收到心跳的时间 LastTime ＝ CurrentTime；

(7) 如果消息类型为 SERVER_REPLY，取出 Server 的回应信息，将其加入的 ServerMap；

(8) 如果消息类型为CLIENT_QUEST，给$node_j$回应自己的IP、Port、标识ID、Handle范围等信息；

(9) 如果消息类型为ROOT_HANDLE且$node_i$为ROOT_META_SERVER，则给$node_j$发送根的元数据信息；

(10)回到第一步继续循环。

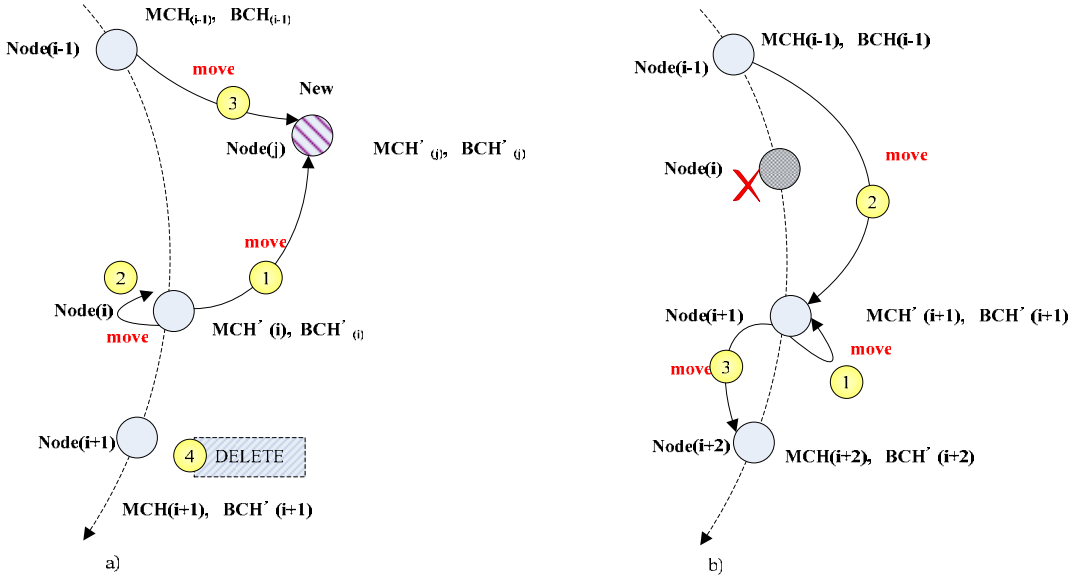


图 5　元数据的迁移过程

另外，每个客户端也会被动的接收各类组播消息以及时更新自己的 ServerMap 表，保证资源定位的一致性，其算法与上类似，但无需做任何数据迁移操作，主要处理

SERVER_REPLY, SERVER_JOIN, SERVER_CRASH, SERVER_LEAVE, ROOT_HANDLE 消息。

3.3. 动态扩展过程

正向扩展过程

如图 6 所示，HANDY 系统的正向扩展包括服务器加入、客户端加入两种。

1） 系统初启时，最先加入系统的节点 A 对外组播 SERVER_REQUEST 消息，在 5 秒中内没有收到任何回应消息，于是将自己作为 ROOT META SERVERE 节点，初始化 HANDY 系统根目录的元数据（如图 6(a)所示）。

2） 随后 Server B 加入，组播 SERVER_REQUEST 消息，收到 A 的 SERVER_REPLY 回应消息和 ROOT META SERVER 的 ROOT_HANDLE 消息(如图 6(b)所示)，从而构建自己的 ServerMap 表，根据自己在 ServerMap 中的位置，建立自己的前后续节点，并给他们发送请求，接管一段 Handle Range，然后根据自己的 Handle Range 和 Root handle 初始化元数据的数据库，最后对外组播 SERVER_JOIN，触发其他的客户端和服务器更新 ServerMap 表（如图 6(c)所示）。

3） 当有客户端 C 加入时，首先发送组播 CLIENT_REQUEST 请求，等待 5 秒接收当前所有服务器节点的 SERVER_REPLY 信息和 ROOT META SERVER 回应的 ROOT HANDLE 信息，然后构建自己的 ServerMap 表和根目录信息。当再有新的服务器节点加入时，客户端的 ServerMap 表也可以被动更新。

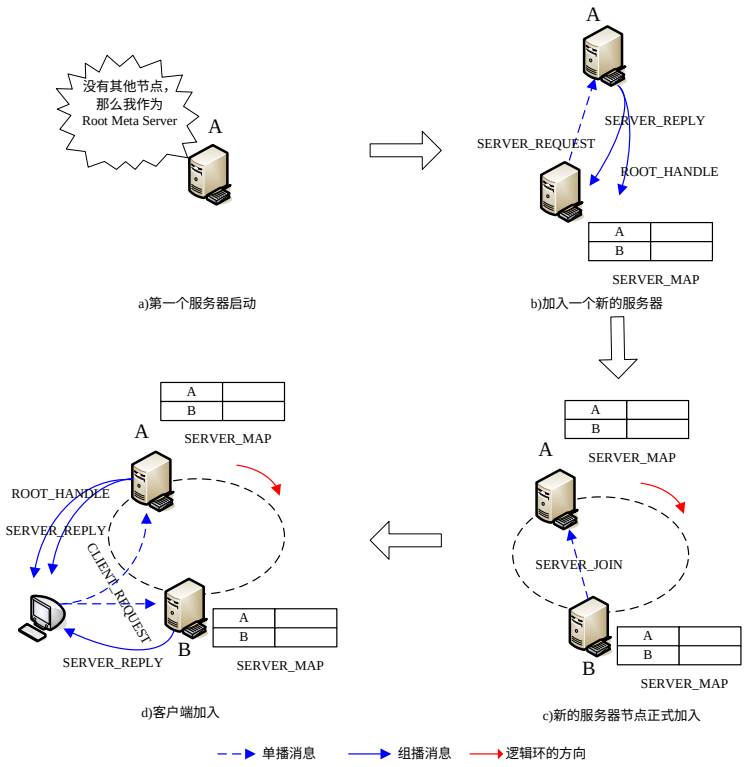


图 6　节点加入过程

反向扩展过程

反向扩展是指服务器节点减少的过程。实现反向扩展需要数据和元数据的容错支持，动态扩展协议的任务只是发现节点失效的异常，触发后续节点更新自己的前续节点信息，合并

Handle Range，并启用备份数据，更新其它节点的 ServerMap 表。如图 7 所示，当 B 失效时，后续的 C 节点发现一段时间没有收到 B 的心跳，断定 B 失效，对外组播消息 SERVER_CRASH，触发所有的 SERVER 和 CLIENT 更新自己的 ServerMap 表，将 B 的 Handle Range 合并到 C。ServerMap 表更新后，原来位于 B 的 Handle 会定向到 C，由于 B 的数据在 C 有备份，所有不会导致用户访问出错。

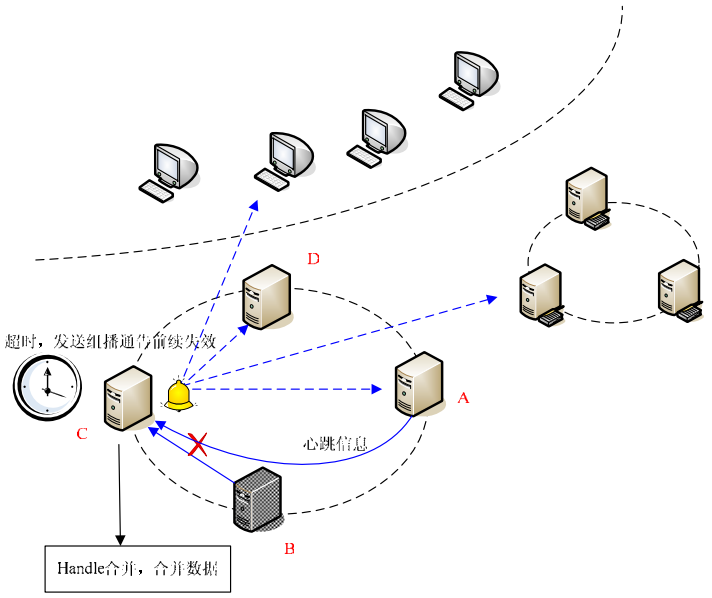


图 7　节点失效过程

4. 性能测试与分析

4.1. 测试环境

如表 2 所示，测试环境是一个 36 个节点的 Cluster 系统，其中包括 12 个数据节点，8 个元数据节点，16 个计算节点，所有节点均安装 Redhat Linux9.0 的系统，采用 100M Fast Ethernet 互连。由 IOZONE 测得本地 EXT2 文件系统的读带宽为 37.5MB/Sec，写带宽为 23.5MB/Sec，由 NETPIPE 测得网络有效传输带宽为 88.56Mb/Sec。测试工具采用通用的文件系统基准测试程序：IOZONE，PostMark，Lmbench。

表 2 测试环境

Items / Role	CPU	MEM	DISK	NIC	NUM	OS	Berkeley DB	GCC
Meta Server	P4 1.6G	256M	40G	100M	8	Redhat Linux 9.0 Kernel2.4	3.2	2.3
Data Server	P4 1.6G	256M	40G	100M	12			
Client	P4 1.6G	256M	40G	100M	16			

4.2. 测试结果及分析

扩展性

HANDY 系统将元数据和数据服务器分离，利用动态扩展协议形成两个独立的逻辑环状结构，实现节点的动态扩展。为了便于分析，本文分别测试了元数据服务器和数据服务器的

可扩展性。

A） 元数据的扩展性测试结果

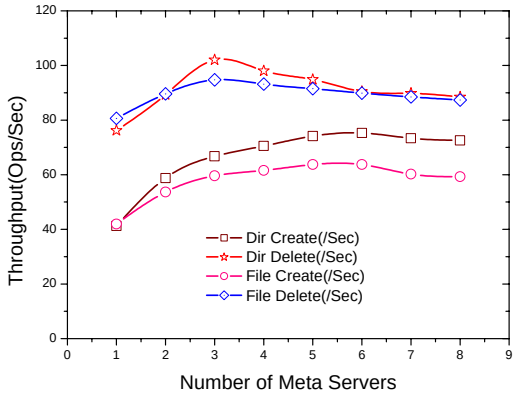


图 11　元数据的吞吐率随元数据节点的变化

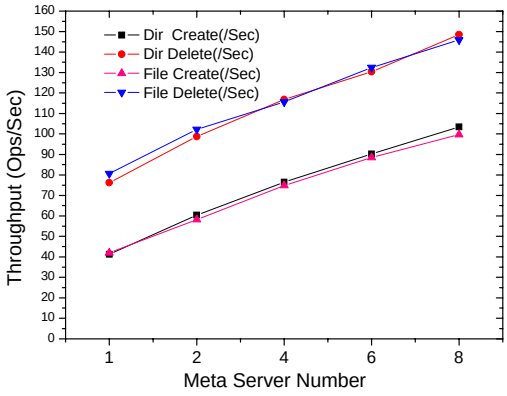


图 12　不同目录下创建和删除的吞吐率

HANDY 元数据服务器的可扩展性主要通过文件和目录创建、删除操作吞吐率变化来体现。测试时保证数据节点数目固定，元数据服务器节点线性增长，测得的单个客户端所取得吞吐率如图 11 所示。从测得的结果看，当元数据服务器数目到达 3 个时，文件和目录删除操作的吞吐率已经达到最大 94.73 个每秒和 102.04 个每秒，之后随着元数据节点的增加吞吐率反而减小；对应文件和目录的创建操作也同样遇到这种情况，当元数据数目达到 6 个时，创建操作的吞吐率也达到最大 63.78 个每秒和 75.29 个每秒。究其原因，发现这种限制主要是因为在同一个目录下做创建和删除操作，任何创建和删除操作都会涉及当前目录元数据的修改，因此存储当前目录元数据信息的元数据服务器成为扩展时的瓶颈。为了证明这种分析的正确性，我们继续测试了客户端在不同目录下运行结果。如图 12 所示，在 8 个节点以内时元数据服务器的扩展性非常明显，但随着节点数的增多，增长的幅度会有所减小，因为分布式元数据会导致用户访问时需要与多个元数据服务器通讯才能完成文件路径的解析。

B）数据服务器的扩展性

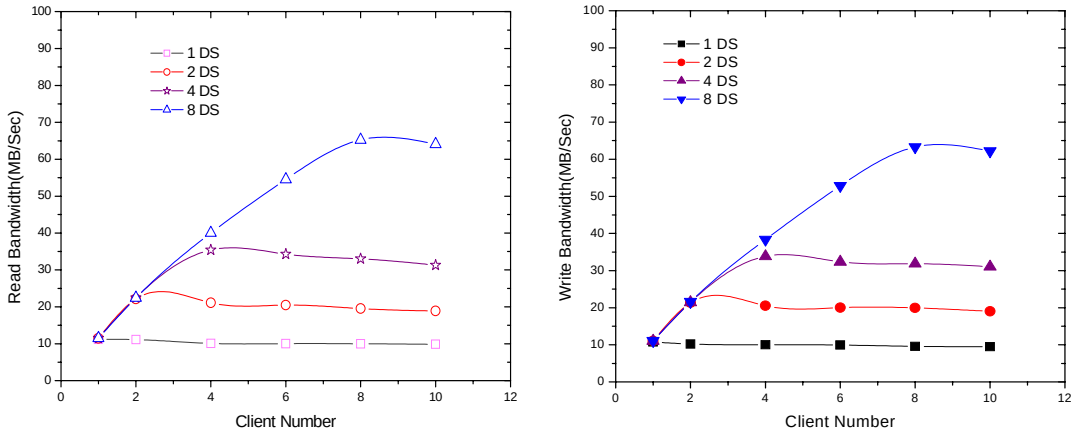


图 13　RAID0 读写带宽

图 13 是用 IOZONE 测得的不同数目数据服务器情况下聚合读写带宽的变化情况。测试时只配置了一个元数据服务器，数据服务器的数目为 1、2、4、8，客户端最多有 10 个，HANDY 的放置策略为 RAID0。 可以看到，当数据服务器增加时 Handy 系统的聚合读写带宽也随之

增加。图 14 显示数据服务器为 1、2、4、6、8 时 HANDY 系统测得的最大聚合读写带宽。1 个服务器的最大读带宽为 11.20MB/Sec，写带宽为 10.75MB/Sec；2 个服务器的最大读带宽为 22.12MB/Sec，写带宽为 21.32MB/Sec；当数据服务器数目增为 8 时，最大读带宽达到 65.25MB/Sec，写带宽为 63.29MB/Sec，这已经大大超过用 IOZONE 测得的本地文件系统的读写带宽。

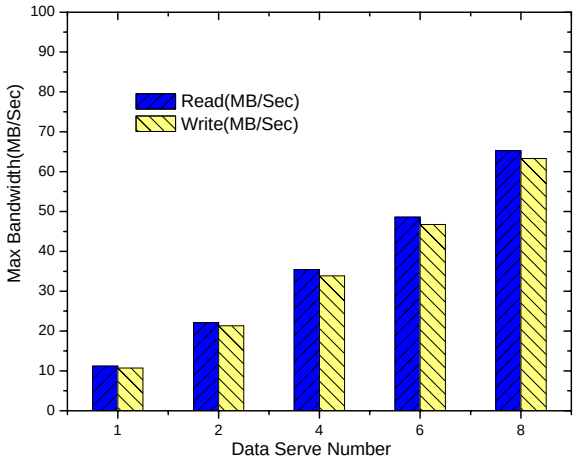
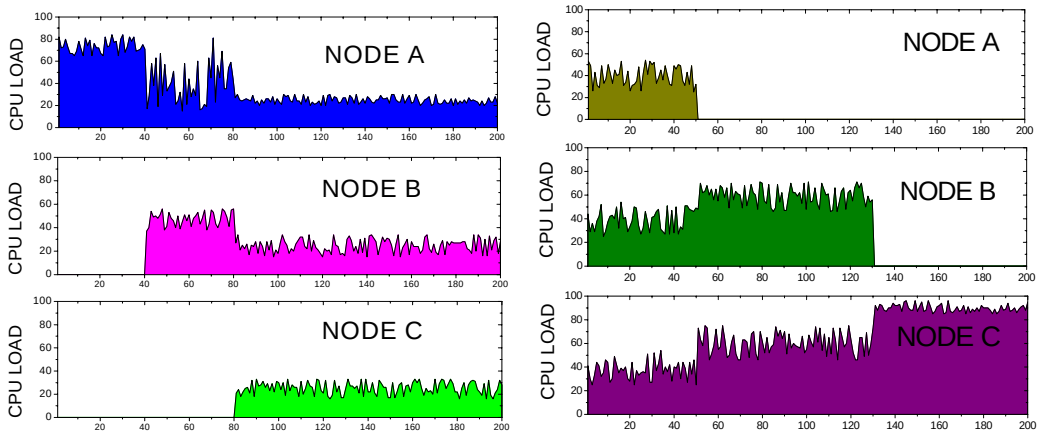


图 14　最大读写带宽的变化

综上所得结果，可以发现：不管是元数据还是数据服务器，随着节点数的增多，系统吞吐率和聚合 IO 读写带宽也随之提高。但相比较而言，元数据服务器的扩展幅度要小于数据服务器，我们认为这主要是由元数据服务器分布后客户端解析路径时的通讯开销导致的。

动态性



a)加入过程　　b)离开过程

图 15　节点加入和离开时各个节点 CPU 利用率的变化

HANDY 系统的动态扩展体现在新节点加入和已存在节点失效时系统的自适应性。为了观察 HANDY 系统的动态性，我们在各个元数据服务器端启运行一个定时收集 CPU 负载的 Daemon 程序 CPU Gather，由 CPU Monitor 启动各个 Gather 开始收集，并且每隔一秒种收集一次。测试时在客户端解压一个较大的 tar 包，在此过程中逐渐加入新的元数据节点，同时收集各个节点的 CPU 负载情况，得到图 15(a)的 CPU 利用率变化曲线。可以看到在 40 秒时，NODE B 正式加入系统，原来由 NODE A 独自承担的负载被平分，NODE A 的负载减少；在大约 80 秒钟的时候又有一个新的节点 NODE C 加进来，原来的两个节点 NODE A 和 NODE

B 的 CPU 负载都减小，NODE A、NODE B、 NODE C 三个节点比较均衡的分担了系统当前的负载。同样，如图 15(b)所示，节点减少会导致剩下的节点的负担加重。如 NODE A 在 50 秒左右离开，NODE B、NODE C 分担目前的负担，但 NODE B 的负载比 NODE C 要高一下，因为 NODE B 是 NODE A 的后续，原来 NODE A 上的存储对象都需要从 NODE B 访问得到。图 15(a)和 15(b)反映出 HANDY 在动态扩展的过程中具有良好的自适应性，达到动态扩展的要求。

可用性

理论分析

本文从理论上分析了基于逻辑环的元数据服务器的可用性。假设各个节点在一定时间 T 内失效的概率为 p，HANDY 系统的元数据的数目为 n，采用邻接复制技术以后，每个节点都有一个后续节点作为备份，因此系统中只要不存在连续的 2 个或 2 个以上的节点失效，系统的元数据信息就可以正常访问，也就是说，当系统中出现 2 个或 2 个以上节点失效时，才会使系统失效。

2 个节点失效导致系统失效的概率P_2：

$$P_2 = C_n^1 p^2 (1-p)^{(n-2)}$$

3 个节点失效导致系统失效的概率P_3：

$$P_3 = C_n^1 C_{n-2}^1 p^3 (1-p)^{(n-3)}$$

m(m<n)个节点失效导致系统失效的概率P_m：

$$P_m = C_n^1 C_{n-2}^m p^m (1-p)^{(n-m)} \quad (\text{m<n})$$

求得系统在时间 T 内完全可访问的概率 P_{access} 的公式如下：

$$P_{access} = 1 - \sum_{i=2}^{n} P_i$$

$$= 1 - \left\{ C_n^1 p^2 (1-p)^{(n-2)} + C_n^1 C_{n-2}^1 p^3 (1-p)^{(n-3)} + ... + C_n^1 C_{n-2}^m p^m (1-p)^{(n-m)} + ... + P^n \right\}$$

$$= 1 - \left\{ \sum_{i=0}^{n-3} nC_{n-2}^i p^{i+2} (1-p)^{n-i-2} + p^n \right\} \quad (1)$$

如果是 n 个节点之间不采用邻接容错构成一个环，那么任何一个节点的失效都会导致系统 Handle 存储对象丢失，系统将无法正常访问。可以得到该情况下系统在时间 T 内完全可访问的概率 P'_{access} 为：

$$P'_{access} = (1-p)^n \quad (2)$$

设 p＝0.05，利用公式(1)和(2)用 Matlab 写程序计算的不同节点个数下系统完全可访问概率如表 3 所示。可以看到，随着节点数的增多，系统的可访问率并不是越来越高，而是越来越小，因为每个节点都处于工作状态，且负责一段 Handle，并不是只做其他节点的冗余设备。图 16 的曲线说明采用基于环状的邻接复制技术后，系统完全可访问率减小速度明显

减小，这也一定程度说明对于分布式系统采用相应容错技术是很有必要的。

表 3　不同节点个数下的系统访问率（单个节点的失效率为 p=0.05）

节点个数 n	基于串联的可访问率 P'_{access}	基于逻辑矢量环的可访问率 P_{access}
2	0.9216	0.9984
3	0.8847	0.9953
4	0.8493	0.9936
5	0.8154	0.9920
6	0.7828	0.9904
7	0.7514	0.9888
8	0.7214	0.9872
9	0.6925	0.9856
10	0.6648	0.9840

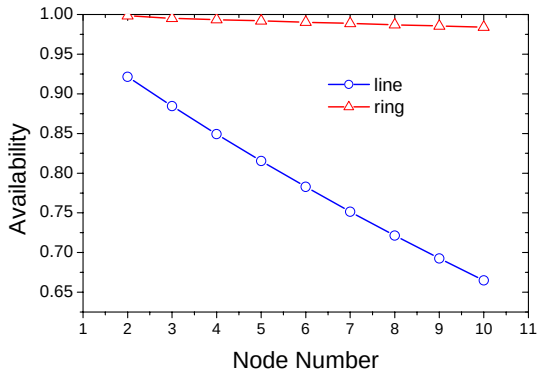


图 16　可用性的理论值变化趋势结果

时间分析

为了达到高可用性，系统必须具有失效发现和失效恢复的能力。HANDY系统的故障发现时间T_{found}为：

$$T_{found} = T_{detect} + T_{announce} + T_{update} \text{(公式 5-13)}$$

其中T_{detect}为心跳检测时间，$T_{detect} \le DeathTime = 4$秒，　　$T_{announce}$为 Crash 消息通告时间，T_{update}为 ServerMap 表更新时间，这两个时间都非常短，因此 HANDY 系统可以在 4 秒钟左右发现节点失效，这一点可以从图 17 的测试结果中看出来。20 秒钟以前，系统中有 3 个数据服务其正常工作，聚合读带宽大约在 25MB/Sec，20 秒钟的时候，人为地 kill 掉一个数据服务器，发现之后的 4 秒内读带宽降低很厉害，这主要是因为系统要读取失效节点的数据，又不知道该节点失效，于是不断的去连接失效节点。4 秒种后，客户端收到节点失效的消息，很快更新自己的 ServerMap 表，将请求定向到自己的后续节点，完成数据的读取。但此时只有 2 个数据节点在工作，所有整体的聚合读带宽在 17～20 之间波动，相对于原来的聚合读带宽有所下降。

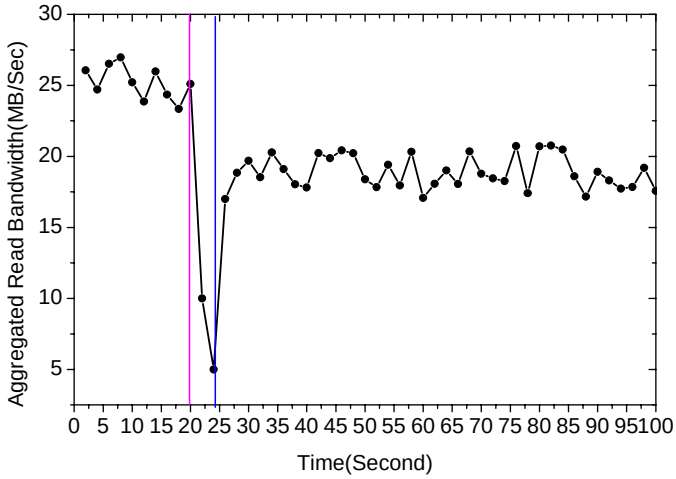
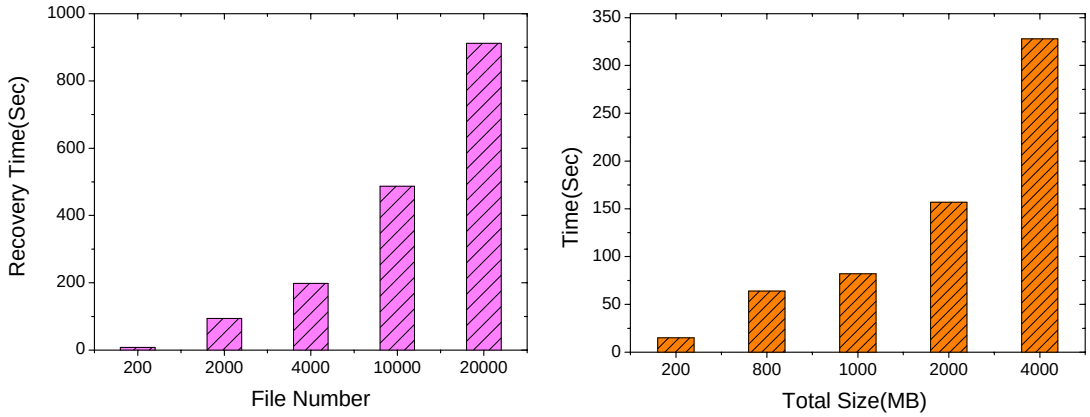


图 17 节点失效时的聚合读写带宽变化

当节点失效后重新加入时需要完成恢复操作，其恢复时间是个不定值，取决于需要恢复的数据量的大小。图 18(a)给出 4 个元数据节点，系统总的文件数为 200、2000、4000、10000、20000 时某个节点失效后重新恢复的时间，可以看到当文件数据总数到达几万个时，元数据的恢复时间比较长，说明系统在元数据的恢复策略上效率有待进一步提高和优化。图 18(b)给出 4 个数据服务器，文件总量为 200、800、1000、2000、4000MB 情况下一个数据节点失效后又重新恢复起来的时间。



a) 不同文件数目下的元数据的恢复时间　　b）不同数据总量下的数据节点恢复时间

图 18 元数据服务器和数据服务器的恢复时间

5. 结束语

动态扩展性对提高分布式系统的可用性、扩展性和可管理性具有重要意义。本文提出一种基于逻辑矢量环的动态扩展方法，实现一种具有高可用性和动态扩展性的集群文件系统 HANDY，解决目前多数集群文件系统中集中式元数据管理所导致的单一失效和潜在系统瓶颈问题，提高系统整体可用性和可管理性，同时结合 HANDY 系统原型的实验结果综合分析了该系统可扩展性、动态性和可用性。

本文主要贡献包括如下四点：

1) 提出一种基于逻辑矢量环的动态扩展方法，并且利用该方法实现集群文件系统中的动态扩展协议，重点分析了协议的格式设计、算法和正反向扩展过程。

2) 借鉴 DHT 中资源的表示方式，将集群文件系统中所有对象用一个唯一的 HANDLE 值标识，利用全局哈西表进行定位，并给出 HANDY 系统动态扩展过程中资源的动态管理算法。
3) 结合原型系统的测试数据重点分析系统扩展性、动态性、容错性、可用性等方面的表现。

参考文献

[1] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A Parallel File System For Linux Clusters. In Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, Oct. 2000, pp. 317-327.

[2] J Xiong, S Wu, D Meng, N Sun and G Li. Design and performance of the Dawning Cluster File System. In Proceedings of 2003 IEEE International Conference on Cluster Computing, 2003. pp. 232-239.

[3] LUSTRE, http://www.lustre.org

[4] Feng Wang, Qin Xin, Bo Hong, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long. File System Workload Analysis for Large Scale Scientific Computing Applications. In proceedings of 21st 12th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST2004), College Park, MD, April 2004.

[5] Sage A. Weil, Kristal T. Pollack, Scott A. Brandt and Ethan L. Miller. Dynamic Metadata Management for Petabyte-Scale File Systems. In Proceedings of the Proceedings of the ACM/IEEE SuperComputing2004 Conference (SC'04). 2004.

[6] Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long and Lan Xue. Efficient Metadata Management in Large Distributed Storage Systems. In proceedings of 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies (MSS'03). April 07 - 10, 2003, San Diego, California.

[7] Hong Tang, Aziz Gulbeden, Jingyu Zhou, William Strathearn, Tao Yang, and Lingkun Chu. A Self-Organizing Storage Cluster for Parallel Data-Intensive Applications. In Proceedings of the Proceedings of the ACM/IEEE SuperComputing2004 Conference (SC'04). 2004.

[8] 赵东，姚绍文，周明天．一种适应性复制协议的研究与设计．电子学报．2002, 12(A):238~245.

[9] I. Stoica, R. Morris, M. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for Internet applications”, In Proc. of ACM SIGCOMM01, San Diego, CA, USA, August 2001.

[10] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris and I. Stoica. Wide-area cooperative storage with CFS. In the Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01), Chateau Lake Louise, Banff, Canada. October 2001.

作者简介：

程斌：男，1979 年生，博士研究生，主要研究方为分布式系统的容错、并行 I/O、调度等。
金海：男，1966 年生，博士生导师，主要研究方向为计算机系统结构、集群与网格计算、网络安全等。
李胜利：男，1952 年生，教授，主要研究方向为并行与分布式计算，性能评价与测试等。
邵志远：男，1976 年生，博士研究生，主要研究方向为集群系统的容错。

作者联系方式

程斌，华中科技大学计算机学院集群与网格计算机实验室 程斌 邮编 430074
Email: showersky@hust.edu.cn, Tel: 027-87557047