高可用并行文件系统的分布式元数据管理

唐维

一、研究意义及课题背景

随着社会经济与科技的发展，对计算的需求也日益增长。为了满足这种需求，高性能集群计算技术被广泛应用到各种领域。随着高性能计算技术的发展，集群的并行 I/O 和文件存储面临巨大挑战，于是研究用于集群计算的优秀的并行文件系统有着重要意义。

目前国内外有各种各样的集群文件系统(并行文件系统)已研究成型或者正处于研究改善状态。象 Clemonson 大学的并行虚拟文件系统（PVFS）及其改进版本 PVFS2，加州大学伯克利分校的 xFS，卡耐基梅隆大学的 Coda，IBM 公司的 GPFS，中科院计算所的 COSMOS，等等。虽然文件系统种类繁多，但是不存在一种文件系统拥有十全十美的优点，他们在都拥有各自的特色，在不同的领域能够体现自己的优势。对于集群文件系统来说，高性能，高可用，和高扩展性是衡量一个文件系统优秀性的三个重要指标。随着硬件技术的发展，对高可用和扩展性的需求逐渐超过了对文件系统高性能的要求。基于这种需求，华中科技大学集群与网格计算实验室并行文件系统小组拟开发一个具有高可用和高扩展性的并行文件系统。命名为 **Handy**。（Parallel Filesystem with High Availability and Dynamic Scalability）。作为集群系统的重要组成部分，该文件系统为未来开发 3H（高可用，高性能，高扩展）的集群超级服务器奠定了基础。

元数据管理是并行文件系统的重要组成部分，**Handy** 元数据管理采用分布式的方式，解决了元数据服务器单一失效点问题，保证了元数据的高可用性。本选题报告的主题即是 Handy 文件系统的分布式元数据管理的实现方案。

二、**Handy** 概述

1． Handy 是什么

Handy 是 CGCL 并行文件系统小组拟开发的用于集群系统的并行文件系统。在 PVFS2 的础上，实现实现元数据分布式管理，提供元数据和数据的高可用特性，提供灵活的可扩展特性。

2． Handy 的特色

Handy 的主要特色有：

- 元数据和数据的高可用性，元数据和数据都采用各自的方法进行冗余存放，容错性好；
- 元数据管理采用无集中服务器方式，存储也采用分布式存储，解决了元数据读写瓶颈和元数据服务器单一失效点问题。
- 节点的可扩展性。元数据和数据节点的没有区分，一个节点可以充当各种角色，并且能够随意加入或离开系统，而不影响系统的正常运行。
- 无需采用特殊存储介质，具有简便易安装的特点。

表一表示了 Handy 与现存其他并行文件系统（集群文件系统）的比较。

Name	Availability	Scalability	Metadata Management	Special Storage Device Based on
GFS	Good	Static Good	Decentralized	OVS
PVFS	Bad	Static Normal	Central	None
xFS	RAIDS	Static Normal	Decentralized	None
Lustre	Good	Static Good	Central	None
NFS	Bad	Static Bad	Central	None
Handy	Excellent	Dynamic Good	Decentralized	None

Figure 1　部分集群文件系统比较

3．体系结构

三、分布式元数据管理

1．什么是元数据

任何文件系统中的数据分为数据和元数据。数据是指普通文件中的实际数据，而元数据指用来描述一个文件的特征的系统数据，诸如访问权限、文件拥有者以及文件数据块的分布信息等等。在集群文件系统中，分布信息包括文件在磁盘上的位置以及磁盘在集群中的位置。用户需要操作一个文件必须首先得到它的元数据，才能定位到文件的位置并且得到文件的内容或相关属性。

2．元数据管理方式

元数据管理有两种方式。集中式管理和分布式管理。集中式管理是指在系统中有一个节点专门司职元数据管理，所有元数据都存储在该节点的存储设备上。所有客户端对文件的请求前，都要先对该元数据管理器请求元数据。分布式管理是指将元数据存放在系统的任意节点并且能动态的迁移。对元数据管理的职责也分布到各个不同的节点上。

大多数集群文件系统都采用集中式的元数据管理。因为集中式管理实现简单，一致性维护容易，在一定的操作频繁度内可以提供较满意的性能。缺点是单一失效点问题，若该服务器失效，整个系统将无法正常工作。而且，当对元数据的操作过于频繁时，集中的元数据管理成为整个系统的性能瓶颈。

分布式元数据管理的好处是解决了集中式管理的单一失效点问题，而且性能不会随着操作频繁而出现瓶颈。其缺点是，实现复杂，一致性维护复杂，对性能有一定影响。由于 Handy 的目标是提供高可用和高扩展的并行文件系统，采用分布式元数据管理能够更好的迎合这个目标。

3．**Handy** 的分布式元数据管理

分布式元数据管理的实现有多种方式。**xFS** 采用将文件系统分为一些子集，每个子集都有负责的元数据管理器的方式，这样减轻了单一元数据管理服务器的瓶颈，但在每个子集内还是相当于集中管理。我们的 **Handy** 采用元数据的完全分布式管理，元数据和数据一样分布在任何节点，任何节点都能扮演元数据管理的角色。并且元数据采用一定算法进行冗余分片，这样随时的增删节点不影响整个系统的元数据的完整性。这样元

数据的高可用和元数据服务器的高扩展都得以保障。

4． **Handy** 分布式元数据管理和数据存储实现的关键技术

Handy 利用 **DHash** 和 **Chord** 技术来实现完全分布的元数据管理。

DHash 是一种分布式存储技术，每个 DHash 服务器保持一张分布式哈希表，能将数据块在分布式系统的其它节点上进行存储和取回， DHash 在存储数据块时，采用 IDA 分片算法，将每个数据块（Block）分成冗余的若干片，分别存储在不同节点上，由于分片算法的特殊性，只需任意取回一定数量的分片，即可组装成原来的块。少数节点的临时增删不会影响系统的正常，可用性和扩展性都得以保证。DHash 将的数据分以 Key/Value 的形式存储在名为 Berkey DB 的数据库中，该数据库是一种方便检索的内存数据库，提高了元数据的读写性能。

Chord 是一种用于 P2P 系统中的路由协议，它仅完成一个操作，对每个提供的 Key，能将它映射到系统中的一个节点，该节点一般负责这个 Key 所对应的内容的管理或者存储。DHash 通过 Chord 提供的 API 函数来定位数据块所存放的节点位置。Chord 提供各种技术来提高节点定位的性能，并且配合 DHash 一起，动态维护整个系统的数据一致性和完整性。

元数据的分布式管理主要由 DHash 来实现，Chord 的作用是给 DHash 提供节点定位函数。所以本文以下的内容就是先简要介绍 Chord，再详细介绍如何用 DHash 来实现 Handy 中的元数据分布式管理，最后给出实现的计划和进度安排。

四、Chord 简介

Chord 是一种的分布式查找协议。它能够提供 P2P 系统高效的节点定位功能。给它一个 key 值，它能映射到唯一的节点。这种特性能够明确分布式系统中各节点的职责。在 Handy 中，正是利用 Chord 找到为某一操作负责的节点，利用 Chord 定位每个数据分片所需的存储节点。

在 Handy 中 Chord 为 DHash 提供两个 API 函数:Get_successor_list(n)和 lookup(k,m)。前者的作用是返回节点 n 的后续节点列表。后者的作用是根据 Key 值 k 定位 m 个后续节点。某个 key 的后续节点是指 Chord 将这个 key 同过某种机制映射到的节点，这些节点负责有关这个 key 的所有操作。关于 Chord 的具体工作机制，参考文献[2]中有详细介绍。

五、DHash 技术要点

１．DHash API

DHash 将元数据以块为单位进行分布式存储。每一个块提 DHash 一个 Key,DHash 将这个 Key 对应的块的内容进行分片后分别存储到若干个节点上。此后若要取回此块，用户程序只需提供它的 Key，ＤＨash 就能取回原先存储到系统中的块。也就是说 ＤＨash 只要提供给用户程序两个ＡＰＩ函数，Put(Key,Block)和 Get(Key)就能完成对数据块的存储和获取操作。存取涉及到数据块的分片，节点的定位，数据库的读写，这些都是对用户完全透明，由ＤＨash 调用其他技术提供的函数接口来完成。

- 块的存储函数: **Put(Key,Block)**
  当应用程序需要存储一块时，会调用 DHash 的 Put(key,block)函数，参数分别为要存储的块的 Key 和块的内容。Put 被调用后，首先利用 IDA 分片算法将 Block 分片分成数个 fragments，在这里是１４片（为什么是 14 片后文将

有说明)。然后，调用 Chord 提供的 lookup 函数，找到 Key k 的 14 个后续节点[*]，即分别存储 14 个分片的节点。然后，逐个将这些分片利用 RPC 发送到相应节点中去，并且存储在相应节点上。

该函数的伪代码如下。

```
void put (Key    k ,Block b)
{
    frags = IDAencode(b);
    succs = lookup(k,14);
    for I (0..13)
      send(succs[i].ipaddr,k,frags[i]);
}
```

- 块的获取函数：**get(Key)**

  应用程序需要读取一块时，只需提供该块的 Key，将它作为参数调用 DHash 的 get(Key)函数，就能返回所需块的内容。当 get 函数被调用，首先根据参数 Key 的值来调用 Chord 的 lookup 函数，定位该 key 的后续节点，这些节点为存放了该 key 所对应内容的分片。然后从这些节点中选一部分网络延迟最少的节点，从他们取回所需数目的分片，然后再利用 IDA 分片组装算法，将分片还原成数据块,返回给应用程序。描述该算法的伪代码如下：

```
Block get(Key k)
{
        frags = [];
        succs = lookup(k,7)
        sort_by_latency(succs)

        for (i=0;i<#succs && i<14;i++){
          //download fragment
           <ret,data> = download(k,succ[i])
           if (ret == OK)
                frags.push(data)
        }
        //decode fragments to recover block
        <ret,block> = IDAdecode(frags)
        if (ret == OK)
             return (SHA-1(Block) != k?FAILURE:block

        if (i == #succs -1){
            newsuccs = get_successor_list(succs[i])
            sort_by_latency(newsuccs)
            succs.append(newsuccs)
        }
        return FAILURE
}
```

2．数据块的高可用

为了提高元数据的高可用性和容错性，DHash 将待存储的数据块按 IDA 算进行分片，分别存储到不同节点。

所谓 IDA 算法[31]是一种信息分散算法（Information Dispersal Algrighm）。它的功能是能将一个长度为 L 的文件(或文件块)F，分成 n 个分片 Fi（$l \le i \le n$），每个分片的长度为 L/m,那么在 Fi 中任取 m 个分片都能重组成原来的文件 F。这种分片和分片重组在计算效率上是很高的。在空间上，可以看出 n 个分片 Fi 的总和是原来文件的 n/m 倍，n/m 大于但可以趋近于 1，所以数据冗余不会占用太多空间。

DHash 采用 IDA 算法将文件块分片，n 和 m 的取值可以根据系统的规模来定。对于较大规模的系统，n，m 可以分别取值 14 和 7。这样，Hash 将待存储的数据块（长度为 8192Bytes）分成 14 片,每片的大小 1170Bytes,为从中任意取 7 片，能够将他还原成原来的块。也就是说，当把这 14 片分别存储在 14 个节点上，只要其中失效的节点数不超过 7 个，那么原来的数据就仍然能够取回。这就保证了元数据的高可 用性。因为 IDA 算法的时间复杂度很低，所以分片与重组不会影响对数据操作的性能。只是在空间上，需要用原来存储空间的 1 倍来保存冗余数据。可以证明，采用以上取值，在一个系统中，10%的节点失效后，仍能够取回原来文件块的概率为 0.99998;而如果采用简单的用一倍空间来保存块的备份，上面这个概率仅为 0.99。[40]可见 DHash 可以大大提高数据的容错能力和系统高可用特性。

3．分片维护

数据块的IDA分片，提供了元数据的高可用性。这种机制除了对每个块需要用IDA算法分片或重组以外，还带来另外了一些附加操作。比如说，一个系统中的每个元数据的块将分成14个分片，分别存储在14个节点上。如果这14个节点有一个退出了系统，那么就只剩13个分片了，虽然只要有任意7个分片存在，就能取回原来的块，但是为了防止情况进一步恶化（节点数变得少于7个），DHash总是维持系统中有14个分片存在。它在发现分片少于14个后，会在某个固定时刻从新创建一个分片，放在另外一个不曾存储这个块的任何分片的节点上。同理，如果在这14个节点中临时加入了一个节点，原来的14个节点就不是Chord中紧随数据块的紧随Key的14个节点，这将不能使Chord的lookup函数效率达到最高。所以DHash将定期将14个分片都迁移到Key的14个后续节点中。这种数据分片的重建和迁移工作成为分片维护(Fragment Maintenance)。

分片维护工作的目标是使系统的分片分布趋于“ 理想状态 ”。所谓的理想状态是指满足以下三个标准：

1. 多样性: 对每个块有14, 15，或16个分片存在。
2. 相异性：所有分片尽可能的内容不同
3. 位置最优化:每个块的14个分片存储在id紧随该Key的14个节点上，接下来的两个节点也可以存储第15或16个分片，其他的节点都不参与存储该块的任何分片。

DHash通过后台的进程定期激活，进行分片维护，使分片的状况达到理想状况，由于节点的临时加入与退出，理想状态很难达到，但是分片维护程序不停的使系统达到相对平衡的理想状态。

为了恢复理想状态，DHash运行两个维护协议：本地的（local）和全局的(global)维护协议。本地维护通过重建分片来达到理想状态的多样性要求。全局维护将错放位置的分片移动到合适的位置去，使DHash达到的分片位置最优化状态。同时，全

局维护也能删除多余的分片，使分片数目维持在14到16之间。

4． DHash分片的存储

DHash存储元数据是以块为单位，每个块有一个Key,通过Key值又Chord层找到若干个节点，来存储该块的若干个IDA分片。每个分片在存储它的节点上保存在Berkey DB数据库中，也就是以Key/Value对的形式保存在每个节点的内存中。为了提高查询效率，数据库以Merkle Tree的形式保存了本地数据库存有哪些Key值的信息。这里用到的Merkle Tree是一个64叉平衡树，叶子节点是Key值，每个内部节点是它的所有孩子节点的SHA-1哈希值。通过查找Merkle tree能快速查找到所指定的key对应的内容，在数据分片存储和获取时可以大大提高效率。另外，通过比较不同节点的Merkle tree，可以实现两节点间的数据库同步。

六、Handy元数据和数据的读写机制

1.元数据的读写

用户程序对元数据的读写请求是以块（Block）为单位。当用户程序要读取某个块，先利用该块的key定位一个为之负责的节点，在调用该节点DHash的get(key)函数，Chord定位存储了该Block分片的节点，并取回足够数目的分片，将他们重组成所需的块，返回个用户程序。

用户程序要写一个块时，将该块的Key定位到这个块需要由哪个节点负责。然后将key和内容一起传给该节点的ＤHash，调用ＤHash的put函数,将块分成若干片后发到Chord定位的若干个节点上，在当地调用数据库插入操作，写到数据库中，如果该Ｋey的内容原来存在与数据库中，将被新的内容覆盖。

2.数据读写

数据读写的基本方式与元数据读写类似，也是以块为单位由于一个数据文件通常比元数据要大得多，包含的块也多得多。我们采用一定的策略提高数据读写的性能。

在写数据时，利用key值定位一个数据节点，将数据块写到这个节点上，既完成了写操作。然后该节点再启动后台线程去调用ＤHash的put函数去将数据块分片后分布到其他数据服务器上去。分布完成后再将本地的数据块删掉。

在读一块数据时，客户端先根据该块的Key值定位一个节点，如果这个节点上有完整的块存在，说明数据块分片还没有结束，就直接从这个节点读回所需的块。如果该节点上不存在所需的数据块，说明该数据块的分片工作已经晚完成。数据块已经分成了若干个分片存储在别的节点上。这时，该节点的DHash调用Chord的lookup函数，得到存储了这些分片的节点的ip地址，从中选择足够数目的节点，将他们的ip地址和返回给客户端，客户端与这些节点建立连接，利用key值直接从这些节点的数据库中取回分片，然后在本地将这些分片组合成所需的块。这样的并行读取提高了读数据的性能，也减少了网络开销，分片不需要经过负责的DHash服务器再传与客户端了。这就要求DHash的分片重组函数需要移植到客户端。

七、进度安排

- 2003 年 2 月~2003 年 4 月 完成各模块的详细设计
- 2003 年 4 月~2004 年 8 月 编码，联调
- 2003 年 9 月~2003 年 12 月 完善并完成毕业设计论文。

八、参考文献

[1]Josh Cates,Robust and Efficient Data Management for a Distributed Hash Table, (thesis,MIT,June 2003)

[2]Ion Stoica,Robert Morris,David Karger,M.Frans Kaashoek,Hari Balakrishnant, Chord:A Scalable Peer-to-peer Lookup Service for Internet Applications

[3] Blake, C., and Rodrigues, R. High availability, scalable storage, dynamic peer networks: Pick two. In Proceedings of the 9th IEEE Workshop on Hot Topics in Operating Systems (HotOS-IX) (Lihue, Hawaii, May 2003).

[4] Castro, M., Druschel, P., Hu, Y. C., and Rowstron, A. Exploiting network proximity in peer-to-peer overlay networks. Tech. Rep. MSR-TR-2002-82, Microsoft Research, June 2002. Short version in International Workshop on Future Directions in Distributed Computing (FuDiCo), Bertinoro, Italy, June, 2002.

[5]Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. Freenet: A distributed anonymous information storage and retrieval system. In Proc. ICSI Workshop on Design Issues in Anonymity and Unobservability (Berkeley, California, June 2000). http://freenet.sourceforge.net.

[6] Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. Wide-area cooperative storage with CFS. In Proc. 18th ACM Symposium on Operating Systems Principles (SOSP '01) (Oct. 2001). http://www.pdos.lcs.mit.edu/chord/.

[7] Freedman, M., and Mazieres, D. Sloppy hashing and self-organizing clusters. In 2nd International Workshop on Peer-to-Peer Systems (IPTPS'02) (Feb. 2003). http://iptps03.cs.berkeley.edu/.

[8] Gribble, S. D., Brewer, E. A., Hellerstein, J. M., and Culler, D. Scalable, distributed data structures for Internet service construction. In Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2000) (October 2000).

[9] Merkle, R. C. A digital signature based on a conventional encryption function. In Advances in Cryptology - Crypto '87 (Berlin, 1987), C. Pomerance, Ed., Springer-Verlag, pp. 369{378. Lecture Notes in Computer Science Volume 293.

[10] Rabin, M. E_cient dispersal of information for security, load balancing, and fault tolerance. Journal of the ACM 36, 2 (Apr. 1989)

[11] Rowstron, A., and Druschel, P. Pastry: Scalable, distributed object location and routing for large-s cale peer-to-peer systems. In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001) (Nov.2001).

[12]Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen Ivy: A Read/Write Peer-to-Peer File System

[13] T. Anderson, M. Dahlin, J. Neefe, D. Patterson,D. Roselli, and R. Wang. Serverless network file systems.In Proc. of the ACM Symposium on Operating System Principles, pages 109–126, December 1995.

[14] Hartman and J. Ousterhout. The Zebra striped networkfile system. ACM Transactions on Compute Systems,13(3):274–310, 1995.

[15] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. In Proc. of the ACM Symposium on Operating System Principles, pages 213–225,1991.

[16] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In Proc. ACM SIGCOMM, August 2001.

[17] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001) (Nov. 2001).

[18] MAZI `ERES, D., KAMINSKY, M., KAASHOEK, M. F., AND WITCHEL, E. Separating key management from file system security. In Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP) (Dec. 1999), pp. 124–139.

[19] TYAN, T. A case study of server selection. Master's thesis, MIT,Sept. 2001.

[20] WALDMAN, M., RUBIN, A., AND CRANOR, L. F. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In Proc. 9th USENIX Security Symposium (August 2000), pp. 59–72.

[21] ZHAO, B., KUBIATOWICZ, J., AND JOSEPH, A. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, Apr. 2001.

[22] Philip H. Carns Walter B. Ligon III PVFS: A Parallel File System for Linux Clusters http://www.pvfs.org