蓝鲸分布式文件系统的客户端元数据缓存模型*)

黄 华[1,2] 张建刚[1] 许 鲁[1]

(中国科学院计算技术研究所 北京 100080)[1] (中国科学院研究生院 北京 100039)[2]

摘 要 在蓝鲸分布式文件系统中,客户端的所有元数据操作都是通过远程过程调用由元数据服务器完成,所有数据读写都是直接与存储服务器交换完成的。由于通信延迟,在客户端进行频繁数据读写时,元数据信息交换影响了整个系统的性能。我们设计了一种在客户端尽量缓存文件元数据信息的模型,有效地减少了元数据通信,缩短了整个读写过程的延迟,极大地提高了蓝鲸分布式文件系统的性能。

关键词 文件系统,分布式文件系统,客户端,缓存,蓝鲸

The Model of Meta-data Cache on Client Node in Blue Whale Distributed File System

HUANG Hua[1,2] ZhANG Jian-Gang[1] XU Lu[1]

(Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080)[1]

(Graduate School of the Chinese Academy of Sciences, Beijing 100039)[2]

Abstract In Blue Whale distributed file system, the meta-data server handles all the meta-data through remote procedure calls, and all the data flows directly between clients and storage nodes. Because of communication delay, the meta-data exchanging between clients and servers has great impact on the whole system performance especially when the load is heavy. We propose a model to maximize the meta-data cache on client node that dramatically reduces the RPC messages when itreads and writes data. This model greatly improves the reading and writing performance of Blue Whale distributed file system.

Keywords File system, Distributed file system, Client node, Cache, Blue Whale

1 引言

网络文件系统(NFS)[1]是最常见、最典型的分布式文件系统,它在几乎所有常见的通用操作系统(包括各种 UNIX, LINUX 和 Windows)中都有实现。但是,随着使用分布式文件系统的客户端数量的增加,以及每一个客户端数据访问速度的增加,NFS 单个服务器的结构遇到了严重的数据传输瓶颈。文[2]的调查显示在文件系统中元数据操作和数据读写操作大概各占 50%左右的系统资源。在需要进行大量数据传输的应用中,文件系统数据读写操作将占用系统更多比例的资源。在蓝鲸分布式文件系统中,元数据操作由元数据服务器处理,数据读写请求直接由存储节点处理。蓝鲸分布式文件系统将数据存储在多个存储节点上,多个客户端在元数据服务器的协调下,并发访问多个存储节点。通过这种方式,一方面减轻了元数据服务器的负载,降低了元数据服务的延迟;另一方面,充分利用了每一个存储节点的数据传输能力,提高了每一个客户端的数据吞吐能力。应用程序在读写数据时,文件系统负责将文件内的相对偏移量转化成实际物理磁盘的实际偏移地址。蓝鲸分布式文件系统和 NFS 一样,客户端的所有元数据都由(元数据)服务器提供。因此即使在应用程序直接与存储服务器交换数据时,它同样需要通过元数据服务器获得关于文件内偏移地址的物理映射。如果蓝鲸分布式文件系统客户端在存取每一个块数据时都需要与元数据服务器进行通信,必然增加其中的延迟开销,严重降低系统性能。我们设计了一个模型,客户端每次向元数据服务器申请多个块映射信息,并且尽量缓存更多的块映射信息,用以减少这样的通信,从而可以减少操作延迟,合并多个块读写请求,提高整个分布式文件系统的性能。

本文第 2 部分主要介绍蓝鲸分布式文件系统将元数据操作和数据读写操作分开的模型和实现;第 3 部分主要讲述如何缓存块映射信息以及由此带来的问题;第 4 部分主要分析对比了是否采用块映射信息缓存机制的性能。

2 元数据和数据分离

2.1 存在的问题

网络文件系统(NFS)是最常见、最典型的分布式文件系统,在几乎所有常见的操作系统中都有实现。另一个类似的分布式文件系统就是通用 Internet 文件系统(CIFS)[3],主要运行在微软公司的视窗系列操作系统中。它们两者都能在局域网内进行文件共享,它们的一个共同特点是采用单个服务器的结构。单个 NFS 或者 CIFS 服务器处理连接到此服务器的所有客户端的文件系统请求,包括元数据操作请求和数据读写请求。随着这些请求数量的增加和频率的提高,文件服务器的负载加重,响应延迟增加,造成整个系统在此处形成数据传输瓶颈,导致整体性能下降。

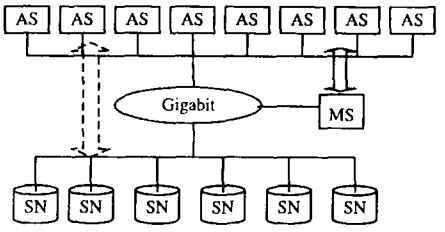
2.2 BWFS 的体系结构

中国科学院计算技术研究所国家高性能计算机工程研究中心研制开发的蓝鲸分布式文件系统(BWFS, Blue Whale

*)项目基金:国家"八六三"高技术研究发展计划基金项目(2002AA112010)。黄 华 博士研究生,主要研究领域为分布式文件系统,海量存储等。张建刚 博士,副研究员,硕士生导师。许 鲁 博士,研究员,博士生导师。

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

File System)摒弃了 NFS 和 CIFS 单个文件服务器的结构，整个分布式文件系统管理多个存储节点，并发提供数据传输服务。在蓝鲸的体系结构中，元数据服务器向文件系统客户端提供元数据操作服务，存储节点向文件系统客户端提供数据存取服务，如图 1 所示。多个存储节点一方面分担了元数据服务器的负载，提供并发的数据访问，显著提高了整个文件系统的数据吞吐能力。另一方面，多个存储节点可以提供更大容量的存储空间，克服了单个服务器存储容量的限制。因此蓝鲸分布式文件系统比传统的 NFS 在性能和容量上具有更好的扩展性。



（图中 AS 表示应用服务器，MS 表示元数据服务器，SN 表示存储节点，实线表示它们之间通过千兆以太网连接，虚箭头表示数据的流向，实箭头表示元数据的流向）

图 1　BWFS 逻辑结构图

2.3　实现机制

在 NFS 协议中，客户端的所有关于文件系统的请求都是通过远程过程调用由文件服务器完成的。在 NFS 协议的第三版本（NFSv3）中，对文件进行读写时分别需要将如下参数用 XDR 形式表示，通过 RPC 传递给服务器：

```
struct READargs {
  nfs_fh3 file; /* NFSv3 的文件句柄 */
  offset3 offset; /* 64 位文件内偏移量 */
  count3 count; /* 32 位希望读取的数据量 */
};
struct WRITEargs {
  nfs_fh3 file; /* NFSv3 的文件句柄 */
  offset3 offset; /* 64 位文件内偏移量 */
  count3 count; /* 32 位希望写入的数据量 */
  stable_how stable; /* 是否同步写标志位 */
  struct {
    u_int data_len; /* 32 位希望写入的数据量 */
    char *data_val; /* 实际即将写入的数据 */
  } data;
};
```

从上面的结构我们可以看出，在客户端进行数据读写时，使用 NFS 文件句柄唯一标识某一个文件，使用一个 64 位的无符号整数表示读写时数据的文件内偏移量。NFS 协议中的数据读写没有涉及将文件内偏移量映射到实际物理磁盘偏移量的计算工作，该部分功能由 NFS 服务器导出目录的宿主文件系统负责计算和分配。

蓝鲸分布式文件系统将多个存储磁盘上的物理存储设备通过改进的 NBD[4] 协议或者改进的 iSCSI[5] 协议，虚拟成一个统一的存储空间。在整个蓝鲸系统中，所有的节点都可以通过标准的磁盘访问接口，像访问本地块设备一样访问虚拟的磁盘，形成“共享磁盘（share-disk）”的架构。

蓝鲸分布式文件系统客户端利用 Linux 的虚拟文件系统（VFS）将用户通过系统调用申请的文件系统服务定位到自身的一个模块。该模块注册了蓝鲸特定的读写（BWFS_read、BWFS_write）函数，以供系统内核调用。为了能够充分利用系统内核本身提供的数据缓存功能，我们又注册了文件系统的按页读写函数（BWFS_readpage、BWFS_writepage）等函数。在读写函数里进行必要的准备工作和采集一些有用的系统信息以后，调用内核函数 generic_file_read 和 generic_file_write，它们最终会分别调用 BWFS_readpage 和 BWFS_writepage。同样，在进行了一番准备工作和采集了有用的系统信息以后，它们分别调用内核函数 block_read_full_page 和 block_write_full_page。在调用这两个函数时，我们需要提供一个函数指针作为参数，系统调用这个参数代表的函数，用以将文件内偏移量映射为磁盘物理地址偏移量，我们称此函数为映射函数，在 Linux 中，它具有如下原型：

```
int BWFS_get_block(struct inode *inode,
        long iblock,
        struct buffer_head *bh_result,
        int create);
```

inode 参数指向每一个文件的内核数据结构；iblock 参数指示需要映射的块在文件内的偏移量，以一个块的大小（通常情况下 Linux 中块的大小为 4k）为映射单位；create 参数指示该块是否需要文件系统分配。操作系统内核调用该函数以后，我们应该将结果存放在 bh_result 所指示的数据结构里边，返回零表示成功或者直接返回一个错误码表示映射或者分配失败。如果映射成功，我们应该在 bh_result 填写最重要的两个值，用以指导操作系统正确读写数据，它们就是：b_dev 和 b_blocknr，分别表示块设备和映射成功的块在该块设备上的物理地址，此地址以块（512 字节）为单位。

蓝鲸分布式文件系统客户端将映射请求通过 RPC 调用发送到服务器端。服务器接收到请求以后再将此映射请求传递给宿主文件系统，由宿主文件系统将此请求执行，并将执行的结果返回，服务器再将此结果返回给客户端。此 RPC 调用的参数和返回结果如下所示：

```
struct GETBLOCKargs {
  nfs_fh3 file; /* NFSv3 的文件句柄 */
  long iblock; /* 32 位文件内块偏移量 */
  int create; /* 是否是新创建 */
};
struct GETBLOCKres {
  int status; /* 指示请求执行的结果 */
  long pblock; /* 32 位块设备物理偏移量 */
};
```

采用以上数据结构和通信协议，每次可以获得一个映射信息，将文件内的偏移量转化成相对于物理设备的偏移量，以便可以正确存取块设备。

在 Linux 内核中通过标准的文件系统接口无法获得宿主文件系统块映射请求。我们通过为宿主文件系统添加一个特殊定义的 IOCTL 调用，通过在 IOCTL 函数里边再调用相关的函数满足此功能。

通过以上实现机制，蓝鲸分布式文件系统实现了元数据操作和数据操作的分离，获得了更好的性能和扩展性。

3　块映射信息的缓存

采用前述方法实现了元数据和数据操作的分离，使得蓝鲸分布式文件系统可以管理多个存储节点，应用节点可以并发存取数据。但是，应用节点在存取文件的每一个数据块时都需要向元数据服务器申请获得数据块映射信息。这个过程不仅消耗元数据服务器一定量的系统资源，还带来通信延迟，影响了应用节点存取数据的速度。在存取大文件时，这种开销尤为明显，它使得应用节点不能充分利用自身的 I/O 带宽。另一方面，如果由于系统内存资源比较紧张，应用程序出现反复读写某个大文件时，由于系统没有缓存文件块的映射信息，导致应用节点再次向元数据服务器申请块映射信息服务，降低效率。

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

3.1 块映射信息的成批获取

我们设计的方法就是应用节点在向元数据服务器申请块映射信息时，不再是每次申请一个块的映射信息，而是申请多个连续块的映射信息，并且在应用节点本地缓存这些块的映射信息。

我们将块映射信息的请求参数变更如下：

```
struct GETBLOCKargs {
   nfs_fh3 file; /* NFSv3 的文件句柄 */
   long iblock; /* 32 位文件内块起始偏移量 */
   long count; /* 32 位文件内块的数量 */
   int create;/* 是否是新创建 */
};
struct GETBLOCKres {
   int status; /* 指示请求执行的结果 */
   long count;/* 返回的数量 */
   long* pblock; /* 32 位块设备物理偏移量 */
};
```

我们将每次申请一个块的映射信息扩展成每次申请多个块的映射信息。元数据服务器在收到相应请求后，分配或者读取这些块映射信息，并将它们返回给客户端。

3.2 块映射信息的缓存

从元数据服务器获得的块映射信息被存放在应用服务器本地内存中，与每一个打开文件的内核 inode 结构相关联，并且将随着内核 inode 结构一起释放。为了在使用相同内存空间的情况下尽量缓存更多的块映射信息，我们采用了基于范围(extent-based)的表示方式。每一个普通文件的 inode 结构都有多个这样的数据项，形成一个块映射信息缓存队列，记录着已经获得的该文件的块映射信息。客户端在进行文件的读写时，大部分情况下可以直接从内存中取得这些信息，减少了数据存取的延迟。蓝鲸分布式文件系统的元数据服务器在给每一个普通文件分配数据块时，尽量保持块号连续，便于客户端记录更多的块映射信息。

每次申请的映射块数、本地缓存的个数等参数，取决于客户端的内存大小、读写模式等因素。用户可以根据需要和当时的情况，调整这些参数，以便获得最好的效果。

3.3 块映射信息的替换策略

当一个文件比较大时，如果客户端缓存所有的块映射信息，势必需要较多的内存，特别是当文件的长度超过十亿字节(Giga bytes)以后。然而这样的大文件应用在科学计算中很常见，所以我们必须选择一套替换策略，保证既不占用过多的内核内存，又有比较好的缓存效果。我们选择最近最少使用(LRU)策略来解决这个问题：当客户端在读写数据时，首先从本地缓存中检索块映射信息，如果成功，就用该信息进行读写；如果本地没有找到相应的块映射信息，就通过远程过程调用向元数据服务器申请服务。在本地成功获得映射信息或者从元数据服务器返回块映射信息之后，将该映射信息存放在缓存队列的最前面。如果原来缓存队列已经充满有用信息，那么队列的最后一项就被最新的一项替换。采用 LRU 策略保证被缓存的信息总是最近访问过的信息，由文件访问的时空局部性保证缓存的有效性。

3.4 块映射信息的失效处理

在并行计算和大规模信息处理等应用中，多个节点同时读写一个文件的情况是很常见的。分布式文件系统不能像本地文件一样保持符合 POSIX 标准的文件一致性，否则节点之间的同步通信信息将极大地降低系统性能。因此蓝鲸分布式文件系统也和其它类似的分布式文件系统一样，保持各个节点之间文件的弱一致性。客户端每隔一段时间就从元数据服务器获得最新的文件元数据信息，或者将已经修改的元数据向元数据服务器报告，以便获得同步。当客户端节点从元数据服务器更新文件属性以后发现文件已经被改变，我们就将该文件的映射信息失效。在下次访问这些缓存时，发现映射信息已经失效，于是客户端节点再去从元数据服务器申请这些映射信息，以便正确存取数据。

4 性能测试对比

这部分主要对比测试了蓝鲸分布式文件系统在使用块映射信息缓存技术前后的性能。我们在测试环境中配置一个客户端、一个存储节点，以及必要的系统服务器，它们之间通过千兆以太网相连。每一个结点均采用 Intel Xeon™ 2.40GHz 的 CPU，客户端节点拥有 1024MB 内存，存储节点拥有 3072MB 内存。存储节点采用 3ware 9500 SATA RAID 控制器连接 10 块 160G 7200 RPM 的 Seagate 硬盘，RAID 0 的磁盘整列。所有的测试均为采用 Linux 的命令 dd 顺序读写一个大文件获得，并且所有测试开始前均没有任何该大文件的缓存。

我们测试的结果如表 1 和表 2 所示，其中横向数据表示客户端每次从元数据服务器申请映射的块的个数，分别是 1、16、64、256 和无穷大，纵向表示每次读写数据块的大小。无穷大表示一次为测试文件分配所有的数据块，客户端只需要一次映射就能获得所有的块映射信息。测试结果用图形表示分别如图 2 和图 3。从中我们可以看出，蓝鲸分布式文件系统采用每次映射多个数据块的策略大大提高了系统的读写性能。

表 1 蓝鲸分布式文件系统读性能(单位：KB/sec)

Block Size \ Cache Blocks	R1	R16	R64	R256	R∞
1K	11586	72315	69905	67650	66363
2K	11902	73843	71332	71820	69905
4K	12108	78643	75984	71331	69905
8K	12192	77672	73326	71331	70849
16K	12577	78252	76538	71820	71331
32K	12577	75984	77672	73843	74898
64K	13443	73843	78840	75983	75437

表 2 蓝鲸分布式文件系统写性能(单位：KB/sec)

Block Size \ Cache Blocks	W1	W16	W64	W256	W∞
1K	10412	59918	67216	69905	69442
2K	10527	63167	68685	69905	68985
4K	10538	64860	69136	69905	69905
8K	10623	64329	68985	69905	69905
16K	10667	64329	68985	69905	69905
32K	10699	64329	69905	69905	69905
64K	10776	64727	69905	70734	70734

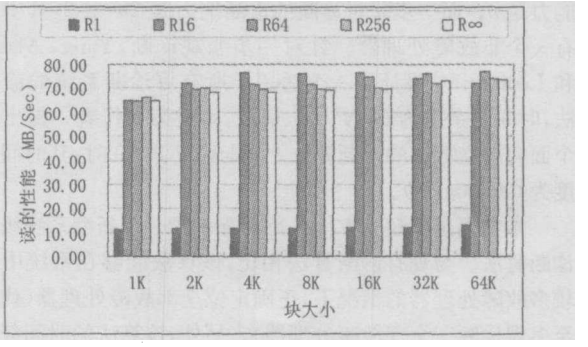


图 2 蓝鲸分布式文件系统读性能

(下转第 248 页)

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

通分图里。
7: if(结点 y 已经属于另外一个连通分图)
8: 通过比较 list[status[x]].count 与 list[status[y]].count 的大小,把小的连通分图连到大的连通分图并修改小的连通分图的状态信息。

定理 3 如果 $n=6$ 并且 $|F|=4n-10$,则算法 COM_SEARCH 可以诊断出一个故障结点集 $V(|V|=4n-7)$,其中最多包含 3 个不可诊断的结点。

证明:算法 COM_SEARCH 把结点间互相诊断结果均为 0 的结点连接起来以形成连通分图,并把分别属于两个连通分图的相邻结点的互相诊断结果均为 0 的结点连接起来,从而把它们所在的两个连通分图连接起来而形成了一个大的连通分图。根据 PMC 模型,两个无故障结点间互相诊断的结果均为 0,所以无故障结点可以形成连通分图。两个有故障结点间互相诊断的结果不定,也可能它们互相诊断结果均为 0,所以也可能形成连通分图。但是,故障结点形成的连通分图和无故障结点形成的连通分图是不会连通的,因为这两个连通分图中相临的结点中总是有一个是无故障结点,它对故障结点的诊断结果总是 1。又因为文[7]中证明了当 $|F|=4n-10$ 时,最大连通分图的大小为 $2^n-|F|-3(n=6)$。所以算法 COM_SEARCH 可以诊断出故障结点集 V,并且 $|V|=4n-7$ 其中最多包含 3 个不可诊断的结点。

引理 4 函数 DIAG_CONN 的时间复杂度是 $O(n)$。

证明:在函数 DIAG_CONN 中,3-7 行程序需要 $O(n)$时间。第 8 行程序需要分两种情况讨论:

1)对于故障结点:因为故障结点互相测试为 0 的概率很小,并且故障结点在超立方体中的分布是稀疏分布。所以,我们可以认为故障结点形成的连通分图的长度为常数。

2)对于无故障结点:因为最多有 3 个可能是无故障结点的结点所组成的连通分图会被故障结点分离出大的无故障结点的连通分图,当一个链表需要被连接到另一个链表时,仅有不超过 4 个无故障结点需要被改变结点状态。

因此第 8 行程序需要 $O(n)$时间。所以函数 DIAG_CONN 时间复杂度为 $O(n)$。 □

定理 5 算法 COM_SEARCH 的时间复杂度是 $O(N\log_2 N)$。

证明:在算法 COM_SEARCH 的第一步中,数组 status 的大小为 N,数组 list 的大小为 $8\log_2 N-20$,所以需要 $O(N)$时间对其进行初始化。在第二步中,算法 BUILD_TREE 的时间复杂度为 $O(N)$。函数 DIAG_CONN 是在算法 BUILD_TREE 中调用的函数,而函数 DIAG_CONN 的时间复杂度是 $O(\log_2 N)$,所以第二步需要 $O(N\log_2 N)$时间。由于连通分图的个数不超过 $8\log_2 N-20$,所以第三步需要 $O(\log_2 N)$时间。

因此,算法 COM_SEARCH 的时间复杂度是 $O(N\log_2 N)$。 □

结论 在这篇论文中,我们设计了一个适用范围较广的时间复杂度为 $O(N\log_2 N)$的新算法。此算法在构造超立方体树 HT_n 的过程中,来建立这个超立方体中的连通分图。从而诊断出故障结点集。当 $n=6$ 并且 $|F|=4n-10$ 时,此算法时间复杂度为 $O(N\log_2 N)$。

参 考 文 献

1 Kavianpour A, Kim K H. Diagnosabilities of hypercubes under the pessimistic one-step diagnosis strategy. IEEE Transactions on Computers, 1991, 40: 232~237
2 Liu J, Chen Y. On the Distributed Subcube-allocation Strategies in the Hypercube Multiprocessor Systems. In: Proc. of the 4th IEEE Symposium on Parallel and Distributed Processing, 1992. 360~364
3 Preparata F P, Metze G, Chien R T. On the connection assignment problem of diagnosable systems. IEEE Transactions on Electronic Computers, 1967, EC-16: 848~854
4 Yang C-L, Masson G M, Leonetti R A. On fault isolation and identification in t1/t1-diagnosable systems. IEEE Transactions on Computers, 1986, C-35: 639~643
5 Yang X. A fast pessimistic one-step diagnosis algorithm for hypercube multicomputer systems. Journal of Parallel and Distributed Computing, 2004, 64(4): 546~553
6 Yang X, Evans D, Chen B, Megson G, Lai H. On the maximal connected component of hypercube with faulty vertices, International Journal of Computer Mathematics, 2004, 81(5): 515~525
7 Yang X, Evans D, Megson G. On the maximal connected component of hypercube with faulty vertices(II). International Journal of Computer Mathematics, 2004, 81(10): 1175~1185

(上接第 245 页)

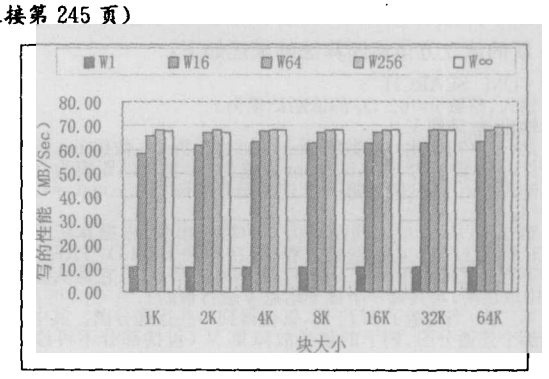


图 3 蓝鲸分布式文件系统写性能

结论 蓝鲸分布式文件系统通过元数据和数据分离机制,管理多个存储服务器,实现数据的并发访问,大大提高了整个存储系统的性能和扩展性。同时,我们通过一次请求多个元数据映射信息,减少了交换元数据映射的网络通信和开销;通过有效的缓存表示、替换机制和失效机制,尽量减少内核内存空闲占用和保持信息有效性。测试结果表明这种模型极大地提高了系统的数据读写性能。

参 考 文 献

1 Shepler S, Callaghan B. RFC 3530: Network File System (NFS) version 4 Protocol. The Internet Society, 2003
2 Klosterman A J, Ganger G. Cuckoo: layered clustering for NFS. Technical Report CMU-CS-02-183. Carnegie Mellon University, Oct. 2002
3 Leach P, Perry D. CIFS: A Common Internet File System. Microsoft Interactive Developer, Nov. 1996
4 Machek P. Network Block Device. http://atrey.karlin.mff.cuni.cz/~pavel/nbd/nbd.html
5 Meth K Z, Satran J. Design of the iSCSI Protocol. In: 20 th IEEE/11 th NASA Goddard Conf. on Mass Storage Systems and Technolog

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net