计算机科学2000Vol.27№.2

一个基于NOW的跨平台并行文件系统的设计和实现

The Design and Implementation of a Parallel File System Based on The Heterogeneous Network of Workstation

赵 欣 陈道蓄 谢 立
(南京大学计算机软件国家重点实验室 南京210093)

Abstract To improve the I/O performance of a parallel system, we designed and implemented a Parallel File System Based On The Heterogeneous Network Of Workstation (PFSHNOW). It is a cross-platform parallel file system, and has some characteristics such as parallelism, efficiency, cross-platform, convenience of management, and intelligence. This article described the model of the PFSHNOW and the advanced technique used in the file system. At the end, we also gave the result of evaluation.

Keyword NOW, PFS, Platform independence, Agent service

1. 引言

近年来,国内外对并行文件系统做了很多的研究[7],著名的有:一些商用的并行文件系统,如Intel为iPSC/2和iPSC/860而设计的CFS[1],以及Intel Paragon的并行文件系统PPFS[1],还有HPF的运行支撑系统PASSION[5,6]。研究性的则有PIOUS[2]等。它们通常是针对并行机设计的,应用范围较为狭窄。

随着对NOW(Network of Workstations,工作站网络)的研究深入,人们认识到NOW具有高速的交换网络,增强的计算能力,和大容量的内存及磁盘的特点,这使得NOW在解决磁盘I/O瓶颈这个问题上具有突出的优势。随着网络的发展,人们越来越希望将NOW建立成一个开放的环境,能将各种结构不同的工作站都连接到NOW环境中共同发挥功效,从而实现一个异构的NOW环境。因此,基于NOW的并行文件系统也必须能适应NOW环境的异构性。然而,不同结构的工作站之间往往存在着硬件体系结构不同、运行的操作系统不同、磁盘组织方式不同等差异,这些差异给基于异构型NOW的并行文件系统的设计和实现带来了许多困难。例如,在DEC的工作站上整数的高、低位字节的排放顺序和PC机上的正好相反,所以同样的数据在网络中传送以后就可能产生差异。又例如,UNIX工作站是将硬盘按照逻辑卷组织的,而PC上常用的Windows操作系统却将硬盘按逻辑分区组织。

JAVA的出现和MPI规范的推出使得我们能跨越平台和网络的异构障碍,解决上面提及的平台异构性问题。在此情况下,我们设计了一个支持异种结构的NOW的并行文件系统PFSHNOW(Parallel File System Based Heterogeneous NOW)。该系统构建在NOW环境中各节点已有的操作系统之上,用MPI为通信支撑、用NOW环境中各节点自己的操作系统提供的底层文件服务为文件操作支撑,用JAVA作为跨平台的工具,将各个节点上的磁盘通过网络连接在一起,构成一个虚拟的共享文件系统,使得用户可以透明地对存放在多种平台上的并行文件进行操作,而不必对文件存放的结构和位置以及相关机器的体系结构有所了解,由系统自动实现跨平台的操作,从而掩盖了各个节点的底层操作。

整个系统有以下特点:(1)并行性。具有传统并行文件系统的并行存取的功能,有方便的适合SPMD(单程序多数据)方式的系统调用。(2)共享性。节点间数据的共享方式自然方便,改变了传统并行文件系统的解决方式。(3)快速性。系统有一套高效的缓冲区管理机制,文件数据尽可能存放在内存中,每一节点可使用其它节点内存中的数据,因而比在磁盘上存取数据快。(4)跨平台性。系统采用JAVA结合MPI作为通信支撑,从而实现了系统的可移植性。用同一套程序就能在运行不同操作系统的、系统结构不同的异种机器上任意运行,系统能将异种机器上的文件系统连接成一个大的并行文件系统,用户只会看到一个完整的文件系统被提供使用了,而屏蔽了下层系统结构的差异性,这是过去的传统并行文件系统所不具备的特点。(5)系统管理的方便性。系统采用的是基于WEB的文件系统管理方式,文件系统管理者能在任何有WEB浏览器的机器上进行文件管理,而不同于过去的并行文件系统是用某种特殊的应用程序或某种特定的操作系统提供的功能来进行文件系统维护,使得文件管理者能在

任何地方、任何操作系统下、任何机器上、任何网络环境中都能通过互联网和 WEB 浏览器进行文件系统的维护。这种管理机制大大方便了系统的管理，由于该管理机制是基于 HTTP 协议的，管理者的操作只能通过系统提供的 JAVA 程序来实现，所以具有很好的系统安全性。(6)智能性。系统能根据过去的文件使用状况智能地调整文件分片的分布，达到负载平衡的目的，从而提高了系统运作效率。

2. PFSHNOW 的结构

PARFSNOW^{++}的结构如图1，它包括文件服务器、I/O 服务器、计算节点(即用户程序)三部分。并行文件系统将文件分片存放，便于并行读写，I/O 服务器(下简称 I/O 节点)负责存放、管理各并行文件分片。I/O 服务器还要负责完成用户对各并行文件分片的读写请求，并为用户的文件访问提供"预输入缓输出"服务。文件服务器则负责并行文件分片信息的存储和管理、文件操作任务的协调等任务。其中文件分片信息是以文件分片表 FAT(即 File Allocation Table 文件)形式存放的，用于存储各个文件分片的位置、状态、读写权限等信息。它还要负责文件系统的管理。文件服务器节点上有基于 WEB 的文件管理的服务程序，能支持文件管理者通过 WEB 的 CLIENT 端程序连接到文件服务器，通过 WEB 服务向文件管理的服务程序发出文件系统调整请求，文件管理服务程序则能根据请求、向各个 I/O 服务器发出相应的具体系统调节命令，由各个 I/O 服务器上的系统管理 AGENT 程序完成文件系统的调节工作。用户节点则是并行文件系统的使用者。当用户要求打开某并行文件时，系统会自动连接文件服务器，将该文件对应的 FAT 信息读出，返回给用户和并行文件系统，便于用户和系统的控制。当用户获取了文件分片表后就可以不必每次访问文件都要通过文件服务器，而可以直接访问相关的 I/O 节点。这样，就减少了文件服务器成为系统性能瓶颈并出现单点故障的问题，能有效提供系统并行度。图中虚线就是描述的用户节点直接访问 I/O 节点的操作。由于文件服务器是整个系统中枢，若出现故障，将导致整个系统的崩溃，所以容易造成单点故障。为此，系统设计了备份文件服务器，由备份文件服务器定期同文件服务器交换数据，文件服务器出现故障时，系统能自动切换到备份文件服务器上继续操作，从而大大提高了系统的可靠性。

图1中所有节点都将自己的部分内存共享出来，通过 DSM 机制连接成为一个大的分布式虚拟共享内存，这样能扩大文件数据缓冲区，有效地提高数据在缓存中的命中率，减少低速的磁盘访问，从而达到提高系统效率的目的。

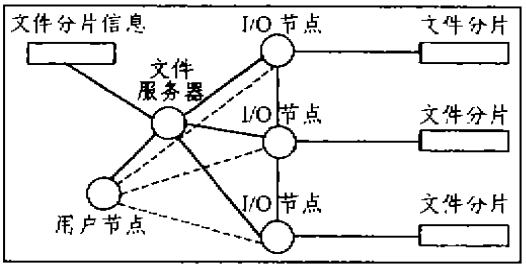

图1

3. PFSHNOW 的运行机制

PFSHNOW 运行中采用了一种独特的分优先级的多命令队列操作机制，它同所有文件 I/O 操作息息相关，能有效利用空闲的处理器和 I/O 资源，提高系统的效率。所以本文先引入这种分优先级的队列调度机制。

3.1 I/O 节点的队列调度机制

分优先级的命令队列调度机制是在 I/O 节点上采用的一种调度策略，能有效地利用空闲的处理器和 I/O 资源，提高系统的效率。结构如图2。

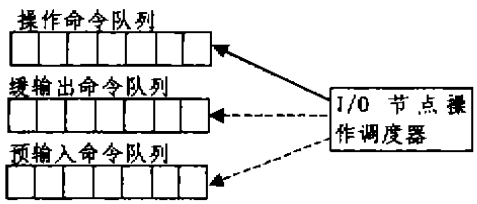

图2 I/O 命令队列机制

命令队列分成三类：按操作优先级由高至低依次为操作命令队列、缓输出命令队列和预输入命令队列。其中操作命令队列里放置的是要立即执行的操作命令，缓输出命令队列中存放的是缓输出命令，预输入命令队列中存放的是预输入命令。调度器总是按照优先级顺序来调度完成队列中的 I/O 命令的，只有当高级队列中没有命令时才依次执行低一级队列中的命令。例如，调度器发现操作命令队列里已经没有命令了，则说明当前 I/O 节点空闲，则利用该空闲时间执行一条缓输出命令。每次执行完毕一条缓输出命令，调度器就要查看一次操作命令队列，看是否在缓输出命令执行时有操作命令来到，若有，则先执行操作命令。当操作命令队列和缓输出命令队列都为空的时候，系统才执行预输入命令。

但是，这种操作顺序并不是绝对的。当操作命令为读某块文件数据，然而，用户曾对该块数据执行过写操作。由于有缓输出机制，该写操作命令可能还在缓输出命令队列中而未被真正写到磁盘上，那么，若简单地按上面的队列顺序进行操作就会造成数据的不一致性。在这种情况下，操作调度器就会分析操作命令和缓输出队列，并将在缓输出队列中取出对应的缓输出命令，将它改成立即输出命令，并插入到操作命令队列的该读操作命令前面，从而保证数据的操作一致性。所以，操作调度器实际上还要起到命令分析的作用。

测试表明，通过操作调度，系统能充分利用空闲时间，减少 I/O 节点有时空闲而有时拥塞，使得整个系统效率得到提高。

3.2 文件 I/O 操作

3.2.1 文件打开 当用户打开文件时，首先向文件服务器发出请求，文件服务器查看该文件是否打开，若文件已经打开，并且使用的是排他模式打开的，就拒绝用户打开文件的请求，并返回原因代码。否则，就读出该并行文件对应的 FAT 信息，并给这次文件打开设定一个全局唯一的序列号，然后根据 FAT 信息向文件各分片对应的 I/O 节点发出打开对应分片文件的请求并传送该文件打开序列号，然后等待回应。当且仅当所有分片对应的 I/O 服务器节点都登记了这次文件打开的序列号并返回了打开成功的消息，系统才认为文件打开成功。此时，文件服务器向用户返回文件打开成功的消息，并传送给用户该文件对应的 FAT 信息。而相应的 I/O 服务器节点则将对应的预输入命令放入预输入队列，以便在系统空闲的时候进行预输入操作，这能在后面文件读的过程中有效地提高文件读的效率。

3.2.2 文件读 用户有两种读文件的方法：1)简单读。2)代理读。简单读操作由一个客户方和若干个 I/O 服务器组成的服务方构成，而客户方就是进行读操作的用户节点。结构如图3。

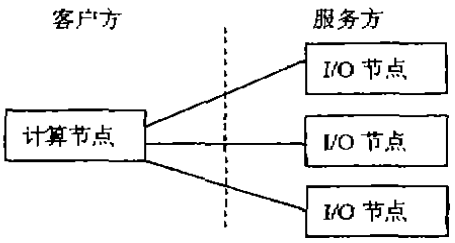

图3 简单读结构图

在客户方：I/O 库函数首先在本地缓冲区中查找对应信息。若本地缓冲区中没有对应信息，则 I/O 库能根据打开文件时文件服务器送来的文件 FAT 信息计算出该部分信息分布于哪些 I/O 节点上，自动将文件读操作分解为多个子操作，并启动一个文件 I/O 线程。由该线程控制向各个相关的 I/O 节点发送子操作命令。

在服务方：在 I/O 节点的命令队列操作模式下，I/O 节点执行读操作命令时先检测预输入队列，看是否有同该操作相重合的预输入命令，若有，则从预输入命令队列中删除该预输入命令。因为再进行这块数据的预输入已经没有意义了，若没有重合的预输入命令，则直接分析该读操作命令，检查在本地缓冲区中是否已经有相关的文件数据。若本地缓冲区中已有相关的文件数据，则直接向用户节点发送缓存的文件数据；否则，就从硬盘中读出数据到缓冲区并向用户节点发送。上面操作完成后，I/O 节点应以当前读取的文件数据结尾为起始地址，生成下一块文件数据的预输入操作命令，将该命令放入预输入命令队列中进行下一次的预输入操作。由于文件读取的连续性，这样进行的预输入能获得良好的效果。当前，由于内存方面的考虑，我们暂时使用了每个文件分配三块缓冲区的策略，若有必要，可以很容易地更改缓冲区的大小及数量。缓冲区的更替我们使用的是最近最少用的原则。缓冲区链表的维护由 I/O 节点负责。通过这种有优先级的命令队列策略，我们既能实现文件的预输入操作以提高文件 I/O 效率，又可以避免文件预输入过多导致真正的 I/O 操作反而等待时间过长的问题。

最后由用户的文件读取线程负责接收并将各 I/O 节点发来的文件数据按照正确的格式在内存中排列。当所有数据都已经收到时，线程结束并通过发送信号提交一个填充了用户要读取的文件数据的缓冲区。值得注意的是，我们在此并未首先检查缓输出命令队列来保证应该输出的数据已经输出到磁盘上，这是因为该一致性控制操作已经由命令调度器完成了，当执行到该读操作的时候已经可以确保没有导致数据不一致的缓输出命令了。

代理读则由一级客户、二级客户和若干个 I/O 服务器组成的服务方构成，其结构如图4。其中一级客户为某空闲节点，二级客户为要读文件的计算节点。代理读往往是在用户节点内存及 CPU 资源非常匮乏，而文件操作又同计算操作并行性较好的情况下，由用户节点在要用文件数据的许多时间前就开始申请一个空闲节点，并将要读取的文件的 FAT 信息和文件 I/O 请求传送给它，而空闲节点上有一个文件 I/O 的 AGENT 程序，负责代替用户节点根据 FAT 信息和 I/O 请求分解成多个子请求，代替计算节点向各相关 I/O

节点发送I/O子请求，并代为接收并缓存I/O节点发来的文件数据，当用户节点需要用这些数据时就能直接向代理的空闲节点上申请，空闲节点能直接从缓冲区中取出数据交给用户节点。这种读模式利用了空闲节点上的CPU和内存资源，扩大了用户所能使用的内存空间，使得操作尽可能地在内存中进行，从而提高了系统操作的效率。但是，当读取的文件数据较少，或同计算任务的并行度较低，由于代理读比简单读多进行了一次通信，所以性能可能反而下降。

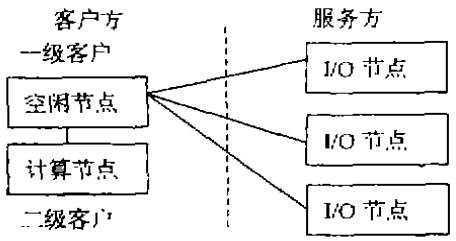

图4 代理读结构图

3.2.3 文件写 类似文件读，文件写也有两种方式：同步写和异步写。顾名思义，同步写是指文件写操作时一定等待各文件分片确实已经写到磁盘上才返回，并允许进行下一个操作。它的好处在于，一旦同步写返回成功，则文件数据一定安全地写入了磁盘；缺点在于，由于一定要等待磁盘操作，所以效率不高。

异步写则融合了缓输出的技术，是指文件写操作时仅仅将输出命令填入各个相关I/O节点的缓输出队列中而不等待各文件分片确实已经写到磁盘上就返回，并允许进行下一个操作。由系统在合适的时候再输出数据到磁盘上。它的好处在于，对用户响应较快，采用输出缓冲，所以效率较高；缺点在于，写操作返回时无法保证文件写成功。由于使用了缓输出操作，所以当I/O节点发生故障时，文件写操作将会失败。虽然系统能通过全局唯一的文件打开序列号发现这个错误，但在此情况下，由于操作失败，异步写的效率将比同步写操作还要低。异步写操作步骤为，由I/O节点为每个打开的文件分片维护一个输出缓冲池，在异步写模式下，当用户节点请求对文件进行写操作时，I/O节点首先将写入数据放入对应文件分片的输出缓冲池，若输出缓冲池中已经有数据，则根据原数据和写入数据的偏移量及大小来判定是否能将这些数据合并，若可以，则将它们合并成一个大数据块。并在合适的时候（包括缓冲的数据已经填满缓冲区、用户关闭该文件分片、用户要读已经写入输出缓冲区但未写入磁盘的数据等）将缓冲区里的数据刷新到磁盘上去。根据实验我们知道，对相同数量的数据而言，以大块数据为单位进行的I/O操作效率要大大高于离散的以小分片数据为单位的I/O操作，所以通过合并数据块，缓输出等操作就能使写操作的效率提高。然而，当I/O节点发生故障的时候，未来得及写入磁盘的数据就会丢失，所以它是不稳固的。但是当文件关闭或下一次文件操作（包括读和写）时若I/O节点无法找到对应序列号的打开的文件分片，则I/O节点就会返回出错信息，用户就能知道文件信息错误了。所以异步写操作不会出现发生错误而用户不可知的情况，因而是安全的。另外，当写操作造成各文件分片发生大小、偏移等变化时，同步写操作立即通知文件服务器进行更新，异步操作则仅在本地的FAT信息中更新，等缓冲数据刷新到磁盘上或文件关闭后再同文件服务器联系，进行FAT信息更新。这是多用户文件系统数据一致性的要求。

3.2.4 文件关闭 文件关闭操作需要将缓冲区内的数据刷新到磁盘上，并通知每个文件分片所在的I/O节点关闭文件分片，并等待关闭成功的信息。当且仅当所有文件分片都关闭成功时，该文件关闭操作才会返回成功信息，并让各I/O节点释放分配的缓冲区内存。否则，就返回出错码。

3.3 文件的创建和删除

进行文件的创建操作时，由用户节点向文件服务器发出请求，由文件服务器分析当前I/O节点的负载情况和文件大小状况，选择分片大小及各分片对应的I/O节点，并生成FAT信息传送给用户节点，用户节点按照FAT信息向各相关I/O节点发送创建分片的请求即可。当所有相关I/O节点返回创建成功的消息后，用户节点通知文件服务器，文件服务器将FAT信息送入FAT文件中保存。

进行文件的删除操作时，同样由用户节点向文件服务器发出请求，由文件服务器读出相关FAT信息并传送给用户节点，用户节点按照FAT信息向各相关I/O节点发送删除分片的请求即可。当所有相关I/O节点返回删除成功的消息后，用户节点通知文件服务器，文件服务器将相关FAT信息从FAT文件中删除。

3.4 文件系统的管理

文件管理员能通过WEB浏览器从文件服务器上下载一个JAVA的Applet Client程序，通过它能在任何地方对文件服务器发出文件管理请求，对文件系统进行管理。实现了文件管理的跨平台性。它能提供文件系统的直观的使用图示化描述，大大方便了文件系统管理员的管理，虽然过去运行文件系统能提供类似的系统图示化描述、管理工具，但是它们都局限于某种固定的操作系统平台，而基于WEB的文件管理工具却

能在任何平台上使用，便于文件系统管理员进行文件管理工作。

4. 系统性能评测

4.1 实验环境

作为国家863项目"分布式一致性平台"的一部分，我们实现了 PFSHNOW，该系统将多台 IBM RS6000 工作站，内存64M，硬盘2G，其上运行的操作系统是 AIX4 1；多台 SUN Ultra 1和 Ultra 30工作站，内存64M，硬盘2G，其上运行的操作系统是 Solaris 2 6；还有多台 PC 机，内存32M，硬盘4G，上面运行的操作系统为 WIN NT4.0连成一个完整的并行文件系统平台。其中 RS6000使用155MB/S 的 ATM 网络连接，SUN 的 Ultra 机采用100M 以太网连接，而 PC 机采用10M 以太网连接。所有机器都已互联。我们采用了 JAVA 为开发工具，使用 MPI 为通信支撑，从而跨越了异种结构机器的藩篱，实现了平台无关性。

4.2 实验结果分析

以下是传输大小为1200KB、1440KB、1920KB 时写操作的数据传输率(KB/S)的直方图和传输大小为3840KB 时读操作的数据传输率(KB/S)的直方图。

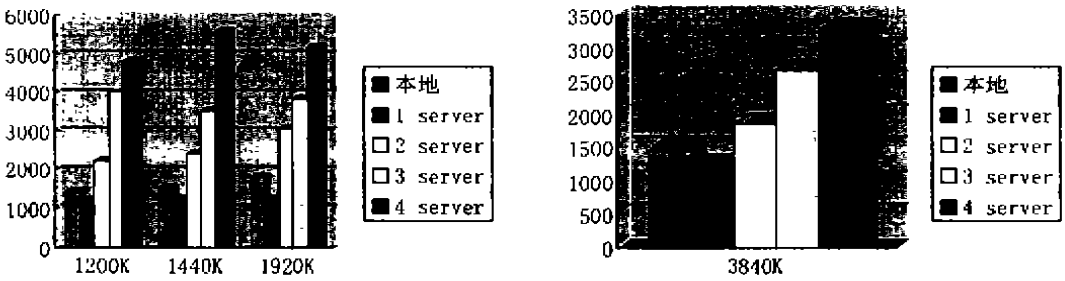

图5 PFSHNOW 的读写试验结果直方图

根据试验的结果图，我们看到，系统成功地将异种平台联系在一起，构成了一个支持异种结构的 NOW 环境的并行文件系统。由于系统采用并行读写和预输入缓输出、多命令队列调度等多种先进技术，系统读写性能得到了明显提高。由于当前 JAVA 和 MPI 性能都不高，所以随着 JAVA 和 MPI 的发展，系统读写性能还将有较大的提高潜力。

参 考 文 献

1 French J C，et al. Performance Measurement of the Concurrent File System of the Intel iPSC/2 Hypercube. J. of Parallel and Distributed Computing，17(1～2)：115～121

2 Moyer S A. PIOUS：a Scalable Parallel I/O System for Distributed Computing Environments. 1994 Scalable High Performance Computing Conf.，71～78

3 Huber J V Jr. PPFS：a High Performance Portable PFS. Supercomputing'95，1995. 385～394

4 Corbett Peter F. Parallel access to files in the VESTA file system. Supercomputing'93，1993. 472～481

5 Rosario J M D，et al Improved Parallel I/O via a Two-phrase Run-time Access Strategy. Computer Architecture News，1993，21(5)：31～38

6 Rosario J M D，et al. High Performance I/O for Parallel Computers：Problems and Prospects. IEEE Computer，1994(March)：59～68

7 李群，谢立，孙钟秀. 并行文件系统的设计. 计算机科学，1996，23(4)：36～39

8 李群. 基于分布式共享存储机制的并行文件系统的研究：[南京大学98届博士论文]