PVFS 文件系统吞吐率分析和改进*)

胡雨壮　孟　丹

(中国科学院计算技术研究所高性能计算机研究室　北京100080)

Throughput Analysis and Improvement of PVFS

HU Yu-Zhuang　MENG Dan

(High Performance Computing Lab, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080)

E-mail: hyz@ict.ac.cn

Abstract　With the rapid developments of cluster systems, the problem of I/O bottleneck becomes more and more severe. A good file system is crucial to the whole systems. Throughput, which means the jobs done per unit time, is an important index of the systems' performance. In this paper, through testing, the authors find that the throughput of PVFS(Parallel Virtual File Systems), which now is popular in the field of parallel processing, is abnormally low. This paper discusses the cause of this phenomenon and improves the throughput of PVFS drastically by eliminating this bottleneck. The performance before and after this improvement has also been compared.

Keywords　Cluster system, Cluster file system, Linux, PVFS, Throughput

1. 引言

近年来工作站机群系统以其低成本、高性能而日益成为高性能计算的潮流。尽管机群系统为许多应用提供了高性能的计算平台，但是其相对低速的文件系统成为了系统的瓶颈，因此提高机群文件系统的性能具有重要的意义。吞吐率是指在单位时间内所处理的作业数，它是评价系统性能的一个重要指标。PVFS 是一个机群文件系统，目前在大规模并行计算中应用较多。本文对 PVFS 文件系统进行吞吐率性能测试时，发现 PVFS 文件系统的吞吐率过低。为此本文跟踪、记录 PVFS 吞吐率测试涉及到的每个操作的用时情况。通过对该数据进行分析，本文找到了 PVFS 吞吐率过低的原因，对 PVFS 进行了改进，并对改进前后的吞吐率进行了比较。

本文简要地对 PVFS 进行了介绍；给出了对 PVFS 的吞吐率测试结果；对 PVFS 吞吐率进行了分析，给出了导致 PVFS 吞吐率过低的原因；对 PVFS 做了改进并进行了性能测试。最后给出了本文的结论。

2. PVFS 简介

PVFS 是一个并行文件系统，用于美国 Argonne 国家实验室的一个由256个结点构成的 Linux 机群系统中。PVFS 是客户/服务器结构的文件系统，结点上所有的创建、删除、读写文件等请求都必须送到服务器方进行处理。PVFS 采用了元数据与文件数据相分离的结构，系统中存在管理元数据的元数据服务器和存储文件数据的存储服务器。元数据服务器仅仅负责管理元数据，如超级块、i 节点等，对所有文件数据的 I/O 由存储服务器处理。PVFS 系统中可以配置多个存储服务器，但只允许存在一个元数据服务器。

元数据中文件数据分布信息包括了机群中磁盘的位置和磁盘上文件的位置。PVFS 将文件数据和元数据存放在本地文件系统的文件中。为便于并行存取，PVFS 文件以分组的形式存放在结点中。文件的分布由三个元数据参数来表示：初始结点号，结点数和分组的大小。通过这些参数和将结点排序，就能完全地确定文件的分布。应用进程在进行文件操作，如打开、创建、关闭、移走文件时直接同管理服务器通讯。当应用进程打开文件时，管理器返回文件数据所在的存储服务器的位置，应用进程通过这些信息直接同这些结点通讯来完成文件存取。为了便于理解，图1给出了 PVFS 的总体结构图。

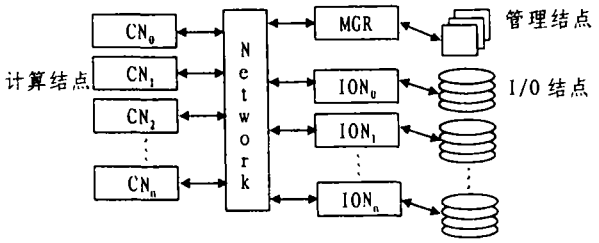

图1

如上所述，PVFS 中结点分为了运行应用的计算结点，处理元数据的管理结点，存储文件数据的存储结点(I/O Node)。管理结点和存储结点也可以用于计算。对于小规模的机群来说，为充分利用资源可以让一个结点同时担负管理、存储或计算的任务，但在机群规模较大的情况下，最好是将它们放在不同的结点上。

PVFS 客户端、元数据管理器端和存储服务器端都是用户级实现。在客户端，核心模块通过队列与客户端进行通讯。由于客户端是用户级实现，因此 PVFS 中生成了一个设备文件(/dev/pvfsd)，客户端通过读写该设备来进入核心。对该设备的读实际上会从队列中取出一个请求，对该设备的写则是向该队列中写入服务端的应答信息。核心模块则对队列直接进行存取。

*)本课题得到国家"八六三"计划资助(项目编号863-306-ZD01-01)。胡雨壮　硕士研究生，主要研究方向为机群文件系统与操作系统。孟　丹　研究员。

在PVFS管理器上，所有的元数据以本地文件系统文件的方式存放。PVFS的inode号是元数据本地文件系统文件的inode号。元数据文件名与PVFS文件对象名相同。在存储服务器上也是利用本地文件系统的系统调用来完成文件读写，并把将PVFS的inode号对101取模所得的数作为数据文件所在的目录，以PVFS的inode号加上".0"作为数据文件名。

3. PVFS吞吐率测试结果

Lmbench的目的是测试整个计算机系统，由一系列小的benchmark组成，其中有相当一部分是用于测试文件系统的性能。它的lat_fs测试创建、删除小文件的延迟，是测试创建、删除小文件吞吐率经常使用的工具。lat_fs在测试创建吞吐率时是首先创建文件，然后对该文件分别写入0k、1k、4k、10k数据，最后关闭文件。因此，lat_fs测得的创建吞吐率实际上包含了创建操作，写操作和关闭文件操作，并且该创建吞吐率还消除了数据缓存的影响。lat_fs分别累计了整个创建过程、删除过程所花的时间，用这两个时间分别除以创建的文件总数和删除的文件总数就得到了创建吞吐率和删除吞吐率。

在利用Lmbench对PVFS进行吞吐率测试时，测试环境配置为4个客户节点，1个元数据服务器、1个存储服务器(与元数据服务器在一台机器上)，所有节点连接在100Mb/s的以太网上。节点的配置为：smp节点(双cpu)，cpu是主频为1GHz的PIII，1GB的内存，每个节点上安装了Turbo Linux6.5，核心版本为2.2.18-2smp。

表1是在上述测试环境下，利用Lmbench测得的PVFS对于小文件的创建、删除吞吐率。其中，创建吞吐率和删除吞吐率之间用/隔开，且0表示客户端与服务器在一台机器上。

表1 PVFS吞吐率测试结果

客户端个数	0	1	2	4
吞吐率(0k)	50/1242	50/995	100/1371	200/1899
吞吐率(1k)	50/1251	50/815	100/1182	181/1500
吞吐率(4k)	33/995	50/869	93/1308	155/1571
吞吐率(10k)	50/1182	33/977	76/1187	127/1808

由上面的测试结果可以看出，PVFS的创建吞吐率过低，并且当客户端与服务器端在一起时，创建吞吐率也并不比客户端与服务器分离的情况下高(在写入1k，4k数据时反而比后者低)。为此本文跟踪分析了PVFS在运行测试程序时，其客户端和服务器端的耗时情况，试图找到上述问题的原因。

4. PVFS吞吐率分析

要分析、提高机群文件系统的吞吐率，就必须要了解机群文件系统吞吐率的时间开销由哪几部分组成，哪个部分是影响机群文件系统吞吐率的主要因素。本文跟踪了PVFS中吞吐率涉及到的所有操作，记载了每部分的时间开销，并对测试的结果进行了较详尽的分析。在对吞吐率涉及到的操作进行跟踪、分析之前，有必要先对文件操作的开销进行一下分类，以确定应该测试哪些方面的数据。

从上面的讨论可以得知，PVFS是客户/服务器模式。本文将文件操作时间开销分为了客户端协议及实现开销、通讯系统开销和服务端协议及实现开销。在文[5]对机群文件系统性能开销的分类中，将性能开销分为了磁盘存取时间、协议开销、少量的CPU处理时间和网络数据传输。这样的分类当然也是可行的，但是不利于在客户端和服务器对时间开销进行跟踪、测试。抛开了实现，纯协议开销是无法取得的。统计单纯的网络数据传输也没有多大意义，需要关注的是具体机群文件系统所采用的通讯系统的整体性能。

对于PVFS，要记录文件操作在PVFS的核心模块(pvfs-.o)、客户端模块(pvfsd)、管理器模块(mgr)和存储服务器模块(iod)的时间消耗。由于PVFS客户端模块是在用户级实现的，存在着核心和用户层的通讯开销，因此还要记录PVFS客户端从核心取得数据和客户端发送数据至收到应答的时间。对于每一个文件操作，要在核心模块产生一个严格递增的数(在程序中是变量xid，以下简称为请求号)。请求号随着每次请求传送到客户端模块、管理器模块和存储服务器模块。在这些模块中分别开辟了一块内存区域，专门记载不同文件操作请求、不同请求号在该模块的用时情况。当该内存区域用完时，程序会自动将该部分数据存储到磁盘上。

PVFS在创建文件时，首先VFS层会做一个验证i节点有效性(revalidate inode)操作。该操作成功后，客户端模块将创建请求传送到管理器模块上，并阻塞等待管理器应答。管理器收到创建请求后，首先创建元数据文件，为该文件分配存储服务器，然后送请求至该组存储服务器。存储服务器创建文件成功后传送应答给管理器，管理器随后回传应答给客户端模块。客户端将结果回传给核心模块，核心模块接着做一次目录查找(lookup)操作，从管理器取回元数据信息。成功创建后，客户端要打开与存储服务器的连接，做一次空(NOOP)操作。因此，一个创建文件(CREATE)操作在客户端要涉及到一次revalidate inode操作，一次创建文件操作，一次lookup操作，一次NOOP操作。在追踪CREATE操作的用时情况时不仅要考虑CREATE，同时还要考虑到另外的三个操作。

对于写操作，如果文件未被打开过，在执行写前PVFS要先打开文件。但在Lmbenc的测试中，写操作紧跟在创建操作后面。因此，写操作只需考虑客户端、存储服务器端写操作和读取文件大小操作(FSTAT)的用时情况。

由于PVFS在关闭文件时，核心模块在产生关闭文件请求后立即返回，客户端在以后的执行中再去管理服务器上关闭文件，因此对于关闭操作暂时可以不予考虑。

PVFS在删除文件时，首先执行revalidate inode操作，从管理服务器取回PVFS的i节点元数据。然后客户端送删除请求到管理服务器上，等待管理服务器的处理结果。管理服务器删除元数据文件后，送删除请求到存储服务器上。收到存储服务器应答后，管理服务器向客户端回传结果。因此对于删除操作除考虑客户端、管理器端、存储服务器端的删除操作时间开销之外，还要考虑revalidate inode操作的开销。

表2 PVFS操作用时情况

	pvfs	pvfsd	xfer	mgr	iod
lookup	559	261	243	145	
create	19379	18653	18633	478	37
noop			191		1
lookup	559	261	243	130	
write	5152	5044	4748		103
fstat			330	231	19
close	14	391	240/133	123	21
revalidate	373	322	305	213	18
unlink	441	421	407	319	88

表2是客户端与管理器、存储服务器在同一台机器时的创

建、写、关闭、删除时间(平均时间)分布情况。

由于在 PVFS 中传送请求和处理回传结果是 pvfs_vl_xfer.c 中的过程执行的,因此在下表中单独列出了在客户端传送请求和处理回传结果(表中的 xfer 项)的时间消耗情况。和本章中其他用时情况分析表一样,表2中的时间单位是微秒。

表2中 pvfs 栏是完成整个文件操作所花的时间,pvfsd 栏是客户端所花时间(包括与元数据管理器和存储服务器的通讯、等待时间),mgr 栏是管理服务器所花时间(包括与客户端和存储服务器的通讯、等待时间),iod 栏是存储服务器所花时间(不含通讯时间)。Close 操作在 xfer 栏有两个值,这是因为根据 PVFS 协议,如果与存储服务器的连接已经建立,总是先到存储服务器上关闭文件,然后再到管理服务器上关闭文件。另外,close 操作在 PVFS 中执行时,核心在将关闭文件请求插入队列后立即返回,真正的 close 处理在以后 pvfsd 从队列中取出请求时执行,因此本表 pvfs 栏中 close 操作的时间大大小于其真正执行的时间。

5. PVFS 吞吐率改进

从表2中可以看出,创建文件操作竟然平均花了接近20个毫秒,与其它操作相比不成比例。并且对于创建操作,管理服务器上只花了478个微秒,也就是说客户端和管理服务器通讯花了十几个毫秒的时间,这显然是不正常的。

PVFS 管理服务器回传创建结果给客户端是分两次完成的。一次是传送管理服务器的应答,一次是传送管理服务器分配给这个文件的存储服务器的信息。通过进一步的跟踪发现,客户端其实很快就收到了第一次管理服务器的应答消息,创建文件操作的大部分开销是在接收随后的存储服务器的信息上。

问题只能是出在管理服务器的这两次回传上。为此,本文模拟 PVFS 客户端和管理服务器的通讯方式编写了两个小程序,在这两个程序中统计发送和接收所花的时间。程序的流图如图2。

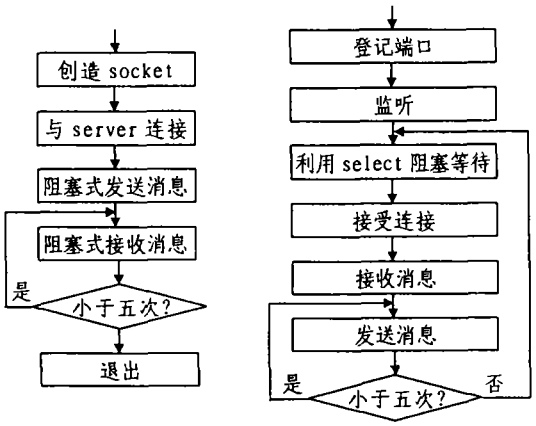

图2 测试程序流程图

图2的左边是 client 的处理流程,右边是 server 的处理流程。每次 client 接收的消息长度为100个字节。在运行上述程序时,我们发现 client 在第三次接收消息时花的时间超过8个毫秒。因此,可以确定 PVFS 在创建文件时,客户端和管理器的通讯模式是有问题的。

在 TCP 通信时,write 操作并没有把数据实际写向网络,而是将数据从应用缓冲区拷贝到 TCP 层的发送缓冲区中。在 PVFS 的创建文件通讯方式中,使用了两个 write 系统调用来发送数据。这两次数据可能被放在不同的 TCP 数据段中,然后在不同的时刻发送到网络中。为了确认这是 PVFS 创建吞吐率过低的原因,需要对 PVFS 管理器端发送创建文件结果的过程进行改动,将管理器端的应答和存储服务器信息合并起来,用一个 writev 系统调用发送给客户端。

同时从表2中可以看到 PVFS 的写操作花了数个毫秒,这也是不正常的。分析 PVFS 的写操作流程,在进行写操作时,PVFS 首先要生成写任务,随后执行该任务。一个写任务包括一个写请求,写传送,写应答这三个子任务。写请求首先被送到存储服务器上。存储服务器根据请求构造写任务,然后执行该任务,从客户端接收数据,并送回应答。从上述过程可以看出 PVFS 在执行写时,其数据也是分次传送的,因而也存在上述问题。对写操作也要进行修改,从客户端一次将请求和数据传送到存储服务器上去。同时 PVFS 在打开文件后,要去存储服务器上做一个空操作(NOOP),以建立客户端与存储服务器的连接。这样做也存在着上面的问题,因此连接要改在写数据时建立。下面是做了上述改动后 PVFS 的吞吐率测试结果:

表3 PVFS 修改后吞吐率测试结果

客户端个数	0	1	2	4
吞吐率(0k)	697/1268	530/997	776/1393	803/1835
吞吐率(1k)	352/1247	246/896	332/1261	317/1317
吞吐率(4k)	290/961	242/932	296/1275	247/1829
吞吐率(10k)	266/1216	96/835	189/1364	220/1972

表4是修改后 PVFS 客户端与服务器在一起时的用时情况:

表4 PVFS 修改后操作用时情况

	pvfs	pvfsd	xfer	mgr	iod
lookup	545	250	232	139	
create	1348	619	599	477	37
lookup	545	250	232	139	
write	860	770	282		91
fstat			336	244	19
close	15	410	258/136	130	22
revalidate	363	313	299	211	19
unlink	438	420	407	310	99

从表4可以看出,CREATE 和 WRITE 操作所花时间大幅减少,这是修改后吞吐率提高的主要原因。表5是1个客户端和4个客户端时用时情况的统计结果。

表5 PVFS 修改后操作用时情况

	1个客户端					4个客户端				
	Pvfs	pvfsd	xfer	mgr	iod	Pvfs	pvfsd	xfer	mgr	iod
lookup	787	377	366	116		2299	1098	1087	149	
create	1675	714	704	426	28	3885	1662	1653	657	31
lookup	787	377	366	116		2299	1098	1087	149	
Write	1973	1910	1277		100	2775	2708	1348		129
fstat			434	194	13			1145	235	13
close	13	575	376/232	126	16	14	1314	1061/305	146	19
reval	471	429	419	185	13	1018	1098	966	224	13
unlink	537	491	482	247	55	1209	1162	1153	347	42

结论 通过将多次调用 write 发送数据改为一次生成数据,用一个 writev 发送所有数据,PVFS 的创建吞吐率得到了

大幅提高。由上述结果可以看出，本文对造成 PVFS 吞吐率过低的原因进行的分析是正确的，所做的改进也是有效的。

参 考 文 献

1 Carns P H, et al. PVFS: A Parallel File System for Linux Clusters
2 Anderson T E, et al. Serverless Network File Systems. ACM Trans. on Computer Systems, 1996, 14(1): 41～79
3 王建勇. 可扩展的单一映象的文件系统:[博士学位论文]. 中国科学院计算技术研究所, 1999
4 Corbett P F, et al. Parallel file systems for the IBM SP computers. IBM Systems Journal, 1995, 34(2): 222～248
5 冯军. 机群文件系统性能优化中的关键问题研究:[硕士学位论文]. 中国科学院计算技术研究所, 2001
6 李善平, 刘文峰, 李程远, 王焕龙, 王伟波. Linux 内核2.4版源代码分析大全. 机械工业出版社, 2002
7 Sun Microsystems, Inc. NFS Version 4 Protocol. RFC3010
8 Shepler S. NFS Version 4 Design Considerations. RFC 2624, June 1999
9 Sandberg R, et al. Design and Implementation of the Sun Network Filesystem. In: Proc. USENIX Summer 1985
10 Ligon III W B, Ross R B. Implementation and Performance of a Parallel File System for High Performance Distrubuted Application
11 Vahalia U. Unix Internals: The New Frontiers. Prentice-Hall, 1996

(上接第131页)

个归约求和算式，而且相互之间有计算顺序关系，只要将归约求和部分定义为中间变量，这种计算的顺序性就消除了，可以归纳为这些中间变量组成的向量的归约求和，从而提高并行计算效率。

不失一般性，归约计算的数学形式表达为：

$$S_{i_1,i_2,\cdots,i_m}=\underset{(i_{m+1},\cdots,i_n)\in\Omega_{red}}{\mathscr{R}} f(u_{i_1,i_2,\cdots,i_n},v_{i_1,i_2,\cdots,i_n},\cdots) \quad (13)$$

$$(i_1,i_2,\cdots,i_m)\in\Omega_{iter}$$

其中，$\mathscr{R}$ 为归约运算符，f 为表达式，$S_{i_1,i_2,\cdots,i_m}$ 为归约变量，Ω_{red} 为归约空间，Ω_{iter} 为迭代空间。整个计算空间 $\Omega=\Omega_{iter}\times\Omega_{red}$。按数据分布特性将 Ω_{iter} 和 Ω_{red} 进行分解：

$$\Omega_{iter}=\Omega_{iter,1}\times\Omega_{iter,2}$$

$$\Omega_{red}=\Omega_{red,1}\times\Omega_{red,2}$$

其中，$\Omega_{iter,1}$ 是由在 Ω_{iter} 空间中数据分布的维组成，$\Omega_{iter,2}$ 是由在 Ω_{iter} 空间中数据不分布的维组成，$\Omega_{red,1}$，$\Omega_{red,2}$ 的定义是类似的，整个空间 Ω 分解为：

$$\Omega=\Omega_{iter,1}\times\Omega_{red,1}\times\Omega_{iter,2}\times\Omega_{red,2}$$

按各子空间先后次序由外层到内层循环展开，可得到最佳的并行向量归约。

按照上述原理，(9)式的并行归约循环层次由外层到内层的次序是 σ、δ、θ、i、j、k。其中(σ、δ、θ)及(i、j、k)二组集合内的循环次序根据存储分配从优化 cache 命中率考虑安排。

4 边界条件的并行处理

在前面的讨论中都忽略了边界条件的处理。在许多实际问题中边界条件的处理比较复杂，它的计算公式往往与内点计算公式不同，变量依赖关系也不同，而且往往会更复杂，从而增加了并行计算的复杂性。

在本文列举的数值模拟算法中，物面和流场外边界处理很复杂。在物面，当气体分子撞击物面并发生了反射，则速度分布函数 $f^{w}_{\sigma,\delta,\theta}$ 是壁面密度 ρ_w 的函数，后者的计算式类似于(9)中第一式，且更为复杂，需要在 Ω_v 空间并行归约计算，如果气体分子未撞击物面，则采用原始方程预测、校正两步在位置空间二阶精度的单边差分格式计算。在流场外边界根据分子逆变速度 W 的方向，计算公式也不一样。为了使 L_5 算子在整个求解空间同时进行并行计算，而且不使多重并行现象出现，需要在变量依赖关系分析的基础上，对这一算子的算法步骤作适当调整。下面就是本文采用的 L_5 算子的算法步骤。

步骤1　在 Ω_v 空间并行归约求解壁面反射密度 $\rho_{w_{i,j}}$。

步骤2　在 Ω_v 空间并行，内层是 Ω_r 空间的 j、i 循环。

步骤2.1　k 循环计算通量 H；

步骤2.2　k 循环内点计算 U^*(预测步)；

步骤2.3　利用边界条件计算物面和外流场的 U^*；

步骤2.4　k 循环计算通量 H；

步骤2.5　k 循环内点计算 U^{n+1}(校正步)；

步骤2.6　计算物面和外流场的中间数据，置于 U^{n+1} 中；

步骤3　在 Ω_v 空间并行归约求解壁面反射密度 $\rho_{w_{i,j}}$。

步骤4　利用边界条件在 Ω_v 空间并行求解物面和外流场的 U^{n+1}。

5 并行程序设计与实验结果

并行程序设计是基于并行程序概念设计系统(PPCDS)[5]进行的，利用 PPCDS 系统的 DPHL 交互编辑器编写 DPHL 源程序，这是一种基于数据并行高层描述语言的形式化建模程序[6,7]，经过编译器进行词法、语法和语义分析后，由并行算法分析器进行分析与优化，由并行程序综合器自动生成并行 HPF 源程序，最后在高性能并行计算机上进行大规模并行计算。计算空间规模是，离散速度空间点16×16×16，物理坐标空间点51×37×63，计算对象为圆球绕流，数据在离散速度子空间 Ω_v 三个方向分布，实测加速比如表2。

表2　并行计算加速比

PE 数	128	256	512	1024
迭代1步平均计算时间 t(秒)	33.23	17.06	9.22	5.15
实测加速比	1	1.94	3.60	6.45
理想加速比	1	2	4	8

由上表可以看出，程序获得了很好的加速比，说明本文采用的各种并行计算策略是可行的，PPCDS 系统的并行识别技术与自动生成并行程序的效率较高。

参 考 文 献

1 李志辉. 从稀薄到连续流的气体运动论统一数值算法研究:[博士学位论文]. 中国空气动力研究与发展中心, 2001
2 张涵信, 等. NND 格式的推广及在年新粘性流计算中的应用. 空气动力学学报, 2000, 12(2): 121～129
3 吕涛, 等. 区域分解算法－偏微分方程数值解新技术. 科学出版社, 1997
4 Wolfe M, Banerjee U. Data Dependence and its Application to Parallel Processing. Intl. Journal of Parallel Programming, 1987, 16(2)
5 并行程序概念设计系统(PPCDS)参考手册:[江南计算技术研究所内部技术报告]. 2001.9
6 陆林生, 等. 面向科学计算的数据并行高层建模语言. 计算机研究与发展, 2001, 38(7. 增刊): 153～159
7 董超群, 等. 域:支持并行程序概念设计的一种抽象手段. 计算机科学, 2001(10): 24～28