一种高性能 I/O 方法*)

霍严梅　鞠九滨　胡　亮
(吉林大学计算机科学与技术学院　长春 130012)

摘　要　并行文件系统是解决 I/O 瓶颈问题的重要途径。研究表明，科学应用中跨越式的文件访问模式与现存并行文件系统访问这些数据的方法的结合，对于大型数据集的访问其 I/O 性能是难以接受的。为了提高并行文件系统中对不连续数据的 I/O 性能，创建了一种新型高性能 I/O 方法：用户自定义文件视图结合合并 I/O 请求。并且在 WPFS 并行文件系统中实现了该方法。研究和实验结果表明，该方法具有增强科学应用性能的潜力。

关键词　跨越式文件访问，不连续数据访问，用户自定义文件视图，合并 I/O 请求

A High Performance I/O Method

HUO Yan-Mei　JU Jiu-Bin　HU Liang
(College of Computer Science and Technology, Jilin University, Changchun 130012)

Abstract　Parallel file system is an important way of solving the I/O bottleneck. Research showed that the strided file access pattern in scientific applications combined with current parallel file system methods to perform these accesses lead to unacceptable performance for large data sets. To enhance performance of discontiguous data access, a new high performance I/O method has been created: the user-defined file view and combination I/O requests. This method has been implemented on WPFS. The research and experimentation results show that this method can substantially enhance the performance of scientific applications.

Keywords　Strided file access, Discontiguous data access, User-defined file view, Combination I/O requests

1　概述

随着处理器和内存技术的飞速发展，在许多高性能计算应用中，I/O 已经越来越成为影响系统性能的瓶颈。开发并行文件系统是解决这一问题的主要方法之一，到目前为止，已经出现了许多并行文件系统，大多数的并行文件系统都建立在机群或网格上[1~3,9,16,5]。并行文件系统能够按照一定的规则，将一个并行文件分布存储到多个节点的磁盘上，然后根据用户程序提出的 I/O 请求并行地访问文件数据。也就是说通过逻辑上聚合多个独立存储设备为一个单一的存储子系统来获得高的 I/O 性能，解决 I/O 的瓶颈问题。

而实际上这种性能的提高要受到很多因素的影响，比如文件的分布、应用程序的访问模式、计算机网络的性能等。如果文件的物理分布能够和用户的 I/O 请求很好地匹配，并行文件在多个节点上的分布以及用户程序对各节点的请求都很平均，并且计算机网络具有足够的总带宽，那这个机群应该能够提供比较理想的可扩展 I/O 性能。但实际上，由于网络性能的限制以及为分布和收集文件数据所进行的机间通信带来的开销，很多用户应用程序无法通过并行文件系统获得理想的可扩展 I/O 性能[6]。这主要是由文件数据的物理分布与用户应用程序所期望的逻辑分布不匹配引起的。文[7]的研究表明，80%的并行文件访问使用一种跨越式文件访问模式。传统的方法是，并行文件系统执行多个连续 I/O 操作来满足这些请求，从而带来大量的 I/O 请求处理开销。

本文描述了一种高性能的不连续数据访问方法：用户自定义文件视图结合合并 I/O 请求方法。在我们的 WPFS 并行文件系统中，用户应用程序可以根据自己的需要，定义自己的访问子集，通过一个函数调用来访问文件中多个不连续的部分，并行文件系统一级使用合并 I/O 方法支持用户在自定义文件视图中描述的不连续 I/O 模式，从而减少了多次调用产生的通信开销。实验结果表明，用户自定义文件视图和合并 I/O 请求方法的结合很好地改善了对不连续数据的 I/O 性能。

在本文第 2 部分将概要介绍 WPFS，第 3 部分分析不邻近 I/O 问题并详细介绍我们的用户自定义文件视图结合合并 I/O 请求的方法，第 4 部分介绍实验结果，第 5 部分是同类工作比较，最后进行总结。

2　WPFS 简介

我们使用 WPFS 系统实现了我们的不连续数据访问方法。WPFS 是一个建立在 Windows 上的机群并行文件系统，应用于高性能低性价比的 PC 机群上。与许多其它的并行和机群文件系统一样，WPFS 也设计为客户-服务员模式，如图 1 所示。

WPFS 系统具有多个 I/O 服务员。I/O 服务员通常运行在机群的不同节点上，这些节点具有自己的磁盘，称为 I/O 节点。WPFS 建立于本地文件系统之上，这样 Windows 的高速缓存可以降低单个本地磁盘访问的开销。每个 WPFS 文件

*)本文得到国家自然科学基金(批准号：60473099)和吉林省杰出青年基金(批准号：20040119)的资助。霍严梅　博士研究生，讲师，从事并行计算与计算机网络的研究；鞠九滨　教授，博士生导师，从事分布式系统与计算机网络的研究；胡　亮　教授，博士生导师，从事分布式系统与计算机网络的研究。

分布于多个 I/O 节点的多个磁盘上。

WPFS 还具有一个系统管理员，主要负责元数据操作，如文件许可，文件大小，分布单元长度，数据在磁盘上的位置等等。当客户端应用程序打开一个并行文件时，WPFS 管理员直接告诉它们相应的 I/O 服务员的位置。管理员不参与读/写操作，客户端的库函数与 I/O 服务员处理所有的文件 I/O 而无须管理员的干涉，这样保证管理员不会成为系统的瓶颈。客户端、I/O 服务员和管理员进程不必运行于不同的机器上。当然，在不同的机器上运行这三者会带来系统的高性能。

应用程序通过客户端的库函数(wpfs_lib)与 WPFS 系统进行交互。

WPFS 是一个为并行应用程序提供高速文件数据访问的并行文件系统。在整个机群范围内，WPFS 提供了一个一致的名字空间，使得用户可以控制数据在不同 I/O 节点磁盘上的分布。

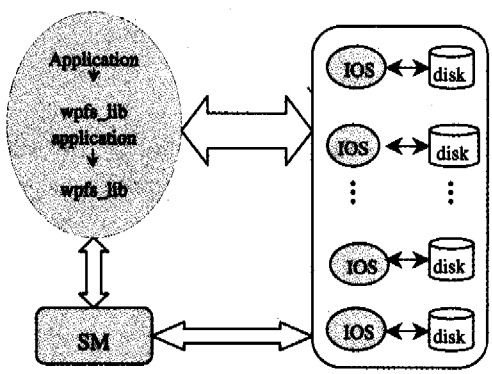

图 1　WPFS 并行文件系统

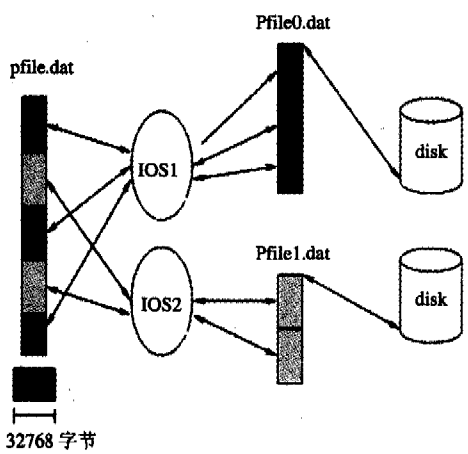

图 2　并行文件分布实例

并行文件系统中的文件，通常称为并行文件，将按照指定的划分单元(stripe)分布于系统中若干个 I/O 节点上，每个节点上的部分称为该并行文件的一个子文件。子文件的数目与存储这个并行文件的所有 I/O 节点的数目是相等的。每个子文件是物理上连续的字节流，但在逻辑上是由若干不连续的划分单元组成。

一个并行文件分布的例子如图 2 所示。

在缺省的情况下，用户要获得不连续的数据必须使用多个请求，每个请求针对一段连续的数据段。这样，在不连续请求中，连续的 I/O 调用的数目就随着连续数据段数目的增加而线性增加。我们期望在文件系统中建立能够优化不连续请求的机制，从而大大减少 I/O 请求的数目。

3　用户自定义文件视图与合并 I/O 请求方法相结合

3.1　不连续数据访问

不连续数据访问即是对文件中不邻近数据的访问。在并行文件系统中，一个子文件中的数据物理上是连续的但逻辑上是不连续的。对文件中数据是连续的情况，为了优化访问，可以通过一个内存操作把数据缓存到内存中，从而仅需一次磁盘文件读/写操作，其余的对这些数据的操作就可在内存中执行了。而当文件数据是不连续的时候，仅有缓存是不够的，因此必须使用其它的方法来执行对这种文件中不连续数据的访问。

研究表明，科学应用通常访问的是文件中许多小的不连续的数据区域[11~15]。文[7]的研究表明，80%的并行文件访问使用一种跨越式文件访问模式。如果必须使用连续的 I/O 请求来执行这些数据访问的话，会产生大量的 I/O 请求处理开销，势必会影响应用程序的运行时间。

因此，我们在 WPFS 的设计中提供了用户自定义文件视图接口，这与 MPI-IO[8] 中的文件视图的概念很相似，并行文件系统 Vesta[9] 和 PVFS[10] 中也提供了类似的功能。这些系统允许用户描述不连续数据的访问模式，但如果文件系统一级不支持不连续访问，对应用程序性能改善的能力非常有限。因此，我们在 WPFS 并行文件系统中提供合并 I/O 请求的支持。

3.2　用户自定义文件视图

结合文[7]的研究结果，为了使用户能在一个 I/O 请求中描述多个不连续的部分，每一部分是一段连续的数据，从而在一个系统调用中产生多个 I/O 请求，我们使用了用户自定义文件视图的方法。

用户自定义文件视图是文件的一个逻辑视图，对于一个文件来说，不同的用户程序或者相同用户程序的不同调用都可以选择不同的逻辑视图。一个用户自定义文件视图由多个记录组成，记录是文件中一段连续的字节流，其长度(rs)由用户程序在打开文件时指定；两个相邻记录之间有固定的间距称为跨距(rd)，跨距也是由用户程序在打开文件时指定；另外在用户程序中还要指定第一个记录距离文件头的偏移(ro)。图 3 展示了文件 file. dat 的一个用户自定义文件视图，根据用户程序中给定的参数：ro、rs、rd，可得该用户视图由三个记录组成。

在用户自定义文件访问模式中，用户程序在打开一个已经存在的并行文件时给出相应的访问参数(ro，rs，rd)，定义自己的文件访问视图，这样在后面的文件 I/O 操作中就可以按照自己定义的文件视图来访问该并行文件。

在 WPFS 中，对应用户自定义文件访问模式，用户能看到的读文件的接口函数为：

```
int wpfs_uread( char * pathname,
                int rs,
                int ro,
                int rd)
```

其中，pathname 是文件名，rs，ro，rd 分别是用户自定义文件视图的三个参数。WPFS 并行文件系统将根据这三个参数进行计算获得具体的 I/O 请求，以便 I/O 服务员恰当地处理这些请求。

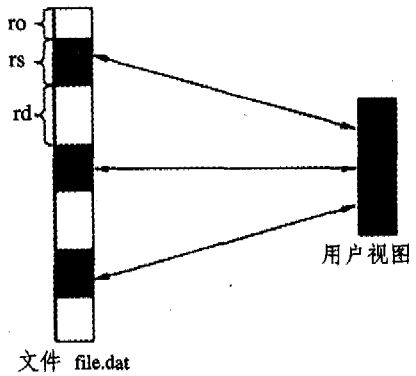

图 3 用户自定义文件视图示例

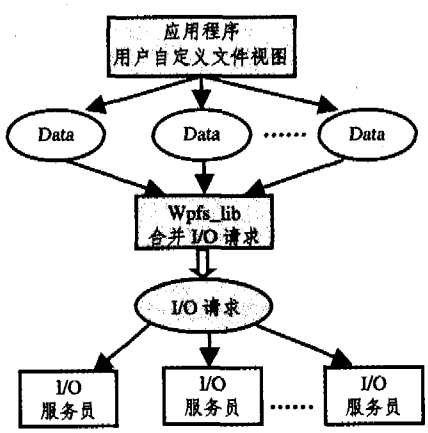

图 4 合并 I/O 请求方法流程

3.3 合并 I/O 请求方法

用户自定义文件视图方法允许用户描述不连续数据的访问模式,但如果文件系统一级不支持不连续数据访问,该方法对应用程序性能改善的能力非常有限。因此,我们在并行文件系统一级使用合并 I/O 请求方法来支持不连续数据访问。

WPFS 客户端通过 wpfs_lib 形成 I/O 请求。这些 I/O 请求包含一个并行文件的信息(如元信息、文件分布参数等),通过这些请求通知 I/O 服务员执行如文件读、写、打开和关闭等操作。为了使一个 I/O 请求能够描述在用户自定义文件视图中提出的不连续数据 I/O 请求,我们在 I/O 请求结构中增加了一个字段来通知 I/O 服务员该请求后面跟着一个长度可变的尾部数据,在尾部数据中描述了对一个子文件的不连续 I/O 请求的文件偏移和文件长度。

我们在 I/O 服务员代码中增加了接收尾部数据的功能,这样 I/O 服务员就可以正确处理这样的请求并完成 I/O 访问。在一个请求的尾部数据中,我们选择最多允许描述 64 个连续的文件区域,如果连续的文件区域多于 64 个,就可以分成多个 I/O 请求来完成。有了这个限制,I/O 请求和其后的数据就可以放到一个以太网帧中通过网络来传输。图 4 显示了用户自定义文件视图结合合并 I/O 请求方法的执行流程。

4 实验结果

本部分我们通过模拟实验测试了我们的用户自定义文件视图结合合并 I/O 请求方法。以下是我们的实验环境、实验设计及数据,最后对实验结果进行分析。

4.1 实验环境

我们用实验室中的 10 台 PC 机组成一个机群,每个 PC 机的配置如下:

(1)一个 500MHz 的 Pentium 3 处理器;

(2)512MB 的 RAM;

(3)一个 100Mbits/s 的高速以太网卡(全双工模式);

(4)Windows XP 操作系统。

4.2 实验设计及结果分析

现存的不连续数据访问方法有多重 I/O 请求方法和数据缝合 I/O 技术。我们通过模拟实验来测试我们的不连续数据访问方法、多重 I/O 请求及数据缝合 I/O 三种方法的并行读性能。在实验中,设置每次数据访问的总数据量为 1GB,通过调整用户自定义文件视图的参数,产生不同的连续数据访问次数,对同一访问次数使用三种方法分别进行访问,获得每种方法所需要的时间,然后对三者进行比较。

在模拟测试中使用该 PC 机群的 8 个节点作为 I/O 服务员节点,其中有一个节点同时充当服务管理员和 I/O 服务员两个角色。分别测试了有 8 个和 16 个客户端应用程序两种情况。实验中并行文件的划分单元尺寸为 32kB。

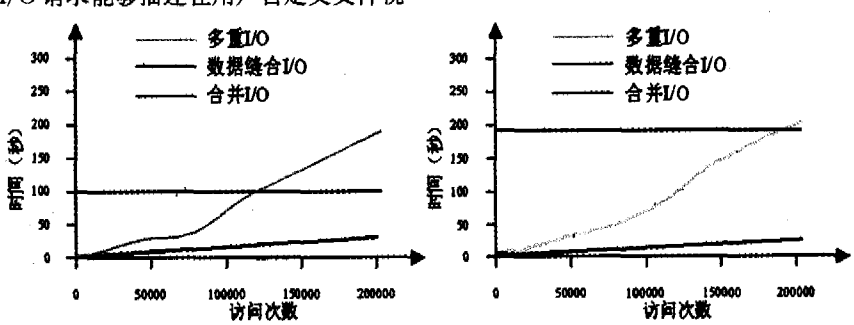

图 5 3 种不连续数据访问方法并行读性能比较

图 5 展示了有 8 个客户端程序(左图)和有 16 个客户端程序(右图)两种情况下的模拟实验结果。

通过图 5 中的实验结果我们可以看出,多重 I/O 与合并 I/O 请求两种方法的访问时间随着访问次数的增加而线性增加。原因是访问的总数据量是一定的,当访问次数增加时,要访问的连续数据区域的个数也在增加,只是每个连续数据区域的长度变小了。多重 I/O 必须相应地增加 I/O 请求的数目,虽然合并 I/O 请求方法也要增加 I/O 请求的数目,但由于一个 I/O 请求可以携带 64 个连续数据区域请求,故合并 I/O 请求方法受访问次数变化的影响要比多重 I/O 受到的影响小得多。这一点在左、右两图中都清晰地体现出来。

对于数据缝合 I/O,在客户端数目一定的情况下,读数据所需要的时间保持恒定,这是因为不论访问次数是多少,经过

(下转第 267 页)

上可以看出应用优化代码消除技术可以明显提高二进制翻译生成的中间代码的质量：减少基本块个数 14.7%～19.8%（平均减少 16.3%），控制流边减少 10.3%～16.5%（平均减少 12.1%），指令条数减少 3.8%～10.3%（平均减少 5.8%）。

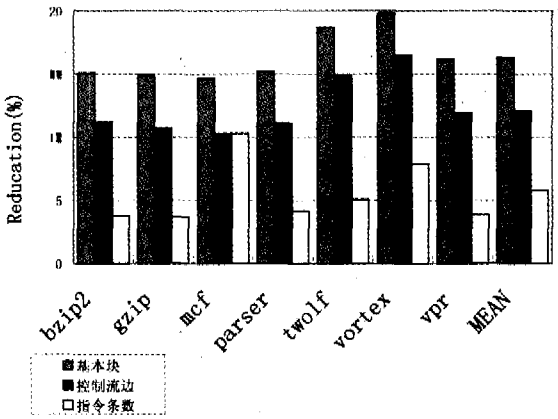

图 3　消除优化代码后控制流图简化情况

结论　编译器充分应用 IA-64 架构特征对程序进行优化，在提高程序性能的同时导致对低级代码结构的深度重构，对二进制代码翻译来说，很难恢复原程序的逻辑，增加工作的复杂性。本文提出的算法可以消除代码优化后的效果，使得识别原程序结构的工作简化，实验数据表明，该方法能够有效地缩减优化后的 IA-64 二进制代码在二进制翻译中产生的控制流图的大小和复杂性。

参 考 文 献

1　Byme E J. Software reverse engineering: a case study [M]. Software-practice and Experience, 1991, 21(12): 1349～1364
2　Intel IA-64 Architecture Software Developer's Manual [R]. Intel Corp, July 2000
3　Pugh W. The Omega test: a fast and practical integer programming algorithm for dependence analysis. [J] August, ACM, 1992, 35: 102～114
4　Debray S K, Muth R, Weippert M. Alias analysis of executable code [C]. In: 25th ACM SIGPLANSIGACT Symposium on Principles of Programming Languages (POPL-98), January 1998. 12～24
5　Cifuentes C, van Emmerik M, Ramsey N. The Design of a Resourceable and Retargetable Binary Translator [C]. In: Proceedings of the Working Conference on Reverse Engineering, Atlanta, USA, IEEE CS Press, Oct. 1999. 280～291

（上接第 261 页）

缝合以后，系统所传输的数据量是相同的。而当客户端加倍时，由于要读取的数据总量不变，每个客户端所读数据变为原来的一半，这样经缝合以后，传输的数据量就变为原来的 2 倍，所以读数据所需时间也加倍。

5　同类工作比较

目前对不连续访问模式问题的解决方案包括：多重 I/O 请求方法和数据缝合 I/O 技术[4,14]。

大多数的并行文件系统接口只允许在一个 I/O 操作中访问一段连续的文件区域。使得一个不连续数据访问请求需要多个 I/O 操作来进行，结果带来大量的传输以及处理 I/O 请求的开销以及磁盘访问的开销。我们称这种对不连续数据访问的处理方法为多重 I/O。对于多重 I/O 来讲，最好的访问模式是仅对几个在内存和文件中都存在的数据区域进行访问的情况。

数据缝合 I/O 技术通过将文件中的一大块数据移动到内存缓冲区——数据缝合缓冲区中来处理不连续数据访问，而必要的数据移动操作都是在客户端的内存中进行的。数据缝合 I/O 方法减少了对物理上邻近的不连续数据的 I/O 请求的数目。主要缺点是要访问和通过网络传输无用的数据。另一个小不足是写操作的情况，当访问的文件数据区域不连续时，必须使用 read-modify-write 的方式。对数据缝合 I/O 模式最理想的是当访问许多不连续的数据区域且两个相邻区域之间间隔很小的情况。

我们的用户自定义文件视图结合合并 I/O 请求方法减少了一个不连续数据访问中 I/O 请求的数目，其方法是在一个单一的合并 I/O 请求中描述多个文件区域。在大多数情况下合并 I/O 请求方法要优于多重 I/O 请求方法和数据缝合方法。

结论　科学应用通常访问的是文件中许多小的不连续的数据区域。在并行文件系统中，如果使用传统的访问方法来访问这些不连续数据，无法获得理想的可扩展 I/O 性能。本文实现了一种不连续数据访问方法：用户自定义文件视图结合合并 I/O 请求方法。实验结果表明，该方法比现存的不连续 I/O 方法具有更好的 I/O 性能。

参 考 文 献

1　Schmuck F, Haskin R. GPFS: A Shared-Disk File System for Large Computing Clusters [C]. In: Proceedings of the Conference on File and Storage Technologies (FAST'02). Monterey, CA, January 2002. 231～244
2　Ligon III W B, Ross R B. PVFS: Parallel Virtual File System [J]. In: Sterling T. ed. Beowulf Cluster Computing with Linux. MIT Press, November 2001. 391～430
3　Nieuwejaar N, Kotz D. PIOUS: A Scalable Parallel I/O System for Distributed Computing Environments [C]. In: Proceedings of the Scalable High-Performance Computing Conference, 1994. 71～78
4　Thakur R, Choudhary A, Bordawekar R, et al. Passion: Optimized I/O for Parallel Applications [J]. IEEE Computer, 1996, 29(6): 70～78
5　Oldfield R, Kotz D. Armada: a parallel I/O framework for computational grids [J]. Future Generation Computer Systems, 2002, 18: 501～523
6　French J C, Pratt T W, Das M. Performance Measurement of the Concurrent File system of the Intel IPSC/2 Hypercube [J]. Journal of Parallel and Distributed Computing, Jan/Feb 1993
7　Nieuwejaar N, Kotz D. Low-level Interfaces for High-level Parallel I/O [C]. In: Workshop for I/O in Parallel and Distributed Systems, IPPS 1995. 47～62
8　Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface [EB/OL], July 1997. http://www.mpi-forum.org/docs/docs.html
9　Corbett P F, Feitelson D G. The Vesta parallel file system [J]. ACM Transactions on Computer Systems, 1996, 14(3): 225～264
10　The parallel virtual file system. http://www.parl.clemson.edu/pvfs/[EB/OL]
11　Baylor S, Wu C. Parallel I/O Workload Characteristics Using Vesta [C]. In: Jain R, Werth J, Browne J, eds. Input/Output in Parallel and Distributed Computer Systems. Kluwer Academic Publishers, 1996. 167～185
12　Crandall P, Aydt R, Chien A, et al. Input-Output Characteristics of Scalable Parallel Applications [C]. Proceedings of Supercomputing '95. ACM Press, 1995
13　Nieuwejaar N, Kotz D, Purakayastha A, et al. File-Access Characteristics of Parallel Scientific Workloads [J]. IEEE Transactions on Parallel and Distributed Systems, October 1996, 7(10): 1075～1089
14　Thakur R, Gropp W, Lusk W. Data Sieving and Col-lective I/O in ROMIO [C]. In: Proc. of the Seventh Symposium on the Frontiers of Massively Parallel Computation, 1999. 182～189
15　Thakur R, Gropp W, Lusk W. On Implementing MPI-IO Portably and with High Performance [C]. In: Proc. of the Sixth Workshop on Input/Output I Parallel Distributed Systems, 1999. 23～32
16　Schwan P. Lustre: Building a file system for 1,000-node clusters [C]. Proceedings of the Linux Symposium, Ottawa, Ontario, Canada. July 2003