计算机科学1996 Vol.23 №.4

并行文件系统的设计

李 群 谢 立 孙种秀

（南京大学计算机科学系 南京210093）

摘 要 Parallel file system is a software approach to alleviating the I/O bottleneck exploiting data declustering technique. This paper presents the components of a parallef file system, that is, file servers, storage servers and clients. Moreover, it addresses the problems in designing a parallel file system, such as caching and prefetching, interface design, data mapping, data distribution and fault tolerance or reliability. To describe the paradigm of the PFS more clearly, the main characteristics of several parallel file systems are also given.

关键词 Parallel file system, Parallel processing.

1 引 言

随着处理机速度和网络传输速度越来越快，外部I/O设备的速度却相对落后了三、四个数量级，已经成为影响整个系统速度的瓶颈。另一方面，诸如多媒体、图像处理这些应用所需要的数据传输率越来越大，因而有必要支持高速的I/O子系统以弥补磁盘与处理机之间速度的差异，支持I/O密集的应用之高数据传输率。为了提高I/O速率，提供大的I/O带宽，在硬件结构上可以并行使用磁盘来解决。对于一个MIMD系统中使用多磁盘输入/输出子系统，一个要考虑的问题是文件的存储结构[7]。单个文件在多个磁盘上的分布有磁盘分条(disk striping)和文件分簇(file declustering)两种方式。前者是指文件数据以很小的粒度分布，每一数据块都分布在所有的磁盘上(如CM的Data Vault)。其好处在于系统软件不用作基础上的改动，对于用户而言好象仅仅是单个磁盘容量变大、速度变快而已。但不足之处是：①磁盘的总带宽通过一个通道到内存，限制了可使用的磁盘数目；②如果计算结点对磁盘进行并行访问，要从结点多路选通到此通道，既限制了输入输出服务，又影响了用户的可扩展性。文件分簇是指将文件的不同块分布在不同的磁盘上(如Bridge文件系统，Intel的CFS，及RAID的不同策略)，分簇的磁盘分布在多个文件存储结点上，可以对文件的不同部分独立并行处理、并行服务，如果在存储结点上有用于输入输出操作的缓冲区就不会有内在的瓶颈了。

另外一个要考虑的问题是I/O子系统的体系结构[9]。一种结构是磁盘直接与网络相连，当两个结点同时对文件数据进行访问时需要不同结点之间的同步，而且在磁盘上也没有空间对不同结点需要的数据进行缓冲，既不灵活效率又低。另外一种结构是每一结点都带有自己的磁盘，但因为其它结点可以向本地结点要求数据，本地应用的执行与I/O中断处理的矛盾会导致低效率，另外结点应用的软件运行故障可能导致整个系统的崩溃，具有低的可靠性。

现有的大部分并行I/O子系统都采用专门的带有磁盘的I/O结点，并以互连网络与其它I/O结点及计算结点相连，每个I/O结点负责对磁盘访问的管理和具体执行，存储并行文件的部分数据，处理块分配和缓冲。在I/O结点与不同的计算结点之间可以同时传送数据而不需要计算结点的同步，专门的I/O结点提供了很好的灵活性和高的I/O带宽。

2 并行文件系统的一般结构

在解决I/O瓶颈问题上有两种方式，一种是上述的硬件解决方法；另一种就是从软件上着手，基于文件分簇在多个文件存储结点上建立文件系统的并行文件系统解决方法。并行文件系统从功能上一般可分为以下几个部分：文件服务器，存储服务器和用

李群 博士研究生，研究方向并行操作系统。谢立 博导，教授，副校长，研究方向为分布式处理系统、并行处理系统。
孙钟秀 院士，研究方向为分布式处理系统、并行处理系统。

户。

2.1 文件服务器(file server)

文件服务器对用户的文件请求进行中间处理，将用户的文件请求转化为存储结点可以执行并且在含义上与用户的要求相一致的命令或操作。当用户发出对文件的请求以后，文件服务器判断请求的合法性和类型，若此请求是合法的，则文件服务器按照元数据将请求分成几个子请求，然后向存储结点传送指令。有的系统不存在文件服务器，文件服务器的功能分担在计算结点或存储结点上。文件服务器的作用是：

1)用来决定文件分簇的策略。当一个并行文件创立以后，文件服务器就要决定这个文件的元数据，并将元数据存储在元数据文件中。文件的元数据主要包括：系统文件的段名，(每一个存储服务器管理一个文件段)，段所在存储服务器的名及段的长度，以及其它类似于通常文件系统中文件的元数据。实际上元数据主要刻画并行文件的组织(比如逻辑记录对物理记录的映射)以及所期望的文件访问模式的信息。

2)为用户的文件打开和文件关闭请求提供服务。在初始时文件服务器就将并行文件的元数据缓冲(Cache)在内存中。对于打开，文件服务器通过对元数据的查找，发送消息通知所有包含此文件数据的存储服务器，指示读出什么物理位置的数据以及多长的数据。文件服务器也协调文件的关闭操作，检查文件的所有用户是否已完全关闭此文件，然后从存储服务器中收集被修改的元数据，对磁盘进行写入。

3)保证对文件的共享服务的原子性，也负责与其它文件服务器的交互，比如在对并行文件访问的共享模式下，实现单写/多读的协议，所有的读/写令牌都由文件服务器维护，每一读写的操作的执行必须首先要得到一个令牌，然后在操作完成以后释放令牌。

4)为了能更大限度地开发并行性，文件服务器能进行负载的平衡，使一个存储服务器不至于成为一个"热点"(hotspot)。

2.2 存储服务器(storage server)

一个存储服务器是对并行文件系统输入输出设备的抽象，它是读写请求的最终解决者，所有物理的输入输出都流过它。存储服务器一般由原有的一个本地文件系统对磁盘数据进行读写、提供对分簇文件的部分数据进行访问的功能，但它不对文件的全局结构进行解释，这样就将文件访问的机制与策略相互分离。存储服务器这种独立对数据的访问，增强了系统的并行性和可扩展性。每一个存储服务器都包括一个文件数据的缓冲，还有一个预取机制，可以依据多种预取算法从基础的文件系统中预先取得数据，当然存储服务器的基本功能还要支持服务器消息传递，以及有一个基础的存储数据的文件系统。一个并行文件的数据可以在存储结点上分为固定大小或可变大小的记录。

2.3 用户(Clients)

在SPMD计算中，一个用户包含有一个用户应用代码的实例、局部缓冲、预取，以及进行并行文件系统的记录工作的软件。在某些系统中，用户可以负担文件服务器的某些功能，用户代码与文件服务器通信，通过元数据找到能够满足一个给定文件的输入输出请求的存储服务器进行文件的访问。

3 设计并行文件系统的几个问题

3.1 缓冲(Caching)和预取[6,7]

并行文件系统在硬件上以并行输入输出子系统作为基础，但为了消除一些性能瓶颈，也需要软件的配合，缓冲就是为避免不必要的磁盘访问而设置的。缓冲从位置上可以有全局的、用户方的及服务器方的。文件的缓冲设计包括缓冲替换策略(在需要一个空闲缓冲块时决定被替换的块)以及写回策略(决定何时将新的数据写回到磁盘)。

缓冲替换策略：缓冲依赖于以下几种局部性：①时间局部性：刚用过的数据不久又会用到；②空间局部性：刚访问的块内或块附近的数据不久就会用到；③前二者的结合，顺序局部性：最近用到的块(MRU)不断被用到；④对于并行应用，还有进程间局部性：由一个进程使用的块常被另一进程使用。可以依照这些应用显示的局部性指导对数据的缓冲。

写回策略：缓冲利用延迟写回(delayed write-back)可以提高写性能，根据空间局部性可以将对同一文件块的写操作合并为一个磁盘写，对于进程间局部性也有好处。写回策略可以有如下几种：①WriteThru：每一文件写请求都强制磁盘写，这对于只访问一次的块是理想的；②WriteBack：磁盘写延迟到另一块数据需要这一块缓冲时；③Write Free：当这块缓冲可替换时发出磁盘写，即在此块缓冲需要被重新使用前，及它不再被某些处理机使用之后，发出磁盘写；④Write Full：当此块缓冲满时发出磁

盘写。

在程序的工作负载体现出高的局部性时，对近期使用的文件块缓冲可以加速块的访问，但在只读一次的块比较多的情况下，比如读大文件，文件缓冲就不那么有效了，而预取将要使用的文件块是减少文件访问时间的最好方法。最好的预取是开发应用本身的知识，即将关于要访问的文件块的信息以提示的形式传给文件系统，系统将通过分析结点对系统资源的竞争要求而预取数据，这种预取方式称作"通知式预取"(informed prefetch)。预取的一个不良效应是，大的预取可能把缓冲中将用到的数据替换掉，因而要处理好缓冲和预取的比例，使得既能有效地预取数据，又能使用已用过的数据。

3.2 并行文件系统的界面

Cormen 和 Kotz[13]从一些常用算法中归纳出 PFS 所需要的功能，并由此评价了已有的商用并行文件系统。这些功能包括：能否对数据的分簇或分条因素进行控制，能否对现在配置询问，能否独立访问磁盘块，能否关闭或打开缓冲功能，能否关闭或打开校验功能，能否优化或支持数据分布。不同的系统对以上的支持也不尽相同，但趋势是使用户能够利用其对应用知识的了解，控制并行文件系统的诸方面以期望得到良好的性能。下面我们从与并行处理更为密切相关的方面对界面进行考虑。

3.2.1 并行文件系统界面对 UNIX 的兼容和改变。因为 UNIX 能掩盖低层并行磁盘结构的细节，具有可移植性，而且又为广大户用所熟悉，所以并行文件系统的界面顺理成章地要与 UNIX 兼容。大多数 MPP 的商用系统提供传统的 UNIX 语义(文件是字节的线性序列)，透明地将数据分布在多个 I/O 结点上，如 Intel 在 iPSC 上的 CFS，Paragon 的 PFS，CM-5 的 sfs，Meiko 的 CS-2，IBM SP2 的 Piofs 及 KSR1 系统。但 UNIX 不支持并行访问的函数和语义，不是有效的并行 I/O 界面，因而有必要对 UNIX 进行修改，这种修改主要表现在对原 UNIX 调用的语义修改和对 UNIX 增加新的功能和调用上[11]。

1)传统文件系统的 open 要求系统认证用户身份并建立内部数据结构，在并行应用中多个结点同时调用 open 是比较浪费的。因此 Kotz 提出 multi-open 调用，为预定义好的进程组打开文件，组内的所有进程有一文件句柄以访问共享文件，Bridge 文件系统提供并行 open 调用。另外，在 Vesta 中文件访问分为 attach(相当于通常文件系统的 open)和 open(用于定义逻辑划分，指出要访问的逻辑划分)。

2)对 read，write 的改变。在 Bridge 系统中当对一共享文件进行 read 操作时，读出的数据将传递到所有共享此文件的进程。在 Parasoft 的 Express 的 Cubix 模型中，不仅读出的数据传送到各结点，而且在一个输出操作时将合作的所有进程所修改的数据项写回。

3)文件创建。在 Vesta 和 CFS 中提供文件创建原语，可以在文件建立时预先分配好磁盘空间，这样在写操作时就不必进行多次磁盘空间分配的请求。

3.2.2 文件的访问模式。程序员经常通过知道逻辑数据的放置和应用的访问特点来优化物理数据在磁盘的放置，产生高效的 I/O 代码，提供多种文件访问模式就是为此设计的。Vesta 将文件的访问模式分为六种，比如在结点对不相交的划分访问或者独立异步访问时可以属于不维护一致性的访问模式，另外 Vesta 也提供共享偏移访问和联合访问模式，在 CFS 及 Express 中也有相似的访问模式。

3.2.3 独立的并行 I/O。在界面中若能提供给结点独立的 I/O 操作并行访问磁盘而不需要结点间的同步，将会大大提高效率。为了支持并行 I/O，避免顺序化操作，一种方法是让每一结点访问它们需要的文件部分，而不与其它结点互相干扰。nCUBE，Meiko 中使用这种方法，这些系统以数据结构的划分方式对文件进行划分，于是可以对数据并行读写。但这种方法使文件数据分为很多小的部分，导致了许多小的读写。第二种方法是界面提供对文件数据放置的直接控制，使数据放置与应用特点相一致，不同的结点可以独立地访问自己划分中的数据。下面所讲的数据的分解就是此类，Vesta 是其中的一个典型例子。

3.2.4 其它。并行应用的编程比较困难，如果使用异步并行 I/O 函数就更为复杂，因而有必要在高层语言库和新的高层语言上提供支持高层 I/O 的文件系统界面。一种可能是提供语言级持久数据类型，比如 HPF 能容纳持久矩阵。另外一种可能是 C++ 库提供持久并行对象和对对象的操作，这种类库可以使程序员避开复杂的低层实现，不必重复类似的并行界面的编程。可扩展文件系统(ELFS)允许定义并行文件对象，包括具体类的访问方法、缓冲、预取，不仅方便编程，而且可依据具体应用灵活定义，也具有可移植性。

3.3 数据的分解与映射

在组织并行文件系统时，一个要考虑的重要问题是数据的分解，即数据在各计算结点上的分配，这

是为了适合并行应用背景以及开发多个机器的并行性而划分问题域所引起的，比如 Vesta 划分方式就是如此。Vesta 将文件看作是二维结构，用户可以以不同的逻辑视图打开文件，形成不同的逻辑划分，而在存储结点上并不需要移动文件数据。比如对于矩阵可以以行、列、矩阵块的方式打开并分布在不同的计算结点上，每个结点独立地访问自己的划分而不用知道其它的划分。

在并行文件系统中，可以分为两种映射关系[9]，一种是数据集合(逻辑数据)与磁盘数据分布的映射关系 M1，另一种是数据集合与数据在各处理机上的映射关系 M2。因此，在数据分布的实现中，系统要依据 M1，M2将处理机中的数据与磁盘上的数据对应起来，从磁盘依据 $M1^{-1} * M2$将数据取到处理机中，或依据 $M2^{-1} * M1$将数据从处理机写到磁盘上去，对于这些映射关系的输入输出操作对应着的是不同的数据传输策略。在对文件数据访问时，依照映射关系可以将数据的源和目的地址计算出来，然后进行传输。CFS 系统中只提供数据集合映射到磁盘数据所对应的传输策略，而将数据在处理机上的分配留给用户实现。通常的并行文件系统不支持数据分解和对数据在磁盘上分布粒度的控制，这些系统往往对不同的访问模式有极大的敏感性，不同的数据分解在性能上有极大的不同；而且如果应用的数据分解与文件的分布粒度大小不匹配，那么结点负载的不平衡会导致瓶颈现象，因而支持数据分解以及对磁盘上文件分布粒度的控制是很有必要的。

在并行文件系统的设计中，将上面所讲的两种映射化归为一种数据传输策略，可能会导致很差的 I/O 性能。在运行库的设计中，复合映射策略较好地改进了并行文件系统对不同访问模式敏感性大的问题，它将一个并行 I/O 任务分解为两个阶段[12]，第一阶段依照于 M1的映射模式以符合数据在磁盘上放置的方式对数据进行访问；在运行时，以第一阶段的结果为基础依据 M2的映射模式以适合应用的方式将数据分布在各处理机结点上。这种方法减少了对磁盘的竞争，减少了输入输出的数量，避免了许多 I/O 配置引起的负载不平衡的开销，较充分地应用了处理机之间的互连网络的高带宽。这种将用户选择的数据分布映射与文件数据在磁盘上的映射相分离，使性能较少地依赖于用户的选择。

3.4 容错与可靠性

对于大的并行系统而言，其故障率也相应地增长，可靠性就成为一个要考虑的因素。在并行 I/O 子系统的设计中，RAID 设计的不同策略为可靠性提供了很好的模式，建立在这种 RAID 元件上的系统，如 Vesta，在单个结点上独立地实现容错。另外 Vesta 也提供由用户控制的对文件有效而一致的检查点(checkpoint)。在应用中，可以回退到已被记录下检查点的早先的状态。其实现中，以元数据的形式而非以文件数据的形式维护检查点。类似地，有的系统采纳数据库系统中容错的解决方式，比如在 PIOUS 中，基于事务、日志的方式对文件进行操作，在故障时可以从原来的保存点继续执行。

4 几种并行文件系统实例

PIOUS[8]是第一个为可扩展性能使用数据分簇的网络并行文件系统，它改进了 PVM 对并行 I/O 系统不支持之不足，PIOUS 的 I/O 操作在事务的上下文中执行，对用户透明地提供访问的顺序性和系统的容错。

PPFS[5](可移植并行文件系统)提供丰富的 I/O 操作集，能使应用控制或定制策略以匹配应用的要求。应用可以：①声明每一用户对文件的访问模式(随机、顺序、隔步长访问等)。②控制多个服务器上文件数据的放置。③控制服务器、用户的缓冲策略。④控制服务器、用户的预取策略。⑤用户自定义的数据一致性。

Vesta[1]并行文件系统是专门为并行计算服务的，提供了一系列有利于并行计算特点的功能。Vesta 在界面中可以让用户自己定义文件的并行视图，允许用户定义文件的划分和再划分。Vesta 用户界面在两个层次上定义文件的并行结构：数据的物理分布和数据的逻辑视图，物理分布决定使用多少存储结点以及划分的最小数据单位，对于一个特定的物理放置，应用可以以多种不同的逻辑视图打开文件，产生不相交的逻辑划分。Vesta 也支持高层并发以及并行 I/O 库。Vesta 为 SPMD 模式的计算提供一个联合 I/O 操作库，提供对并行文件的协调访问。在所有进程同时发出相同的库调用的时候，可以将库调用看作进程之间需要同步的操作，也可以使每一进程独立执行操作，减少同步开销。使用不同的库调用能在保持并行语义的前提下，可以将并发的读写操作进行分解，这当然是以 attach(取得文件元数据的操作)和 open(文件打开)的同步作为代价的。Vesta 定义了文件访问的六种模式，每一种模式都依照进程如何共享文件以及其间的访问如何交叉而有其特点。

Pasta 并行文件系统是为了在通常的分布系统中支持高数据传输率的并行输入输出系统。Pasta 采用的是基于 cluster 的体系结构，Pasta 中的 cluster 是工作站组和在局域网络上的一个或多个文件服务器，另外在每个 cluster 中还有多个存储服务器。

PASSION[12](Parallel and Scalable Software for Input-Output)是一个编译器和运行支持系统，使用并行输入输出将外围(out-of-core)的 HPF 程序转化为结点间消息传递程序。在系统中使用了二阶段的策略。访问数据时，一种方法是直接文件访问，任一个处理机都可以访问任一磁盘；另一种方法是显式的通信，每一处理机仅访问自己的本地数组文件，数据先读入到内存中再送到其它处理机。第一种方法可能导致许多处理机同时访问一个磁盘而引起竞争，而且即使访问一小部分数据，对于每一处理机访问的整个块都要被传送。第二种方法可以避免这些缺点，但是在 I/O 之外还需要一个通信的阶段。在后来的 VIP-FS[3] 中还引入了假定请求(assumed request)，只允许指定的 client 向 server 发出请求，每一 server 根据数据的划分信息将数据发送给 clients，通过大幅度减少对每一 I/O 设备的请求数目提高了读性能。

对于一些商用的并行文件系统，如 CM-5 的 spf (Scalable Parallel File System)，Intel 为 iPSC/2 和 iPSC/860 而设的 CFS(Concurrent File System)，以及 Intel Paragon 的并行文件系统，这些都提供文件数据的分簇，并提供少量并行文件访问模式，但不能给应用程序提供足够的控制。分布式文件系统，如 Zebra[4] 和 Swift 利用 RAID-4/5 容错机制也在文件服务器上将数据分簇，但不对应用层提供分布控制或策略。

5 结束语

在并行文件系统中，经常提到的问题是访问模式的多样性以及 I/O 系统对这些访问模式的敏感性。无论从并发文件系统的特点上，还是从并行文件系统对并行的支持上，以及并行应用的更进一步的要求上都可以看出，访问模式的多样性是很有必要的，但现在的系统对于多种模式的资源管理方法仍不能满足。从更细的方面看来，正如我们在上面所讲的，在文件系统层次上，在运行系统(runtime system)上，都有许多问题要靠大量实验解决，包括：文件系统内的可靠性，文件服务器的可靠性，不同体系结构下的数据分簇策略以及数据在不同结点的分布，传输策略，缓冲和预取策略以及整个系统的灵活性和可扩充性。

参考文献

[1]Peter F. Corbett. Parallel Access to Files in the Vesta File System, Supercom' 93

[2]James C. French et al., Performance Measurement of the Concurrent File System of the Intel iPSC/2 Hypercube, J. of Parallel and Distributed Computing 17(1-2)

[3]Michael Harry, et al., The Design of VIP-FS: A Virtual, Parallel File System for High Performance Parallel and Distributed Computing, ACM Operating System Review

[4]John H. Hartman et al., The Zebra Striped Network File System, SIGOPS' 93

[5]J. V. Huber Jr., PPFS: A High Performance Portable PFS, Supercom' 95

[6]Ramakrishna Karedla et al., Caching Strategies to Improve Disk System Performance, IEEE Computer, March 1994

[7]David Kotz et al., Caching and Writeback Policies in Parallel File System, same to[2]

[8]Steven A. Moyer, PIOUS: A Scalable Parallel I/O System for Distributed Computing Environments, 1994 Scalable High Performance Computing Conf.

[9]J. del Rosario and A. Choudhary, High Performance I/O for Parallel Computers: Problems and Prospects, same to[6]

[10]Rajeev Thakur et al., Compilation of Out-of-core Data Parallel Programs for Distributed Memory Machines, Architecture News, 1995

[11]Anand R. Tripathi et al., Trends in Multiprocessor and Distributed Operating Systems Designs, The Journal of Supercomputing, 9, 1995

[12]J. M. del Rosario et al., Improved Parallel I/O via a Two-phase Run-time Access Strategy, Computer Architecture News 21(5), 1993

[13]T. H. Cormen and D. Kotz, Integrating Theory and Practice in Parallel File Systems, DAGS 1993 Symposium on Parallel I/O and Databases