计算机科学2000Vol. 27№ 4

并行文件系统PARFSNOW++中的协作式缓冲技术研究*)

The Cooperative Cache Techniques Used in PARFSNOW++

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 The Parallel File System based on NOW is focus more and more. This paper briefly describes a Parallel File System based on NOW—PARFSNOW++. We put special focus on the technique of Cooperative Cache. Aiming at the existed problem of current Cooperative Cache mechanism, we introduce a new mixed Cooperative Cache mechanism used in the PARFSNOW++, we also present the cache coherence mechanism and the data replace mechanism used in the mixed Cooperative Cache mechanism.

Keywords NOW, PFS, Cooperative Cache, MCS

1 引言

近年来,国内外对并行文件系统做了很多的研究[1],提出了许多有特色的并行文件系统,如xFS[2],ParfiSys[3],Galley[4],Scotch[5],Zebra[6],许多系统采用了协作式高速缓冲机制。使用这种机制,所有节点都提供自己的部分内存供共享,并通过协同机制统一调度这些分布式的共享内存,为并行文件系统提供一个虚拟的高速全局共享缓冲区。这种协作式缓冲扩大了数据缓冲区的容量,能有效提高文件数据在缓存中的访问命中率,减少磁盘的访问次数,以便提高系统性能。但是,协作式高速缓冲在带来性能提升的同时也带来了一些非常难以解决的问题:如何维持共享数据的一致性。通常情况下,使用协作式高速缓冲机制的文件系统往往需要实现一套非常复杂的数据一致性维护机制,但该机制的运行开销十分高昂,无形中又降低了系统运行的效率。

针对这些问题,我们提出了一种有权限控制的复合式协作式高速缓冲管理机制,并在这种新的协作式高速缓冲管理机制的基础上设计了一种简单而高效的主动式内存数据一致性方法。这种方法能有效减少内存数据一致性的开销,提高了系统效率。现在,这种复合协作式高速缓冲管理机制已应用在我们设计并实现的基于NOW[2]的并行文件系统PARFSNOW++中。通过实验,我们证明了文中提出的协作式高速缓冲机制的有效性。

2 PARFSNOW++模型简述

PARFSNOW++是一种基于NOW的并行文件系统,系统中每个节点都通过高速网络互相连接,并运行各自的基于微内核的操作系统,并行文件系统则构建于这些操作系统平台之上,作为服务程序向用户提供并行文件服务。

2.1 PARFSNOW++结构

如图1,PARFSNOW++包括文件服务器、I/O服务器、计算节点(即用户程序)三部分,这种并行文件系统通常将文件分片存放,便于并行读写。I/O服务器(以下简称I/O节点)就是负责存放、管理各并行文件分片的。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。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

*)本项目受到国家"863"基金的支持。赵 欣 博士生,研究方向:分布式计算环境。陈道蓄 教授、系主任。谢 立 教授、博导、副校长。

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1中虚线就是描述的用户节点直接访问I/O节点的操作。

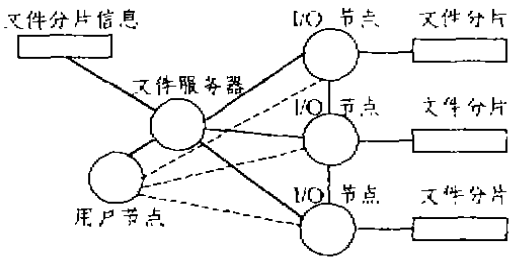

图1

图1中所有节点都将自己的部分内存共享出来，通过协作式高速缓冲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。既然协作式高速缓冲机制是用来缓存文件数据的，那么系统采用的协作式高速缓冲机制是否先进就直接关系到系统的运行效率，所以，我们在设计并实现 PARFSNOW^{++} 的时候对系统应该采用什么样的协作式高速缓冲机制进行了深入的研究。

3 协作式高速缓冲简介

如图2，协作式高速缓冲是一种数据缓冲机制，能协调并行环境中各个节点上数据缓冲的内容，使它们能互相协作，将多台工作站上的局部内存通过网络互联成为一个虚拟的全局共享地址空间，提供给系统使用者一个平坦的共享地址空间，以掩盖使用网络上其他主机上内存的底层实现细节，用户能象访问本地内存一样访问这种分布式共享内存，这样，就给用户提供了一个方便、高效的全局数据缓存。将协作式高速缓冲机制引入并行文件系统的设计中，能通过协作式高速缓冲机制将用网络连接的工作站上的空闲内存联成一个虚拟的全局内存空间，用来对 I/O 数据进行缓存，提高高速缓冲的命中率，减少对磁盘的读写操作，从而有效提高文件 I/O 的效率。实际上，这种数据共享还是通过网络通信实现的，但它提供了一种抽象，掩盖了消息传递方式的不方便性，方便了节点对复杂数据结构的访问。在 PARFSNOW^{++} 中，每个节点提供一部分它自己的本机内存作为共享的全局高速缓冲。由于这块内存已经提交作为全局共享内存，所以本地节点也不能任意修改这块内存。

4 复合的协作式高速缓冲管理机制

4.1 现有协作式高速缓冲管理方式存在的问题

过去提出的许多协作式高速缓冲系统往往采用两类高速缓冲管理模式：基于多备份的分布式共享内存管理模式和基于单备份的分布式共享内存管理模式，前一种模式允许对同一个数据块在全局内存中存在多个数据备份，即系统内同一个数据块可能同时存在主、副备份，用户节点访问这个数据块时就能在本地数据备份中读取数据，这样就能有效减少用户到远地缓存中读取数据的次数，减少网络数据传输的开销，从而提高数据访问效率，但是，由于同一个文件分片数据在全局共享内存中可能有多个备份，而各个备份都能被用户修改，这就造成了各个数据备份可能存在的数据不一致性。因此，系统不得不实现一套复杂的数据一致性机制来保证系统数据的一致性。实验结果和实践经验都证明，为维护主、副备份的数据一致性，系统必须经常进行主、副备份数据同步，占用了大量网络带宽和系统资源，使得这种数据一致性维护的代价非常高昂，直接影响了整个系统的运行效率。另一种基于单备份的分布式共享内存管理模式，基于减少数据一致性维护代价的考虑，采用了全局数据单备份机制，即在全局共享内存中仅仅维护一个数据备份，没有主、副备份之分。当某用户节点访问一个全局共享数据块时，系统会从数据块实际所在的节点内存中读出数据并传送给用户，但是，若所有数据访问都要通过网络传输，势必给网络带来沉重的负担，同时也会影响数据访问效率。为了提高 I/O 效率，系统常常把用户要读取的数据块从全局虚拟地址空间中（可能实际上是远地内存）迁移到用户节点上来，供用户在本地访问。然而，在多用户状态下，当几个用户轮流对一块数据进行 I/O 操作的时候，由于系统只有一个该数据块的缓冲备份，系统就会根据用户需求将缓存数据在各个节点上频繁迁移，占据了大量带宽，降低了系统效率。由于存在上述问题，所以设计一个新的协作式高速缓冲管理机制就是非常重要的了。

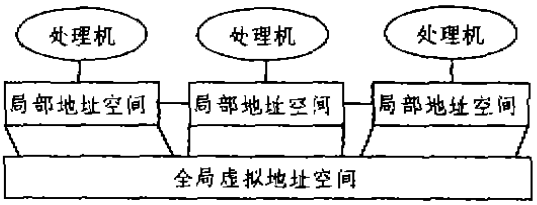

图2

4.2 术语定义

在引入我们设计的协作式高速缓冲管理机制前，首先引入一些该机制中用到的术语。

预输入：即数据预取。PARFSNOW^{++}采用的是Look Forward预输入技术。

延迟写：在并行文件系统中，当用户提交一组对文件数据的修改操作的时候，系统不立即将更新数据刷新到磁盘上去，而是将用户对文件数据的修改保存在缓存中，等到合适的时候再刷新到磁盘上去。这种推迟刷新的操作就称为"延迟写"，又称为"缓输出"。

"脏"缓冲数据：由于系统采用了延迟写操作机制，当用户修改文件数据后，文件系统就会在该文件数据对应的数据缓冲中进行修改，但是，在这些修改尚未被刷新到磁盘上之前，缓冲数据是同对应文件数据不一致的。我们称这时的缓冲数据是"脏"的。

4.3 有权限控制的复合协作式高速缓冲管理模式

通过对现有协作式高速缓冲机制的分析，我们可以看出，它们存在着一些问题。为此，我们引入了有权限控制的COMA[9]机制和本地高速缓存相结合的复合协作式高速缓冲管理机制（以下简称"复合机制"）。在复合模式下，所有节点都将本机上的部分内存提供出来，作为全局共享，这些通过网络互联的共享内存构成一个大的虚拟共享内存。其具体结构如图3所示。系统在全局虚拟地址空间内为共享数据块保留且只保留一份权威数据备份。同时，为了提高数据读取效率，减少网络通信，系统在用户本地还维护了一个本地高速缓冲，这个缓冲同系统维护的权威缓冲数据备份相映射，使得用户能尽可能地在本地读取数据，而不必到存放在远地主机内存中的缓冲数据备份中读取数据，从而有效地提高了数据访问效率，因此整个系统实现了层次型的缓冲机制：当用户要读取某块数据的时候，首先在本地缓冲中查找数据，若本地缓冲有该数据并确实同权威的缓冲数据备份相一致，那么用户就在本地读取数据。当本地缓冲没有该数据或本地缓冲数据同权威的缓冲数据备份已经不一致（由于用户对权威缓冲数据备份进行修改）的时候，系统就自动到存放在全局共享内存中的权威缓冲数据备份中取得当前最新数据来更新本地高速缓冲，并提交给用户。但是，这种本地高速数据缓冲和权威数据备份相结合的复合协作式高速缓冲机制并不同于基于多备份的分布式共享内存管理模式。本地高速数据缓冲和权威数据备份不象主、副本那样在系统中享有等同的数据访问权限。复合机制在本地高速数据缓冲和权威数据备份上引入了权限控制机制。只有权威数据备份是具有可修改权限的，而本地高速数据缓冲则是只读权限。权威数据备份确保的是相应的文件数据块内的最新数据，它允许用户在上面进行修改操作，并定期自动将更新数据刷新到文件中去。用户若要修改文件数据，就必须到系统维护的缓冲数据备份中进行修改。系统并不向维护相关节点通告修改操作的发生，这样能有效减少数据一致性通告的开销，系统也不需要维护各用户节点本地的高速缓冲的状态，从而能降低系统实现的复杂度。但是，权威数据备份同本地高速缓冲在一定时间内还是会出现数据不一致的状态。为此，我们设计了主动式内存同步策略来使本地高速数据缓冲同权威数据备份保持一致，并访问到最新的数据。由于系统内只有一个权威数据备份，数据同步大为简化，配合上主动式内存同步策略，可以明显减少数据一致性维护开销。

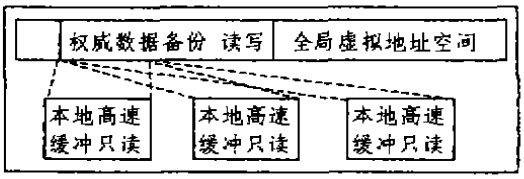

图3

5 复合协作式高速缓冲机制中的内存主动同步机制

采用了复合内存组织方式后，用户既可以从权威缓冲数据备份中读取数据，又可以从本地高速数据缓冲中读取数据，显然，从本地高速缓冲中读取数据比从远地的权威数据备份中读取数据开销要小。所以，用户将尽可能从本地高速缓冲中读取数据。但是，由于系统不会对权威数据备份的修改进行通告，所以可能会出现权威数据备份同本地高速缓冲数据不一致的状态。因此，能否保证用户读取数据的权威性和高效性就成为系统实现成功与否的关键。所谓保证数据的权威性，是指当权威的缓冲数据备份为用户修改后，用户本地维持的高速数据缓冲要知道权威数据已经发生修改，并刷新本地高速缓冲以便同权威数据备份保持一致。也就是如何保持权威数据备份和本地高速数据缓冲的数据一致性的问题。要解决这个问题，就必须实现一个高效的内存一致性规范MCS。

通过对实际应用的分析，我们发现，许多时候用户虽因以前的操作保留了某块数据的本地高速缓冲，但是并不一定会立即继续使用，甚至不会再用。若在用户使用该高速缓存的数据前，有另外的用户对权威数据备份进行了多次修改，那么为了维护数据一致性，本地高速缓冲就必须一次次更新。这实际上是没有必要的，

因为本地高速缓冲只需在用户需要使用前更新一次就可以了，基于这种认识，我们设计了"主动一致性方法"。这种控制机制由两个部分组成：对权威数据备份的控制和对本地高速数据缓冲的数据一致性控制。

5.1 权威数据备份的一致性维护

当多个用户对权威数据备份进行修改时，必须有数据同步策略控制来保证用户程序能正确运行，否则，就可能出现同时间有关的错误。例如：用户 A 在时间1在文件中写入一个整数 I，并希望能在以后读出这个整数以供使用。但是，在时间2的时候，用户 B 将文件中的这个数据修改为 I'。这样，当用户 A 在时间3上来读取文件中的这个数据时，就无法读取原来他写入的数据了，这就会造成程序运行的错误，所以我们必须对权威数据备份的访问加以同步控制。为此，系统引入了一种类似于 Sprite[7] 和 AFS[8] 的"修改锁"机制。即用户对权威数据备份进行访问的时候，若用户是读权威数据备份，则没有任何限制。若用户是修改该权威数据备份，则必须先向权威数据备份的管理节点发送修改锁请求消息，管理节点收到该请求消息后就检查当前该权威数据备份相关的修改锁是否已经被别的用户申请去了，若某权威数据备份已经被修改锁锁定，申请该修改锁的用户将处于等待状态，等修改锁持有者释放修改锁后，系统将按申请顺序通知申请修改锁的用户。当申请到了修改锁后，用户就可以对该权威数据备份进行修改了，直到他释放该修改锁前他都不必再申请修改锁而自动拥有修改权限。尽管这种修改锁机制限制了对权威共享数据备份的修改，但是其他用户仍然可以任意读取权威数据备份。在我们目前的实现中，我们没有对修改锁的作用范围进行管理，而是认为若申请了修改锁，则用户就将整个权威数据备份块的修改权限都锁定了。当然，若在修改锁的定义中添加锁定的范围，则有利于提高系统修改的并行度。但是，这将增加系统维护的开销和系统实现的复杂度。所以，我们目前使用的是无锁定范围定义的修改锁，使用这种同步机制，能有效避免一些同时间相关的错误。

5.2 本地高速数据缓冲和权威数据备份的数据同步

本地高速数据缓冲和权威数据备份的同步由"用户主动一致性方法"来保证，这种方法有两个原则。第一，权威数据备份和本地高速缓冲分别同时间戳相联系的原则。时间戳是一个逻辑非负数，用来描述两个动作之间的先后关系。在"用户主动一致性方法"中有三种时间戳：系统时间戳、权威数据备份时间戳 Ta、本地高速缓冲时间戳 T1。系统时间戳是 I/O 节点在初始化的时候创建的。每次对权威数据备份进行更新后该系统时间戳就会增大1。权威数据备份时间戳 Ta 等于权威数据备份被更新时的系统时间戳。本地高速缓冲时间戳 T1 则等于它从权威数据备份中获得数据时的权威数据备份时间戳 Ta。由于每次对权威数据备份进行修改都会增大权威数据备份时间戳，所以通过比较本地高速缓冲的时间戳 T1 和权威数据备份的时间戳 Ta，就能判定权威数据备份是否已经被修改过。若 Ta=T1，则表明当前高速数据缓冲内的数据就是对应文件数据块的最新数据。此时可以直接使用本地数据。若 Ta>T1，则表明由于用户修改，该高速缓冲对应的文件数据块已经被刷新过，而本地高速缓冲却还未刷新，则本地高速缓冲中的数据是失效的数据。此时系统会根据权威数据备份向用户本地传送最新数据，并刷新本地高速缓冲。第二，"使用时主动同步"的原则，即只有当用户要使用本地高速缓冲的某数据块的时候，才主动向权威数据备份发出数据同步请求。系统在请求中赋上本地高速缓冲的时间戳，这种同步报文非常小，所以系统开销不大。当权威数据备份收到该同步请求时，通过比较请求中赋的时间戳和自身时间戳，就能确定用户本地缓冲中的数据是否是最新的。若是，则发送"无需更新"的回答。若不是，则直接发送更新后的数据。这样，在权威数据备份上进行"写"操作后，系统允许出现临时性的数据不一致。在某个进程要得到当前某些共享数据的最新版本时，由它主动地询问当前数据状况并进行同步操作，而文件缓冲数据的更新无需向每个相关进程发送更新通告，能省去许多通告开销。

这种主动式方法有效解决了前面我们提及的那些问题。它能支持临时性的数据不一致状态，减少了系统通告操作的麻烦，提高了系统操作的效率，最重要的是它省去了许多不必要的一致性维护的开销。只有那些要用到这块数据的进程才会提出同步操作请求，并获得最新通告。也只有那些通过同步操作请求的数据才参与数据一致性维护操作，从而进一步减少了数据的不必要的共享操作。

6 缓冲数据替换

通常情况下，权威数据缓冲备份保存在该备份所对应的文件所在的 I/O 节点上，然而，当 I/O 节点被大量访问的时候，仍然可能出现本地内存不足以保存所有的权威数据缓冲备份的问题。此时，就必须要考虑将缓冲内存中的某些页面替换出去。过去常用的技术是将缓冲数据替换到速度较低的磁盘上，所以替换操作往往效率不高。利用协同式缓冲提供的服务，权威数据缓冲备份可以在全局内存空间中自由迁移。这意味着缓存的数据在内存中的位置可以是动态调整的。这种机制非常有利于并行计算的工作模式。它能在某个节点内存不够时，将某些缓冲数据迁移到其他节点上

去。我们注意到，在高速网络环境下，当本地内存不够缓冲所有需要缓冲的数据的时候，把数据缓存在其他空闲节点比将之替换到磁盘中更能有效提高性能，特别是对于某些节点使用数据过多的情况下，配合有效的替换算法，其他节点实际上就变成了此节点的文件服务器，能将对单个节点的 I/O 请求分流。当然，这种角色可以根据应用状态的改变而变化。

要将数据缓冲备份替换到它机上，就涉及两个问题：第一、应该将哪个页面替换到其他节点上去；第二，一个权威数据缓冲备份不应该在没有用的时候仍然保存在内存中，占用内存空间，那么，如何判断将一个权威数据缓冲备份抛弃的合适时间。对于第一个问题，我们采用的是一种改进的 LRU 算法。该算法在节点内部维护了一个 LRU 先进先出队列，当节点创建了某文件数据的权威数据备份后，就将该数据备份的索引加到队列尾部。当某权威数据备份被访问后，系统就从该队列中将这个权威数据备份块对应的索引从原来位置抽出来添加到队列尾部。这样，就可以保证队列首部是当前最少访问的权威数据备份的索引。通常的 LRU 算法是将当前最少用的缓冲数据块替换出去，但是，由于 PARFSNOW^{++} 采用了"延迟写"机制，所以 LRU 队列的首部可能是一块已经被修改但是尚未刷新到磁盘上的"脏"的缓冲数据块。显然，在本地将"脏"数据块刷新到磁盘上效率要高于从远地内存中将数据刷新到磁盘上，所以，我们尽可能不将这种"脏"缓冲数据块替换到其他节点上。根据上面的原则，当需要进行缓冲数据替换的时候，系统就从该 LRU 队列首部开始向尾部查找，寻找第一个"干净"的权威数据备份块，将它迁移到一个具有空闲内存资源的节点上，并修改本机上对该文件数据块所处位置的索引，使它指向接受该数据备份块的节点。而接受这个权威数据备份块的节点则将迁移来的权威数据备份块添加到它自身的 LRU 队列尾部。当然，一个权威数据缓冲备份不应该在没有用的时候仍然在协同式缓冲系统中来回迁移，占用内存空间，所以，有必要决定一个缓冲数据块何时应被抛弃。在我们的协同式缓冲系统中，每个权威数据缓冲备份块都有一个 TTL(Time To Live)域，每次将权威数据缓冲备份替换到其他节点都要使它的 TTL 域加1，当 TTL 达到一定大小 N 后，系统就不会再替换该数据块，而只是简单抛弃这块权威数据备份块。这种策略如同 TCP/IP 协议中对数据报文设定 TTL 域的策略一样，能有效避免无用的数据块在系统中无限制地传送，PARFSNOW^{++} 中设定 N=2。

结论 本文介绍了基于 NOW 的并行文件系统 PARFSNOW^{++} 采用的协同式缓冲技术，为了测试其各项技术的性能，我们建立了一个试验环境，该环境由多台 IMB RS6000工作站组成，每台工作站的内存为 64M，硬盘2G，其上运行的操作系统是 AIX4.2 1。

1)对传输大小为1200kB、1440kB、1920kB 时写操作的数据传输率(kB/S)的测试结果显示，采用了协同式缓冲策略的并行文件系统 PARFSNOW^{++} 确实使系统文件读写操作的效率有所提高。

2)对传输大小为3840kB 时读操作的数据传输率(kB/S)的测试是描述的两个用户的随机共享读写，其中读写的记录及次数均随机产生，1/3为写操作，2/3为读操作，且两个用户各读写100次的时间(s)。测试结果显示，随高速缓冲的增大，无协作式缓冲机制的共享访问时间逐渐减少，这是因为 I/O 节点上缓存了一些共享数据，但总是比用协作式缓冲机制时耗时要多。在协作式缓冲机制下则变化不大。这显示出系统采用的协作式缓冲能有效缓冲数据，提高系统性能。

在以后的工作中，我们将着重研究文件系统的高效率预取机制，使文件系统能智能地分析用户对文件数据的 I/O 操作状况，并准确地指导数据预取。现在，我们正在使用单奇偶块技术实现单机系统容错(系统能容忍单个 I/O 服务器出错)，下一步工作将研究如何使用多奇偶块技术实现多机系统容错。

参考文献

1 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

2 Anderson T E，et al. A Case for NOW (Networks of Workstations) IEEE Micro，1995，(2)：54～64

3 Carretero J，et al. ParFiSys. A Parallel File System for MPP. ACM SIGOPS，1996，30(2)：74～80

4 Kotz D. Nieuwejaar N. Flexibility and Performance of Parallel File Systems ACM *Operating Systems Review*，1996，32(2)：63～73

5 Gibson G A，et al. The Scotch Parallel Storage Systems. *IEEE Compcon*，1995

6 Hartman J H，Ousterhout J K. The Zebra Striped Network File System. Transactions on Computer System，1995，13(3)：274～310

7 Nelson M，et al. Caching in the Sprite Network File System. ACM Trans. on Computer Systems，1988，6(1)

8 Howard J，et al. Scale and Performance in a Distributed File System. ACM Trans. on Computers，1988，6(1)

9 Hagersten E，et al. DDM-A Cache-Only Memory Architecture. IEEE Computer，1992(9)：44～45