万方数据资源系统 WANFANG DATA (CHINAINFO)
计算机应用研究
APPLICATION RESEARCH OF COMPUTERS
2000 Vol.17 No.2 P.16-19
数字化期刊 DIGITIZED PERIODICAL

存储一致性模型研究

胡曙 廖湘科 罗军 唐晓东 张屹

摘 要 讨论了分布式共享存储系统(DSM)的通用结构及页面管理机制，给出了各种存储一致性模型的正确性条件，并以互斥问题为例，说明了在分布式环境下编程的一些特点。最后，基于作者本人的工作，还讨论了如何提高DSM系统性能的问题。
关键词 分布式共享存储 一致性模型 延迟

1 研究背景

目前，随着网络技术的急剧发展及日益普及，计算机系统正朝着分布化、并行化的方向发展。如何让多台计算机协同工作来完成一个大型任务，提高计算机系统(非单机)的整体处理能力，一直是国内外研究的重点。为达到这个目标，存储一致性的研究是基础，它是保证分布式系统正确运行及提高效率的关键。

2 系统结构

DSM系统具备一个客户/服务器的结构，而且包含了内核及用户级的模块，如图1所示。其中，客户进程是指使用DSM系统的用户级应用程序，当然，在单个节点上可运行多个客户进程，为了支持客户进程访问DSM系统的操作，提供一个单一的用户态下的DSM服务器。

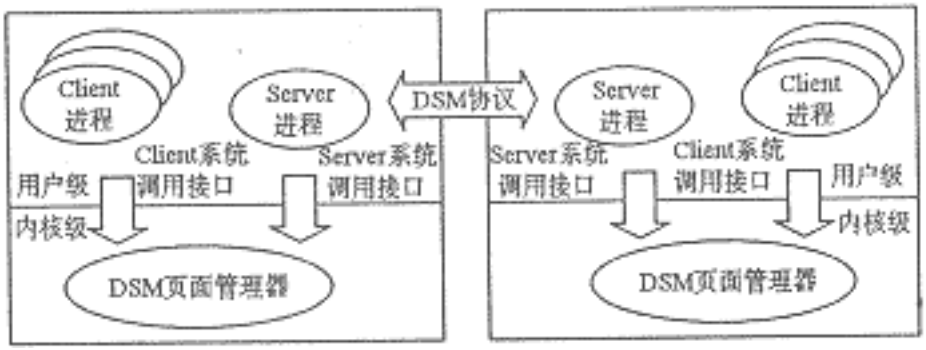

图1 DSM系统的通用结构示意图

DSM系统的页面管理结构如图2所示，它描述了一个页面从网络上取回并映射到用户或内核的地址空间及可交换到本地Swap区的全过程。

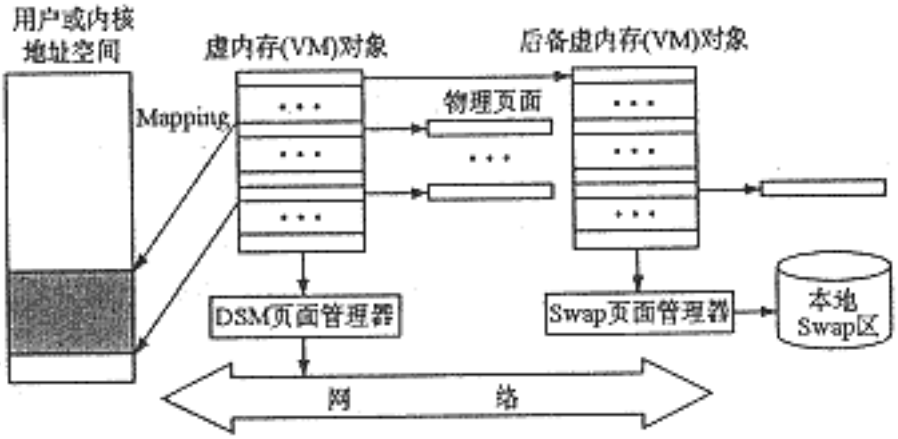

图2 DSM系统的页面管理结构

3 一致性模型

所谓存储一致性模型(Memory Consistency Model)，实际上是系统设计者与应用程序员之间的一种约定。如果应用软件遵从一定的规则访问虚内存系统，则应用软件可获得正确的存储访问结果；反之，若破坏约定的规则，则存储访问的正确性不受保证。

从某种意义上来讲，存储一致性模型对共享存储系统中多处理机的访存次序作了限制，从而对性能有影响。分布式共享存储系统(DSM)的一个根本目标就是让一个通过局域网连接起来的工作站集群，共享单一的分页虚地址空间，使在工作站集群之上运行程序的效果类似于程序在单机之上的运行。在最简单的变体中，每页存在于一个确定的机器中，对本地页面的查询速度等同于对内存的访问速度，而访问远程机器的页面查询请求将引发段违例信号(SIGSEGV)，这将导致程序陷入到操作系统之中，由操作系统来处理缺页中断，操作系统随之发送一个消息至远程机以找到所需的页面并等待远程机将该页面回送过来。本地机器获得远程页面之后，引发段违例信号的指令将被重启并能继续往下执行。

为达到上述目标，需要构造一个虚内存子系统，由它来捕获在DSM系统中的页面访问错及负责从网络上的其它结点处取回数据并完成必要的同步操作。

在前述的DSM系统中，某只读页面可能在多个结点上均有副本，而对可写页面，一般都由一个宿主(Owner)来维护其一致性，远程访问时，由远程结点向宿主结点发出访问请求并从宿主结点处取回该可写页面。通常情况下，对一个可写页面，DSM系统中最多只能有一个副本，当应用程序的数据相关性比较大时，这种对每一可写页面只维护一个副本的策略将引发严重的性能瓶颈。维护多个副本的策略能减缓性能的瓶颈效应，但又带来一个新的问题，即如何在多个副本之间维护数据的一致性？为解决这个问题，提出了一致性存储模型。大致来说，按照对一致性的要求由强到弱的顺序，可区分为以下几类。

3.1 严格一致性模型(Strict Consistency)

这是对一致性要求最严格的一种模型，它由以下条件来描述：任何对内存位置X的读操作将返回最近对位置X进行写操作而存入的值。

在DSM系统中，这是一种理想模型，但受网络延迟的影响，不可能实现。在DSM系统中实现的多种一致性模型，都是对严格一致性模型在不同程度上的放松；而在单机

环境下，任何存储访问序列都满足严格一致性的要求。严格一致性模型与非严格一致性模型的存储访问序列的比较见图3。其中W(x)a表示对应左边的进程将x的值置为a，R(x)b则表示对应左边的进程读取x的值，读到的x的值为b。

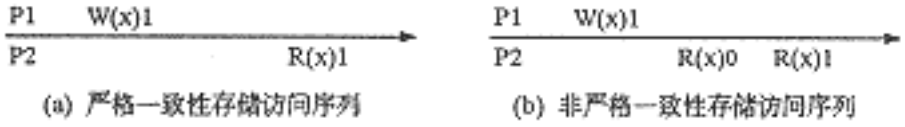

图3　严格一致性模型(水平轴时间)

3.2 顺序一致性模型(Sequential Consistency)

顺序一致性对存储器的限制较严格一致性稍弱，顺序一致性的存储器要满足以下的条件：

(1)每个进程内部各个操作的顺序是确定不变的；

(2)假如所有CPU上的进程都对某个存储单元执行操作，那么，它们的操作顺序是确定的，即任一进程都能感知到这些进程同样的操作顺序。

上述条件的含义是指，当多个进程分别在不同的机器上并行执行时，只要所有的进程都保持同样的顺序访问存储器，那么，任何有效的交叉访问执行都是可接受的。在顺序一致性模型中，时间不再是影响一致性的因素，它关心的是：所有进程必须能感受到一致的内存访问序列。

顺序一致性模型不确保进程的一次读操作能够返回由另一进程所写入的最新值。产生图4中(a)结果的程序在下一次执行时可能会产生图4中(b)的结果。在没有显式的同步操作的情况之下，再一次运行同样的程序不能确保能获得同样的结果。

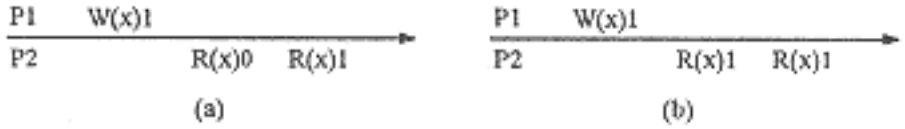

图4　运行同一程序的两种可能的结果(水平轴表示时间)

3.3 因果一致性模型(Causal Consistency)

它是对顺序一致性模型的弱化，它要求在具有潜在因果关系的操作之间保持其一致的顺序。一般性的描述如下：有潜在性因果相关的写操作必须以同样的顺序被各进程所感知，而并发的写操作在不同的机器上可以有不同的顺序。

在因果一致性的存储管理系统中，上述操作序列是合法的，而在顺序一致性或严格一致性模型中，它们是非法操作。

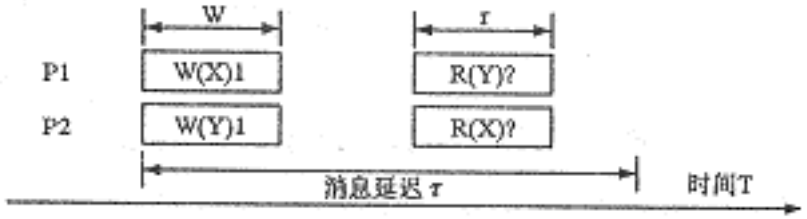

图5　一个全法的因果一致性存储访问事件序列

3.4 管道一致性模型(Pipelined RAM)

管道一致性模型是对因果一致性模型的进一步弱化，它满足以下条件：由某一个进程完成的写操作能够被所有别的进程按序地感知到，而从不同进程中来的写操作对不同的进程可有不同的顺序。如图6示例。

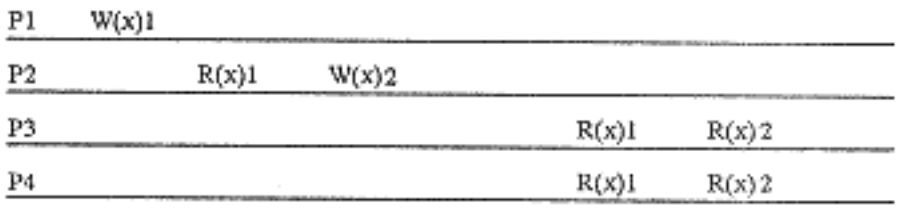

图6　在PRSM模型中的一个合法的事件序列

PRAM模型由于其易于实现，故有一定的吸引力。实际上，它对不同进程所感知的写操作顺序并没有保证，除非是从同一个进程(源)来的写操作，则必须按序到达别的所有的进程，类似于它们处在一条管道中一样。除此之外，在PRAM模型中，由不同进程产生的写操作完全是并行的。

3.5 弱一致性模型(Weak Consistency)

尽管PRAM模型与较之严格的一致性模型相比，能够获得更好的性能，但因为它对同一个进程(源)所产生的结果仍然作了必须按序地送达所有别的进程的要求，故对很多应用程序而言，仍然受到了模型的限制。并非所有的进程都要求看到所有的写操作的结果，让它们按序地看到写操作的结果则更是没有必要。在此，模型的开销能够严重地影响应用程序的性能。考虑到运算的中间结果在大多数情况下并没有必要传播出去的情况，我们必须对一致性要求作进一步的放松，我们只有在需要传播写操作结果的时候才将结果传播出去，除此之外，一切都是完全并行的。为了同步操作的目的，在弱一致性模型中引入同步变量。

弱一致性模型必须满足的条件：

(1)对同步变量的访问满足顺序一致性的要求。

(2)对同步变量的访问，只有在以前的写操作在各处都完成之后才能完成。

(3)对数据的操作(读或写)，只有在以前的对同步变量的访问完成之后才能完成。

第一点说明所有的进程都能以同样的顺序感知到所有对同步变量的访问。当一个进程访问某同步变量时，它会把对该同步变量的访问广播出去，在该进程对该同步变量的操作成功之前，任何别的进程对同步变量的访问都将被阻塞。

第二点说明对同步变量的访问会导致对内存进行刷新的结果。当一个同步访问完成之后，那么，所有先前的写操作可以同时确保已完成。当某进程对一个共享数据作了更新之后，它可以通过同步操作将新值传播出去。

第三点说明当一个进程在读一个共享数据(非同步变量)时，通过同步操作，它能获得该共享数据的最新值。

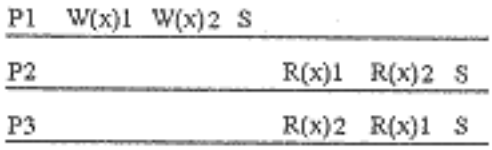

图7　一个合法的弱一致性访问序列

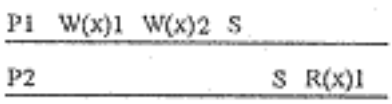

图8　一个非法的弱一致性访问序列

3.6 释放一致性模型(Release Consistency)

对于同步变量的访问，弱一致性模型有一个问题：就是无法区分进程是准备进入临界区还是已经完成对共享变量的操作而准备退出临界区，其后果就是进程在以下两种情况下都必须采取同步操作：

(1)将局部写操作的结果传播出去。

(2)从别的机器上收集共享数据的最新值。

如果能将进入和退出临界区这两个动作区分开来，则能实现一种更为高效的存储一致性模型—释放一致性模型。释放一致性模型提供了两类同步操作：Acquire和Release。某进程将要进入临界区时执行Acquire操作，退出临界区时执行Release操作。也可以不用临界区而用栅栏(Barrier)同步来实现释放一致性协议。栅栏是一种同步机制，它要求所有的进程全都到达程序的某一同步点之后，各个进程才能继续往下执行。

借助于Acquire和Release操作，我们可以把某些特殊的共享变量保护起来，并维护它们的一致性。当然，应用程序必须确知它需要维护其一致性的数据，这也给应用程序的编制增加了一些协议开销，但整体性能提高了，如图9所示。

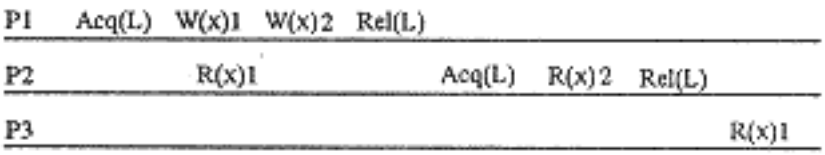

图9　一个合法的释放一致性访问序列

通常，如果一个分布式共享存储系统满足释放一致性，则它必须遵从以下的规则：(1)某进程只有在成功地完成Acquire操作之后，才能确保对一般共享变量(非共享同步变量)访问的正确性。(2)某进程只有在完成对共享数据的读写操作之后，Release操作才能完成。(3)Acquire和Release操作必须满足管道一致性要求。

为了更进一步提高性能，另外一种实现释放一致性模型的协议被称之为懒释放一致性(Lazy Release Consistency)，与之对应，一般的实现被称之为勤释放一致性(Eager Release Consistency)。它们之间的区别在于，勤释放一致性协议在Release操作结束之后，将所有已修改的数据传送给所有别的进程，这样，别的进程都拥有此最新数据的一个副本并可在本地访问它们。但实际情况是有可能别的进程并不需要该共享数据，这样就浪

费了带宽并给程序带来不必要的延迟。懒释放一致性协议在Release操作结束之后，并不急于传送新的数据，而是在别的进程执行Acquire操作之后，由别的进程向它提出获取新数据的请求时，它响应该请求并把共享数据的最新值传送给特定的进程，这样，系统的性能又获得了提高。

3.7 单项一致性模型(Entry Consistency)

另外一种用于提高临界区操作并行性的一致性模型是由Bershad等人于1993年提出的单项一致性模型。它要求编程人员在临界区的开始和结束时使用Acquire和Release操作，但与释放一致性不同的是：它要求每个共享变量都与某同步变量相关联，同步变量可以是锁或者栅栏。以并行访问一个数组的不同元素为例，它要求给不同的数组元素加不同的锁，只有当对同步变量的Acquire操作完成之后，相关的共享数据才得到一致性保证。单项一致性模型也不同于懒释放一致性模型，后者并不将共享数据与锁或栅栏相关联，而是在对同步变量作Acquire操作之后，才能确定它需要哪些共享数据。

因为只有少量的共享变量需要同步，所以如果将每一个同步变量与多个共享数据相关联，则会减少对同步变量的Acquire和Release操作的开销。同时，它还允许多个包含不同变量的临界区并行操作，从而提高了并行性。它的代价主要是每个共享变量均要与一个同步变量相关联，从而带来一些开销及复杂性。

同步变量的使用规则如下：

每个同步变量有一个当前拥有者(即最近获取到该同步变量的进程)，拥有者可以对临界区进行重复的操作而无需在网络上发送任何消息。这时，一个想获得此同步变量的进程需要给当前拥有者发送一个消息，以获得对同步变量操作的所有权及与此同步变量相关联的共享变量的值。在非互斥的情况之下，允许多个进程同时拥有某同步变量，但只能对共享数据进行读操作，而不能写。满足单项一致性的条件是：

(1)在当前拥有者对数据的更新操作未完成之前，不能执行另一进程对同步变量的Acquire操作。

(2)如果某进程以互斥模式访问某同步变量，则在该进程释放此同步变量之前，任何别的进程即使是在非互斥模式下都将无法获得该同步变量。

(3)如果某进程以互斥模式访问一个同步变量，则在该进程完成操作之后，任何进程对此同步变量的后继非互斥访问，都只有在成为该同步变量的拥有者之后才能完成。

4 编程模式

由一系列共享页面所组成的虚地址空间是DSM系统的核心，不妨称之为DSM对象，试图访问虚地址空间的进程在访问前必须满足两个先决条件：

首先，它必须获得一个唯一的DSM对象标识符，借此以区别于系统中的其它DSM对象。其次，它还必须知道控制该DSM对象的DSM服务器的网络地址。

一个DSM对象标识符可通过请求创建一个新的DSM对象或者在已存在的DSM对象集中查询而获得，上述过程被称之为Attach操作。一旦某个进程满足上述两个条件，它将向DSM服务器发出请求，将相应的DSM对象通过Map系统调用映射到本进程的地址虚空间中，从而完成内存空间的分配任务。进程在完成对DSM对象的访问之后，将向DSM服务器发出Detach请求。

以互斥问题为例，传统的互斥算法，如Peterson的入口协议(Entry Protocol)算法，在分布式境下都将失效。其中，¬ 表示取非操作，∨表示或操作。

Peterson的两进程互斥算法在图10中(a)给出。当X_i为真时，进程i将试图获得访问临界区的入口，当两个进程都在试图获取临界区的入口时，变量Y起到仲裁作用，由它来指示哪个进程可以真正进入临界区。此代码段在单机环境下可以很好地工作，但在DSM环境之下，由于没有考虑到通信延迟的影响，上述代码将导致错误的结果，出错情况如图10中(b)所示，造成两个进程均进入了临界区的后果。

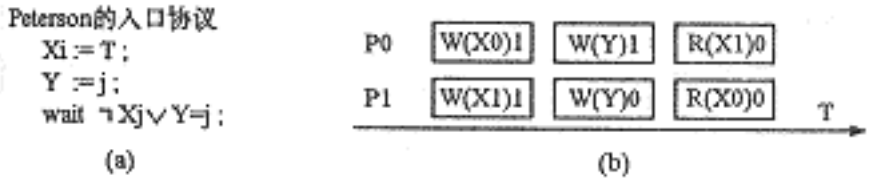

图10　传统互斥算法在DSM环境下失效的示意图

在DSM环境下，互斥算法是可能的，如图11所示，客户通过向互斥服务器发Require请求及等待服务器来的Ack等操作,可以保证能够互斥地进入临界区，通过Realease操作，可以向服务器通告本进程已退出临界区。服务器在接受各客户的互斥请求时，进行排队处理，它只会给处在队列首位的客户发Ack消息，别的客户由于等待接受服务器的Ack消息而处于阻塞状态。其实，以上算法也是对某些存储一致性模型中同步操作(如加锁Lock及栅栏Barrier)的形式化描述。

```
Client i :: {                    Mutex Server :: *{
    ...                              Req_i: =F;
    NCS;                             Ack_i: =T;
    Req_i: =T;                       wait ¬Ack_i;
    wait Ack_i;                  }
      CS;
    Ack_i: =F;
}
```

图11　在DSM环境下集中式的N-进程互斥算法

5 结论

考虑一个包含两个变量x、y的程序，x、y的值均初始化为0。现在由两个进程P_1、P_2来执行该程序，其中P_1把x的值先赋为1，然后读取变量y的值，P_2则把y的值先赋为1，然后读取变量x的值，方框代表操作的时间间隔。在一致性存储器中，P_1或P_2读到0值的情况是不允许的。

假如写操作的最短时间为w，读操作的最短时间为r，消息延迟的最短时间为τ(如图12所示)，如果$w+r<\tau$，则P_1和P_2读到的变量的值都只能是0，即未更新的值。因为消息延迟时间实际上就是信息传输的时间，在某一进程没有收到消息之前，它不可能知道发生在别的进程中的事件，所以，在一致性共享存储系统中，必然满足$w+r\geq\tau$。从

而，可以得出以下结论：不管一个协议多么复杂及多么智能化，只要它实现的是一个一致性共享存储器，那么它必然是慢的(Slow)，即系统的性能必然受到通信延迟的影响。同样的原因，对一致性要求严格的模型，如严格一致性模型或顺序一致性模型，其可扩充性能也是不好。

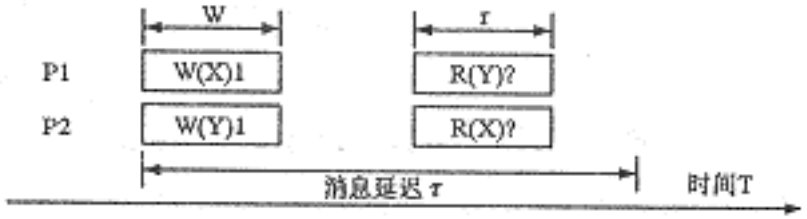

图12　消息延迟的影响

6 相关工作

针对中国科学院计算所开发的JIAJIA系统，我们做了大量的测试及改善性能方面的工作。JIAJIA实现的是一种被称之为域一致性的协议，通过显式的加锁(Lock)及栅栏(Barrier)操作来实现同步。主要的改进之处在于：

(1) 采用块传输方式

由于JIAJIA采用的是UDP协议，实现的是一种不可靠的数据包通信，而且，对具有大通信量的应用程序而言，采用UDP协议通过以太网(EtherNet)将数据逐个地发送出去显得太慢。为此，我们采用了硬件提供的特殊BLT功能，通过PUT/GET操作实现块传输，通信不再通过以太网。因为块传输提高了通信的带宽，所以系统的性能获得了改善。在块传输方式下运行的应用程序随着结点的增加有良好的加速比，并且，与之在以太网通信下的时间耗费相比，性能也有提高。

(2) 将部分功能下移至系统内核实现

DSM系统中带普遍性的一个问题是：应用程序在运行过程中由于需要访问远程数据而不得不陷入到操作系统之中。这种进管/出管的时间耗费是相当大的，在采用块传输方式提高了通信带宽之后，这种进/出管的开销就成为限制系统性能的瓶颈。现在，这部分工作正在进行之中。

(3)进一步在寻求硬件方面的支撑

本项目获国家自然科学基金(基于单映象并行分布式操作系统研究)资助
胡曙（国防科学技术大学计算机系）
廖湘科（计算机学院 长沙 410073）
罗军（计算机学院 长沙 410073）
唐晓东（计算机学院 长沙 410073）
张屹（中国电子设备系统工程公司研究所软件中心 北京 100039）

参考文献

1, Andrew S. Tanenbaum. Distributed Operating Systems

2，Phillip W.Hutto and Mustaque Ahamad. Slow Memory: Weakening Consistency to Enhance Concurrency in Distributed Shared Memories. IEEE 1990
3，Pedro Souto and Eugene W.Stark. A Distributed Shared Memory Facility for FreeBSD. USENIX Association. 1997 Annual Technical Conference
4，Hu Weiwu, Shi Weisong, Tang Zhimin and Li Ming. A Lock-Based Cache Coherence Protocol for Scope Consistency. Vol.13 No.2 J.of Comput. Sci.&Technol. Mar.1998
5，L.lftode, J.P.Singh and K.Li. Scope Consistency: A Bridge between Release Consistency and Entry Consistency. Theory Comput.Systems 31. 451～473(1998)
6，Steven K. Reinhardt, Member, IEEE, Robert W.Pfile, and David A.Wood, Member, IEEE. Hardware Support for Flexible Distributed Shared Memory IEEE 1998

收稿日期：1999年9月21日