计算机科学 1999Vol. 26№. 2

PDSM：一个可移植的分布式共享存储系统*)

PDSM: A Portable Distributed Shared Memory System

徐大杰 章 锋 李宇峰 陈国良

(中国科技大学计算机系 合肥 230027)

Abstract Parallel programming on independent computers has done using systems that support for inter-computer communication through message passing, the programmer has to worry about low level message passing details. DSM mitigates the programming complexity by providing a shared memory abstraction on top of a network of computers. This paper describes PDSM, a flexible and efficient COW-based DSM system in Linux, which we developed to provide an easy to use tool to build high performance computing applications under fine-grained access control and application-specific protocols without complicating the system design.

Keywords Message passing, DSM, COW, False sharing, Thread, Page fault

1. 引言

科学计算是一门迅速发展的学科，传统上，这些问题是用超级计算机或工作站机群来解决的。在互相独立的计算机上的并行程序设计是在 PVM[1]这样的网络并行计算和分布式编程环境下通过消息传递实现结点通信的。但是，由于编程者要了解底层消息传递的细节，基于 PVM 的并行编程十分困难，而科学家们又没有很多精力用于细致的程序设计。DSM(分布式共享内存)通过在工作站机群上建立一个共享内存的抽象层来降低这种程序设计的复杂度。

目前已实现的 DSM[2~4]系统或是基于特殊的操作系统或者需要程序设计者遵守严格的规范，其中绝大多数在细粒度的并行程序下有很高的通信开销。直到今天，在 UNIX 平台上还没有一个专门 DSM 的较通用的实现。因此，有必要在 UNIX 上实现一个有效的 DSM 系统，既可降低并行编程的难度，又保持了相当的加速比。

PDSM 是作为一个建立在工作站机群上解决大计算量的并行应用的平台。它将运行在现今的 UNIX 工作站(或微机)上，并且为科研团体使用，而这些团体目前是在相似的硬件环境上用 PVM 来编写并行应用程序的。我们有二个主要的设计目标。首先，该系统与现存的程序环境兼容，可在不同硬件间移植。为便于用户转向 PDSM，我们决定，不修改内核和编译器，仅将 PDSM 作为一个可与应用程序链接的运行时库。应用程序设计界面简单，很类似于 ANL 宏。为了保证可移植性，我们决定，尽量不使用与机器相关的特性。我们编写了在不同机器环境下处理与硬件特性有关的代码。

其次，该系统将提供好的性能和可扩放性，不低于基于消息传递的 PVM 的性能。为了在问题规模变大时还维持有相当的性能，特别要注意处理器增多的情形。

除了这两个目标，我们对该系统还有其他一些要求。为了能实现不同协议下的性能，我们未将一致性协议在硬件上实现。可定义许多不同的协议，在编译时才绑定某个具体的协议，更进一步，每一页的一致性都可基于不同的协议，这样一个应用可运行多种不同协议。最后，该软件的细致的实现也使得未来的扩展更容易。特别地，我们避开了可能妨碍将来改进的实现特征。

本文提出一个 UNIX 上的基于 DSM 的并行程序设计环境，PDSM(Portable Distributed Shared Memory)，用以方便地实现高性能计算，而且我们将

*)本课题得到国家自然基金和 863 计划的资助。

该系统源码完全公开，使用者可以通过互联网免费得到。

2. 相关工作

关于分布式共享内存系统的最早实现是IVY系统[2]。尽管该系统基于写无效的Cache一致性协议易于实现，但它存在着假共享(false sharing)问题，页面发生抖动，因为两个进程同时对一页中非重叠区域同时写，Munin[3]则实现了基于写共享协议，利用释放一致性(release consistency)来延迟更新广播，因此减少了要发送的消息数量，但它是在V系统上实现的，而V系统并不象UNIX那样为科学团体广泛使用，Treadmarks[4]，是UNIX上的一个DSM系统，它使用懒惰释放一致性，但在请求同步(acquire)时要对整个共享数据写保护，因此，会有很高的开销(diff的产生和更新广播)。我们认为在*Midway*[5]中的一个更类似于入口一致性(entry consistency)的方法，可以将维护一致性的消息减少到一个可以接受的数量。

3. 设计方法

PDSM上的并行程序由多个并发地运行在共享地址空间的合作进程组成，PDSM通过一个共享内存的服务器和与客户程序链接的运行时库来实现共享地址空间抽象层。

3.1 共享内存服务器

该服务器管理共享内存空间，负责空间的分配和回收；也负责同步原语(锁和路障)的分配。尽管该服务器是系统的中心点，但它并不会造成瓶颈，因为它只在客户程序执行之前负责内存的分配，在客户程序执行完毕后负责内存的回收。在客户程序的整个执行期间，只有与客户程序相链接的运行时库与其他处理器上的相应部分通信，以实现一致性协议和同步；而服务器并不参与。

3.2 运行时库

运行时库与应用程序相链接，由许多模块组成，我们称与运行时库相链接的应用程序的一个运行实例为一个结点，PDSM的结点组织如图1。DSM子系统提供了内存一致性，消息子系统是建立在UDP协议之上的，但提供有序且可靠的消息传递，全局线程模块则是建立在CThreads之上并且提供全局线程。以下各节详细描述了这三个子系统。

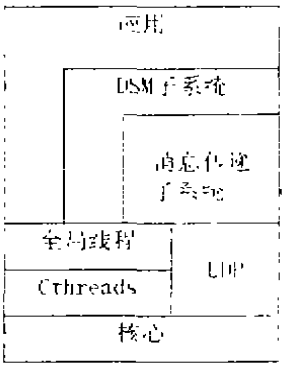

图1 PDSM结点的组织

3.3 全局线程

PDSM通过全局命名策略来支持全局线程概念。一个线程的全局名在该系统中的所有线程之中是唯一的，用于区分消息的发送方和接收方。这些线程并不真正是全局的。

除非是主线程，线程是不能由其他结点远程地创建的，并且它们不可在结点间转移。在SPLASH[6]基准测试程序中，PDSM假定，先启动一个主线程(根线程)，分配且初始化共享内存和同步原语，最后在远程主机上创建(fork)出一系列计算线程。全局线程实际上是用Cthreads来实现的，Cthreads是UNIX上的一个用户级的线程软件包。全局名由二部分连结而得：创建线程的结点的结点id号和该线程的id号。

3.4 DSM子系统

共享内存抽象层是由PDSM的DSM子系统提供的。有两个实现上的问题：应用程序怎样去申请分配共享内存，PDSM怎样维护内存的一致性。

3.4.1 共享内存块 应用程序分配共享内存块，运行时库向共享内存服务器申请分配共享内存块。共享内存块用一个整数标识符来命名。相同id的块在不同结点上是共享的。一旦一个块被分配了，它能象通常的内存一样地存取。PDSM保证共享块的数据一致性。应用程序可分配命名的或未命名的内存块。如果还没有分配名字，服务器将分配给一个名字，但该过程对应用程序是透明的。命名内存块用于在应用程序间共享数据。

3.4.2 内存一致性 共享内存块中的数据一致性由一致性协议来保证。协议是基于页的，应用程序对不同页可使用不同的协议。目前定义了二种协议：基于顺序一致性的写失效协议[2]和基于释放一

致性的写共享协议[3]。UNIX 中的 mmap 和 mprotect 二个系统调用提供了实现这些协议的虚拟内存支持，一个共享内存块是页的集合，这些协议则在页这级粒度上实现了一致性。

3.5 同步

PDSM 提供了三个同步原语，锁，路障和条件变量。锁是用基于分布式队列的算法实现的，而路障和条件变量则是集中地处理的。这些原语都有一个由共享内存服务器分配的 id 号，申请和释放锁的算法要保证公平性，但不必保证申请和释放顺序的一致性。

3.6 消息子系统

消息子系统在不可靠的数据报服务(UDP)上提供一个有序、可靠的消息传递系统。消息在线程间交换，一个线程可以将消息发送给其他结点上的任一线程。发送可以是同步的，即线程发送消息之后，在接收到确认之前是阻塞的；发送也可以是异步的，即发送方不要求确认。接收方线程可以无限期地等待接收，或者在延迟一段时间后超时要求重传。我们仔细地减小了基于 UDP 之上的消息子系统的额外开销，消息缓冲区是预分配的且消息头的各域尽可能填满。消息子系统被 DSM 子系统调用，对应用程序编写者是完全透明的。

模块化是 PDSM 的消息子系统的一个重要特征。消息子系统为高层提供了一个编程接口。这个接口可以用任意的通信协议来实现。PDSM 现在是基于 UDP 协议的，我们认为用其他协议来取代 UDP 实现 PDSM 是很容易的。

作为对消息子系统的进一步优化，为一些常用的消息如页和锁的请求提供可靠传输的工作可以由消息层向上移，即这类消息的丢包和重复并不由消息子系统处理，而是由负责页和锁请求的高层例程处理。在一个典型的 LAN 中，丢包的概率很低，如果由消息子系统处理错误检测和恢复，需要消除大量不必要的确认信息。高层例程可以使用异步发送请求，对丢包使用超时重传，并通过维护一个序号来解决包的重复。

4. 实现

当前，PDSM 已在 SunOS4.1，Irix 和 Linux2.0.30 上实现。库是用 C 实现的，线程软件包则是 UNIX 的一个用户级的 Cthreads 库，通信基于 UDP 协议。下面详细讨论 PDSM 实现的不同方面。

4.1 全局线程

正如 3.3 节所说，全局线程是由创建线程的结点的结点 id 号和线程 id 号二部分连接来获得一个唯一的命名，每个线程在本地表(local table)中有一个条目用于保证消息传递的可靠性。

图 2 示出了表的条目。Cthread_id 是 Cthread 标识符，用以实现由条目代表的全局线程，条目的其他部分则与通信有关，将在 4.5 节中讨论。一个相类似的表，称为远程表(remote table)也维护着消息子系统所需要的关于远程线程与本地线程的有关消息。该表的条目可见图 2，将在 4.5 节中讨论。

4.2 共享内存块

共享内存的分配单元是一个块(region)。应用程序通过运行时库向共享内存服务器申请。尽管应用程序可申请任意尺寸的内存块，PDSM 的内存分配则是按页对齐的。每个结点维护一个已打开块的表(open region table)，由块号(region_id)来索引。该表的条目可参见图 3。Name 是块号，length 是由服务器分配的块尺寸。

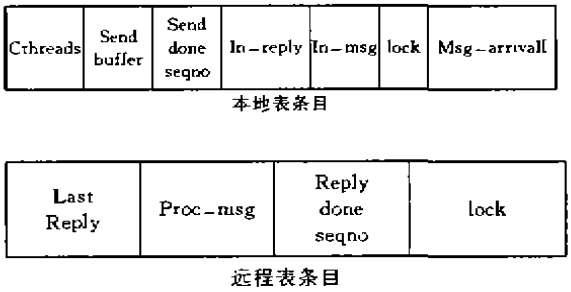

图 2 本地表和远程表条目结构

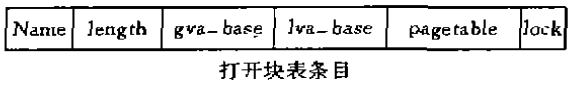

图 3 打开块表条目结构

一旦申请到内存块，一个结点便可用 mmap 系统调用将其映射到它的地址空间。mmap 映射后的块地址与服务器分配的地址可以不同，类似地，同一块可被不同结点映射到各自的虚存空间的不同地址。因此，我们需要二级的地址变换策略。由服务器分配的地址组成全局地址空间，每个结点则在本地地址空间内分配块。所有结点上同一给定块的全局地址相同，但在本地地址空间可映射至不同地址。打开块表中 gva_base 和 lva_base 二个域即表示了这二个不同地址。结点是利用块的全局地址相互通信的，因此，将一个地址从一个线程传递到另一个线程需要二个附加的变换：在发送结点一方将本地地址

映射为全局地址，在接收结点一方将其再映射为本地地址。

pagetable 域是一个指向该块中页的页表的指针。页表结构可见图 4。address 域是该页中第一个字节的地址（在本地地址空间）。在调用 mprotect 中使用。lock 用于保证对页表条目的互斥存取。probOwner 是该页拥有者所在结点的结点 id 号，当发送结点发送页给接收结点，接收结点中该页的 probOwner 域将更新为接收结点的 id 号，因此 probOwner 域是一个线索，一个消息可以从 probOwner 构成的链追溯到页的实际拥有者。copyset 是拥有页拷贝的结点集。access 则标识着页的保护状态，是下面三种状态之一：无效、只读或可读写的。

Address	lock	probOwner	copyset	access

页表条目

图 4　页表条目结构

4.3　内存一致性协议

目前，PDSM 实现了二种协议：类似于 IVY[2] 的写无效协议和由 Munin[3] 提出的写共享协议。尽管每个页都可以绑定不同的协议，但目前这种绑定是在编译时完成的，所有的页都使用了相同的协议。我们正试图提供给编程者一个每页可绑定不同协议的界面。

在写无效协议中，一个结点上的页可映射为无效的、只读的或可读可写的。对无效页的读或写和对只读页的写都会造成页面失效，运行时库向 probOwner 申请页，接收到该页后再将其映射，并提供适当的保护。一个写操作也将拥有同一份拷贝的其他结点的相应页变为无效。一个结点在接收到对一个不属于它的页的请求，它将其转发给该页的 probOwner。而转发结点则将更新该页的 probOwner 为请求结点。

在写共享协议中，多个结点缓存一个页的拷贝，且可以同时对它们写。如果一个页存在多份拷贝，它们都是只读的，如果线程写页失败，则由页面失效例程创建该页的一份拷贝，称为 twin，twin 是可写的，于是线程可继续对该页的存取。当线程到达同步点（释放锁或到达路障），所有拥有 twin 的页将通过一个 diff 消息，通知自上个同步点以后所有的对页的修改。这些 diff 消息发送给拥有该页拷贝的所有结点，页面更新后 twin 被销毁，页也恢复为只读的，以捕获其后的写。

4.3.1　虚存支持　UNIX 下的 mmap 系统调用在进程地址空间中映射页。mprotect 用于修改页的存取模式。对无效页的存取，对只读页的写都会造成段失效，并产生 SIGSEGV 信号。该信号将被运行时库的 segv_handler 消息句柄所捕获，句柄函数确认失效地址是在共享块中的，利用页表，它向 probOwner 发送页面请求。一旦接收到页，句柄函数用 mmap 映射它，用 mprotect 为它设置存取模式，一个称为 dsm_thread 的线程运行在每个结点上，负责处理这些页请求。

4.3.2　同步　锁、路障和条件变量提供了对应用程序必要的同步支持。路障由集中式算法实现，而共享内存服务器则扮演仲裁者。当一个客户程序到达一个路障，它将发送一个消息给同步服务器线程，等待回应。当同步服务器从客户程序接收到足够数量的消息后，它将发送应答给每个客户结点，这样客户程序可继续执行。条件变量也可由服务器来处理，服务器中条件变量或真或假。在条件变量上等待的线程，在条件不满足时将被阻塞。服务器把在该条件变量上等待的所有线程排队。一个信号可唤醒一个线程，而一个信号广播则可启动所有线程。

锁用分布式算法来实现。每个结点都保存着它到目前为止存取的锁的信息。其中的一条信息是锁的当前可能拥有者，类似于页表条目中的 probOwner 域。结点向锁的可能拥有者发送申请锁的请求。拥有锁的结点将所有对锁的申请排队。当锁释放时，锁和队列一并发送给队头结点。队列保证了锁算法的公平性。

4.4　通信

消息子系统在 UDP 之上提供了可靠和有序的消息传递。这是对结点间虚通道状态的维护获得的。本地表 local_table 和远程表 remote_table 维护着这样的状态（见图 2）。因为一个线程只能等待在一个消息上，只有一个消息缓冲区用于发送消息已经足够了。send_buffer 域指向该缓冲区。每个消息都在发送前分配给一个递增的序号，以避免接收的重复。消息到达时产生 SIGIO 信号。asyncio_handler 句柄函数负责读取发送来的消息，并将其送入接收方的消息队列。

本地表的 send_done_seqno 域是被线程成功发送且已收到确认的最后一条消息的序号。in_reply 域存放接收到的确认，而 in_msg 域则是消息队列。lock 用于对表条目的互斥存取，msg_arrival 条件变

量则用于唤醒因等待接收而阻塞的线程。

远程表的 last_reply 域存放远程线程最近收到的确认。如果线程收到一个重复的请求，asynco-handler 利用这个域来重传确认。如果又从远程线程接收到带有新的序号的请求，该确认被丢弃。proc_msg 域则是一个从远程线程处接收，正被处理消息的序号。换言之，确认还未发送。这可防止在处理请求的时间大于发送方的超时时间间隔情况下，发送方重复地给目标线程发送请求。reply_seqno 域则表示已被确认的消息序号，它用于丢弃重复请求。lock 锁保证了对表条目的互斥存取。

对同步消息，发送方线程在接收到应答前阻塞，实际上，它处于循环中，不断检查 in_reply 域。在每次循环后都为其他线程让出处理机。如果应答未在给定时间内到达，线程超时重传。如果经一定数量的重试后仍得不到应答，便可认为传送失败。超时界限以毫秒为单位存放在全局变量 timeoutTime 中。对异步消息、发送线程则不会因等待应答而阻塞。

5. 性能测试

为了测试我们设计的 PDSM 的性能，我们选取了四个典型的应用。它们是：红黑 SOR、TSP，及 Water，Barnes-Hut[6]。我们的测试环境是：四台运行 Linux2.0.30 的 PC 机，其 CPU 为 Intel Pentium 100，内存为 64M EDO RAM，通过 100 Mbps 的 FDDI 网络相连。试验结果是：PDSM 最低能够获得 0.67 的单机效率（Water），最高达到 0.93（SOR）。PDSM 还可以对系统运行过程中的消息量、数据量等用户关心的因素进行统计。

结束语 尽管在分布的网络工作站建立共享内存很吸引人，结点间通信延迟却限制它只能用于粗粒度程序。因此需要工具来微调应用程序以得到高的加速比。使用混合的一致性协议是一种方法，我们正在研究它的优点。类似地，在高度同步的应用程序中，预取也可以减少花在页和锁等待的时间。我们目前正将一些应用移植到 PDSM 上，在研究它们的行为后，我们将开发出必要的微调工具。

参考文献

1 Sundaram V S. PVM: A Framework For Parallel Distributed Computing, Concurrency. Practice and Experience, 1990. 315～339

2 Li K, Hudak P. Memory Coherence in Shared Virtual Memory System. ACM Trans. on Computer Systems, 1989, 7(4): 321～359

3 Carter B, et al. Implementation and Performance of Munin. In: Proc of the 13th ACM Symposium on Operating Systems Principles, 1991. 152～164

4 Keleher P, et al. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In: Proc of the 1994 Winter USENIX conference, 1994. 115～131

5 Bershad B N, et al. The Midway Distributed Shared Memory System. In: Proc. of the '93 COMPCON conf, 1993. 528～537

6 Singh J P, et al. SPLASH: Stanford Parallel Applications For Shared-Memory: [Technical Report CSL-TR-91-469]. Stanford University, 1991

7 Dwarkadas S, et al. An Integrated Compile-Time/Run-Time Software Distributed Shared Memory Systems. In: ASPLOS-VII, 1997

8 Erlichson A, et al. Soft-FLASH: Analyzing the Performance of Clusterd Distributed Virtual Shared Memory. In: 6th ASPLOS, 1996

9 Kontothanassis L, et al. VM-based Shared Memory on Low-Latency Remote-Memory-Access Networks. In: ISCA24, 1997

10 Scales D, et al. Towards Transparent and Efficient Software Distributed Shared Memory. In: SOSP-16, 1997

11 Schomas I, et al. Fine-grain Access For Distributed Shared Memory. In: 6th ASPLOS, Oct 1996

12 Speight E, et al. Brazos: A third Generation DSM Systems. In: USENIX Workshop on Windows-NT, 1997

13 Sets R, et al. Casbmere-2L: Software Coherent Sbared Memory on a Clusterd Remote-Write Network: In: SOSP-16, 1997

14 Yeung D, et al. MGS: A Multigrain Shared Memory System: In: ISCA23, 1996