工作站网络系统进程迁移机制*

裴 丹 汪东升 沈美明

(清华大学计算机科学与技术系 北京 100084)

E-mail: peidan@mail.cic.tsinghua.edu.cn

摘要 进程迁移是工作站网络系统实现负载平衡、提高系统可用性功能的重要手段.该文提出了一种基于接收/发送方消息记录的进程迁移技术.它在消息传递库 PVM(parallel virtual machine)之上实现,具有对用户程序透明、可移植性好、开销小和实现简单等特点.此技术已实际应用于作者自行研制的"并行程序运行回卷恢复与进程迁移系统——ChaRM(checkpointing-based rollback recovery and migration system)"中.

关 键 词 进程迁移,工作站网络,PVM,进程状态.

中图法分类号 TP393

在工作站网络(NOW)系统中,各结点的负载分布情况在很大程度上影响着系统的执行效率.进程迁移技术能动态地改变系统的负载分布,因而它是支持系统动态负载平衡、合理有效地利用资源、提高系统整体性能和系统可用性的关键技术.文献[1~4]指出,利用进程迁移,(1) 在计算过程中,可以把进程从负载较重的结点迁移到负载较轻的结点上,实现动态负载平衡;(2) 可以充分利用 NOW 中的空闲工作站,并且在主人要求独占其工作站资源时及时迁出进程;(3) 在长时间计算的过程中,可以使某结点退出计算以进行系统维护,提高系统的可用性;(4) 使 I/O 操作在设备所在的结点上进行,减少网络通信量.

PVM(parallel virtual machine)是一个应用广泛的并行编程环境,但其本身并没有支持进程迁移.我们在 PVM 的通信库上对其功能进行扩充,开发了一个基于检查点机制的并行程序运行回卷恢复和进程迁移系统——ChaRM(checkpointing-based rollback recovery and migration system).ChaRM 系统提供了并行计算环境下的容错功能和进程迁移功能,并能与负载平衡调度工具相结合,实现对工作站网络系统的智能化资源管理.

进程的状态是指,为了保证进程在迁移之后能够继续正确地执行所必需的信息.进程状态应该包括进程的数据段、用户栈内容,还应包括程序计数器 PC、处理机状态字 PSW、栈指针 SP 内容、活动文件信息以及中断、信号等信息.在进程迁移时,迁移进程捕获其进程状态,并将它直接通过 UNIX 的 socket 通道传给系统为它在目标结点上创建的一个骨架进程;骨架进程则从 socket 通道中读出进程状态,并在自己的进程空间内重建该状态.进程状态传输完毕后,迁移进程自动终止,由骨架进程代替它继续运行.

在并行系统中,还必须保证迁移后其他进程以原来的任务号访问该迁移进程对应的骨架进程,并且保证那些在迁移前和迁移过程中发往该迁移进程的消息都能够以正确的顺序接收,否则将造成消息的丢失并引起程序执行错误.

1 丢失消息的处理

在进程迁移过程中由于迁移进程与骨架进程任务号不同而引起的几种消息丢失的情况如图 1 所示.进

---

* 本文研究得到国家 863 高科技项目基金资助.作者裴丹,1973 年生,硕士生,主要研究领域为并行/分布系统,容错与负载平衡.汪东升,1966 年生,博士,副教授,主要研究领域为并行处理,容错计算.沈美明,女,1938 年生,教授,博士生导师,主要研究领域为并行/分布计算机系统.

本文通讯联系人:裴丹,北京 100084,清华大学计算机科学与技术系

本文 1998-07-20 收到原稿,1998-10-09 收到修改稿

程迁移从 $t0$ 时刻开始,$t1$ 时刻结束(各进程的 $t0$ 和 $t1$ 可能不一样).MP 代表被迁移进程,SP 代表 MP 在目标结点上对应的骨架进程,NMP 代表应用程序中的非迁移进程.在迁移过程中,MP 挂起,NMP 继续运行.迁移结束后,MP 消亡,由 SP 代替 MP 在应用程序中参与计算.

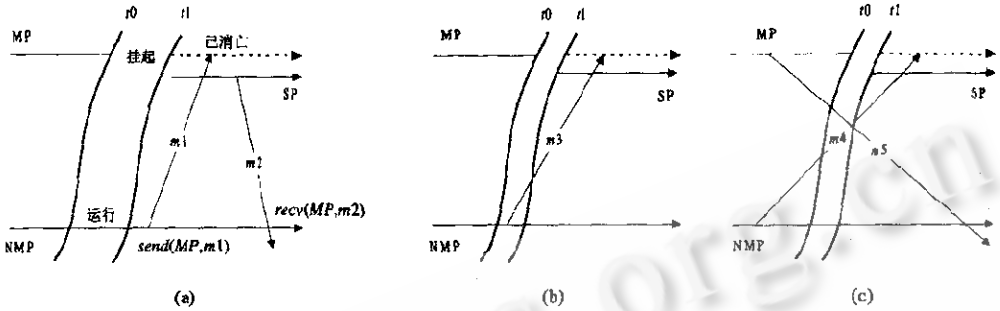

图 1　进程迁移可能造成的消息丢失

在图 1(a)中,由 NMP 在 $t1$ 时刻之后向 MP 发送的消息 $m1$ 找不到接收方 MP,而 SP 也无法收到,消息 $m1$ 丢失.如果 NMP 在 $t1$ 时刻之后等待来自 MP 的消息,由 SP 发来的消息 $m2$ 由于发方不是 MP,NMP 无法接收,$m2$ 也丢失了.为了避免这两种完全由于迁移造成的进程任务号的改变而导致丢失消息,一般支持进程迁移的系统都采用了一种 MP,SP 的任务号之间的匹配机制.这种机制在系统中维护 MP 与 SP 之间的任务号的匹配表,当其他进程要与 MP 通信时,先通过匹配表查到对应的 SP 的任务号,然后再与 SP 进行实际的通信,这样就解决了 $m1$ 和 $m2$ 的丢失.但是,单纯的任务号匹配机制并不能处理像图 1(b)中的 $m3$ 和图 1(c)中的 $m4$ 那样的丢失消息.当 NMP 在 $t0$ 到 $t1$ 时刻之间向 MP 发送 $m3$ 时,本次迁移导致的任务号变化还无法反映在 NMP 当时的匹配表中,因此 $m3$ 被发往 MP 而不是发往 SP;而此时 MP 已经停止计算,在 $t1$ 时刻之后将消亡,它无法收到 $m3$,消息 $m3$ 丢失.图 1(c)中的 $m4$ 和 $m5$ 是在 $t0$ 时刻之前已经发送出去,但是接收进程尚未收到的消息.与 $m3$ 的情况类似,NMP 发送 $m4$ 时并不知道任务号的变化,因此 $m4$ 被发往 MP 而丢失.消息 $m5$ 是由于采用了任务号匹配机制而产生的丢失消息,在 $t1$ 时刻之后,NMP 通过查询匹配表等待接收发自 SP 的消息,所以无法收到发自 MP 的消息 $m5$,消息 $m5$ 丢失.在有多个进程同时进行迁移时,迁移进程之间的丢失消息与图 1 中所示类似,但由于它们彼此在 $t0$ 到 $t1$ 时刻之间不发送应用消息,所以不存在 $m3$ 一类的消息.

在支持进程迁移的系统中,有如下几种维护任务号匹配表的方法.(1) 直接在每个用户进程中维护.这种方法不依赖于特定的并行环境,但是每个用户进程的虚存空间都因此而增长,而且要同时更新匹配表,以维护匹配表内容的唯一性.(2) 由并行编程环境系统在每个结点上的 daemon(如 pvmd)进程维护.这种方法比前一种方法更能节省空间,匹配表更新容易,但需要修改并行编程环境系统源码,可移植性差.(3) 在每个结点上增加一个管理进程,专门负责维护匹配表,并接管进程的通信.这种方法也不依赖于特定的并行环境,但是在进程间通信频繁的情况下有可能成为系统的瓶颈.

处理 $m3$,$m4$,$m5$ 这样的丢失消息的方法一般有两种.(1) 同步方式.在迁移的整个过程中,挂起 NMP(暂停计算),因此 $m3$ 这样的消息根本就不会出现;在传输进程状态前,MP 和 NMP 分别阻塞接收 $m4$ 和 $m5$,避免这两类消息的丢失.这种方式实现简单,但是由于迁移会导致整个应用程序暂停计算,开销大,效率低.(2) 异步方式.这种方式允许 NMP 在迁移过程中继续运算.为了避免丢失 $m3$,$m4$ 或 $m5$ 一类的消息,MPVM[2]等系统引入了复杂的消息转发和排序机制.这种方式执行效率高,但实现复杂,且需要修改并行编程环境系统源代码.

2　ChaRM 系统进程迁移

我们为 ChaRM 系统设计了一种新的进程迁移机制,它允许 NMP 在迁移过程中继续计算,只是在迁移开始和结束时进行简单的协调工作.为了支持在不同并行环境之间的可移植性,ChaRM 系统由一个专门的管理进程 C-Manager 进程负责系统总体控制.为了处理图 1 中的几种消息丢失,ChaRM 系统采取了 PVM 函数封装、任务号隐式匹配、消息驱赶和消息的延迟发送等技术.

(1) PVM 函数的封装. 在执行真正的 PVM 函数前后,进行一些额外的工作以完成 ChaRM 系统的功能,但是提供给用户的接口与原 PVM 函数的接口完全一致.这种封装技术避免了对 PVM 源码的直接修改,而且使 ChaRM 所做的工作对用户透明.

(2) 任务号隐式匹配. ChaRM 在每个用户进程空间中维护一个任务号匹配表,并使进程在每次通信时都先查询匹配表再进行真正的通信.在迁移结束前,更新各个进程的匹配表.在整个计算过程中,应用程序只需知道各进程创建时的任务号,而完全不必知道该进程是否发生过迁移.这种机制有效地避免了图 1 中的 $m1$ 和 $m2$ 一类消息的丢失.

(3) 消息驱赶机制. ChaRM 利用 PVM 通信机制本身的 FIFO 语义将一个进程向另一个进程的消息通道清空.假设在迁移过程中,A 进程向 B 进程发送一个特殊的就绪消息后就不再向 B 发送其他消息;而 B 进程等待接收该就绪消息,将接收到的应用消息妥善保存到预先指定的接收缓冲中.根据 FIFO 语义,当 B 进程收到来自 A 进程的就绪消息时,可以推断从 A 进程到 B 进程的单向通信通道中的所有消息已被驱赶至 B 进程.这种基于收方的消息记录避免了图 1 中的 $m4$,$m5$ 的丢失,同时也保证了正确的消息接收顺序.

(4) 消息的延迟发送机制. 为了避免产生图 1 中的消息 $m3$,ChaRM 对 PVM 的消息发送函数进行了封装,使得当 NMP 要在 $t0$ 到 $t1$ 时刻之间向 MP 发送应用消息时,并不进行真正的发送,而是将该消息记录到一个延迟发送消息表中.等到迁移完毕再根据更新过的匹配表将延迟发送消息表中的应用消息发送出去.这实际上是一种基于发方的消息记录.

算法 1. ChaRM 系统的迁移算法.

1. **C-Manager**: **IF** 接收到迁移发起信号 MIGRATE_SIG **THEN**
   向所有 MP 发出迁移启动信号 MIG_CHECK_SIG;
   向所有 NMP 发出迁移协调信号 MIG_SYNC_SIG;
2. **MP**: **IF** 收到 MIG_CHECK_SIG **THEN**
   暂停计算,建立起一个用于与 SP 通信的 socket 管道;
   向 C-Manager 发送 PRT_MSG,通知它的 socket 的地址、端口号等信息;
   向其他所有进程发送就绪消息 RDY_MSG;
   持续接收向它发送的消息,直至收到来自其他所有进程的 RDY_MSG.
   IF 收到除 RDY_MSG 以外的其他消息(被驱赶而来的中途应用消息) **THEN**
   把这个消息保存到预先指定的接收缓冲中;
   NMP: **IF** 收到 MIG_SYNC_SIG **THEN**
   向所有 MP 发送就绪消息 RDY_MSG;
   继续进行计算;
   IF 在迁移完毕之前,要向 MP 发送应用消息 **THEN**
   将消息记录到延迟发送消息表中,不进行真正的发送;
   C-Manager: **IF** 收到从 MP 发来的 PRT_MSG 消息 **THEN**
   在目标结点上派生出 MP 对应的骨架进程 SP;
   将 MP 的 socket 地址、端口信息通过 PRT_MSG 传送给骨架进程;
3. **MP**: **IF** 已经收到其他所有进程发来的 RDY_MSG **THEN**
   等待与相应的 SP 建立 socket 连接;
   捕获进程状态,并将进程状态通过 socket 传给骨架进程;
   状态传输完毕后消亡;
   SP: **IF** 收到 C-Manager 发来的 PRT_MSG **THEN**
   与相应的 MP 建立 socket 连接;
   从 socket 中读出进程状态,并在自己的进程空间内重建该状态;
   将任务号变化通过 RJN_MSG 通知 C-Manager;
4. **C-Manager**: **IF** 收到所有 SP 发来的 RJN_MSG **THEN**

向所有 NMP 再次发送 MIG_SYNC_SIG;
并向所有进程发送 LST_MSG 用于各进程更新其任务号匹配表;

NMP: **IF** 收到 MIG_SYNC_SIG 信号 **THEN**
持续接收向它发送的消息,直至收到所有发自 MP 的 RDY_MSG;
IF 收到除 RDY_MSG 以外的其他消息(被驱赶而来的中途应用消息) **THEN**
把这个消息保存到预先指定的接收缓冲中;
接收 C-Manager 发来的 LST_MSG,更新各自的任务号匹配表;
根据新的匹配表将延迟发送消息表中延迟发送的消息发送出去;
继续执行;

SP: **IF** 接收到由 C-Manager 发来的 LST_MSG **THEN**
更新任务号匹配表;
继续执行;

在算法的第 2 步中,MP 向所有其他进程发送就绪消息,NMP 向所有 MP 发送;然后,MP 将所有其他所有进程向它发送的应用消息驱赶到缓冲区中;NMP 则继续进行计算,而并不在此时就进行消息驱赶,这是因为在第 2 步中进程间彼此发送的消息较多,将这一工作推迟到第 4 步再做可以使 NMP 因消息驱赶所花费的开销最小.在第 4 步中,在 NMP 消息驱赶完毕后,它将延迟发送的消息发送出去并继续执行.这种延迟发送机制带来的附加开销很小,这有两方面原因:一方面,NMP 执行完消息发送命令后不必理会接收进程是否收到而可以继续计算,即发送应用消息并不在 NMP 执行的关键路径上,所以这种机制对 NMP 的计算过程无直接影响;另一方面,由于延迟发送的消息是在迁移完毕后立即被发送出去,而 MP 直到迁移完毕后才继续计算和接收应用消息,所以它不会因为长时间等待接收发自 NMP 的应用消息而明显增加执行时间.

在同步方式的进程迁移中,NMP 的开销与 MP 的开销大致相当;而在算法 1 中,NMP 除了在迁移发起时和迁移结束时做一些协调工作以外,一直处于计算过程中,其开销远小于同步方式.在异步方式的进程迁移中,虽然 NMP 没有明显的协调工作,但这种方式引入的复杂的消息转发和排序机制也会给应用程序的正常运行带来一定的开销;而算法 1 中以 NMP 在迁移过程中参与协调为代价,使得算法易于实现,并且正常运行开销小于异步方式.综上所述,算法 1 中基于发送/接收方的消息记录的算法有效地结合了同步方式的易于实现和异步方式的效率高的优点.

3　性能测试

测试环境为由 4 台 SUN Ultra2(每个结点两个 CPU,主频 200MHz)经 100M 快速以太网连接而成的工作站网络系统,操作系统为 Solaris2.5.测试用例为计算密集型的矩阵幂程序,该程序由一个根任务在 4 个结点上各派生出两个子任务,总共 8 个子任务.测试结果如表 1 所示,其中状态空间是指通过 socket 传送的进程状态信息的总和;挂起时间是指从 MP 在源结点上暂停计算到在目标结点上继续计算所花费的时间,它等于状态传输时间加上协调时间;MP 协调时间是指除了在网络上进行进程状态传输以外,它所进行的所有其他工作所花费的时间,其中包括发送及接收就绪消息、与 C-Manager 通信、骨架进程的派生、维护任务号匹配表等工作;NMP 最大协调时间是指各 NMP 为进程迁移所花费的时间中的最大值,包括发送及接收就绪消息、与 C-Manager 通信、延迟发送消息、维护任务号匹配表等工作.

从表 1 可以看出,进程状态传输时间是挂起时间的主体部分,它随着进程状态空间的增加,大致呈线性增加,其上下波动是由网络传输的波动所致,这部分开销的大小主要取决于网络硬件;协调时间随着进程状态空间的增加虽然呈增加趋势,但变化并不显著,在 0.2s 左右,这是由于协调工作与进程状态空间的关系很小所致.从图 2 中可以更直观地看出 MP 各部分开销与状态空间的关系.表 1 中的 NMP 的协调时间很小,与 MP 挂起时间相差两个量级,这验证了我们对算法 1 的分析:NMP 开销远小于同步方式,而用毫秒级的协调开销来避免实现复杂的异步方式进程迁移也是完全可以接受的.这一开销逐渐变大的趋势是由于传输较大的进程状态空间导致网络通信变慢所致.

表 1 矩阵幂程序进程迁移开销

状态空间(MBytes)	挂起时间(s)	状态传输时间(s)	协调时间(s)	NMP 最大协调时间(ms)
1.68	0.418	0.219	0.189	2.0
2.00	0.438	0.253	0.185	1.7
3.01	0.584	0.368	0.196	2.6
4.00	0.661	0.472	0.189	2.5
5.00	0.759	0.565	0.194	3.5
6.01	0.938	0.684	0.254	5.8
7.01	1.039	0.819	0.220	7.5
8.02	1.120	0.916	0.204	2.6

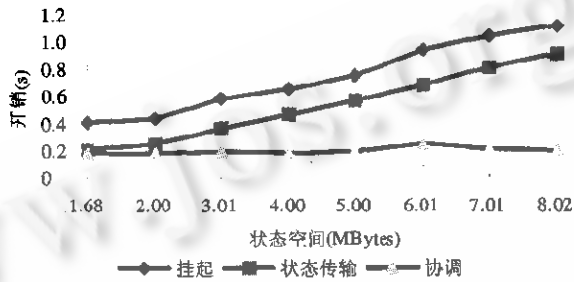

图 2 矩阵幂程序 MP 迁移开销折线图

另外,我们还对其他一些程序进行了测试,结果见表 2.其中偏微分方程、哈达码变换为计算密集型,ASE、分数维、FFT 为通信密集型.可以看出,计算密集型与通信密集型在 MP 的协调时间上区别很小,证明消息驱赶和延迟发送机制的效率较高;而在通信密集型的应用程序中,由于 NMP 要保存和重发的消息比计算密集型的多,所以其协调时间有一定的增加,同时这一增加也受状态空间大小的影响.

表 2 应用程序进程迁移开销

应用程序	状态空间(MBytes)	挂起时间(s)	状态传输时间(s)	协调时间(s)	NMP 最大协调时间(ms)
哈达码变换	1.6	0.438	0.212	0.215	2.0
偏微分方程	2.2	0.471	0.273	0.196	3.1
分数维	1.7	0.437	0.225	0.204	3.3
ASE	8.0	1.088	0.864	0.223	1.9
FFT	3.7	0.445	0.679	0.234	4.4
FFT	22.1	2.232	2.445	0.211	6.2
FFT	66.1	7.348	7.687	0.249	7.6

4 相关工作及结论

进程迁移在许多并行系统中得到了支持和应用.早期的系统 Charlotte,V system,Mosix,Sprite 和 Mach,大都是通过修改操作系统内核,在系统级实现了对进程迁移的支持[1].这种方法效率高,透明性好,但可移植性差.用户级实现的进程迁移只使用标准的 UNIX 系统调用,因而可移植性好.其中 MPVM[2]修改了 PVM 源码,并引入了复杂的消息转发和排序机制以支持异步进程迁移;Condor[3,4]支持的进程迁移是先将进程的检查点文件存入磁盘并中止该进程,等到系统找到空闲机后在该机上恢复;CoCheck[5]利用了 Condor 提供的单进程检查点设置和卷回恢复工具,在 PVM 通信库之上实现,但只支持同步方式的进程迁移.DPVM[6]利用进程迁移进一步支持了动态调度的功能.

与国际上的同类工作相比,ChaRM 系统具有下面的一些功能和设计特点.(1) 对用户程序透明,以运行时库的方式提供给用户.用户只需将 ChaRM 提供的库与原有 PVM 应用程序链接,即可用命令方式或 API 函数调用进程迁移功能.(2) 在并行编程环境通信库上实现,并且由专门的管理进程 C-Manager 而不是选择 PVM

的资源管理器 RM 负责总体控制,从而不受限于特定的并行编程环境,可移植性好.(3) 通过引入消息驱赶和消息延迟发送机制,支持一种新的、基于收/发方消息记录的迁移技术以提高迁移效率,而未采用复杂的消息转发和排序机制.(4) 支持手工、半自动、全自动 3 种方式的进程迁移发起方式.即用户可以指定把哪个进程迁往哪个结点,也可以提出对某些进程的迁移请求,由系统决定迁往哪个结点,还可以由系统自行决定迁移发起的时机、迁移的进程以及目的结点.(5) 对进程状态进行了精简,排除了数据段中不必保存的区域,从而减少了由于迁移引起的状态捕获、传输和重建的时空开销.

本文提出的进程迁移技术可以在 PVM,MPI 等多种消息传递库上实现,采用了任务号隐式映射、消息驱赶和消息延迟发送技术,是一种基于接收/发送方消息记录的迁移技术,具有实现简单、对应用程序透明、可移植性好和开销小等特点.

在下一步的工作中,我们将在 ChaRM 系统中设计并实现动态负载平衡调度算法,对工作站网络系统进行智能化资源管理.

参考文献

1 Eskicioglu M R. Design issues of process migration facilities in distributed systems. IEEE Technical Committee on Operating Systems Newsletter, 1989,4(2):3~13

2 Casas J, Clark D L *et al*. MPVM: a migration transparent version of PVM. Computing Systems, 1995,8(2):171~216

3 Litzkow J *et al*. Condor——a hunter of idle workstations. In: Proceedings of the 8th IEEE International Conference on Distributed Computing Systems. Los Alamitos, CA: IEEE Computer Society Press, 1988. 104~111

4 Tannenbaum T, Litzkow M. The condor distributed processing system. Dr. Dobb's Journal, 1995,(2):40~48

5 Stellner G, Pruyne J. Resource management and checkpointing for PVM. In: Proceedings of the 2nd European PVM Users' Group Meeting. Lyon, France: Edition Hermes , 1995. 131~136

6 鞠九滨,魏晓辉,徐高潮等.DPVM:支持任务迁移和排队的 PVM.计算机学报,1997,20(10):872~877
(Ju Jiu-bin, Wei Xiao-hui, Xu Gao-chao *et al*. DPVM: an enhanced PVM supporting task migration and queuing. Chinese Journal of Computers, 1997,20(10):872~877)

Process Migration for the Network of Workstations

PEI Dan　WANG Dong-sheng　SHEN Mei-ming

(*Department of Computer Science and Technology　Tsinghua University　Beijing* 100084)

Abstract　Process migration is an important tool on NOWs (network of workstations) for load-balancing and high availability. In this paper, the authors present a log-based approach to provide process migration for parallel applications. Because this approach is implemented on top of PVM (parallel virtual machine), it is transparent to users and portable. This approach has been used in the ChaRM (checkpointing-based rollback recovery and process migration system) system. The experiments show that the overhead is low.

Key words　Process migration, network of workstations (NOW), PVM (parallel virtual machine), process state.