利用检查点机制在PVM中实现进程迁移*

鞠九滨 魏晓辉 郭 雷

(吉林大学计算机系 长春 130023)

摘要 本文通过对PVM进程通信机制进行修改及引入Condor的检查点机制,实现了对PVM用户进程的迁移.

关键词 检查点机制,进程通信,进程迁移.

目前,工作站和高档个人微机的数量正在快速增加,而且越来越多地使用局域网把这些机器连接起来.这种新的计算系统,已成为并行计算的一个主流.PVM[1]是支持这种系统的一个优秀软件.PVM所提供的功能简单实用,目前已被广泛使用,其用户已超过10 000个,是较为标准的并行计算环境.PVM可以让用户把一个松散耦合的异构计算机系统当做一个统一、方便的并行计算资源来使用.在系统启动时,系统中的每一台机器都运行一个PVM daemon进程(pvmd)负责管理所在的机器.但PVM不支持在进程迁移基础上的动态负载平衡和容错.

Condor[2]是一个自动寻找空闲处理机以执行长时间运行任务的软件.Condor的检查点机制CKPT(checkpointing)已被应用于CODINE,LoadLeveler等分布式计算机系统中.另外一些著名的分布计算系统如DQS,Load Balancer,LSF,NQE和PBS也正在进行对CKPT的开发工作,支持它们的检查点机制和进程迁移的功能.[3]Dynamic PVM [4]试图利用CKPT支持PVM进程迁移达到动态负载平衡.Fail—safe PVM[5]利用CKPT支持容错.

我们已把Condor的CKPT从Condor系统中分离出来成为独立的机制.用户在使用这种CKPT时,不需要修改源程序,只需要在编译时与我们所提供的库进行链接即可.当需要中止正在运行的进程做检查点时,只需向该进程发送SIGTSTP信号.该进程收到SIGTSTP信号后立即中止运行,保存当前的进程环境,形成一个检查点文件.当这个检查点文件重新启动时,首先恢复保存的进程环境,使进程从断点处继续运行(这部分工作将在另一篇文章中介绍).本文叙述的工作是在此基础上把CKPT引入PVM中,用以实现进程迁移,并支持PVM所提供的进程通信函数.

在PVM中实现与多个进程有通信关系的进程迁移,关键要解决如下问题:

(1)Socket通信的原子性.在Unix环境下,进程间通信主要使用Socket.在进行通信的

* 本研究得到国家863高科技项目基金资助.作者**鞠九滨**,1935年生,教授,博士导师,主要研究领域为分布式系统.**魏晓辉**,1972年生,硕士生,主要研究领域为分布式系统.**郭雷**,1972年生,硕士生,主要研究领域为分布式系统.

本文通讯联系人:鞠九滨,长春 130023,吉林大学计算机系

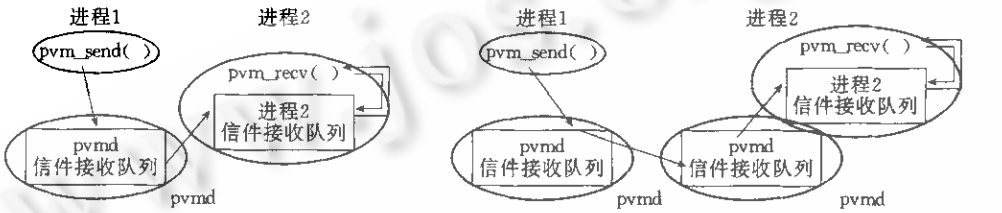
本文1995—08—23收到

过程中,有部分信息保存在 Unix 系统的内部,我们无法得到.因此,使用 CKPT 对进程做检查点时,如果这时正好打断了 Socket 通信,我们无法在检查点文件中保存 Socket 的完整信息.因此在进程迁移时应选择 Socket 不进行通信的时刻做检查点.

(2)相关进程通信的处理.由于要迁移的进程与系统中的多个其它进程通信,因此在做进程迁移时应该保证该进程与其它进程的通信不受进程迁移的影响.

1　PVM 的进程通信机制

在 PVM 系统中,用户编程时通过使用 PVM 所提供的进程通信函数进行进程通信.PVM 中的用户进程之间的通信有 2 种方式:(1)间接方式(见图 1);(2)直接方式(见图 2).



(a) 发送进程与接收进程在同一机器上　　(b) 发送进程与接收进程不在同一机器上

图1 进程间接通信

方式(1)是 PVM3.0 版本中提供的进程通信方式.在这种方式中,PVM 用户进程之间的通信都是通过 *pvmd* 实现的:PVM 用户进程把要发送的数据发送给本地的 *pvmd*,若数据的接收进程在同一台机器上,本地 *pvmd* 就把数据转发给本地的接收进程;否则,本地 *pvmd* 把数据发送给接收进程所在机器的 *pvmd*,再由这个远程 *pvmd* 把数据转发给远程的接收进程.在这种通信机制中,用户进程之间不直接通信.用户进程只需保留一个 *Socket* 端口用以与本机的 *pvmd* 进程相互交换信息.这种通信方式易于保证通信的原子性,而且进程的地址发生变化后,只需要通知每个机器上的 *pvmd* 进程,所以该通信方式易于实现进程迁移.但这种通信方式使 *pvmd* 进程不仅要负责整个系统环境的维护,还要负责用户进程间的信件管理,使 *pvmd* 成为系统的瓶颈.

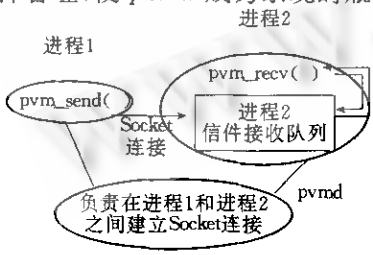


图2 进程直接通信

方式(2)是 PVM3.3 版本中引入的一种新通信方式.在这种通信方式中各机器上的 *pvmd* 只负责在 2 个 PVM 用户进程之间第一次通信时进行握手,使 2 个 PVM 用户进程之间直接建立 *Socket* 连接,以后这 2 个 PVM 用户进程通信就直接使用该 *Socket* 而不再通过 *pvmd*.这样 *pvmd* 的负担大大减轻,不再成为系统的瓶颈,提高了整个系统的效率.但每个用户进程需要与每一个与之进行通信的用户进程建立一个 *Socket* 连接,这就为实现进程迁移带来了很大的困难.首先,每个进程在迁移之后,要打断所有与之有通信要求的进程,把自己的新地址通知给它们,使整个系统的开销增大.其次,要选取一个恰当的时刻(使要迁移的进程的所有 *Socket* 连接都符合通信的原子性要求)做检查点是异常困难的.

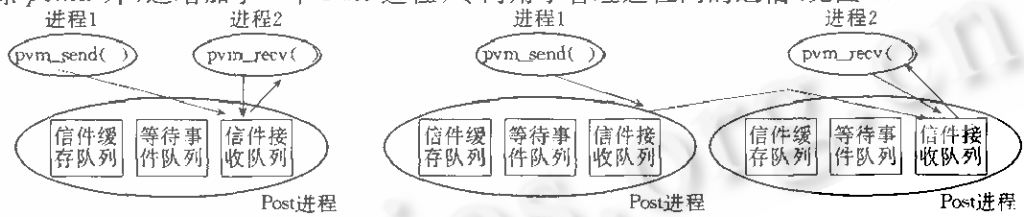
2　修改 PVM 进程通信机制

由上面的分析可知,虽然 PVM 的进程通信函数是用 *Socket* 实现的,但它是由 PVM 系统进行控制. 我们可以通过修改 PVM 进程通信机制控制 *Socket* 的通信,解决前述的 2 个关键问题,在 PVM 系统中实现进程迁移. 为此我们对 PVM 的进程通信机制作如下修改:

(1) 使用原子进程通信的概念. 我们对 PVM 的进程通信函数进行了修改:使这些函数在执行期间阻塞 *SIGTSTP* 信号,直到函数执行完毕以保证进程通信的原子性. 在这些函数的执行过程中,任何进程迁移的请求都将被推迟,直到函数执行完毕.

(2) 设置 *Task_tid* 转换表. 每个在 PVM 上运行的用户进程都有一个全局唯一的整型值(*tid* 值)作为标识. 通过进程的 *tid* 值可以确定该进程在系统中的位置. 每个 PVM 进程初始创建时通过调用 PVM 的 *pvm_mytid*()函数得到自己的 *tid* 值,系统中的其它进程与之进行通信时都以这个 *tid* 值作为该进程的标识,我们称之为进程的逻辑 *tid* 值. 而进程迁移之后,继续运行的进程将得到一个新的 *tid* 值来标识该进程在系统中的新位置. 这个新的 *tid* 值在该进程下一次迁移之前实际标识该进程,我们称进程迁移后得到的 *tid* 值为进程的实际 *tid* 值. 因此,为保证进程迁移后信件能够正确发送给新的进程,就需要一个 *Task tid* 转换表,用来存放系统中所有进程的逻辑 *tid* 值和实际 *tid* 值之间的映射关系.

(3) 对相关信件的处理. PVM 系统程序要负责当一个进程正在迁移期间,对发往该进程的信件进行缓存. 在该进程迁移完成之后,PVM 系统程序还要把缓存的该进程信件发送给新的进程,并负责更新 *Task_tid* 转换表中该进程的实际 *tid* 值. 每个机器上运行的系统程序除 *pvmd* 外,还增加了一个 *Post* 进程,专门用于管理进程间的通信(见图 3).



(a) 发送进程与接收进程在同一机器上　　(b) 发送进程与接收进程不在同一机器上

图3 修改后PVM进程通信方式

PVM 的发送函数只是把要发送的信件和 SEND 命令一起发送给本机上的 *Post* 进程,得到 *Post* 进程的应答后立即返回,然后由 *Post* 进程负责完成这次发送操作. PVM 的接收函数只是向本地的 *Post* 进程发送 RECV 命令,然后等待本地 *Post* 进程把自己索要的信件发送给自己.

3　进程迁移的控制过程

设置一个 *Manage* 进程,负责对整个系统的运行状况进行管理,收集各机器的负载信息,作出进程迁移的决定,并在进程迁移期间协调系统中各机器的动作. *Manage* 进程对进程迁移的控制操作如下:

(1)在需要进程迁移时向系统中的所有 *Post* 进程发送 MIG_BEGIN 命令,并把要迁移

进程的逻辑 *tid* 值放在命令包中;等待所有 *Post* 进程的应答;

(2)向迁移进程所在机器的 *Post* 进程(源 *Post* 进程)发送 MIG_FROM 命令,并把"目的 *Post* 进程"地址放在命令包中;向进程迁移的目的机器的 *Post* 进程(目的 *Post* 进程)发送 MIG_TO 命令,并等待"目的 *Post* 进程"发来的迁移进程的新实际 *tid* 值;

(3)向"源 *Post* 进程"发 MIG_MESSAGE 命令,并把"目的 *Post* 进程"地址放在命令包中;等待从"源 *POST* 进程"发来的应答;

(4)向所有的 *Post* 进程发送 MIG_END 命令,并把"目的 *Post* 进程"地址和迁移进程的新实际 *tid* 值放在命令包中;

最后,在系统退出时向所有的 *Post* 进程发送 QUIT 命令.

Post 进程接收从本机 PVM 用户进程发来的进程通信命令和从 *Manage* 进程发来的管理命令,然后依照命令执行相应的操作.每个 *Post* 进程管理 3 个队列(见图 3)和 1 个表.

这 3 个队列是:(1)信件接收队列,保存着所有发送给本机上的 PVM 用户进程的信件.(2)信件缓存队列,用于对那些暂时还不能确定其目的地址的信件进行缓存(这些信件的接收进程正在迁移过程中).(3)等待事件队列,对那些不能马上处理的用户发送、接收命令进行登记,用于以后进行处理.

这个表是 *Task_tid* 转换表.表的每一项结构如下:

Enable	进程的逻辑 tid 值	进程的实际 tid 值

Enable:整型,等于 0 时表示进程正在迁移的过程中,这一项不可用;等于 1 时,可用.

Post 进程工作过程如下:

```
RUN=TRUE;
WHILE(RUN) DO
   BEGIN
     接收命令;
     SWITCH(命令)
        BEGIN
          CASE SEND:
             向发出命令的用户进程发应答包;
             以信件的目的进程的逻辑 tid 值为关键字查找 Task_tid 转换表;
             IF (Enable) THEN
                    把信件发送给该目的进程所在机器的 Post 进程;
             ELSE 添写等待事件队列;
                    把该信件放入信件缓存队列;
          CASE RECV:
             从信件接收队列中查找符合条件的信件;
                 IF 查找成功 THEN 把找到的信件发送给接收进程;
                 ELSE 添写等待事件队列;
          CASE MIG_BEGIN:
             从命令包中取出要迁移进程的逻辑 tid 值;
                 使 Task_tid 表中相应项的 Enable=0;
                 向 Manage 进程发应答;
          CASE MIG_FROM:
             从命令包中读取"目的 Post 进程"的地址;
                 向要迁移的用户进程发送信号 SIGTSTP;
                 把生成的检查点文件发送给"目的 Post 进程";
          CASE MIG_TO:
             接收检查点文件并重新启动该检查点文件;
                 把进程的新实际 tid 值发送给 Manage 进程;
```

```
        CASE MIG_MESSAGE:
          从命令包中取出"目的 Post 进程"的地址;
              把迁移进程的信件从信件接收队列中取出转发给"目的 Post 进程";
              向 Manage 进程发应答;
        CASE MIG_END:
          从命令包中取出"目的 Post 进程"地址和迁移进程的新实际 tid 值;
              把迁移进程的信件从信件缓存队列中取出发送给"目的 Post 进程";
              更新 Task_tid 表中迁移进程的实际 tid 值,并使表中相应项的 Enable=1;
        CASE QUIT: RUN=FALSE;
      END /* SWITCH */
       依次处理等待事件队列中的事件;
END /* WHILE */
```

4　进一步的工作

我们在 PVM 系统中使用本文的方法实现了进程迁移. 在并行计算中,系统的可靠性和系统中各机器的负载分布都在很大程度上影响并行任务的执行效率. 我们已在本文工作的基础上在 PVM 系统中实现了动态负载平衡的功能,在今后的工作中将进一步实现系统的容错.

参考文献

1　Geist Al *et al*. PVM user's guide and reference manual. February 1993.

2　Litzkow Michael J, Livny Miron, Muttka Matt W. Condor—a hunter of idle workstations. IEEE 8ICDCS, 1988.

3　Kaplan J A, Nelson M L. A comparison of queuing, cluster and distributed computing systems. NASA TM 109025, June 1994.

4　Leen Dikken *et al*. Dynamic PVM. In: Proc. of High Performance Computing and Networking, Apr. 1994.

5　Leon Juan, Fisher Allan L, Steenkiste Peter. Fail—safe PVM: a portable package for distributed programming with transparent recovery. Report CMU—CS—93—124. Feb. 1993.

IMPLEMENTING PROCESS MIGRATION IN PVM WITH CHECKPOINTING

Ju Jiubin　Wei Xiaohui　Guo Lei

(*Department of Computer Science　Jilin University　Changchun*　130023)

Abstract　This paper introduces an implementation of process migration in PVM system using checkpointing and modifying the process communication mechanism of PVM.

Key words　Checkpointing, process communication, process migration.