NOW中支持进程迁移的运行时增量式集中式调度*)

Runtime Incremental Concentrated Scheduling Supporting Process Migration on NOW

陈锡明　卢显良

(电子科技大学计算机系8020教研室　成都610054)

Abstract This paper proposes and implements a method of process migration among workstations—CSR_PM. Based on CSR_PM, a method of Runtime Incremental Concentrated Scheduling Supporting Process Migration on NOW(NRICS_SPM)is proposed, which has four important characteristics: runtime scheduling, incremental scheduling, concentrated scheduling and exchanging load among nodes parallelly, supporting task and process migration among nodes. With this method, the load imbalance in one system phase can be improved and corrected in the next system phase, so all nodes' loads are continuously trending to balance. NRICS_SPM has good performance, some experimental results are given and are compared with relative works at the end of this paper.

Keywords NOW, Process migration, Task, Incremental scheduling, Concentrated scheduling

NOW(Network Of Workstations)是多个独立计算机(以下简称结点)通过高速网络和特定网络操作系统连接而成的一个可用来进行大规模并行处理和负载平衡的网络计算环境[1]。由于NOW中各结点计算能力的差异,相同任务在不同结点上的执行时间可能存在较大差异,导致结点间负载失衡。进程迁移技术可以将粒度大的任务在执行过程中从计算能力弱的结点迁移到计算能力强的结点,加快进程执行速度,促进结点间负载平衡,提高系统整体性能和可用性[2]。

在NOW中实现并行处理和负载平衡的调度策略主要有三种:静态调度、动态调度和并行调度。静态调度严重地依赖于对程序所产生负载的估计的精确程度,其自适应性、可伸缩性较差;动态调度适应了应用程序可变化、可伸缩等特点的需要,但由于每个结点上任务提交的动态特性,结点之间很难及时、准确地掌握相互的负载信息,出现"任务颠簸"的几率较大。并行调度集成了静态调度和动态调度的优点,使所有处理机合作起来进行任务调度,结点之间尽可能并行地交换负载,能够精确地进行全局负载平衡,主要应用于多处理机并行计算环境。NOW是由多个独立计算机组成的网络环境,与多处理机系统有较大差异,因此并行调度方法并不直接适用于NOW。

本文提出了NOW中一种支持进程迁移的运行时增量式集中式调度方法NRICS_SPM(Runtime Incremental Concentrated Scheduling Supporting Process Migration on NOW),该方法综合了静态调度、动态调度、并行调度、进程迁移等的优点,并以运行时调度、增量式调度、集中式负载收集和调度、结点间并行式负载交互、支持结点间任务和进程迁移等为核心。文末给出了NRICS_SPM的部分实验结果及与相关工作的比较,表明NRICS_SPM具有良好的并行计算和负载平衡性能。

1. 一种结点间进程迁移方法CSR_PM及其时间开销分析

根据NOW中结点间进程迁移的特点,我们实现了一种基于检测点保存和恢复的进程迁移方法CSR_PM(a Process Migration of Checkpoint Saving and Recovering)。CSR_PM由四部分组成:进程迁出函数,进程迁入函数,结点进程迁移维护进程及系统进程迁移管理进程。

进程迁出函数的流程为:(1)保存断点;(2)将进程当前工作目录、程序计数器、处理机状态字、打开文件信息(包括打开文件的读写指针)、中断及信号信息、数据段和堆栈段的起始指针及内容等写入到一个检测点文件Checkpoint_File中;(3)进程终止。为了在完成正常的文件打开、关闭、复制等操作的同时记录这些文件的打开状况及相关信息,以便在进程迁入时予以恢复,

*)国防科技九五预研项目资助。陈锡明　博士生,主要研究领域为:分布式系统与并行处理,操作系统。卢显良　教授,博士导师,主要研究领域为操作系统,分布式系统与并行处理。

我们用系统调用替换法对 open()、close()、dup()等系统调用进行了修改和替换。

进程迁入函数的功能为：(1)从 Checkpoint_File 中读入进程迁出函数中保存的内容并予以恢复；(2)将迁入进程的 I/O 操作重定向到其源结点上；(3)从断点处恢复迁入进程继续执行。

为了确保进程迁移后进程间通信的正确执行，CSR_PM 要求所在的并行环境必须能为每个结点的每个进程在系统全局空间内赋予一个唯一的一维或二维编号，CSR_PM 在每个结点的进程迁移维护进程中维持一张进程迁移前后全局编号对照表，通过在具体的通信操作前查询这张表的方式来保证进程迁移后进程间通信的正确进行。

结点进程迁移维护进程为每结点一个，它根据系统进程迁移管理进程的指示进行下列操作：(1)若指示为迁移一个进程至目标结点，则向被迁移进程发送信号、使其调用进程迁出函数，实施进程迁出；(2)若指示为从源结点迁移一个进程至本结点，则 fork()一个新进程，使其调用进程迁入函数重新构造迁移进程的运行状态，恢复迁移进程的继续执行，并协助进程迁入函数完成迁移进程的 I/O 重定向操作；(3)若指示为更新进程迁移前后全局编号对照表，则实施相应操作。

系统进程迁移管理进程为整个系统一个，其作用是：接受系统负载平衡机构的指示，向各结点的进程迁移维护进程发出进程迁出或迁入指示，同时记录进程迁移前后其全局编号的变化情况，发送给各结点的进程迁移维护进程。

CSR_PM 进程迁移方法的时间开销 T_{pm_total} 主要包括：(1)进程迁出函数时间开销 T_{pm_out}；(2)进程迁入函数时间开销 T_{pm_in}；(3)迁移进程执行文件与 Checkpoint_File 在源与目标结点之间传送的时间开销 T_{file_tr}；(4)等待目标结点允许进程迁入的时间开销 T_{wait_pm}。即：

$$T_{pm\_total} \approx T_{pm\_out} + T_{pm\_in} + T_{file\_tr} + T_{wait\_pm} \quad (1)$$

在磁盘的读写速度明显高于网络传送速度的慢速网络中，T_{file_tr} 是构成 T_{pm_total} 的主要因素；在网络传送速度高于磁盘读写速度的快速网络中，T_{pm_total} 综合决定于以上各要素。

2. NRICS_SPM 的实现方法

NRICS_SPM 是 NOW 中一种综合了静态调度、动态调度、并行调度、结点间任务和进程迁移等的优点的任务调度和负载平衡方法，具体用运行时调度、增量式调度、集中式调度等三种调度方式加以实现。

NRICS_SPM 的运行时调度是指系统宏观上在执行用户程序的同时进行任务调度和负载平衡。增量式调度是指将整个 NOW 从执行时间上明确地划分为两个交替循环的阶段，即系统调度阶段 SSP(System Scheduling Phase)和系统执行阶段 SEP(System Executing Phase)。SSP 负责系统范围内任务的调度和全局负载平衡，SEP 将调度好的任务投入运行，系统初启于 SSP。同时，将每个结点从执行时间上也明确地划分为两个交替循环的阶段：局部调度阶段 LSP(Local Scheduling Phase)和局部执行阶段 LEP(Local Executing Phase)。LSP 按照 SSP 的指示和其他结点交换负载以使其拥有的任务数达到其定额(SSP 中为达到全局负载平衡每个结点应当承担的任务数)；LEP 则将分配至该结点上的任务投入运行，系统调度阶段和每个结点的局部调度阶段如图1、2所示。

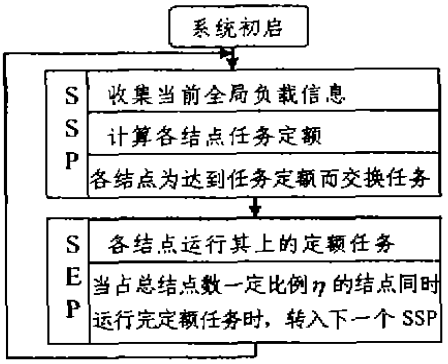

图1 NRICS_SPM 全局阶段转换图

NRICS_SPM 中所有任务只有经过调度才能投入运行，每个结点上的局部转换策略用两个队列：可调度任务 ST(Schedulable-Tasks)队列和执行任务 ET(Executing-Tasks)队列来实现。ST 队列保存用户提交的任务，这些任务虽已被系统接受，但要等到 SSP 进行统一调度并放入相应结点(可能是本结点，也可能被迁移至其他结点)的 ET 队列中之后才能投入执行；ET 队列存储经过 SSP 调度之后本结点上可投入运行的任务。经过 SSP 的集中式调度，各结点 ET 队列中的任务即为了达到全局负载平衡该结点中分配到的定额任务(见后面算法1)，在 LEP 这些任务将被取出来投入执行。在任一时刻，各结点上均可能有新任务产生(若为并行任务，则将按照其并行度被划分成若干个子任务)，这些任务将被放入提交结点的 ST 队列；与此同时，各结点 ET 队列中的任务则在不断地投入执行，当一个结点 ET 队列中所有任务均执行完时，该结点就具备了从 LEP 向 LSP 转换的局部转换条件。当符合一定比例数 $\eta \in [0,1]$ 的结点同时具备了上述局部转换条件时，master 结点便将整个系统从 SEP 转换至

SSP，并指示其余不具备上述局部转换条件的结点从LEP转换至LSP。此时，具备上述局部转换条件的结点ET队列为空，其余结点ET队列则不为空。这是因为尽管各结点所分得的任务定额基本相同(见算法1)，但由于各任务粒度及各结点计算能力不同，任务执行速率也不同，这一负载失衡情形将在下一个SSP中通过将新提交的和各结点ET队列中遗留的任务统一进行调度的方法来得到进一步改进和纠正，这正是NRICS_SPM增量式调度的精髓。

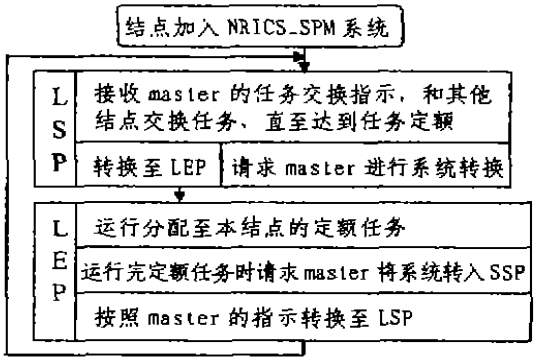

图2 NRICS_SPM局部阶段转换图

NRICS_SPM的集中式调度不同于纯粹动态调度中的集中式调度，而是根据NOW中并行处理和负载平衡的特点，充分利用并行调度[3]和动态调度中的集中式调度两者的优点而形成的一种调度方法，具体体现为集中式负载收集和平衡、并行式负载交互，支持结点间任务和进程迁移等。NRICS_SPM中，整个系统中设置一个master结点来完成全局负载收集和全局负载平衡计算工作(类似于动态调度中集中式调度)，而具体的任务交换则由其他各结点按照master的指示并行地进行(类似于并行调度中的并行任务交换)。通过这种方法，可准确地掌握全局负载信息、精确地进行全局负载平衡，并可详细地记录每个结点的任务和进程迁移情况，这对NOW并行处理中必须解决的I/O重定向来说是极其重要的。同时，通过设置一个master结点，可管理全局范围内结点及并行任务的加入、退出或由于结点中途失效而造成的并行任务异常终止等。NRICS_SPM中，对每个并行或顺序任务赋予一个全局唯一的网络标志号npid，对每个并行子任务赋予一个所在并行任务范围内唯一的子任务号pstn，在master结点的协助下可方便地实现同一并行任务的各子任务之间的同步和通信等操作。NRICS_SPM的集中式调度算法主要完成：全局负载信息收集，各结点任务定额计算，为使各结点达到任务定额而须进行的结点间任务(包括未实际投入执行的任务和已投入执行而形成进程的任务，除非特别指明"任务"或"进程"，下同)交换情况计算，各结点为达到其任务定额而并行地进行结点间任务和进程迁移等，详见算法1。

算法1 支持进程迁移的NRICS_SPM集中式调度算法

说明：NOW中各结点按照其综合处理能力(CPU速度，内存大小等的一个单调增函数f(cpu，mem))大小进行编号，编号方式为：设当前NOW中共有$n+1$个结点，master不执行用户提交的任务，编号为N_0；其余结点按照其综合处理能力从大到小的顺序依次编号为$N_1,N_2,\cdots,N_n$。

Step 1：master向各结点$N_i(i=1,2,\cdots,n)$发送负载收集消息，收集N_i的ST队列中新提交的任务数S_i和其ET队列中未执行完的任务数E_i以及两者之和$w_i=S_i+E_i$，并用$W_0=\sum_{i=1}^{n}w_i$表示各结点任务总数。

Step 2：master计算各结点$N_i(i=1,2,\cdots,n)$的任务定额q_i，令$w_{avg}=\lfloor W_0/n\rfloor$，$R=W_0 \bmod n$，并令$q_i=\begin{cases}W_{avg}+1 & \text{若 } i\leqslant R,\\ w_{avg} & \text{若 } i>R\end{cases}$，称$q_i$为结点$N_i$的任务定额。显然有：$\sum_{i=1}^{n}q_i=W_0$。

Step 3：master根据N_i的w_i和q_i计算为使各结点$N_i(i=1,2,\cdots,n)$的任务数都达到q_i而应进行的结点间任务交换情况。引入数组$\delta_1,\delta_2,\cdots,\delta_n$，初始时$\delta_k=w_k(k=1,2,\cdots,n)$，计算方法为：

```
for(i=1;i<n;i++){
  for(j=i+1;j<=n;j++){
    if((δi-qi)×(δj-qj)<0){
      结点Ni与Nj可以进行任务交换，用(Ni,Nj)表示Ni可与Nj交换的任务个数(初始为0)，
      (Ni,Nj)>0表示可将Ni中的(Ni,Nj)个任务迁移至Nj；(Ni,Nj)<0表示Ni将等待接收从
      Nj迁移来的|(Ni,Nj)|个任务；用(Nj,Ni)表示对结点Nj而言可与Ni进行交换的任务个
      数，含义与(Ni,Nj)相似。令：
        (Ni,Nj)=sgn(δi-qi)*min(|δi-qi|,|δj-qj|);(Ni,Nj)=-(Nj,Ni);
      其中sgn(x)：x>0时sgn(x)=1，x<0时sgn(x)=-1，x=0时sgn(x)=0
      if((δi-qi)>0){δi=δi-(Ni,Nj),
        δj=δj+(Ni,Nj)}
      else{δi=δi+|(Ni,Nj)|,δj=δj-|(Ni,Nj)|}
    }}}
```

Step 4：master将Step3计算出的各结点$N_i(i=1,2,\cdots,n)$应与其他结点进行任务交换的信息以元组

$[(N_i,N_j),N_j](j=1,2,\cdots,n,j\neq i)$的形式同 q_i 一起封装在一个消息中传送给各结点 N_i。

Step 5：各结点 $N_i(i=1,2,\cdots,n)$接收到 master 传送来的本结点应与其他结点进行任务交换的消息后，从中分解出所有元组$[(N_i,N_j),N_j](j=1,2,\cdots,n,j\neq i)$及 q_i，从 LEP 转换至 LSP，在 LSP 中执行：

将本结点 N_i 的 ST 队列首部中的 $\theta_i=\max(0,\min(\omega_i,q_i-E_i))$个任务迁移到本结点 ET 队列的尾部，令 $S_i=S_i-\theta_i,E_i=E_i+\theta_i$；

```
for(j=1;j<=n;j++){
  if (j==i||(N_i,N_j)==0)continue;
  if ((N_i,N_j)>S_i && S_i>0){
  将结点 N_i 的 ST 队列首部的 S_i 个任务迁移到
  结点 N_j 的 ET 队列的尾部；
  *对 N_i 的 ET 队列尾部的(N_i,N_j)-S_i 个任务
  中的每个任务，若其尚未实际投入运行，则将该
  任务迁移至 N_j 的 ET 队列的尾部(任务迁移时
  要进行 I/O 重定向等操作，较进程迁移简单，
  在此不做赘述)；若其已经投入运行而形成进
  程，则采用 CSR-PM 进程迁移方法，向 N_i 发出
  进程迁出消息，向 N_j 发出进程迁入消息，将该
  进程从 N_i 迁移至结点 N_j，并将对应任务放入
  N_j 的 ET 队列尾部；
  S_i=0; }
  else if((N_i,N_j)>0)
  对结点 N_i 的 ET 队列尾部的(N_i,N_j)个任务
  中的每个任务，处理方法与 Step5中的*相似，
  即按照该任务是否实际投入运行而实施任务
  或进程迁移至结点 N_j；}
  else{等待结点 N_j 将其 ST 队列首部或 ET 队
       列尾部的|(N_i,N_j)|个任务或进程迁移至
       本结点 N_i，并将其放入本结点 ET 队列尾
       部；}
  本结点 N_i 从 LSP 转换至 LEP，向 master 发送
  将系统从 SSP 转换至 SEP 的请求消息；
```

Step 6：master 接收到各结点发送来的将系统从 SSP 转换至 SEP 的请求后将系统转换至 SEP；

文[4]对算法1的性能进行了详细分析，并证明：经过算法1的 Step3，各结点 $N_i(i=1,2,\cdots,n)$的 δ_i 值将达到其定额值 q_i。根据算法1，各结点 $N_i(i=1,2,\cdots,n)$的任务定额值 q_i 最多相差1，故算法1使 NRICS-SPM 在每个 SSP 后能够获得良好的全局负载平衡性。同时，由于使结点之间按照 Step5直接进行任务或进程交换，不需中间结点转发，故算法1使一个 SSP 内为了达到全局负载平衡而进行的任务迁移数最少，任务局部性最好。

3. 部分实验结果及与相关工作的比较

文[4]对 NRICS-SPM 的实现进行了详细描述，我们利用 NRICS-SPM 进行了大规模矩阵相乘运算，排序，群集专家系统，14、15皇后问题求解等多项实验，均表现出了良好的性能。表1 列出了用 NRICS-SPM 求解14皇后问题的实验结果及与其他相关系统的比较结果，实验环境为100M ATM 网络，提交结点输出重定向到一个文件中，执行时间单位为：秒。

表1 NRICS-SPM 系统在求解14皇后问题时的实验结果及与其他相关系统的比较结果

结点配置	节点数	具备从 LSP 向 LEP 转换的结点数当占总结点数的比例 η	串行执行时间	NRICS-SPM 加速比	NOW 中不支持进程迁移的运行时增量式集中式调度加速比	PVM 加速比	GLUnix 加速比	RID 加速比
联想 PⅡ450	16	η=1.0	329.6	10.18	9.91	9.99	8.07	10.03
		η=0.5		10.34	9.92			
		η=1/16		10.47	10.26			
联想 PⅡ450	8	η=0.5	329.6/	7.16/	6.68/	6.07/	5.47/	6.16/
联想 PⅠ266	8		577.4/	12.02/	11.33/	9.62/	9.30/	9.75/
联想 P5/166	8		1270	25.4	23.54	20.22	19.91	20.8
Sun Sparc60	5	η=0.5	1023/	4.99/	4.78/	4.65/	不能运行	4.52/
Sun Sparc5	5		2932	14.8	13.57	13.21		13.05

说明：1)表中"1023/2932"表示14皇后程序分别在 Sun Sparc60和 Sun Sparc5上的串行执行时间，"4.99/14.8"表示在 Sun Sparc60和 Sparc5各5台组成的 NOW 中分别在 Sparc60和 Sparc5上提交14皇后程序时测得的串/并行执行时间之比；其余相似表示含义与此相似；2)RID 加速比为采用 RID(Receiver-Initiated Diffusion)[5]方法时测得的加速比；3)GLUnix 为一个基于 NOW 的并行计算环境[6]。

从表1可以看出，NRICS_SPM 在14皇后问题求解中表现出良好的并行计算和负载平衡性能，总体效率优于 NOW 中不支持进程迁移的运行时增量式集中式调度、PVM、RID 及 GLUnix 等。

NRICS_SPM 是结合了静态调度、动态调度、并行调度、进程迁移等的优点而形成的一种运行时增量式集中式调度方法。它与静态调度的不同之处在于：1) NRICS_SPM 是一种运行时调度方法，可以处理动态问题，静态调度只能处理静态问题；2)由于采用增量式调度，NRICS_SPM 中一个 SEP 中由于各任务粒度和各结点处理能力不同而造成的结点间负载失衡可以在下一个 SSP 中得到改进和纠正；而静态调度中负载一经调度，就无法重新分配，也就无法消除上述负载失衡。

NRICS_SPM 与动态调度的调度决定都依赖和自适应于运行时系统信息。然而，在动态调度中，任务的调度和执行是混合在一起进行的，必然造成对 CPU、网络等资源的竞争，使结点之间不能及时、准确地进行负载信息和负载的交互；NRICS_SPM 中任务的调度和执行被清晰地分开在 SSP 和 SEP 两个交替循环的阶段中，相互之间互不影响，可更加精确地进行全局负载平衡。

NRICS_SPM 借鉴了文[3]中并行调度的部分思想，使所有结点在 master 的指导下并行地交换负载。但 NOW 与文[3]中的调度环境—多处理机系统在构成方式和特点上有较大区别，NRICS_SPM 中要进行比文[3]的并行调度更为复杂的结点管理、任务管理、进程管理以及 I/O 重定向等操作。NRICS_SPM 中各结点负载信息的收集及根据总任务数和结点数计算各结点任务定额、并进而计算为达到全局负载平衡结点之间具体任务交换情况等功能由 master 结点统一完成，并据此指导其他各结点并行地进行任务或进程迁移。文[6]的研究表明，在 NOW 中设置一个 master 结点作为全局调度和控制的中心，通过对打开文件个数限制做适当的修改等方法，可使该结点不构成整个系统的瓶颈，文[6]同时介绍了一种基于 NOW 的并行计算环境 GLUnix，与 NRICS_SPM 一样，它采用一个 master 结点作全局控制和负载平衡的中心，并指出即使在96个结点的 NOW 中该结点依然不构成整个系统的瓶颈，系统的伸缩性较强。但 GLUnix 采用实时负载收集和任务一次性分配方法，无增量式调度能力，且当空闲结点数小于并行度时无法提交和执行一个并行任务，表1的实验结果表明其总体效率低于 NRICS_SPM。

PVM3.3版以前没有提供负载平衡措施，PVM3.3以后版本中提供了一个资源管理器(RM)来进行负载平衡，该 RM 采用 Round-Robin 轮转法，每到来一个任务就根据各结点当前 PVM 任务数多少将该任务放到任务数量少的结点上执行，系一次性任务分配机制，若任务分配目的结点计算能力较差，也无 NRICS_SPM 增量式调度那样的纠正能力。这对于任务间粒度和结点间执行能力差异较大的系统，将导致结点间严重的负载不平衡，表1的实验结果表明了这一点。

鸣谢 本文特别感谢纽约州立大学计算机系 Wu Min-You 教授，北京大学计算机系系主任李晓明教授，上海华东计算技术研究所陈涵生研究员。Wu Min-You 教授帮助作者确立了研究方向，并提供了不少具体的研究建议，李晓明教授对本文进行了详细的修改，陈涵生研究员仔细地审查了本文。有兴趣的读者可以通过 email：chenxm@mei29.scgb.com 与作者联系，作者可以给出 NRICS_SPM 的更详细的实现方案。

参考文献

1 Anderson T E, Culler D E, et al. A Case for NOW(Network of Workstations). IEEE Micro, Feb 1994

2 裴丹，汪东升，沈美明. 工作站网络系统进程迁移机制. 软件学报，1999，10(10)：1032～1037

3 Wu Min-You. On Runtime Parallel Scheduling for Processor Load Balancing IEEE Transactions On Parallel And Distributed Systems，1997，8(2)，173～185

4 陈锡明. 基于 NOW 的负载平衡机制研究：[电子科技大学博士学位论文]. 2000

5 Willebeek-LeMair M，et al. Strategies for Dynamic Load Balancing on Highly Parallel Computers. IEEE Trans. Parallel and Distributed Systems，1993，4(9)：979～993

6 Ghormley D P，et al. GLUnix，A Global Layer Unix for NOW. Available at：http：// now. cs. berkeley. edu/Glunix/glunix. html