一个动态自适应的迁移和协同调度模型*

陆桑璐 谢 立

(南京大学计算机科学系 南京 210093)

摘要 本文基于工作站网络NOW(networks of workstations)的特性,提出一个自适应可伸缩的进程迁移和协同调度模型DASIC(dynamic adaptive scalable process migration and coscheduling),试图解决其动态自适应要求越来越高的问题.DASIC模型在系统协同调度的基础上增加动态自适应的可伸缩功能,使其能够动态适应NOW环境下的变化,进行可伸缩的调度和负载平衡调节.在保证工作站用户独占特性的同时,提高了整个系统资源的利用率,为当前NOW环境中系统资源充分利用的研究提供了有效的模型.

关键词 自适应,协同调度,负载平衡,迁移,可伸缩性.

中图法分类号 TP311

由于工作站性能/价格比的绝对优势以及网络性能的不断完善,当前工作站网络NOW(networks of workstations)受到了越来越广泛的重视,成为并行计算领域的一个发展方向.在NOW环境中,一个关键问题就在于用户如何有效地利用整个网络的系统资源.为了使工作站用户们能够共享这些动态变化的资源,相应的协调管理机制必须被开发.对于其中最为重要的处理机资源而言,一方面应该提供并行或大型应用项目大规模使用整个网络处理机的能力,另一方面还要使那些希望独占工作站的用户们避免过多的干扰.

本文针对NOW环境下处理机管理的这些要求,提出一个动态自适应的进程迁移和协同调度模型DASIC(dynamic adaptive scalable process migration and coscheduling),在NOW环境下一方面对并行或大型应用项目的交互进程进行协同调度,另一方面根据系统的动态变化,自适应地进行伸展(Growing)、收缩(Shrinking)和负载平衡(Load Balancing)调整.

1 协同调度

让我们先来回顾一下Ousterhout1982年就已提出的协同调度(Coscheduling)概念.[1]由于进程间交互通信的广泛化,对于独立进程的假设往往会引发进程颠簸(Process Thrash-

---

* 本文研究得到国家863高科技项目、国家基础研究攀登计划和国家教委博士点基金资助.作者**陆桑璐**,女,1970年生,博士生,主要研究领域为分布式计算,并行处理.**谢立**,1942年生,教授,博士导师,主要研究领域为分布式计算,并行处理.

本文通讯联系人:陆桑璐,南京210093,南京大学计算机科学系

本文1996-11-19收到修改稿

© 中国科学院软件研究所 http://www.jos.org.cn

ing),即当挂起一个进程,调度另一个进程运行时,可能被挂起的进程正是被调度的进程很快就要进行通信服务的进程.这种情况类似于内存管理中页面颠簸的情况.协同调度针对这一问题,基于并行应用各进程间需要通信交互的特性,尽可能让同一组进程在不同处理机上同时调度运行,以保证进程在需要通信时总能处于调度状态,从而避免进程颠簸的发生,并且减少了由于进程被挂起的环境转换次数,进而减轻操作系统开销.

Ousterhout 在文献[1]中讨论的 3 种算法中,Matrix 算法的思想基于二维进程空间的考虑,将进程空间看作二维矩阵 $P*Q$. P 指系统所具有的处理机数,Q 指每一台处理机上能够拥有的进程数.这样,进程空间即包括 $P*Q$ 个可供进程分配的进程点(在本文稍后部分对 Matrix 算法还有具体叙述).而其他两个算法的思想均基于一维进程空间的考虑,将进程空间看作一个线性序列.由于 Matrix 算法直观、实现简单,且性能不很低于其他两个算法,整体效率最高,在 DASIC 的设计中,也采用二维矩阵方法.

Ousterhout[1]在对协同调度的实现中,系统处理机数目固定不变,P 为定值,不允许处理机数的动态变化.同时对任务组的大小进行限制,不允许一个任务组中的进程数超过 P,也不考虑经过一段时间的运行后,由于任务组的动态变化而造成的分配点不规整、碎片增多的情况.并且,Ousterhout 还限制进程的动态迁移.这样的协同调度对于 NOW 显然是不能满足要求的.

2 DASIC 的模型设计

2.1 设计思想

对于一个用户来说,他独占使用本地工作站的同时,可能由于负载较重而希望借助网络范围内其他"空闲"工作站的系统资源进行扩展.而在一个 NOW 环境中,大部分工作站在大部分时间都处于低利用率的状态.据 Berkeley 实验显示[2],每天即使在下午最忙的时间,也有 60%的工作站处于空闲可利用状态(由于不同系统对"空闲"定义的不同,得到的百分比也各不相同,但对于系统中任何时刻都存在可利用的空闲结点的结论都是一致的,参见文献[3,4]).但这些空闲工作站是一个动态的集合,成员可能不断地变化.这是因为对于一个工作站用户来说,从分时系统中解放出来使用工作站的一个主要目的就是可以随时独占一台工作站的所有资源而不受其他用户的打扰,因此,当用户扩展地使用其他空闲工作站资源的同时,还应考虑当这些工作站的用户返回使用时,这些工作站应该退出空闲工作站集,而其上正在运行的外来进程也应该立即撤出(Eviction),以避免降低双方的性能.

本地工作站为了利用网络中这些闲置的工作站来辅助完成负载,必须进行自适应地协调管理,这也正是 DASIC 的目标.

早期的研究已指出,对于频繁通信的进程,局部调度将导致执行时间不可接受.加州 Berkeley 分校的 R. Arapaci 等人在 NOW 环境下对局部调度和协同调度进行了实验比较,其结果表明,当系统资源被两个并行应用共享时,局部调度使每个应用都减慢了,并且随着参与竞争的并行应用数目的增多,减慢的程度越为严重.而协同调度比局部调度具有更好的性能,并且保证其性能可达到一定的可接受程度.因此,在 NOW 环境下,协同调度的重要性和必要性依然存在.

© 中国科学院软件研究所 http://www.jos.org.cn

因此,在NOW环境下DASIC一方面采用协同调度以避免进程颠簸的发生,另一方面进行伸展、收缩和负载平衡的调整以自适应于系统的动态变化.

下面主要针对DASIC中自适应可伸缩方面的设计进行介绍,对于"空闲"工作站的搜集机制将暂不予讨论.

2.2 DASIC模型设计

如图1所示,DASIC包括3大部分:伸缩调度(Scalable)模块、负载平衡(Load Balancing)模块以及协同调度(Coscheduling)模块.

图1 DASIC的模块设计

协同调度模块,对Ousterhout的Matrix方法进行改进,应用到DASIC系统中,对并行或大型应用项目的交互进程进行协同调度,包括Matrix算法和替补进程选择(Alternate Selection)两个小模块;伸缩调度模块,包括伸展(Growing)和收缩(Shrinking)调度两个部分,主要用于提供当系统中增加或减少了"空闲"处理机时的相应处理;负载平衡模块,包括填充(Filling)和压缩(Packing)两个部分,主要用于平衡系统中的负载,尽可能保证最大程度的并行.

下面将提出DASIC模型的具体设计和实现,首先引入DASIC的几个主要数据结构,然后依次介绍上述模块的具体设计和实现.

3 DASIC算法的设计和实现

3.1 数据结构

对应于文献[1]中Matrix算法的P,在DASIC中给出相应的P'.P'为变量,指示可利用的动态处理机数目.同时,DASIC还引入一个指针Q',用于指示相应于处理机的动态变化系统自适应调整的分时片数.在系统运行过程中,可能由于空闲结点的动态加入、退出,而使P'从P_1变化到P_2,也可能由于某些进程的完成而使Q'从Q_1变化到Q_2.下面介绍DASIC中的3个主要数据结构.

(1)空闲度λ

模型中引入了空闲度λ的概念.空闲度λ由进程空间($P'*Q'$)中空闲点与总分配点的百分比计算得到

$$\lambda=\frac{\text{idle slots in process space}}{P'*Q'}$$

空闲度的引入是为了显示系统资源的利用程度和进程空间分配的密集程度,由此指示DASIC是否需要对系统的分配调度进行自适应负载平衡调整.为了避免对进程空间过多地调整变化而引起系统较大的开销,只有当λ达到一定的百分比时(可以考虑为30%),DASIC才进行系统调整.

(2) 分配调度矩阵

© 中国科学院软件研究所 http://www.jos.org.cn

DASIC 也有一个可变的进程空间,如图 2 所示.

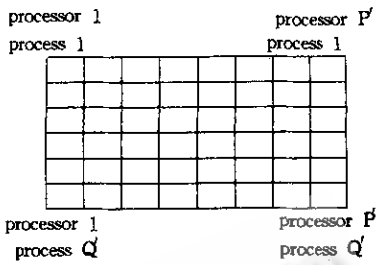

图2 DASIC的可变进程空间

基于该进程空间,DASIC 产生一个分配调度矩阵 $[P_{ij}]$(一个包含了分配和调度信息的矩阵).第 j 列指示第 j 个处理机($1\leqslant j\leqslant P'$)上分配到的进程(分配信息),第 i 行指示第 i 个时间片时($1\leqslant i\leqslant Q'$)各个处理机被调度的进程(调度信息),由此记录系统的分配调度状况.

由分配调度矩阵的行和列,DASIC 分别累计生成 2 个向量:分配向量 *allocation_vector* 和调度向量 *scheduling_vector*.分配向量记录各处理机上进程的分配情况,由各行累计而得,第 j 个元素的值指示第 j 个处理机上分配到的进程数,其中 $1\leqslant j\leqslant P'$.而调度向量记录每个时间片时进程在系统中处理机上被调度情况,由各列累计而得,第 i 个元素指示第 i 个时间片时整个调度系统中有多少进程被同时调度,其中 $1\leqslant i\leqslant Q'$.

由调度向量我们再引入一个空闲向量 *idle_vector* 的概念.空闲向量相当于调度向量的逆向量,由等式　*idle_vector* $[i]=P'-$*scheduling_vector*$[i](1\leqslant i\leqslant Q')$ 计算而得.其中 P' 为调度系统中的处理机数.空闲向量指示矩阵中系统在各时间片(行)时的空闲情况.以下算法的设计大多基于这个数据结构.

(3) 进程组表(Process Team Table)

在 DASIC 算法中,分配调度的最小单位为进程,但协同调度的单位却是进程组(Process Team).一个进程组为系统中有通信交互关系的一组进程,需要进行协同调度.一个进程组可以是一个应用,也可由一个应用中有通信交互关系的几个进程单独组成,也即一个应用可分为一个或多个进程组,主要由其通信关系决定.

DASIC 用一个表结构对进程组进行记录,见表 1.

表 1　进程组数据结构

	name	*size*	*coscheduling status*	*allocating*
1	pt_name_1	$size_1$	0 或 n	$(R_1,R_2,\ldots)$
2	pt_name_2	$size_2$	0 或 n	$(R_1,R_2,\ldots)$
3	pt_name_3	$size_3$	0 或 n	$(R_1,R_2,\ldots)$
…	…	…	…	…

在进程组的数据结构中,"*name*"标识一个进程组;"*size*"指示进程组中所包含的进程数;"*coscheduling status*"显示该进程组是否处于协同调度状态,取值 0,$n(1\leqslant n\leqslant Q')$,0 指示处于协同调度状态,而 n 指示未协同调度的进程组的主行,也即包含该进程组最多进程的分配调度矩阵中的行;"*allocating*"记录该进程组的分配情况,其元素依次为所分配的行 $(R_1,R_2,\ldots)$,$1\leqslant R_1,R_2,\ldots\leqslant Q'$.

由此数据结构,DASIC 又引入一个进程组向量 *process_team_vector*,记录各进程组的大小情况.其元素的值或为 *size* 值(协同调度态),或为"*size*" *MOD* P' 值(非协同调度态).

3.2　算法实现

© 中国科学院软件研究所 http://www.jos.org.cn

(1) 初期调度

DASIC 采用 Ousterhout 提出的 Matrix 算法进行初期的调度，只有当某一进程组的"*size*"> P'，Matrix 算法不能处理时，DASIC 将其按大小分为两个子进程组，一个子进程组 1 包含 P' 个进程，另一个子进程组 2 包含("*size*"— P')个进程（若仍大于 P'，则可再作同样的处理）. 然后搜寻一全空的行 R_1(可能引起 $Q'=Q'+1$)分配给子进程组 1 独占，再对子进程组 2 作同样的分配处理.

这样的进程组在表项中只占据一栏，只是"*coscheduling status*"为 R_1，且"*allocating*"具有不止一个元素($R_1,R_2,\ldots$). 在进程组向量中的相应元素值应为"*size*" *MOD* P'.

系统开始运行后，由于空闲结点的加入和退出以及进程组的执行结束，可能造成进程空间的改变，这就需要进行自适应地可伸缩调度和负载平衡调整. 下面我们给出这些算法的具体设计和实现，其中伸展算法和收缩算法包含在可伸缩模块中，填充算法和压缩算法包含在负载平衡模块中，最后给出的替换选择算法属于协同调度模块. 这些算法的一个共同准则是尽可能维持协同调度.

(2) 伸展算法(Growing Algorithm)

当系统中增加了 S 台空闲工作站到处理机组中时，进程空间将扩充 S 列，这对于那些由于处理机不足而尚未协同调度的进程组将是一个协同调度的机会. DASIC 中伸展算法选择这样的进程组，将其不在主行的进程迁移到这 S 台空闲工作站上，以便于该进程组的协同调度：

```
repeat
    Search the process team table to find some process team T whose "coscheduling status"≠ 0;
    Compare process_team_vector(T) with S;
    If process_team_vector(T)≤ S
        Migrate the subteam of process team T to the additional idle processors of its key row
            so as to be coscheduled;
        Modify the corresponding data structures;
until the end of the process team table
```

(3) 收缩算法(Shrinking Algorithm)

当某台工作站宣布不再空闲时，其上的外来进程应该尽快撤出，以便于工作站的用户可以独占使用. 这样，这些工作站将离开系统中的处理机组，使得进程空间减少相应列. 在最初的考虑中，我们希望当撤出的进程属于一些协同调度进程组的成员时，应尽可能考虑保证它们的继续协同调度，也就是尽可能在同一行内寻找空闲点平行地进行迁移，以保证其所在进程组仍然可以在同一时间片内一起调度. 只有找不到这样的空闲点时，才迁移到其他具有较少进程的处理机上.

然而，对于收缩算法来说，首要的目标应该是快速迁移，如果算法过于复杂，可能大大降低系统的效率，因此 DASIC 当前采用的是一个简单的收缩算法，仅将撤出的进程尽快地迁移到具有较少进程的处理机上，只有当系统调整负载平衡时，再考虑协同调度. 因为每趟扫描结束，压缩算法都要进行一定的调整处理，因此其协同调度性能不会降低太多，而另一方面简单的收缩算法又保证了系统的效率.

```
repeat
    Look up allocation_vector to find the processor P which takes the least processes,
        namely allocation_vector(P) is the least;
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
        Migrate one of evicted processes to processor P;
    until all evicted processes migrated
```

(4) 填充算法(Filling Algorithm)

当一个进程组运行结束之后,进程空间中将增加若干空闲点.若该进程组是协同调度的,则增加同一行的空闲点,否则增加不同行(假设为 $R_1,R_2,\ldots,R_t$ 时间片,$1\leqslant R_1,R_2,\ldots,R_1\leqslant Q'$)的空闲点.为了重新利用这些空闲点,DASIC 提供了填充算法.当一个进程组的结束引起某一时间片(某一行)全空时,该行可以消去,将其下各行的优先级升高 1.否则检查是否当前有某一时间片上调度的进程数少于某 $R_i(1\leqslant i\leqslant t)$ 时间片上空闲点数,若存在,则迁移其上的进程到 R_i 上(对于同一台机器上的进程迁移仅仅是优先级的改变),以减少一行,提高并行效率.

```
for i = 1 to t do
    Calculate idle_vector(R_i), 1≤R_1,R_2,...,R_t≤Q' ;
    If idle_vector(R_i) = P'
      (the row R_i is completely empty, namely at time slice R_i all processor is in idle state,
      so row R_i can be eliminated.)
        Improve 1 to the priorities of all processes of following rows of R_i;
    else
        Compare scheduling_vector with idle_vector(R_i);
        If scheduling_vector(j)≤idle_vector(R_i), 1≤j≤Q', j≠R_i
            Migrate all processes of row j to those empty slots of row R_i, and eliminate row j;
              (Some processes may be "migrated" in the same colomn if there is
              empty slot in row R_i in the same colomn, it will only cause priority changing.)
end for
```

(5) 压缩算法(Packing Algorithm)

随着时间的推移,系统中的调度点可能越来越不规整,存在过多的空闲点,造成系统资源的不完全利用,为此,每次从时间片 0 到时间片 Q' 进行了一次调度扫描之后,DASIC 计算 λ 的值,当 $\lambda>30\%$ 时,需要进行压缩处理,以减小 Q',尽可能使调度向量 *scheduling_vector* 中各值接近 P',也即尽可能使调度分配矩阵紧凑,增加各处理机的并行度.类似于填充算法的处理,压缩算法也是尽可能让只有少量进程的时间片消去(将这些少量进程迁移到其他空闲点上).当然,这些算法都是以协同调度作为前提条件的.

```
Calculate λ;
If λ>30%
    Calculate idle_vector;
    /* Adjust to be coscheduling */
    repeat
      Search the process team table to find some process team T whose "coscheduling status"≠ 0
        and then read its key row value, suppose R;
      Compare process_team_vector(T) with idle_vector(R);
      If process_team_vector(T)≤ idle_vector(R)
        Migrate the subteam of process team T to the additional idle processors of its key row
            so as to be coscheduled;
        Modify the corresponding data structures;
    until the end of the process team table
    /* Packing process space */
    For row i, 1≤i≤Q', get size of each process team or subteam T_1,T_2,...,T_t in row i,
        and compare the size of T_j with idle_vector, 1≤j≤t;
    If the size of T_j≤ idle_vector(k^j) for all j in 1..t, 1≤k^j≤Q', i≠k^j
```

© 中国科学院软件研究所 http://www.jos.org.cn

Migrate process team T_j to corresponding row k^j respectively, and eliminate row i.
(The same as above, some processes may be "migrated" only in the same colomn.)
Repeat until all row are scanned;

(6) 替补进程的选择

由于系统的动态特性以及对协同调度的维持,进程空间的分配或多或少总会存在一些空闲点,对于这些空闲点的处理,DASIC 在前面已提出一些自适应调整的办法,但由于协同调度的要求,空闲点的存在是不可避免的.那么,在系统实际的调度运行中,这些空闲点的处理机周期只能白白浪费吗?为此,Ousterhout 提出一个替代运行的算法,让各处理机从其上(同一列)的进程中任意选择一个替补进程代替空闲点调度运行,以避免空闲点的浪费.对于空闲点或被堵塞的进程,DASIC 中同样采用替补进程的选择方法,尽可能选择同一列中的进程顶替空闲点调度运行.采用各机独立选择替补进程的方法,虽然可能造成协同调度的不充分,但较为简单,避免集中选择可能引起的较大开销.

4 相关工作及其比较

Shoch 和 Hupp 为开发分布并行计算而设计的 Worm 机制[5],以自适应的可伸缩性为迁移的设计目标,采用这样一个模式:一个程序或计算,可以不断从一个处理机移动到另一个处理机以利用需要的资源,必要时可以进行自身的复制,也可被删除.这种模式提供了对计算分布并行性的开发,对于系统资源的有效利用提供了较好的模式.但这个系统的"伸缩"只是作简单的复制和删除,对于扩展后的 Worm 也还缺乏有力的控制机制.

Nicholas 等人在文献[6]中描述了自适应并行和一个 Piranha 系统.Piranha 是一个自适应的并行 master-worker 结构,允许系统中空闲的处理机周期被重新捕获并运行并行应用.Piranha 系统基于共享的元组空间,由空闲结点驱动从元组空间中取任务运行.这在负载较重的系统中比较有效,对于 NOW 并不适用.并且当一个进程被追撤回时,它需要应用明确给出采取的步骤.

对于 Outsterhout 的协调调度算法,前面已作了详细的分析,此处不再赘述.

DASIC 不仅利用协同调度来对那些交互进程同时进行调度,而且利用自适应的进程迁移来满足系统的动态性,它能够对进程空间进行伸展、收缩和负载平衡的调节,自适应地满足 NOW 环境的需要.

5 结束语

当前,随着对 NOW 研究的不断发展,动态自适应的要求越来越高,从而对进程迁移的研究提出了新的挑战.本文据此提出一个自适应可伸缩的进程迁移和协同调度模型 DASIC,在系统协同调度的基础上增加动态自适应的可伸缩功能,使它能够动态适应工作站网络 NOW 环境下的变化,不仅提高了整个系统资源的利用率,而且保证了工作站用户的独占特性,为当前 NOW 环境中系统资源充分利用的研究提供了有效的模型.

参考文献

1 Ousterhout J. Scheduling techniques for concurrent systems. In: Proceedings of the 3rd International Conference

© 中国科学院软件研究所 http://www.jos.org.cn

on Distributed Computing Systems, 1982. 22～30.

2 Anderson T, Culler D, Patterson D *et al*. A case for NOW (networks of workstations). IEEE Micro, Feb. 1995. 54～64.

3 Douglis F, Ousterhout J. Transparent process migration: design alternatives and the sprite implementation. Software Practice and Experience, 1991, **21(8)**: 757～785.

4 Litzkow M, Livny M, Mutka M. Condor——a hunter of idle workstations. In: Proceedings of the 8th International Conference on Distributed Computing Systems, 1988. 104～111.

5 Shoch J, Hupp J. The "worm" programs——early experience with a distributed computation. Communications of the ACM, 1982, **25(3)**: 172～180.

6 Carriero N, Freeman E, Gelernter D *et al*. Adaptive parallelism and Piranha. Computer, Jan. 1995. 40～49.

A MODEL FOR ADAPTIVE MIGRATION AND COSCHEDULING

LU Sanglu　XIE Li

(*Department of Computer Science　Nanjing University　Nanjing*　210093)

Abstract　Based on the scalability on NOWs (networks of workstations), this paper proposes an adaptive migration and coscheduling model DASIC (dynamic adaptive scalable process migration and coscheduling). DASIC takes not only coscheduling to schedule interactive processes simultaneously, but also adaptive migrating to satisfy the system's dynamics and scalability. It can grow, shrink and load balance to adjust the processes scheduling status in the process space, and the processor pool can be expanded or shrinked.

Key words　Adaptive, coscheduling, load balancing, migration, scalable.

Class number　TP311

© 中国科学院软件研究所 http://www.jos.org.cn