维普资讯 http://www.cqvip.com

RIPS 调度策略研究

Research of RIPS Parallel Scheduling Policy

吴 思 陈莘萌
(武汉大学数学与计算机科学学院 武汉430072)

Abstract This paper proposes a parallel scheduling policy of distributed system. Runtime incremental parallel scheduling(in short of RIPS), which utllizes the parallel technique in task scheduling so that it combines the advantages and disadvantages of static policy and the dynamic policy, and discards the disadvantage of them

Keywords Distributed system, Parallel scheduling, Load balancing, Topology

1 前言

分布式系统中的任务调度策略可大致分为两类:静态策略和动态策略,所谓静态策略指的是在系统运行之前,根据整个系统的负载状况把用户提交的任务比较均匀地分配到各站点,在运行过程中各站点负责完成分配给它的任务,不再重新分配,这种方法的优点是实现简单,不足之处是系统事先难以准确掌握每个任务的运行时间,且合作任务间的通信和同步容易造成不确定的时间依赖,加之系统中各站点故障(进程故障)等因素,这种策略难以实现系统的效率最优。所谓动态策略是指把系统中各站点上已有的负载作为参考信息,在运行过程中根据系统中各站点的负载状况随时调整负载的分配,使各站点尽可能保持负载的平衡,所以动态任务调度策略又常被称为动态负载平衡策略,其优点是充分利用了各处理机的能力,不足之处是实现起来较为复杂。

本文讨论了一种分布式系统中的负载平衡策略RIPS(Runtime Incremental Parallel Scheduling)[1,2],就是通过在任务调度算法中引入并行技术,使其既具备了静态调度和动态调度的优越性,又能克服二者的不足。本文给出了RIPS的定义;介绍了增量调度的局部子策略和全局子策略,局部子策略的两种调度任务的方式:主动型和被动型;三种类型的全局子策略:All型、Anyone型、Half型;并简单介绍了并行调度的过程。

2 RIPS 的定义

RIPS是分布式系统中调度任务的一种策略,它将整个调度过程分为两个阶段:系统调度阶段和用户执行阶段(如图1)。一个RIPS开始于一个系统调度阶段,这个阶段执行一些初始化工作,调度初始化任务,第二个阶段是用户执行阶段,执行调度的任务,这个阶段有可能产生新的任务。当整个系统进入下一轮系统调度阶段时,上一轮调度没有执行完的任务与新产生的任务一同调度,二个阶段交替进行,直到全部的任务计算完成。

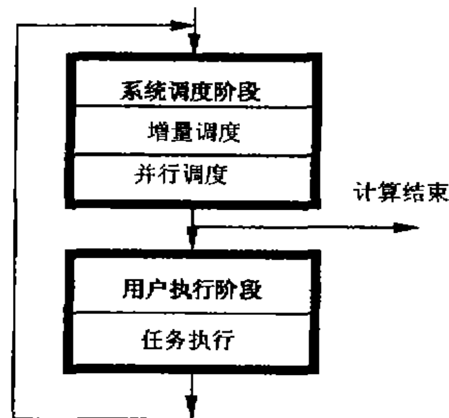

图1 RIPS示意图

系统调度阶段可进一步分为两个组成部分:增量调度和并行调度,增量的含义是指在两次相邻的并行调度之间,系统中可能有新的任务产生,所以在下一轮的并行调度是对发生了数量变化的任务集进行调度,增量调度决定系统什么时间开始下一轮并行调度,并行的含义是指整个系统并行地收集负载信息并完成任务的调度。

吴 思 博士生,研究方向为分布信息处理.陈莘萌 教授,博导,研究方向为分布信息处理.

维普资讯 http://www.cqvip.com

3 增量调度

一般来说，动态调度策略应有四个组成部分，转换策略、选择策略、定位策略、信息策略，RIPS 也有四个组成部分，只是含义有所区别：转换策略决定在什么时间开始下一轮并行调度；选择策略决定选择系统中哪些的任务进行调度；定位策略决定一个任务调度到哪个站点；信息策略决定何时收集信息。选择策略和转换策略是增量调度的核心，定位策略和信息策略则是并行调度的重要组成部分。

增量调度实际上是一个检测阶段，检测每个站点是否满足一定的局部条件，一旦整个系统满足了一定的全局条件，整个系统就要终止正在执行的任务，同步地进入下一轮并行调度，增量调度的转换策略应包括有两个子策略：局部子策略和全局子策略。

3.1 局部子策略

局部子策略是每个站点决定其是否准备好进入下一轮并行调度时所遵循的标准，本文选择每个站点是否完成了调度来的任务作为局部条件，完成了调度的任务的站点就处于准备好进入下一轮并行调度的状态，即 ready to be scheduled。由于在用户执行阶段可能有新的任务产生，如何调度、管理这些新产生的任务有两种方式：主动调度和被动调度，采用主动调度方式时，每个任务必须首先被调度才能被执行，那么在用户执行阶段新产生的任务，由于未经调度就不可能执行；采用被动调度方式时，新产生的任务未经调度也可以被执行。

3.1.1 *主动调度方式* 采用主动调度方式(Eager)时，在每个站点用两个队列来管理任务：执行队列(ready to execute，简称 RTE)，经调度后的任务进入这个队列，调度队列(read to schedule 简称 RTS)，新产生的未被调度的任务进入这个队列。如图2(a)所示。主动调度方式的过程如下：

系统初始化时，RTE、RES 均为空；

当系统第一次进入用户执行阶段时，系统中所有站点的 RTS 为空，而系统中所有站点的 RTE 中任务的数目就是系统中所有准备投入运行的任务数目。在用户计算阶段，RTE 中的任务被执行，如果产生新的任务则进入 RTS。

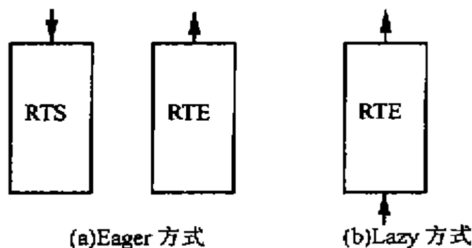

图2 局部子策略的两种队列

当一个站点 RTE 中的任务被全部执行后，即满足了局部条件，该站点就处于 ready to be scheduled，在下一轮的并行调度阶段，如果有些站点的 RTE 中有剩余的任务，则被转入 RTS 中，同新产生的任务一起调度。

3.1.2 *被动调度方式*(Lazy) 其实质是系统不区分调度来的任务和新产生的任务，每个站点只用一个队列，即 RTE，调度到本地的任务和用户执行阶段产生的新任务都进入 RTE，这样一来，RTE 中就可能有调度来的任务和新产生的任务，在系统转换到下一轮的并行调度之前就有一些在本地新产生的任务未经调度就被执行了。被动调度方式的过程如下：

系统初始化时，RTE 为空；

当系统第一次进入用户执行阶段时，系统中所有站点的 RTE 中任务的数目就是系统中所有准备投入运行的任务数目。在用户计算阶段，RTE 中的任务被执行，如果产生新的任务则进入 RTE。

当一个站点 RTE 中的任务被全部执行后，即满足了局部条件，该站点就处于准备好进入下一轮并行调度状态。

局部子策略的二种任务调度方式相应于二种队列策略，如表1所示。

表1 二种局部策略相应的二种队列策略

局部策略	队列策略	新任务
主动型	两个队列(RTS/RTE)	进入 RTS
被动型	一个队列(RTE)	进入 RTE

3.2 全局子策略

全局子策略是所有站点同步进入下一轮并行调度时所遵循的策略，即整个系统满足一个什么样的全局条件时，整个系统开始下一轮并行调度。我们给出三种全局条件，即三种类型的全局子策略：全部站点都满足局部条件，即全部站点的 RTE 都为空(简记 All 策略)；任一站点满足局部条件，即只须一个站点的 RTE 为空(简记 Anyone 策略)；半数站点满足局部条件，即半数站点的 RTE 为空(简记 Half 策略)。

3.2.1 All 型 对于 All 型全局子策略，为了检验整个系统是否满足全局条件，系统要周期性地进行检测，如果条件满足，系统就从用户执行阶段转换到下一轮并行调度；否则，继续用户执行阶段。检测周期是一个难以控制的变量，周期太短将增加检测过程所需的通信开销，周期太长又可能导致不必要的站点空闲，最佳的周期可在系统的运行过程中靠经验去摸索。本文采用了一种基于消息通信的方法，即由满足局部条件的站点发出 ready 消息，系统通过发出 ready 消息的站点数目来判断整个系统是否满足全局条件。

维普资讯 http://www.cqvip.com

如对于树状拓扑结构的(见图3),假定每个节点为一个站点,节点间的连线为通信线路,具体的方法如下:

一个节点只有当其自身满足局部条件(即RTE为空)且其收到所有的孩子节点发出的ready消息,它才向其父节点发送ready消息(对于叶节点,只要其自身满足局部条件就可以向父节点发出ready消息)。只有当一个系统的根节点满足局部条件且收到所有的孩子节点的ready消息时,该系统才满足全局条件,根节点向所有节点广播init消息,通知整个系统开始下一轮并行调度。

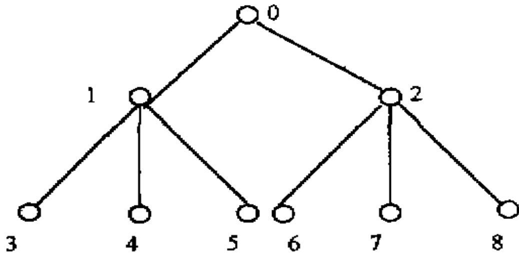

图3 树状拓扑结构的系统

在图3中,节点1只有当其自身满足局部条件且都已收到节点3、4、5发送的ready消息,才向其父节点发送ready消息;整个系统向并行调度转换的条件是根节点0满足局部条件且已收到节点1、2的ready消息。

又如对于环状拓扑结构的系统(如图4),假定环中的消息是顺时针方向前进的。具体的方法如下:

在每一节点设置一个表用于记录哪些节点已发出ready消息,表中有N个单元(R为环中节点数),每个单元对应一个节点,初始化时,每个单元置0,收到一个节点发出的ready消息后,置相应单元为1。

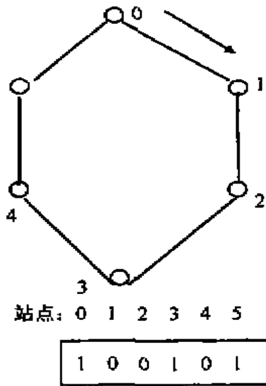

图4 环状拓扑结构的系统

当一个节点满足局部条件后,在本节点的表中相应单元置1,查看表中哪些节点还未发出ready消息,向这些节点发送一个ready消息,收到ready消息节点,读出发送ready消息的节点号,将表中相应位置1。

当系统中最后一个节点满足局部条件后,该节点就向环中所有节点发出init消息,通知它们节点进入并行调度。

例如,在某时间节点2满足了局部条件,同时将本节点表中的对应位置1,由于节点1、4未满足条件,向节点1、4发ready消息,节点1、4收到节点2的ready消息后,置其表中相应位为1。

3.2.2 Anyone型 对于Anyone型子策略,当任一站点满足局部条件时,也就同时满足了全局条件,该站点就可发出一条init消息给所有的站点,收到init消息的站点,从用户计算阶段转换到下一并行调度阶段。由于通信延时,可能会有多个站点发出了init消息,从而导致一个站点收到多个init消息,通过时间管理模块在每个init消息加上时间印,通过检测时间印,收到多个init消息的站点只响应处于同一阶段的最先收到的消息,其余均认为是多余的。

当一个站点收到init消息时,该站点可能正在运行任务,这时有三种方案,一是继续当前正在执行的任务直到完成,二是夭折该任务,三是中断正在运行的任务,将其运行状态、相关数据保存起来,并对该任务作标记,标记该任务正在运行,下次调度时,该任务同运行状态、相关数据一起调度,该任务将在新的站点上恢复运行状态,继续被中断的计算,这一点对粒度较大的任务非常有利。

3.2.3 Half型 是介于All型和Anyone型之间的一种策略,可令Half=[N/2]。一旦检测过程检测到系统中有大于、等于Half个节点处于ready状态时(有可能在两次检测之间有多个节点进入ready状态),就发出init消息,系统转换到并行调度阶段。(也可以使用与All型子策略类似的消息通信方式判断系统是否满足全局条件)

当并行调度任务的数目少于站点的数目时,接下来的用户计算阶段就是所谓低并行阶段。在这种情况下将会有两个问题:

第一个问题是可能有些站点上没有调度来任务,在Anyone策略下,那么它就要因为空闲而发出init消息,如果init消息到达的站点还未开始执行一个任务,这个站点将在下一并行调度阶段开始前不执行任何任务。在极端的情况下,系统可能循环地从用户计算阶段转换到并行调度阶段,而未执行任何操作,为了降低这种可能性,对Anyone型子策略,对站点发出init的资格要进行控制,即至少在并行调度阶段调度来一个任务的站点,才能是具有资格发init消息的。

第二问题是由被动型调度导致的对低并行状态缺乏感知,当采用Anyone-Lazy策略时,具备发出init资

维普资讯 http://www.cqvip.com

格的站不停地产生任务，而同时不具备资格的站点却处于饥饿状态，对 All-Lazy 策略，至少有一个任务的站点的执行可能导致其余的站点饥饿。为了解决这个问题，被动方式在低并行阶段应自动调整为主动方式，即通过采用主动方式提高在低并行阶段的并行性。

在 Half 策略下，当系统中的任务数少于站点数的一半时，也存在上述的两个问题，解决的方法是全局子策略自动调整为 All 型子策略，被动方式在低并行阶段应自动调整为主动方式，这样就可以保证所有的任务都能被执行。

图5的三个示意图分别反映了三种全局子策略从用户执行阶段转换为并行调度时各站点上任务的执行状况，其中 P1、P2、P3、P4、P5为五个站点，实线表示该站点上的任务执行完毕，虚线表示该站点上的任务未执行完毕，灰色阶段代表并行调度时间，其余的时间为用户计算时间。那么从下图中可以看到：

对于 All 型子策略（图5(a)），在第一个用户计算阶段，P4的任务首先计算完毕，满足了局部条件，处于 ready to be scheduled，但它要等到执行时间最长的 P1 最后计算完毕，整个系统满足了全局条件，系统开始下一轮并行调度。

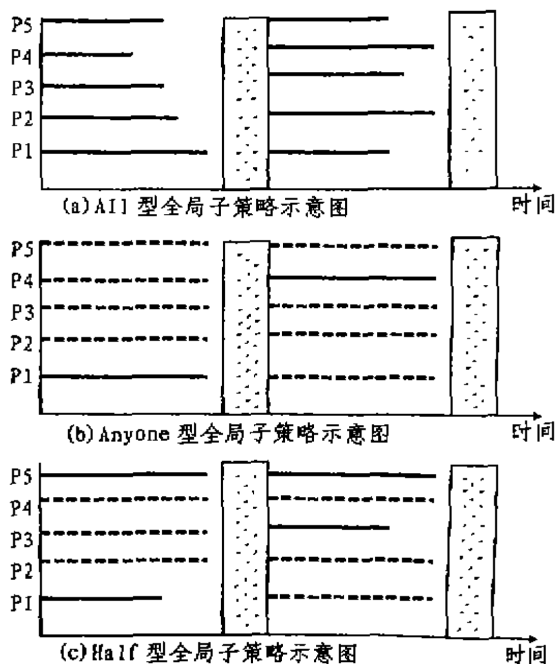

图5　三种类型全局策略比较

对于 Anyone 型子策略（图5(b)），在第一个用户计算阶段，P2上的任务首先计算完毕，即满足了全局条件，它就要向所有站点发送 init 消息，其他站点收到 init 消息后，终止正在执行的任务，（这里就假定其他站点收到 init 消息后夭折其正在运行的任务，下同），系统开始下一轮并行调度。

对于 Half 型子策略（图5(c)），Half＝[5/2]＝2，在第一个用户计算阶段，所以当 P1 和 P5 上的任务计算完毕后，即满足了全局条件，检测过程就要发出 init 消息，其他站点收到 init 消息后，终止正在执行的任务，系统开始下一轮并行调度。

4　并行调度

系统同步进入并行调度后，各站点的并行调度过程被激活，它们并行收集整个系统的负载状况，并以此为根据将整个系统中的全部任务并行、均衡地调度到各站点。并行调度是动态执行的，但它与动态调度是有区别的：

动态调度时站点间并发地交换信息和负载，当一些站点在运行用户程序时，其他的站点可能正执行调度算法；动态调度只收集一小部分区域的负载信息，不可能达到全局的平衡，因此调度结果可能是不稳定的，可能造成站点间的任务颠簸。

并行调度并行地执行调度算法，所有的站点间相互合作并行地收集负载信息，通过并行调度，可得到高度的负载平衡和并行规模，调度结果稳定。

不同的网络拓扑结构需要不同的调度算法，文[3]给出一个适合于环状结构的并行调度算法，文[4]给出一个适合于树状结构的并行调度算法。

参考文献

1　Wu M Y. On Runtime Parallel scheduling for Processor Load Balancing. IEEE Transactions on Parallel and Distributed Systems，1997，2(8)：173～186

2　Wei Shu，Wu Min-you. Runtime Incremental Parallel Scheduling on Distributed Memory Computers. IEEE Trans on Parallel an Distributed Systems，1996，6(7)：637～649

3　He Yan-xiang. Wu Si Zhongyong algorithm-A parallel Scheduling Algorithm Based on Ring Topology Distributed System. J Wuhan Univ(Nat Sci. Ed.)，1999，45(6)：279～282

4　Wu Si. Research of Runtimes Incremental Parallel Scheduling Policy in Distribute System[D]，Wuhan University：College of Computer Science an Technology，1998 (Ch)