松耦合多处理机系统工作池技术①

唐俊奇, 黄廖山

(湄洲湾职业技术学院 信息工程系, 莆田 351254)

摘 要: 分析了多处理机系统中的集中式、分散式和全分布式这三种工作池的动态平衡的工作原理, 通过仿真实验, 提出了利用工作池技术, 及时将工作池技术中待处理的工作任务分派给松耦合多处理机系统中处于空闲状态的处理器, 使松耦合多处理机系统(尤其是异构机群平台)中各机器能够有机地协调工作, 有效地提高海量数据的处理速度.

关键词: 海量数据; 多处理机系统; 工作池技术

Work Pool Technology of Loosely Coupled Multiprocessor System

TANG Jun-Qi, HUANG Liao-Shan

(Department of Information Engineering, Meizhouwan Vocational Technology College, Putian 351254, China)

Abstract: The paper analyzes the dynamic balance working principles of centralized work pool, decentralized work pool and distributed work pool in the multiprocessor system. Through the simulation experiment, it brings、forward a loosely coupled multiprocessor system using the work pool technology to assign the work tasks to the idle processors in a timely manner, which makes each machines of the loosely coupled multiprocessor system (especially for heterogeneous cluster platform) work coordinately. The speed of massive data processing is efficiently improved.

Key words: massive data; multiprocessor systems; work pool technology

引言

在智慧城市的建设中, 视频监控摄像头广泛应用于主要道路、热点地区、地铁和居民小区的安全监视. 1 个 8Mbps 摄像头产生的数据量是 3.6GB/小时, 1 个月为 2.59TB. 很多城市的摄像头多达几十万个, 1 个月产生的数据量是达到数百 PB, 若需要保存 3 个月则存储量达 EB 量级. 当前我们处理的数据量在急剧增长, 所以通常所用的并行数据库的规模也势必跟着不断增大, 这样就导致了企业成本的快速增长. 为了降低成本且能够满足现实的需求, 我们采用中低端硬件构成的机群平台[1](这种由中低端硬件构成的机群平台也称为松耦合多处理机系统)来替代高性能的服务器. 然而, 利用松耦合多处理机系统处理大数据的处理时, 经常出现系统中某些处理器一直处于繁忙状态, 而另外一些处理器一直处于空闲状态, 严重影响了系统的性能. 为了使松耦合多处理机系统(尤其是异构机群平台)中各机器能够有机地协调工作, 我们引入了工作池方式的负载平衡技术, 并作深入研究.

1 松耦合多处理机系统集中式动态负载平衡

在松耦合多处理机系统的集中式动态负载平衡中, 主进程(主处理器)拥有有要执行的一系列任务集(称为工作池[2])中的每个任务由主进程分发给各个从进程, 每个从进程完成一个任务之后, 它们都会向主进程请求另一个任务. 这种工作机制是工作池方法的本质. 所谓工作池方法就是:当整个处理系统中有一个处理器处于空闲时, 主进程就会向其分派工作, 工作池中拥有所要完成的任务集(池), 工作池技术虽然可以用于那些任务很不相同、大小不同的问题. 但是, 最好优先分配较大或最复杂的任务, 因为如果在计算过程

① 基金项目:福建省莆田市科技项目(2013G16);福建省教育厅 B 类科技项目(JB14195)
收稿时间:2015-06-10;收到修改稿时间:2015-10-22 [doi: 10.15888/j.cnki.csa.005133]

中, 大的任务分配较晚时, 那些已经完成小任务的从进程就会一直等待持有较大或最复杂的任务的进程完成任务.

在各个进程执行任务期间, 如果出现任务数经常发生变化的情况时, 在这种情况下采用工作池技术才会发挥它优势. 在海量数据的处理期间, 往往会出现这样一种情形:工作池中某一个任务在执行期间产生了另外一些新的任务. 这时的解决方法是: 如图 1 所示,把当前等待的任务存放在一个队列中. 如果工作池中所有任务大小一样而且优先级相同, 那么我们就可以采用先进先出队列这种简单的方法来实现; 如果工作池中某些任务比其他任务更重要(如优先级更高且期望更快地得到解), 这时就要优先把这些任务从工作池中送到从进程内. 其他的信息(如当前的最佳解等)可以用主进程加以保存.

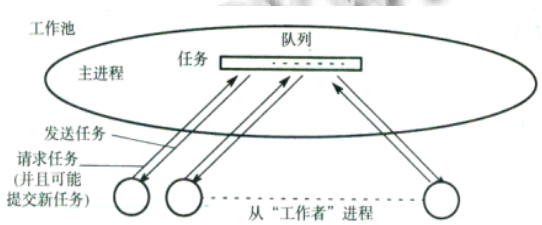

图 1 集中式工作池

集中式动态负载平衡最明显的特点是:主进程能够及时判断计算何时终止. 对一个计算, 假如当前所处理的任务是从工作池的任务队列中获取的, 那么当任务队列为空且每个进程已经请求了另一个任务, 而又没有任何新的任务产生时, 计算就终止.

在判断计算何时终止时, 我们还要注意一个问题: 如果工作池中有一个或多个进程仍在运行, 此时以任务队列为空来确定终止计算是不合理的, 因为那些正在运行的进程可能产生额外的新任务并为工作池任务队列提供新的任务. 至于那些正在运行而不产生新任务的进程, 在任务队列为空并且所有从进程结束时就可以判断计算终止.

综上所述, 集中式工作池技术最好用在任务大小相近和复杂程度较为均匀的松耦合多处理机系统中, 因为在计算中如果有大任务存且分配较晚, 则系统中已经完了成小任务的从进程会一直空闲着, 直到分配到大任务的进程完成任务, 这样就会造成资源浪费, 而且, 集中式工作池中还有一个严重的缺点是: 主进程一次只能发送一个任务, 在初始任务发送后, 它只能一次一个地响应新的任务请求, 因此, 当很多从进程同时请求时就存在着潜在的瓶颈[3]. 为了充分发挥松耦合多处理机系统中的资源, 我们对分散式工作池进行深入研究.

2 分布式动态负载平衡

在任务粒度较细和从进程较多的情况下, 如果能够把工作池分布在多个地点的话, 那将会使系统的功能更加强大. 如图 2 所示, 我们把工作池分布在多个地点称为分布工作池. 在分布式工作池[4]中, 我们可以看到: 主进程将初始的工作池分成几个部分, 并且将每一部分发送给一个小型主进程“(Mo 到 Mn-1)”中的一个. 每个小型主进程控制一组从进程. 在任务的执行过程中, 那些小型的主进程会找到各自的本地最优解, 然后再将它返回给主进程, 主进程再从众多小型的主进程送来的解中选出最优的解, 这样有效地实现问题的优化. 由此可见, 我们可以通过几个层次的分解来发展这种方法. 把从进程放在叶结点上, 采取内部结点分割工作的方法, 就可形成一棵树, 这是将一个任务等分成子任务的基本方法. 对于一棵二叉树, 可在树的每一层进程把任务的一半送给一棵子树, 而把另一半送给另一棵子树. 这样有效地克服了集中式工作池中因较大或最复杂的任务, 在计算中如果大任务分配较晚, 完成小任务的从进程会空闲着的缺点.

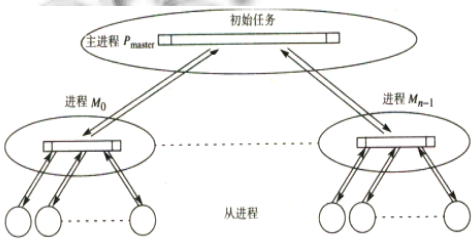

图 2 分布式工作池

3 全分布式工作池

当松耦合多处理机系统中出现某一些进程分配到工作任务, 而且这些已经得到工作任务的进程在任务的执行过程中它们又都分别产生了一些自己的新任务, 这样也就产生了进程之间相互执行任务的可能性(如进程 1 在执行过程中会产生一个新的任务, 而这个新的任务则必需由进程 2 来执行; 反过来进程 2 在执行过程中又产生了另一个新的任务, 而这个新的任务则

必需由进程 1 来执行), 为了解决进程之间相互执行任务这个问题, 我们采用全分布式工作池方法: 即让从进程实际持有工作池的一部分并对这一部分求解. 如图 3 所示, 任务的传递方法有两种: 由接收者启动的方法和由发送者启动方法.

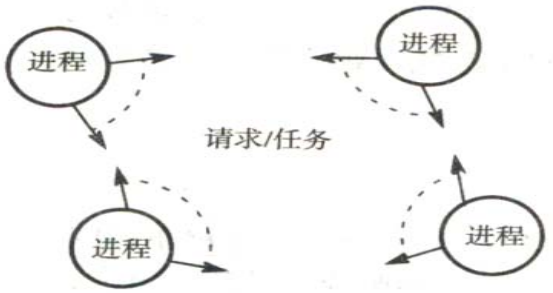

图 3　全分布式工作池

在接收者启动方法中, 一个进程向它选择的其他进程请求任务. 通常, 当一个进程执行的任务很少或没有任务执行的时候, 它就会向其他进程请求任务. 这就表明该方法在高系统负载时会工作得很好. 在授者启动方法中, 一个进程向它选择的其他进程发送任务. 在这种方法中, 通常, 一个负载很重的进程会向愿意接收的其他进程传递一些它的任务. 已经表明这种方法在整个系统负载较轻时工作得较好. 另一种选择是将两种方法结合起来. 但是, 确定进程负载状况的代价比较昂贵. 在系统负载非常重时, 由于缺少可用进程, 负载平衡也可能会很难实现.

现在讨论接收者启动环境中的负载平衡(它也适用于发送者启动的方法). 由于在现实中, 我们要处理的数据非常庞大且非常复杂, 为此我们可将进程组织成一个环, 进程向最近的邻居请求任务. 环形结构适合一个用环形互连网络构成的多处理机系统. 与之相似, 在一个超立方体中, 进程可向每一维上与其直接相连的一个进程请求任务. 当然, 对于任何策略, 都要小心不要让已接收的任务不停地传递.

当进程收到一个任务请求时, 它会将自己还未处理的部分任务发送给请求进程. 例如, 假定问题是用深度优先搜索法遍历一棵搜索树, 从根开始向下访问结点列表, 这些结点与一个进程要访问的结点通过边相连, 进程将从该列表选择一个适当的未访问结点集返回给请求进程. 可以使用多种策略来决定返回多少结点以及返回哪些结点.

(1) 由接收者启动的方法

接收者启动策略与发送者启动策略的不同点是在由轻负载节点启动时, 接收者启动策略要求其它节点把任务发送给它, 其余的大体相同.

接收者启动策略和发送者启动策略中, 我们都采用一个负载节点的负载量阀值 M 来区分重负载节点和轻负载节点(负载量 m>M 时则是重负载, 否则就是轻负载), 同时也采用相关域来确定他们的交互范围.

在进程启动时, 系统中的所有负载节点都同时开始执行各自的计算任务, 若干时间之后, 只要有一个负载节点发现它本身已经成为了轻负载节点, 那么, 该节点就会试图向它所在的相关域中均匀地分布负载. 具体的做法是:

设已经成为轻负载节点的负载量为 l_p, 并且在这个相关域中共有 K 个负载节点, 各个负载节点的相应负载分别为 l_1, ……,l_k, 那么整个相关域中各节点的平均负载量 L_{avg} 为:

$$L_{avg}=\frac{1}{K+1}(l_p+\sum_{k=1}^{K}l_k)$$

为了实现各节点负载均匀分布这个目标, 我们就必须算出相关域中节点应该向轻负载节点传递的负载量 m_k, 然后把他们传递给轻负载节点. 我们首先引入权 h_k 以避免负载从负载更轻的相关域中的节点被迁移到该节点. 如果 $L_{avg}> l_k$, 则 $h_k=h_k= L_{avg} - i_k$, 否则 $h_k=0$. 那么 m_k 为:

$$m_k=(L_{avg}-l_p)h_k/\sum_{k=1}^{K}h_k$$

随后该节点就可以按照 m_k 发出接受任务的请求了.

(2) 由发送者启动方法

我们在发送者启动的策略中引入了一个阀值 M, 并且采用这个阀值 M 来把所有的处理节点负载划分成轻负载节点和重负载节点两种, 在当前所有剩余的负载中, 负载量为 t > M 的节点我们都认为它是重负载节点, 而负载量为 t < M 的节点我们都认为它是轻负载节点. 同接收者启动策略一样, 在发送者启动策略中, 我们也必须为每个节点定义一个相关域(即: 把所有与之相邻的节点作为相关域), 而每个节点也只能与它的相关域中的有关节点进行数据交换或传递任务.

在进程启动时, 在进程启动时, 系统中的所有负

载节点都同时开始执行各自的计算任务, 若干时间之后, 相关域中的所有负载节点都必须开始检查并判断他们自身是否已经变成了重负载节点. 如果已经变成了重负载节点, 那么它们就会力求在相关域中均匀地分布任务. 具体的做法是:

设已经成为重负载节点的负载量为 l_p, 并且在这个相关域中共有 K 个负载节点, 各个负载节点的相应负载分别为 l_1 , ……, l_k , 那么整个相关域中各节点的平均负载量 L_{avg} 为:

$$L_{avg}=\frac{1}{K+1}(l_p+\sum_{i=1}^{K}l_i)$$

实现各节点负载均匀分布这个目标, 重负载节点就应该把相关任务传递给每个相关域中节点的负载量 m_k. 我们引入 h_k 以避免负载被迁移到相关域负载最重的重负载节点. 如果 $L_{avg}>i_k$, 则 $h_k=L_{avg}- i_k$, 否则 $h_k= 0$. 那么 m_k 为:

$$m_k=[(l_p-L_{avg})h_k/\sum_{i=1}^{K}h_i]$$

随后该节点就可以按照 m_k 向各个相关节点发送任务了.

4 仿真实验

4.1 仿真实验设计

在现实实验中, 由于机器设备和实验环境的欠缺, 我们只好利用面向对象语言 VC++软件中所提供的多线程技术来模拟多台处理机[4].

4.2 仿真实验结果

在本次的仿真实验中, 我们分成三组进行实验, 即采用的处理器数分别为 1 个、5 个和 6 个. 在这三组实验中都对同一个数据(大数据)进行仿真实验, 其并行计算的时间如表 1 所示.

表 1 仿真运行结果(不包含通信时间开销)

处理器个数	1	5	6
并行计算时间 Tp/ms	1.109	0.504	0.314

计算加速度的公式[5]: $s_p=T_1/T_p$;

计算效率的公式: $E_p= s_p/P$; 其中 $T1$ 为串行算法在单个处理器上处理一个指定数据的时间开销(即:表中 1 个处理器的执行时间), T_p 表示并行算法在 P 个处理器上处理同一个指定数据的时间开销. 按加速度和效率的计算公式分别把表 1 中 5 个处理器和 6 个处理器的仿真实验数据进行计算, 并把计算结果列在表 2 中.

表 2 计算结果

处理器个数	5	6
加速度 s_p	2.200	3.532
效率 E_p	0.440	0.589

从表 2 的计算结果可知 $s_6 >s_5$, 即系统中处理器数越多, 加速度越大则数据处理的运行速度也相应得到了提高. 由于我们在做仿真实验时所处理的数据量不够大, 所以数据处理的效率提高得不够明显.

5 结束语

综上所述, 我们利用仿真手段, 利用工作池技术在由中低端硬件构成的机群平台(也称为松耦合多处理机系统)上, 及时把工作任务分配给系统中已完成工作任务且已处于空闲等待状态的处理器, 避免在系统中出现部分处理器处于繁忙状态而另一部分处理器则处于空闲等待状态, 有效地实现了所有处理器之间均衡协调地工作, 提高了对大数据处理的工作效率.

参考文献

1 覃雄派,王会举,杜小勇,王珊.大数据分析——RDBMS 与 MapReduce 的竞争与共生.软件学报,2012,23(1):32−45.

2 杨柳,刘铁英.GPU 架构下的并行计算.吉林大学学报(信息科学版),2012,30(6):29−32.

3 唐俊奇.多处理机工作池方式负载平衡技术在机器人的应用.计算机系统应用,2008,17(11):82−86.

4 唐俊奇.单处理机上模拟多处理机的方法研究.莆田学院学报,2007,14(2):75−78.

5 唐俊奇.松耦合多处理机系统在大数据处理中的应用研究.吉林大学学报(信息科学版),2014,32(2):205−210.