

基于异构环境的子任务可分解调度策略研究

宋曰聪

(绵阳师范学院, 四川 绵阳 621000)

摘 要: 针对异构环境下具有数据依赖关系的任务随着业务量增加时, 子任务处理时间急剧增长及计算机资源得不到充分利用的问题, 提出了子任务可以分解的调度策略, 此调度策略根据子任务在 DAG 的层次关系逐层调度, 利用子任务可分解的特性, 将 DAG 图中的一些可分解的子任务分解成一棵 2 层的 m 叉树, 从而使得系统能够动态调整计算机的负载, 并使得系统中各台计算机的负载保持平衡, 其效率得到了明显的提高。

关键词: 异构环境; 任务调度; Min—Min 算法; 调度策略

中图分类号: TP39

文献标识码: A

文章编号: 1000—7180(2011)07—0024—04

Sub-task Decomposition Scheduling Strategy Based on Heterogeneous

SONG Yue-cong

(Mianyang Normal University, Mianyang 621000, China)

Abstract: In heterogeneous environment with data dependencies volume increases with the task, rapid growth and task processing time can make full use of computer resources, puts forward the problems can be decomposed job scheduling strategy, the task scheduling strategy according to the children in the DAG hierarchical relationships by using job scheduling, biodegradable characteristics, some of the graph can be decomposed DAG's job down into a two layer tree layer, thus make the system to adjust the dynamic load, and makes the computer system load balance of the computer, the optimum performance remarkably improved efficiency.

Key words: heterogeneous environment; task scheduling; Min—Min algorithm; scheduling strategy

1 引言

目前在异构环境下具有数据依赖关系的任务调度算法主要包括基于任务拷贝的调度算法、基于优先级的任务调度算法、基于簇的任务调度算法等^[1]。很多调度算法使用 DAG 来进行任务的优化处理, 根据 DAG 的特点, 常采用分层调度、遗传算法、Min—Min 算法等进行调度, 或者采用几种启发式的综合调度方法来分配任务。在对具有数据依赖关系的任务调度时, 无论哪种调度算法, 都必须考虑任务之间的依赖关系, 在 DAG 图中, 只有某个任务的前驱任务完成后才可执行。这些调度算法执行的前提是, 在任务规模不变的情况下, 系统中各个子任务不能够分解, 因而可以确定应用程序中各个子任务的粒

度, 在系统资源一定的情况下, 系统能够很好地满足任务完成的需求, 然而这些调度算法没有充分利用系统中的计算机资源^[2]。文中提出了子任务可以分解的调度策略, 该策略使得系统能够动态调整计算机的负载, 并使得系统中各台计算机的负载保持平衡。实验结果表明, 当子任务可以分解时, 系统更能充分利用计算资源。

2 子任务分解方法

文中采用图 1 所示的子任务分解方法, 即子任务被划分成一些相互独立的更小子任务, 它们负责任务的计算, 在图中子任务用叶子节点表示, 而根节点负责向叶子节点发送和收集数据, 以及向下层子任务发送数据。对系统来说, 并不是所有任务随着任

收稿日期: 2010—09—20; 修回日期: 2010—10—25

基金项目: 四川省教育厅青年基金(2006B082); 绵阳市科技局基金(07Y004-4)

务规模增长而处理时间增长得特别快,对一些任务来说,尽管任务规模增长得很快,处理时间却几乎不增长,此时这样的子任务就不需要进行分解,加上一些子任务随着处理规模和系统资源的变化而需要分解成多个更小子任务,所以系统中的子任务数量随着规模增长而动态变化^[3].

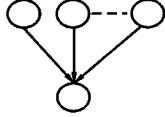


图1 2层结构的 m 叉树

图2给出了子任务可以分解的DAG图.为了使系统能够充分利用资源,必须要有合适的任务调度算法来处理这种子任务可分解的应用系统.

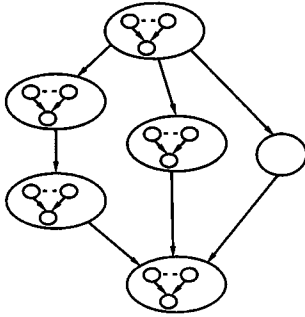


图2 子任务可分解的DAG图

3 子任务可分解的调度策略

当系统中的子任务可以根据任务规模、系统中的计算资源来划分成更小的任务时,就需要合适的调度算法来分配任务.

3.1 系统模型

对于异构系统来说,各台计算机的处理能力不同,这里假设系统中有 M 台计算机, $M = \{m_1, m_2, \dots, m_m\}$. 对一个应用系统来说,处理的数据可能不在一台计算机上,这里假设系统 s 个文件服务器,保存系统工作时需要的数据 $S = \{s_1, s_2, \dots, s_s\}$. 系统中各台计算机的负载分别为 $L_1, L_2, \dots, L_m$. 对异构环境来说,各台计算机CPU的主频可能不一样,在其他硬件满足条件的情况下,这里采用各台计算机CPU主频与1 G的比值来衡量它们计算能力,其比值用 $P_1, P_2, \dots, P_m$ 来表示. 用任务在主频为1 G的计算机上的执行时间来衡量任务粒度.

对一个子任务可以分解的并行应用程序来说,可以用两层嵌套的DAG图表示,如图2所示, $DAG = \{\text{Sub_DAG}, E, T, C\}$,其中 $\text{Sub_DAG} = \{DAG_1, DAG_2, \dots, DAG_n\}$ 表示DAG中的子任务的集合,

DAG_i 表示子任务, E 表示 DAG_i 之间通信边的集合, T 表示各 DAG_i 完成时间的集合, C 表示 DAG_i 之间通信时间的集合, $c_{ij} \in C$ 用来表示 DAG_i 之间的通信代价,根据子任务采用图1的分解方法,它不会因为子任务可分解而发生变化,而是由子任务之间的关系和任务规模来确定. $DAG_i = \{\text{Sub}V_i, \text{Sub}E_i, \text{Sub}T_i, \text{Sub}C_i\}$ 表示子任务的有向无环图, DAG_i 是一棵 m 叉树,如果 DAG_i 不能分解,则 m 始终为0,否则 m 的值和分解策略有关. $\text{Sub}V_i$ 表示 DAG_i 中子任务的集合,子任务的个数为 $m+1$, $\text{Sub}E_i$ 表示 DAG_i 中各个子任务之间通信边的集合,一共有 m 条边,而 $\text{Sub}T_i$ 表示 DAG_i 中各子任务完成的时间集合, $\text{Sub}C_i$ 表示 DAG_i 中叶子节点和根节点之间通信时间集合.

3.2 子任务在DAG图中分层处理

在DAG图中,子任务之间存在数据依赖关系,为了满足子任务的数据依赖关系,只有子任务的所有父节点完成后,它才可以执行,为了使得任务能够满足这种关系,这里给出任务前驱和后续的定义. 对任意节点来说,可以利用公式(1)和(2)来求出其前驱节点集合和后续节点集合.

$$\text{pred}(i) = \{j \mid e_{ji} \in E\} \quad (1)$$

$$\text{succ}(i) = \{j \mid e_{ij} \in E\} \quad (2)$$

对于分解后的 DAG_i 任务来说, DAG_i 是一个2层的 m 叉树,各个子任务之间的前驱和后续关系明确,所以不用前驱和后续关系来求各个子任务之间的关系.

对任意一个子任务 DAG_i 来说,粒度大小和任务的规模用公式(3)表示,其中 b_i 表示初始化时间, a_i 表示不同子任务随着规模增长的任务粒度线性增长因子, x_i 表示任务规模, T_i 表示执行时间.

$$T_i = b_i + a_i x_i \quad (3)$$

对任意子任务 DAG_i 来说,用公式(4)表示这种关系, $\text{Data_}T_{ij}$ 表示任务 i 将数据传输给 j 所需要的时间,其中 $\text{Data_}b_{ij}$ 表示传输初始化数据需要的时间, $\text{Data_}a_{ij}$ 表示线性因子. TCP流量模型用公式(5)表示.

$$\text{Data_}T_{ij} = \text{Data_}b_{ij} + \text{Data_}a_{ij} \quad (4)$$

$$T(t_{\text{RTT}}, s, p) = \frac{c \times s}{t_{\text{RTT}} \times \sqrt{p}} \quad (5)$$

对任意一个分解后的子任务 DAG_i 来说,它是一棵 m 叉树,各个叶子节点用来完成计算任务,所以子任务规模被划分成 m 份,但这 m 个子任务的粒度不同. 可以用公式(6)来表示子任务规模划分后

的关系,根据任务执行时间和任务规模成线性关系的假设,可以知道分解后各个子任务的执行时间用公式(7)表示,同样,叶子节点和根节点之间的数据传输时间可以用公式(8)计算.

$$x_i = x_{1i} + x_{2i} + \cdots + x_{mi} \quad (6)$$

$$\text{Sub_}T_{ki} = b_i + a_i x_{ki} \quad (7)$$

$$\text{SubDa_}T_{ki} = \text{Data_}b_{ij} + \text{Data_}a_{ij} x_{ki} \quad (8)$$

$$k = 1, \cdots, m$$

其中 $x_{1i}, x_{2i}, \cdots, x_{mi}$ 表示子任务的规模, $\text{Sub-}T_{ki}$ 表示子任务的执行时间, $\text{SubDa-}T_{ki}$ 表示子任务的传输时间.

对一个 DAG 图来说,图中的各个结点表示任务,有向边表示任务之间的执行顺序,利用 DAG 任务流之间的关系,可以将这些任务划分成不同的层.根据 DAG 图的顺序关系定义,可以将 DAG 图中节点分成不同的层.在 DAG 中查找没有前驱的子任务,即 $\text{pred}(i) = \Phi$ 的子任务,这些子任务为第一层;根据公式(2),逐个求出第一层中各个子任务的直接后续,所求出的这些任务是第二层,然后利用同样方法求第三层的子任务,如此下去,直到所有任务被遍历,这样 DAG 图就被分为多层结构的 DAG 图.

3.3 逐层调度策略

在采用逐层调度策略时,首先是要对 DAG 做分层处理.根据子任务之间的数据依赖关系,可以将子任务划分成不同的层次,其次是采用 Min-Min 调度策略来分配每一层中的子任务.在 DAG 图中,对每一层的子任务来说,各个子任务之间是相互独立的,各个子任务可以并行地执行,不用考虑同层子任务之间的依赖关系.这里使用的 Min-Min 调度策略是:对任意一个可以开始的子任务来说,在系统所有计算机中寻找最早完成的计算机,这样每个任务都有一个最佳任务计算机配对,然后在所有任务和计算机的最佳配对中选出最早完成的任务计算机配对,然后逐步进行,直到所有任务被分配完成.

再次是在 Min-Min 调度策略分配完一层的子任务后,利用子任务的可分解特性,动态调整系统的负载,使得系统中各台计算机之间的负载保持平衡.为了使得每层中的任务能够在最短的时间内完成,利用子任务可分解的特性,将子任务划分成更小的子任务分配给系统中其他计算机,使得它们的负载保持平衡.对每一层子任务分配过程来说,系统中计算机的负载都保持平衡,并且它们在最短时间内完成,从而对所有层次的子任务来说,也能够使系统保持负载平衡,且能够使得应用程序能在最短时间内

完成.最后是当一层的子任务分配后,采用同样的办法,将 DAG 图中各层中的子任务从上往下逐步进行分配,直到所有任务被分配完.子任务可分解调度算法如下所示:

```

Level-by-Level Scheduling Algorithm
Begin //分层处理
Do level partitioning for the DAG
//分层调度,开始使用 Min-Min 方法调度,然后负载
平衡处理
For level=1 to n
{
Set Ready to be the set of all subtasks which can exe-
cute at this level
Do{
Find EARLY(DAGi, Taskset) for all subtasks in Read-
y, where Taskset presents the task set can be used to sched-
ule //Min-Min 调度
Min-Min: Choose the subtask DAGk with the minimum
completion time.
Schedule subtask DAGk to machine mmin and data re-
pository sk.
Remove DAGk from Ready.
While(Ready is not empty)
Load balance transaction using decomposing some sub-
tasks method, which make computers keep load balance.
}

```

3.4 子任务分解的负载平衡处理策略

在使用 Min-Min 调度策略分配完一层任务后,可能会造成各台计算机负载不平衡,因为 DAG 图中不同层次之间的任务存在数据依赖关系,一方面会造成一些计算资源没有得到利用,而另一方面任务处于等待上层任务的完成状态.为了充分利用计算资源,利用子任务可以分解的特性,将一些可分解的任务重新分配,使得各台计算机负载保持平衡^[4].

当 DAG 图中的同一层任务数量没有系统中计算机数量多,并且存在粒度大的可分解任务时,往往需要对任务进行分解^[5].下面给出同一层子任务分解的负载平衡处理策略,处理策略中的所有子任务都是在同一层中.

(1) 从 $L_1, L_2, \cdots, L_m$ 中求出负载最小值 L . $\text{sup_}g_i$ 表示各台计算机可以提供的任务粒度, $\text{Sum_}G$ 表示统计所有计算机提供的任务粒度之和.

$$\text{sup_}g_i = \Delta L_i \times P_i \quad (9)$$

$$\text{Sum_}G = \sum_{i=1}^m \text{sup_}g_i \quad (10)$$

(2) 在负载大的计算机中查找粒度合适的任务,使得负载小的计算机能够满足 $\text{sub_}g_i$ 绝对值小于 2. 并且将满足条件的计算机从负载平衡处理队列中删除,并更新 L_i . 如果所有计算机都从负载平衡处理队列中删除,则退出负载平衡处理过程.

(3) 在负载大的计算机中查找可分解的任务,将任务分解出一部分,将它分配给负载小的计算机并且使得它的 $\text{sub_}g_i = 0$,将负载小的计算机从负载平衡处理队列中删除,并更新 L_i .

这样一层的子任务就被分配到系统中的各台计算机,在分配下一层的子任务时,采用同样的方法进行负载平衡处理.

4 实验与性能分析

使用 5 个具有数据依赖关系的应用程序进行实验,比较使用子任务可分解和子任务不可分解的情况^[6]. 这 5 个应用程序包含的子任务数量分别为 5、10、15、20、25,应用程序中的每个子任务都可以分解,各个子任务之间的依赖关系满足完全二叉树. 测试应用程序中的子任务满足:可以分解、睡眠 1000 次、每次睡眠时间为 100 ms,因为测试程序中子任务传输数据量很小,所以子任务之间的数据传输时间很短. 如图 3 所示.

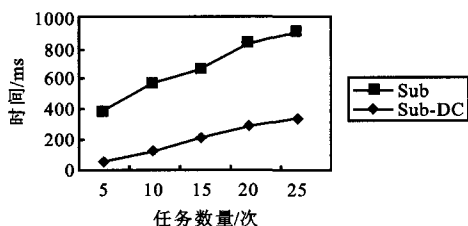


图 3 不同应用程序两种调度算法的时间对比

从图 3 可以看出,当应用程序的子任务能够分解时,系统效率得到明显的提高. 在子任务数量不变的情况下,子任务数量为 15,各个子任务可以分解,逐步增加任务的规模,将子任务睡眠次数逐步增加,对没有使用子任务分解的系统来说,其执行时间比使用子任务分解平衡处理策略的多,实验结果如图 4 所示.

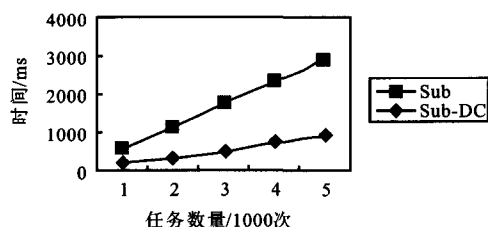


图 4 任务规模逐步增加时的两种调度算法的效率比较

从图 4 可以看出,当增加任务规模时,两种方法都几乎呈线性增长,逐层调度策略使用了子任务可分解的特性后,其效率得到了明显提高,并且可以看出,当任务规模线性增长时,处理时间也是线性增长.

5 结束语

文中阐述了异构环境下子任务可以分解的调度算法,该算法根据子任务在 DAG 的层次结构逐层调度,每层使用 Min-Min 调度算法,在 Min-Min 调度完后,利用子任务的可分解特性,将 DAG 中的一些可分解的子任务分解成一棵 2 层的 m 叉树,动态调整计算机的负载,并且使得系统中各台计算机的负载保持平衡,这样不但可以使每一层任务在尽可能短的时间内完成,而且使得整个应用程序能够在尽可能短的时间内完成. 实验结果显示,当子任务可以分解时,系统更能充分利用计算资源.

参考文献:

- [1] Sameer Shivle, Ralph Castain, Siegel H J, et al. Static mapping of subtasks in a heterogeneous ad hoc grid environment[C]// Proceedings of the 13th International Parallel and Distributed Processing Symposium (IPDPS 2004). SantaFe, New Mexico, USA: IEEE Computer Society, 2004: 26-30.
- [2] Ammar H Alhusaini, Viktor Prasanna, Raghavendra C S. A unified resource scheduling framework for heterogeneous computing environment[C]// Proceedings of the 8th Heterogeneous Computing Workshop. San Juan, Puerto Rico: IEEE Computer Society Press, 1999: 156-165.
- [3] Floyd S, Fall K. Promoting the use of end-to-end congestion control in the internet[J]. IEEE/ACM Transaction on Network. 1999, 7(4): 458-472.
- [4] Padhye J, Firoiu V, Towsley d, et al. Modeling TCP throughput: a simple model and its empirical validation [C]// Proceeding of the SIGCOMM. Vancouver: ACM Press, 1998: 303-314.
- [5] 万本庭, 陈明. 基于 SVG 的矢量图形库管理模型[J]. 计算机工程, 2006, 32(3): 65-67.
- [6] 都志辉. 高性能计算之并行编程技术——MPI 并行程序设计[M]. 北京: 清华大学出版社, 1997: 13-55.

作者简介:

宋曰聪 男, (1963-), 教授. 研究方向为并行计算、复杂网络算法.