一种异构实时系统中任务流均衡负载的算术模型

杨 娟 白 云 邱玉辉

(西南师范大学计算机与信息科学学院 重庆400715)

摘 要 异构环境下如何提高系统自治并发处理的能力,是分布式系统研究领域的重要课题。而异构实时系统中系统自治并发处理的能力则体现在事件驱动任务流控制的资源最优分配。本文针对已有的资源分配策略的缺陷给出了一个通用的任务流均衡负载的算术模型,除了对任务流进行量化,使其可以动态分配以外,还可以在任务流发生变化时在满足系统最大利益的前提下最优化代价函数。另外,由于这个算术模型具有通用性,因此适用于各种资源分配的启发式搜索算法。

关键词 资源分配,事件驱动,任务流均衡负载

A Balanced Workload Mathematical Model in Heterogeneous Real-time Systems

YANG Juan BAI Yun QIU Yu-Hui

(Dept. of Computer and Information Science, Southwest China Normal University, Chongqing 400715)

Abstract How to improve the systems' concurrent processing capability in the heterogeneous environment has gotten the growing attention. The resource allocation of the event driving tasks flow in realtime systems can be used to depict the concurrent processing capability of realtime systems. Many resource allocation policies have emerged to get this purpose, however some defects still hamper the policies employing. One of them is how to quantify the workload and how to protect the systems' cost function without breaking the QOS constraints. We propose a balanced workload mathematical model to solve this problem. Besides this, BWMM's generality make it easily adapt to any heuristic research algorithm.

Keywords Resource allocation, Data driving, Balanced workload

1 引言

异构环境下的并发处理已经成为分布式系统研究领域的一个重要课题,因为各种各样的应用系统(如天气模拟计算、实时系统中的数据流控制,图像处理,等)都潜在地要求在这样的背景下进行代价函数的构建。这里所指的代价函数除了最大化系统吞吐量和最小化整体响应时间(多数学者的工作中心)以外,还包括特定环境下的QoS限制,如实时系统中数据流从输入到输出必须要小于的时延限制[1],等。因此,问题最终衍变成了如何提高系统进行自治并发处理的能力。这种能力又体现在了系统资源分配的合理性上。

资源分配的最优化问题根据所针对的任务流类型的不同[2]可分为:1)事件驱动(event driving)和2)数据驱动(data driving)两大类。事件驱动型的任务流其特点主要体现在任务流间没有数据依赖性[3,4],针对这种类型的资源分配策略主要指实时系统中的任务流控制,而数据驱动任务流则指任务间存在数据依赖关系,任务流可用DAG图准确描述[5~7]。在此基础上构建的代价函数(cost function)也多为:最小化整体响应时间[3,4,6,7]、在保证任务流QoS的基础上最大化系统容错性[8,9]以及最大化系统吞吐量[5],等。因此在实时系统中资源分配主要的工作是进行事件驱动任务流的代价函数构建,并进行资源最优化分配。而应用于实时系统的资源分配策略主要有:

基于权重的分配策略[3,10]。这种方法将优先级赋给各个任务,并根据优先级进行资源分配,典型的算法是文[3]中提出的MMP算法,但它也存在致命的问题,例如:MMP运用的前提是必须知道任务数为λ时其时间执行的上下限,而它们则受任务流性质的影响,无法准确确定。另外,当任务流发生变化时,MMP无法在保障系统整体利益最大的前提下最小化系统整体响应时间。

基于聚类的分配策略。其主要思想是将一类在关系上相互依赖紧密的任务定位到同一个处理器上,这样可以有效地减少通信开销。单独的聚类算法由于其使用情况单一已逐渐不再被使用,替代的则是基于聚类的任务复制策略[6,7,11,12]和扩散收敛策略[4]。基于聚类的任务复制策略多用于DAG模型,其主要思想是将相互依赖紧密的任务定位到同一处理器上后,再判断处理器的空闲情况而进行任务的复制。这种方法的问题在于如何对复制任务后节省的计算耗费与移动任务所增加的通信耗费之间进行有效的平衡,而这一点除了受网络状况的影响外还受任务大小和性质的决定。基于聚类的扩散收敛策略则是通过收敛因子进行邻接节点间的任务传递控制,当收敛因子收敛时则说明系统整体已取得最大效益。但这种方法存在的最大问题在于无法模型化每个节点发送和接受的任务量,也无法设定一个通用的参考标准。

分析上述的各种策略,不难看出其存在的主要问题在于如何在任务流发生改变时在保证整体利益最大的前提下最优

杨 娟 博士研究生。白云 博士研究生。邱玉辉 博士生导师。

化系统代价函数，以及如何将任务量模型化，并选取一个有效的参考标准。针对这两点，我们提出了一种适用于实时系统的任务流均衡负载算术模型——BWMM(Balanced Workload Mathematical Model)。在这个模型中除了对任务流进行了量化，使其可以动态分配外，当系统任务流发生变化时BWMM还在保证满足系统最大效益的前提下最优化代价函数。模型采用的代价函数为最小化系统处理时间，针对的任务流为实时系统中的时间驱动任务流。由于BWMM有很强的通用性，因此可适用于各种资源分配的启发式搜索策略。

本文第2节主要对BWMM的问题空间以及求解进行了描述，第3节给出了一个算术模型应用示例，最后是试验、工作小结以及将来工作阐述。

2 任务流均衡负载算术模型(BWMM)

2.1 问题描述

因为现在几乎所有的并行计算中间件/系统都支持面向对象，所以我们将给定的一个任务 J，初始化为 k 个以类(不能再进行划分)为单位的子任务序列$(j_1,j_2,\cdots,j_k)$，而子任务 j_i 的执行则是对实例化了的这些类的对象进行计算。

给定图 $G(J,P,F)$。J 代表要进行计算的任务，若 j 代表划分的子任务，则有$\forall\ j\in J$，P 代表网络中可以进行并行计算的进程集合，p 代表进程，则有$\forall\ p\in P$。f 为子任务 j 与进程 p 之间可能关联的边，有$\forall\ f\in F$。若子任务 j 在 p 上执行运算，则有 $R_f=1$，否则 $R_f=0$。假设子任务 j 的标准计算时间为 T_j，那么我们用 $\delta(p,v)$来实际描述不同计算节点间的性能差异。则 $T_j\cdot\delta_{(j,(p,v))}$为子任务 j 在 p 上执行真正所需时间，如果网络中总共有 n 个进程，且其中同时有 kp 个进程并发执行任务，则定义这 kp 个进程上分别有 $ks_i,i\in[1,kp]$个子任务进行串行操作。我们用 $T_{L_i}^i$代表执行并发操作进程 $p_i,i\in[1,kp]$上子任务执行的总共时间，则任务 J 的执行时间为 $T=\max\{T_{L_1}^1,T_{L_2}^2,\cdots,T_{L_{kp}}^{kp}\}$，最终问题求解当网络中同时进行计算的进程数分别为$1,2,\cdots\min\{k,n\}$时执行时间最短 $\min\{\min\{T|kp=1\},\min\{T|kp=2\},\cdots,\min\{T|kp=\min\{k,n\}\}\}$。

2.2 相关参数定义

(1)$I_f^l=\{(f,(p,v)\in F|(g_j(j),g_j(p,v))=f\}$。$g_j$ 为一个映射子任务 j 与进程(p,v)所有可能连线的函数。$I_f^l=h(I_f)$，h 为将一个集合 I_f 中的子任务名 j 映射为数字标记 l 的函数。

(2)$R_f,f\in F,R_f=R(j,(p,v))$。用 R_f 将 $G(J,P,F)$中的每条边都关联起来。当子任务 j 在进程(p,v)上执行时，$R(j,(p,v))=1$，否则 $R(j,(p,v))=0$。

(3)δ_f 代表(p,v)元组中进程 p 所在节点 v 的性能参数。若节点 v 的性能优于标准性能指标 δ，则$0<\delta(p,v)<1$，否则 $\delta(p,v)\geqslant 1$。

(4)$T_{L_i}^i=\sum_{m=1}^{ks_i}(\sum_{f\in I_f^l}(T_j\cdot R_f\cdot\delta_f))$代表任务 J 的 k 个子任务中若有 ks_i 个子任务在节点 v 上的进程 p_i 中进行串行计算。则这 ks_i 个子任务总的计算时间为其串行执行时间的累加总和。

(5)$T=\max\{T_{L_1}^1,T_{L_2}^2,\cdots,T_{L_{kp}}^{kp}\}$。$T$ 为执行任务 J 最终时间耗费，因为若 kp 个节点同时进行并行计算，那么最后 J 的执行时间应该取其中最后一个节点计算完成后的值，这里我们实际上对任务 J 的子任务划分数为 kp 个，而不是最开始定义的 k 个，从而实现了子任务大小的动态划分。

2.3 限制条件

(1)$\sum_{j\in I_f^l}R_f=1,\forall\ j\in J$。在每种计算情况下，子任务 j 只可能被一个进程执行。因此在这种情况下只有一条 $f=(j,(p,v))$边为真，其余都为假。

(2)$\sum_{i=1}^{kp}ks_i=k,kp\in[1,n],ks_i\in[1,k],kp$ 与 ks_i 都为整数。

(3)$\delta_f\in(0,+\infty)$。

(4)$\begin{cases}J=\bigcup_{i=1}^{k}j_i\\ F=\bigcup_{i=1}^{k}I_f^l=\bigcup_{i=1}^{k}I_f\end{cases}$。

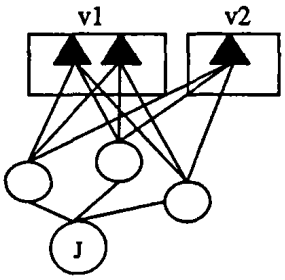

图1 G(J,P,F)

3 BWMM应用示例

如图1所示，若网络中有节点 v_1,v_2，并且分别拥有进程 p_1,p_2和 p_3(图中黑色三角形)，任务 J 被初始划分为三个最小子任务 j_1,j_2和 j_3(空心圆圈)，并且，有节点 v_1的性能参数 $\delta_{v_1}=1.2$，节点 v_2的性能参数 $\delta_{v_2}=0.8$，且子任务的标准执行时间为 $T_{j_1}=8$(ms)，$T_{j_2}=6$(ms)，$T_{j_3}=5$(ms)。求解 $\min\{\min\{T|kp=1\},\min\{T|kp=2\},\min\{T|kp=3\}\}$。在满足限制条件后，BWMM计算得出结果：当子任务1在节点 v_2上进行运算，子任务2和3分别在 v_1上任意一个进程中运算，最后所得的计算时间都为各种情况中的最优。我们通过节点性能参数来进行网络中任务负载平衡的控制，当计算节点间的性能参数差距不大时，BWMM总能找出一种情况可以使计算时间最优，并且网络中计算负载平衡。

4 BWMM性能测试试验

4.1 试验机群的节点性能参数确定

考虑 $n\times n$ 的两个矩阵相乘的不同机型执行时间算出性能参数。

表1 两个n*n矩阵执行时间

machine	n=(50)/T	n=(100)/T	n=(200)/T	n=(500)/T	δ
联想万全	24(ms)	57(ms)	446(ms)	9292(ms)	1.02
DELL PE2550	18(ms)	46(ms)	257(ms)	6480(ms)	0.71
DELL GX240	15(ms)	47(ms)	203(ms)	9360(ms)	1

可以看到，节点的性能参数不仅与任务本身性质相关(有些对内存要求高，有些对CPU计算能力要求高)，而且同任务划分数量相关，通常说来当子任务规模达到一定程度时节点

(下转第227页)

客户端显示的数据。该应用方案的处理流程如图3所示。

通过改进后的应用方案，我们可以将渲染组件中的样式处理集中化，可以不用修改代码就可以改变显示的样式，增强了渲染组件的可扩展性，使渲染操作的条理更加明晰。

结束语 JSF 作为新一代的 Web 应用框架，通过引入用户接口组件模型及其相应的处理流程，从而使 Web 应用程序的开发具有了与桌面应用程序的开发相似的特点和便利。此外，JSF 的定义均基于 Java API，并提供了 JSP 自定义标签库，为 JSF 与 JSP、Servlet 等现有技术实现很好的集成提供了条件。在实际的应用中，针对 JSF 现有标准在渲染组件方面的一点不足，本文还提出了一个 JSF 与 JSP、XML 相结合的解决方案，从而充分发挥了 JSF 在用户界面显示方面的可扩展性和多样性。

JSF 自身诸多的优点使包括 SUN、IBM 等大公司在内的越来越多的人投身到 JSF 的发展中，这让我们有理由相信 JSF 将成为多层 Web 应用程序开发的主流技术。

参 考 文 献

1 Coward D, Yoshida Y. Java Servlet Specification 2.4[S/OL]. http://jcp.org/aboutJava/communityprocess/final/jsr154/, 2003-11-24/2004-8-2

2 Roth M, Pelegrí-Llopart E. JavaServer Pages Specification 2.0[S/OL]. http://jcp.org/aboutJava/communityprocess/final/jsr152/, 2003-11-24/2004-8-2

3 McClanahan C, Burns E, Kitain R. JavaServer Faces Specification 1.1[S/OL]. http://java.sun.com/j2ee/javaserverfaces/, 2004-5-27/2004-8-2

4 Burns E, Horwat J. An Introduction to JavaServer Faces[Z/OL]. http://java.sun.com/j2ee/javaserverfaces/jsfintro.html, 2004-6-28/2004-8-2

（上接第223页）

间的计算性能差距会随着规模的增大而增大。我们取节点平均计算时间的比例来确定 δ。

4.2 BWMM 计算性能试验

因为 BWMM 的状态空间数为 (n^k) 个，因此我们用动态规划法来代替枚举法，分别考虑进程数从10到30时，子任务数为5，10，15时的计算时间。

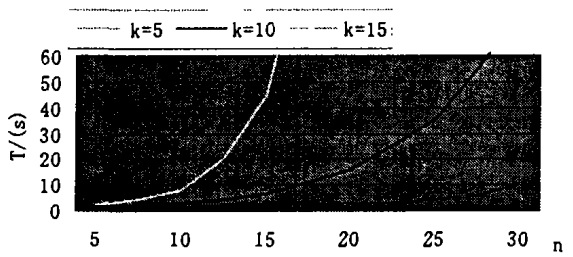

图2 BWMM 计算时间

从图2中可以看到，随着规划的子任务数数量增多，使得计算时间发生突变的进程数越小。如果我们考虑1分钟内结果都是可接受的，那么当进程数达到20，子任务数为10时性能最好。

工作小结及将来工作 本文针对现有实时分布式系统中事件驱动任务流资源分配策略的问题提出了一种通用的任务流均衡负载算术模型，BWMM，并且实现了动态任务规划的必须需求：

(1)子任务大小的动态划分。BWMM 以面向对象的方式对任务 J 进行子任务划分，将 J 初始化为不可再分的以类为单位的子任务，因此每个子任务的执行实际上是这个实例化了的这个类的对象，而我们根据网络中节点的性能差异允许部分子过程进行串行计算，这实际上是将多个子任务合并成为上级子任务 j'，这样就实现了 J 的子任务大小不定的动态划分。

(2)子任务执行进程的动态选择。因为网络中参与节点的环境异构，因此导致了节点间计算性能的差异。BWMM 中用 $\delta(p,v)$ 来表示进程 p 所在节点 v 的性能参数，选择各种情况中计算时间最短的一种为子任务分配策略。当子任务的标准计算时间发生变化时（例如类中新增添了代码），任务分配策略也动态地发生变化。

(3)计算时间总能最优。虽然 BWMM 解决的是一个 NP 问题，但最终计算结果中总能找到一种任务分布情况为计算时间最优。

但 BWMM 针对的是事件驱动的任务流类型，即相对于数据驱动的任务流来说，忽略了子任务间数据的关联（例如 j3 的执行必须要 j1的计算结果）。另外，节点的性能参数准确性的提高以及改进问题算法也是下一步要进行的工作。

参 考 文 献

1 Kafil M, Ahmad I. Optimal Task Assignment in heterogeneous Distributed Computing Systems. IEEE Concurrency on Complex Distributed Systems, 1998, 6(3): 42～51

2 Jaroodi AI, Jiang Hong, Swanson D. A Comparative Study of Parallel and Distributed Java Projects for Heterogeneous Systems. In: Proc. of the 16th Intl. Parallel and Distributed Processing Symposium, 2002

3 Georgiadis L, Nikolaou C, et al. A fair workload allocation policy for heterogeneous systems. Journal of Parallel and Distributed Computing, 2004, 64(1): 507～519

4 Rotaru T, Nageli H-H. Dynamic load balancing by diffusion in heterogeneous systems. Journal of Parallel and Distributed Computing, 2004, 64(4): 481～497

5 Ali S, Kim J-K, etc. Greedy Heuristics for Resource Allocation in Dynamic Distributed Real-Time Heterogeneous Computing Systems. 2002 International Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'02), Las Vegas, NV, 2002. 519～530

6 Ravindran B, Li P, et al. Proactive resource allocation for asynchronous real-time distributed systems in the presence of processor failures. Journal of Parallel and Distributed Computing, 2003, 63(12): 1219～1242

7 Ranaweera S, Agrawal D P. A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems. IPDPS 2002. 445～450

8 Boloni L, Marinescu D C. Robust Scheduling of Metaprograms. Journal of Scheduling, 2002, 5(5): 395～412

9 Ali S, Maciejewski, et al. Measuring the Robustness of a Resource Allocation. IEEE Transaction on Parallel and Distributed Systems, 2004, 15(7): 630～641

10 王永炎，王强，等. 基于优先级别表的实时调度算法及其实现. 软件学报, 2004, 15(3): 360～370

11 Bajaj R, Agrawal D P. Improving Scheduling of Tasks in a Heterogeneous Environment. IEEE Transaction on Parallel and Distributed Systems, 2004, 15(2): 107～118

12 Bansal S, Kumar P, et al. An Improved Duplication Strategy for Scheduling Precedence Constrained Graphs in Multiprocessor Systems. IEEE Transaction on Parallel and Distributed Systems, 2003, 14(6): 533～544