基于 GridSim 模拟器的网格资源调度算法研究*)

李 炯 卢显良 董 仕
(电子科技大学计算机科学与工程学院 成都 610054)

摘 要 网格资源调度策略是网格计算领域中的关键研究方向之一,网格模拟器是资源调度策略优化和改进研究的重要平台。本文研究了 GridSim 模拟器,对此模拟器的整个框架结构和运行机制作了阐述;对基础的 Minmin 算法和 QoS Guided Min-min 算法进行研究和改进,并通过基于 GridSim 包设计了应用程序,对改进后的算法进行了相应的模拟。模拟研究结果表明,改进后的算法在任务平均完成时间上优于以前的算法。

关键词 网格,GridSim,Minmin 算法,QoS Guided Min-min 算法

Research on Resource Scheduling Strategies for Gird Based on GridSim

LI Jiong LU Xian-liang DONG Shi
(School of Computer, UESTC, Chengdu 610054, China)

Abstract Grid resource scheduling strategy is an important component of researching fields in the grid computation. Grid simulator is an important platform about research on optimizing and amelioration of resource scheduled strategy. This text studies GridSim and sets forth the whole framework and running mechanism. In addition, this text studies the Minmin arithmetic and QoS Guided Min-min arithmetic. Puts forward improving on them. Designs a application based on GridSim package, and the application simulates new arithmetic. Research result indicates that new arithmetic excells former arithmetic on average completely time of task.

Keywords Grid, GridSim, Minmin strategies, QoS guided Min-min strategies

1 引言

网格资源调度策略是网格计算领域中的关键研究方向之一,在网格计算中,通过采取适合于网格任务特征和资源特点的调度策略,将网格计算中的资源分配给匹配的网格任务,从而使网格资源利用率最大化。本文对 GridSim[1] 模拟器进行了深入探讨和研究,对目前基础的 Minmin 算法和 QoS Guided Min-min 算法进行了研究,提出一些不足,并对 Min-min 算法和 QoS Guided Min-min 算法进行了改进。

2 GridSim 模拟器

GridSim 模拟器的研究

GridSim[1] 由澳大利亚墨尔本大学 Rajkumar Buyya 领导开发,它的首要目标是通过模拟来研究基于计算经济模型的有效资源分配方法。GridSim 通过资源的"买"和"卖"来引入"经济模型",从而达到控制网格资源的使用的目的。图 1 显示了 GridSim 的体系结构。

GridSim 是在 SimJava[2] 的基础上开发的,它提供丰富的函数库以支持模拟网格环境中的异构资源(时间共享和空间共享)、用户、应用程序、用户代理和调度器。网格资源、用户和用户代理被视为不同的实体,它们通过消息事件(输入和输出)来进行通信。除了通过手工编程来实现模拟外,GridSim 还提供了一套图形接口工具 Visual Modeler(VM)帮助用户配置网格环境并产生相应的代码。仿真结束后,用户可以调用 GridSim 中的称为 GridStatistics 的库函数来收集各种模拟的统计资料。

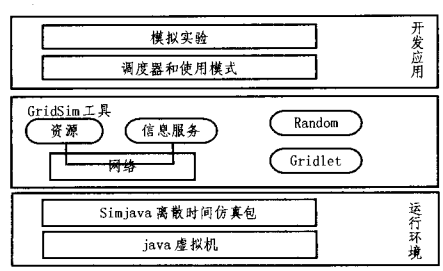

图 1 GridSim 的体系结构

GridSim 模拟的主要流程如下:初始化各个离散对象→启动仿真→资源的注册→代理 broker 向信息中心查询资源→broker 映像计算→提交任务→资源处理任务→资源返回结果→结束仿真。按照这一过程可以对采用不同策略时的网格资源调度进行模拟。

3 MinMin 算法研究

Minmin 算法是启发式算法中的经典算法。对于传统的 Min-Min, Max-Min, A* 和 GA 等静态启发式算法,Tracy D. Bra 等人已经做了详细的研究[3]。结果表明,GA 算法在不同 ETC 矩阵下的性能是好的,Min-Min 和 A* 次之。Tracy D. Braun 的研究表明:对于每个 ETC 矩阵 G 算法的平均执行时间是 60 秒,而 A* 算法是 20 分钟。由于算法中各项参数是通过 NWS 等服务得到的一个提前预测值,因此在参数随时间剧烈变化的网格环境下 GA 或 A* 的计算时间会显得很长,从而得到的调度策略也很不合理。特别是当有主机的性能出

*)本文受电子科技大学青年博士平台基金支持(基金号:05BS01601)。李 炯 讲师,博士研究生;卢显良 教授,博士生导师;董 仕 硕士研究生。

现突变时，算法性能会变得更为低下。Min-Min 算法仍然是目前网格调度算法的研究基础之一，该算法的主要思想如下：

当需要调度的任务集合非空时，反复执行如下操作直至集合为空：

(1)对集合中每一个等待分配的任务 Ti，分别计算出把该任务分配到 n 台机器上的最小完成时间。假设任务在第 k 台机器上的完成时间为最小，记为 Min $Ti(I)=MCT(i,k)$，可得到一个含有 m 个元素的一维数组 Min-Time；

(2)设第 a 个元素是 Min-Time 数组中最小的，其对应的主机为 b，把任务 a 分配到机器 b 上；

(3)从任务集合中把任务 a 删除，同时更新 MCT 矩阵。Max-Min 算法的实现思路和 Min-Min 算法很相似，只是把上面第(2)步找 MinTime 数组中最小的元素改为找最大的元素。Min-min，Max-min，Sufferage，Xsufferage 等基本启发算法模式如下：

(1)while there are tasks to schedule

(2)for all task i to schedule

(3)for all host j

(4)Compute CTi,j=CT(task i,host j)

(5)end for

(6)Compute metrici=f1(CTi,1,CTi,2,……)

(7)end for

(8)Select best metric match(m, n) = f2(metric1, metric2,)

(9)Compute minimum CTm,n

(10)Schedule task m on n

(11)end while

众所周知，Min-Min 算法的一个最大的毛病就是负载不平衡，这是因为当网络传输等条件对局部任务调度影响很小的时候整个网格处在一个异构的环境中，这时候机器的处理能力将完全体现任务的调度策略。换句话说，如果某个节点的计算能力大于同处于一个局域网内的其它节点的时候，所有调度到这个局部的任务都会调度到这台机器上去，从而引起其它机器长期处于空闲状态，大量的任务处在等待调度的状态，这使得这台机器负载过大，造成局部的负载严重不平衡。

3.1 Sufferage 算法

该算法的主要思想如下：

1)初始时各个主机上的已分配任务队列为空；

2)当待调度的网格任务集合为非空时，反复执行如下操作直至集合为空：

(1)对集合中每一个等待分配的任务 T，根据 ETC(Ti, Hj)和 START(Hj)分别计算出把该任务分配到 n 台主机上执行相应的最小完成时间。假设任务 Ti 在主机 Hj 上的完成时间为最小，记为 Min-Time(i)=C(Ti, Hj)，可得到一个含有 m 个元素的一维数组 Min-Time[m]；

(2)遍历存放每个任务的最早完成时间的一维数组 Min-Time，找到其中的最小值和次小值并获得其相应的任务和主机的编号，计算该任务的 Sufferage 的值，将该任务分配到相应的主机上；

(3)根据该主机上的已分配任务队列中每个任务的 Sufferage 值排序，将那些 Sufferage 值小于该任务 Ti 的已分配的任务从已分配任务列表中删除，添加到待调度网格任务集合中；

(4)从待分配的任务集合中把任务 T；删除，同时更新 ETC(Ti, Hj)，START(Hj)，Comp(Ti, Hj)等矩阵。

3.2 QoS Guided Min-min 算法

该算法先对高 QoS 作业使用 Min-min 算法进行调度，将其分配到高 QoS 资源上执行；然后再对低 QoS 作业使用 Min-min 算法进行调度，将其分配到所有网格资源上执行。这种算法将高 QoS 作业优先调度，解决了高 QoS 资源被低 QoS 作业占据的问题。

Sufferage 算法是解决没有任务有 QoS 要求时的一种改进 Minmin 算法存在负载缺陷的算法，而 QoS Guided Min-min 算法则是对存在任务要求(QoS)的一种改进算法。

从这两种针对 Minmin 的改进算法对以上两种不同情况提出了各自的解决方案。如何同时解决这两种情况并存时的一种情况呢？下面我们将提出一个新的解决方案 Qos-Sufferage 算法。

4 Qos-Sufferage[4] 算法

具体算法的思想为：

基于任务所能够被执行的计算资源的个数，即 QosLevel 向量所表示的个数进行分组，向量中的每一个分量作为一个 QoS 优先级分组。由于 QosLevel 向量是一个 n 维向量，因此可以将所有的网格任务分为 n 组。按照 QoS 优先级由高到低的顺序根据 ETC 子矩阵 $SubETC_{i,m}$，对这些子矩阵中的任务采用传统的 sufferage 算法进行调度。

算法流程的伪代码如下：

(1)获得所有任务在所有主机上的执行时间并且将其保存在 ETC 矩阵中；

(2)根据任务的可执行情况将 m 个任务分成 n 个 QoS 优先级分组；

(3)划分 ETC 并获得 $SubETC_{i,m}$；

(4)int $i=1$；

(5)While ($i<=n$){

(6)Do the Sufferage algorithm for the $SubETC_{i,m}$；

(7) $i++$;}

5 仿真设计与结果模拟

为验证改进后的算法，本文在 windows 环境下，使用 eclipse 研发的基于 GridSim 包的应用程序，对改进前后的资源调度算法进行了仿真试验，具体配置和实验模拟结果如下。

5.1 参数的配置

表 1 资源调度参数配置表

任务数	机器数	仿真次数	调度策略
50	10	100	QoS Guided Min-min 或 Qos-Sufferage 算法
100	10	100	QoS Guided Min-min 或 Qos-Sufferage 算法
150	10	100	QoS Guided Min-min 或 Qos-Sufferage 算法

5.2 模拟结果

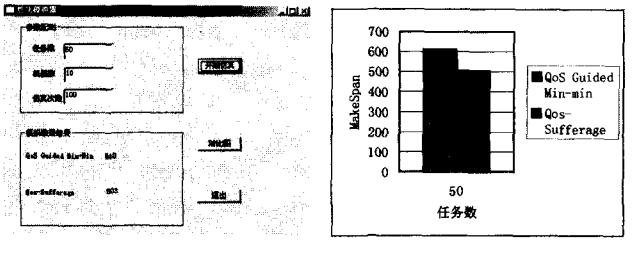

图 2 任务数为 50 时，QoS Guided Min-min 与 Qos-Sufferage 算法的对比

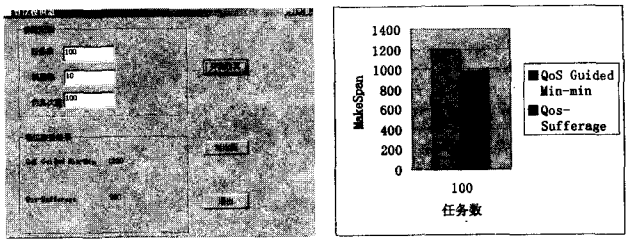

图 3 任务数为 100 时，QoS Guided Min-min 与 Qos-Sufferage 算法的对比

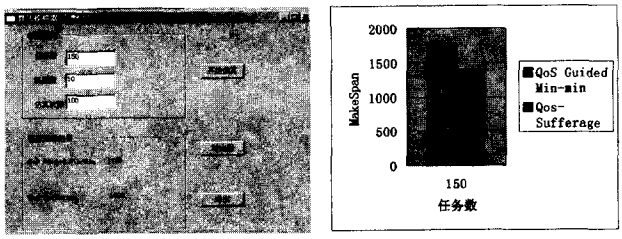

图 4 任务数为 150 时，QoS Guided Min-min 与 Qos-Sufferage 算法的对比

从图 2,3,4 可以看出 Qos-Sufferage 算法与 QoS Guided Min-min 算法相比，降低了平均完成时间，提高了资源调度的性能。

结束语 网格资源的调度是网格计算中的核心问题。本文通过对 GridSim 模拟器的深入探讨和研究，对 Minmin 算法和 QoS Guided Min-min 算法进行了研究，提出一些不足，并对 Minmin 算法和 QoS Guided Min-min 算法进行改进，在改进的算法中融入了 QoS 机制和 Sufferage 算法思想。实验结果表明：改进后的算法，在平均完成时间上有很大的改进，提高了资源的调度性能。

参 考 文 献

[1] Murshed M, Buyya R. Using the GridSim Toolkit for Enabling Grid Computing Education // International Conference on Communication Networks and Distributed Systems Modeling and Simulation (CNDS2002). San Antonio, Texas, USA, January 2002

[2] Braun T D, Siegel H J, Beck N, et al. A comparison study of staticmapping heuristics for a class of meta-tasks on heterogeneous computing systems // Proeeedings of the 8th IEEE Heterogeneous Computing Workshop(HCW,99). IEEE Computer Society Press, 1999: 15-29

[3] Bruna T D, Siegel H J, Beck N. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Journal of Parallel and Distributed Computnig, 2001, 61

[4] He Xiaoshan, Sun Xian-he, von Laszewski G. A QoS Guided SchedulingAlgorithm For Grid Computing[J]. Journal of Computer science and Technology, 2003, 18(4)

(上接第 92 页)

min 算法以及 Sufferage 算法在性能上做了比较。由于这四类算法作为启发式算法，并不能保证每次都能找到最优解决方案，因此测试给出的结果均是多次计算结果的平均值。

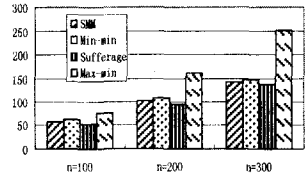

图 3 调度跨度比较 $m=16$

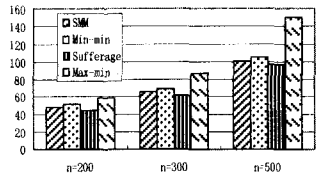

图 4 调度跨度比较 $m=32$

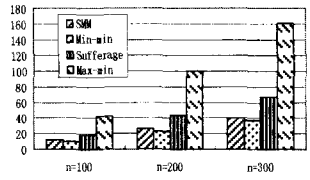

图 5 平均等待时间 $m=16$

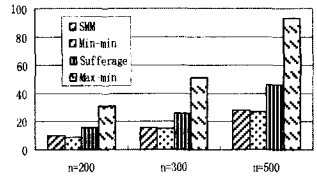

图 6 平均等待时间 $m=32$

测试一共进行了 2 组，图 3 至图 6 分别给出了各自有关调度跨度和平均等待时间的比较结果。第一组测试条件为：$m=16, n=100,200,300$；第二组测试条件为：$m=32, n=200,300,500$。

3.4 测试结果分析

从图 3，图 4 中可以发现，在不同的处理机规模和任务数量下 SMM 算法、Min-min 算法和 Sufferage 算法的调度跨度远优于 Max-min 算法，且 SMM 算法调度跨度低于 Min-min 算法，接近于 Sufferage 算法。同样图 5，图 6 说明了在平均等待时间上，SMM 算法远低于 Sufferage 算和 Max-min 算法。总的来说，SMM 算法做到了调度跨度低与平均等待时间小的统一，因此综合性能略优于其他三个算法。

结束语 本文提出了适用于异构环境的独立任务调度算法：基于任务调度损失的最小最早完成时间算法（SMM 算法）。该算法将任务调度损失引入 Min-min 算法中，在综合权衡任务最早完成时间与任务调度损失的基础上进行调度，克服了 Min-min 算法片面追求局部最优的局限性，使算法更适合于异构环境。经测试表明，该算法在任务调度跨度和平均等待时间这两项指标中均有较优性能。

参 考 文 献

[1] Armstrong R, Hensgen D, Kidd T. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions[A] // Proceedings of 7th IEEE Heterogeneous Computing Workshop(HCW '98)[C]. 1998: 79-87

[2] Maheswaran M, Ali S, Siegel H J, et al. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems // Proceedings of the Eighth IEEE Heterogeneous Computing Workshop: 30-44

[3] Freund R F, Gherrity M, Ambrosius S, et al. Scheduling resourcesin multi-user, heterogeneous, computing environments with SmartNet // Proceedings of Heterogeneous Computing Workshop. 1998: 184-199

[4] Braun T D, Siegel H J, Beck N. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing System. Journal of Parallel and Distributed Computing, 2001: 810-837

[5] Ibarra O, Kim C. Heuristic algorithms for scheduling independent tasks on nonidentical processors. Journal of the ACM, 1977, 77(2): 280-289