网格任务调度算法的研究

郭 创,余 谅
(四川大学 计算机学院,四川 成都 610064)

摘 要:为了提高网格任务调度算法的性能和效率,同时在任务调度过程中让网格资源的负载达到平衡,通过对网格中三种典型调度算法的原理进行分析研究,结合网格计算环境的特点,针对这些典型算法存在的不足,并以这几个经典的调度算法原理为基础,提出了一种适用于网格计算环境的改进算法。通过对改进的算法进行试验分析,发现改进的算法较好地弥补了传统算法中存在的一些不足。最后提出了笔者对网格环境下任务调度算法的一些见解。

关键词:网格;任务调度;完成时间;负载平衡;排序
中图分类号:TP393 **文献标识码**:A **文章编号**:1673-629X(2009)06-0005-04

Research on Algorithm for Tasks Scheduling in Grid

GUO Chuang, YU Liang
(Department of Computer Science, Sichuan University, Chengdu 610064, China)

Abstract: To improve the algorithm for independent tasks group scheduling in the grid system and keep the load balance during the scheduling described and analyzed the theories of three typical algorithms in detail. To resolve the defects on these typical algorithms, proposed an improved algorithm according to the environment over the grid system which based on these typical algorithms. By analyzing data from experiments on these algorithms, it is proved that this improved algorithm has fixed some deficiency on typical algorithms. Finally proposed some viewpoint on the algorithm for tasks scheduling in grid system.

Key words: grid; tasks scheduling; makespan; load balance; sort

0 引 言

网格技术的出现迎合了并行与分布计算技术的发展趋势,它是以资源共享为目的,支持对可计算资源的远程和并发的访问,用高速网络连接的地理上分布的可计算资源所组成的一个具有单一系统映像的高性能计算和信息服务环境[1]。在网格技术飞速发展的同时,由于网格系统由不同的计算机资源组成,网格计算中的任务调度问题也变得越来越重要,一个好的任务调度算法不但要考虑到要使所有任务的最大完成时间(Makespan)的值尽量小,同样也要考虑到整个系统机器间的负载平衡问题[2]。任务调度包括任务映射和调度两个方面:任务映射是逻辑地为每个任务匹配一个最合适的机器,并没有真正地将任务分配到这个机器上;任务调度则是将任务传输到其映射的机器上运行[3]。文中所讲的任务调度是指的前者即任务到资源的映射。

收稿日期:2008-09-28;**修回日期**:2009-02-04
基金项目:四川省科技计划项目资助(2006j13-101)
作者简介:郭 创(1981-),男,硕士研究生,研究方向为网格计算、网络通信、多媒体;余 谅,副教授,硕士研究生导师,研究方向为通信技术、多媒体。

1 任务调度模型

通过数学模型的方法对网格计算中的任务调度问题进行形式化的描述。网格环境中的资源指加入到网格系统中、所有可被共享的资源。所需要执行的任务可以分为计算密集型、通信密集型和计算通信相对平衡型[4]。为了描述问题的简单,文中所指的任务特指计算密集型(任务中计算数据所需要的时间远远大于数据通信所需要的时间)。而且这些任务之间不存在依赖性,是互相独立的,并且是在批模式下进行调度。调度模型定义如下:

(1)参与调度的任务集合为 $M=\{m_1,m_2,\cdots,m_n\}$,其中 m_i 代表的是任务 i;

(2)参与调度的机器集合为 $H=\{h_1,h_2,\cdots,h_m\}$,其中 h_j 代表的是机器 j;

(3)参与调度的机器的相对速度集合为 R,$R=\{R_1,R_2,\cdots,R_m\}$,这里 m 为机器的数量;

(4)定义任务 m_i 在机器 h_j 上的期望执行时间 t_{ij} 为机器 h_j 在不考虑其它负载情况下执行任务 m_i 所需

要的时间；

(5)定义任务 m_i 在机器 h_j 上的期望完成时间 c_{ij} 为任务 m_i 映射到机器 h_j 上执行完的时间；

(6)定义机器 h_j 的期望就绪时间为 r_j，则向量 r 存储了所有机器的期望就绪时间；

(7)据(4)、(5)、(6)的定义有 $c_{ij} = r_j + t_{ij}$；

(8)设全部任务的最大完成时间为 Makespan，它等于任务集合 M 中的全部任务调度完成以后的全部机器的最大期望就绪时间。

2 算法描述

2.1 Min－Min 算法

Min－Min[5~7]算法是一种实现起来很简单的算法，算法的执行时间也很快。算法的思想是首先映射小的任务，并且映射到执行快的机器上。执行过程为：计算要参与映射事件的任务集中每个任务在各个机器上的期望完成时间，找到每个任务的最早完成时间及其对应的机器；从中找出具有最小最早完成时间的任务，将该任务指派给获得它的机器；指派完成后，更新机器期望就绪时间并将已完成映射的任务从任务集合中删除。重复上面的过程，直到所有的任务都被映射完。该算法形式化描述如下：

M 为所有未调度的任务的集合

① 判断任务集合 M 是否为空，不为空，执行 ②；否则跳到步骤 ⑦；

② 对于任务集中的所有任务，求出它们映射到所有可用机器上的最早完成时间 c_{ij}；

③ 根据 ② 的结果，找出最早完成时间最小的那个任务 m_i 和所对应的机器 h_j；

④ 将任务 m_i 映射到机器 h_j 上；并将该任务从任务集合中删除；

⑤ 更新机器 h_j 的期望就绪时间 r_j；

⑥ 更新其他任务在机器 h_j 上的最早完成时间；回到 ①；

⑦ 此次映射事件结束，退出程序。

2.2 Max－Min 算法

Max－Min[5~7]算法非常类似于 Min－Min 算法。同样要计算每一任务在任一可用机器上的最早完成时间，不同的是 Max－Min 算法首先调度大任务，任务到资源的映射是选择最早完成时间最大的任务映射到所对应的机器上。该算法形式化描述如下：

M 为所有未调度的任务的集合

① 判断任务集合 M 是否为空，不为空，执行 ②；否则跳到步骤 ⑦；

② 对于任务集中的所有任务，求出它们映射到所有可用机器上的最早完成时间 c_{ij}；

③ 根据 ② 的结果，找出最早完成时间最大的那个任务 m_i 和所对应的机器 h_j；

④ 将任务 m_i 映射到机器 h_j 上；并将该任务从任务集合中删除；

⑤ 更新机器 h_j 的期望就绪时间 r_j；

⑥ 更新其他任务在机器 h_j 上的最早完成时间；回到 ①；

⑦ 此次映射事件结束，退出程序。

2.3 Max－Int 算法

最大时间跨度优先的算法吸取了前两种算法的优点，除了利用历史调度信息之外，还利用预测信息来减少调度任务的时间。该算法首先计算所有未指派任务在所有可能机器上的预测完成时间，找出每个任务的最小最早完成时间和获得该时间的机器，再找出每个任务的次小最早完成时间，然后计算每个任务的次小最早完成时间与最小最早完成时间的差值即时间跨度，找出时间跨度最大的以及获得该值的任务，再把这个任务指派给获得最小最早完成时间的机器。如此重复，直到所有任务被指派完毕。该算法形式化描述如下：

M 为所有未调度的任务的集合

① 判断任务集合 M 是否为空，不为空，执行 ②；否则跳到步骤 ⑦；

② 对于任务集中的所有任务，求出它们映射到所有可用机器上的最小最早完成时间 C_{ij} 和次小最早完成时间 S_{ij}，然后计算出所有任务的时间跨度 $\mathrm{Int} = S_{ij} - C_{ij}$；

③ 根据 ② 的结果，找出时间跨度最大的那个任务 m_i 和所对应的最小最早完成时间的机器 h_j；

④ 将任务 m_i 映射到机器 h_j 上；并将该任务从任务集合中删除；

⑤ 更新机器 h_j 的期望就绪时间 r_j；

⑥ 更新其他任务在机器 h_j 上的最早完成时间；回到 ①；

⑦ 此次映射事件结束，退出程序。

2.4 基于“排序”思想改进的 S－M－M 算法

前面的三个经典算法都存在各自的不足，Min－Min 和 Max－Min 算法造成系统的负载不平衡，Max－Int[5~7]算法和 Min－Min 算法的 Makespan 偏大。提出一种改进的算法来弥补这些缺点，首先计算每个独立任务在网格系统中的计算机上的平均预测执行时间，然后根据这个平均值对要调度的任务进行降序排序，再把这个序列分成相等的几段，逐步对这几个段采用 Min－Min 进行调度。这样先被调度的任务是平均

执行时间较大的任务,但它不是最大的任务,因为每段内部仍是采用的 Min－Min 算法进行调度。这样既能保证让相对较大的任务先被调度,让它能与其它较小的任务并行执行,又能避免 Min－Min 算法中过多的小任务先被集中调度到性能较好的计算机上造成网格系统中其他性能较差的计算机资源闲置的情况。S－M－M 算法首先根据任务的初始 ETC 矩阵[8](ETC 矩阵是 m 个任务在 n 个不同机器上的预测执行时间矩阵,etc(i,j) 表示第 i 个任务在第 j 台机器上的预测执行时间)进行排序,即根据每个任务在所有机器上的平均预测执行时间将任务排列为一个从大到小的有序序列;然后将这个任务序列分成相同大小的段,先映射大任务段,后映射小任务段。对于每个任务段,仍然使用 Min－Min 算法进行任务调度。算法描述如下:

① 计算每一个任务的排序值 avgetc,计算初始 ETC 矩阵中每一行的平均值:

$$\text{avgetc}_i = \left(\sum_{j=0}^{n-1} \text{etc}(i,j)\right)/n$$

② 根据排序值,降序排列任务集合,形成一个有序序列。

③ 将排序后的任务序列平均分成 4 段(分为 4 段的原因:一方面,分的段数越多机器的负载越趋于均衡,但是段数值过大会使 S－M－M 算法的效率基本和 Min－Min 算法相同,没有达到改善的目的。在选取不同的段数时,算法相对 Min－Min 算法的改进度是不同的,通过试验发现当分段数为 4 达到最高的算法改进度)。

④ 逐步调度各任务段,从排序值大的任务段开始,每一段内部都用 Min－Min 算法调度。

3　实验及结果分析

3.1　实验准备

定义网格系统内有 5 台同构机器,其性能比为 R = {1:3:5:7:8}。试验中使用随机生成的包含 60 个独立的被调度任务的集合 M,任务集合中的任务都是独立运行的任务,采用批模式进行调度。任务数组中的数字代表每个任务在性能为 1 的机器上的期望执行时间。要将这 60 个任务通过不同的算法映射到上面的 5 台同构非均一的机器集合上,然后通过观察试验结果来分析这些算法的性能和特点。

M = {436, 5659, 9247, 4472, 9695, 4920, 146, 5369, 594, 5817, 1042, 6266, 1491, 6714, 1939, 7162, 2388, 7611, 2836, 8059, 3284, 8508, 3733, 8956, 4181, 9405, 4630, 9853, 5078, 303, 5527, 752, 5975, 1200, 6423, 1649, 6872, 2097, 7320, 2545, 7769, 2994, 8217, 3442, 8665, 3891, 9114, 4339, 9562, 4788, 13, 5236, 461, 5684, 910, 6133, 1358, 6581, 1806, 7030}

3.2　实验结果

＊Min－Min 算法:

Makespan = 16725

机器 1 到 5 号映射的任务个数为:[1, 6, 12, 19, 22]

＊Max－Min 算法:

Makespan = 12026

机器 1 到 5 号映射的任务个数为:[5, 8, 12, 17, 18]

＊Max－Int 算法:

Makespan = 14971

机器 1 到 5 号映射的任务个数为:[10, 14, 11, 12, 13]

＊S－M－M 算法:

Makespan = 12729

机器 1 到 5 号映射的任务个数为:[10, 8, 12, 14, 16]

3.3　实验结果分析

从实验结果中可以看出在 Makespan 上,Max－Min 算法最小为 12026,其次是 S－M－M 为 12729,再次是 Max－Int 为 14971,最大是 Min－Min 为 16725。

图 1 描绘出了不同机器在不同算法的情况下分配到的任务个数的情况。

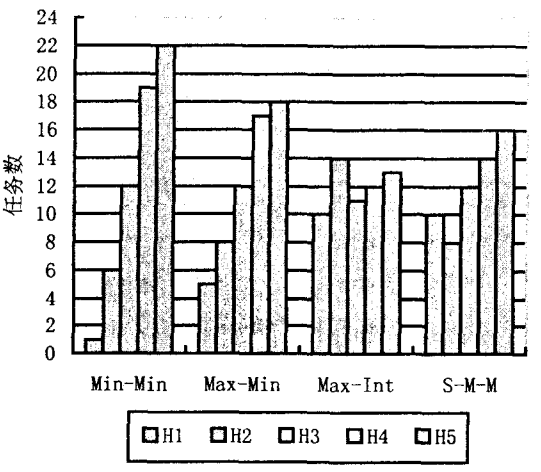

图 1　系统负载分布图

Min－Min 算法中 5 个机器执行的任务数很不平均,机器速度越快的获得的任务数越多,1 号机器速度最慢它只获得了 1 个任务,5 号最快,它获得了最多的任务 22 个,可以看出 Min－Min 算法造成了机器负载的极度不平衡,而且它的 Makespan 也是几个算法中最大的。

Max－Min 算法中 5 个机器的任务数还是不平

均,机器速度越快的获得的任务数越多,1 号机器速度最慢它只获得了 5 个任务,5 号最快,它获得了最多的任务 18 个,可以看出 Max－Min 算法比起 Min－Min 造成的机器负载的不平衡没那么严重,但是它的 Makespan 却是几个算法里面最小的。

Max－Int 算法中 5 个机器的任务数基本是平均的,机器速度最快的获得的任务数不再是最多的了,2 号机器速度适中,它却获得了最多的任务 14 个,可以看出 Max－Int 算法是这几个算法中机器负载最均匀的,但是它的 Makespan 却比 Min－Min 和 S－M－M 都要大。

改进的 S－M－M 算法中 5 个机器的任务数基本接近于平均,平均程度比 Min－Min 和 Max－Min 都要好,比起 Max－Int 还是要差一些,虽然在机器的负载平衡方面它不是最好的,但是它的 Makespan 只比 Max－Min 大一点,在几个算法里面是第二小的。所以可以看出这个改进的算法在追求尽量小的 Makespan 和保持机器的负载平衡方面都有优秀的表现。

图 2 描述了被调度的独立任务组的任务个数在不同数目的情况下各种算法的在 Makespan 性能上的表现,可以看出在同构非均一的网格系统中 Max－Min 算法的 Makespan 在这几个算法中总是保持最小的,Min－Min 算法的 Makespan 总是保持最大的,在大多数情况下 S－M－M 算法的 Makespan 大小都在 Max－Min 和 Max－Int 算法之间,有时候 S－M－M 算法和 Max－Int 算法很接近。总的来说 S－M－M 算法在 Makespan 性能指标上还是比较优秀的。

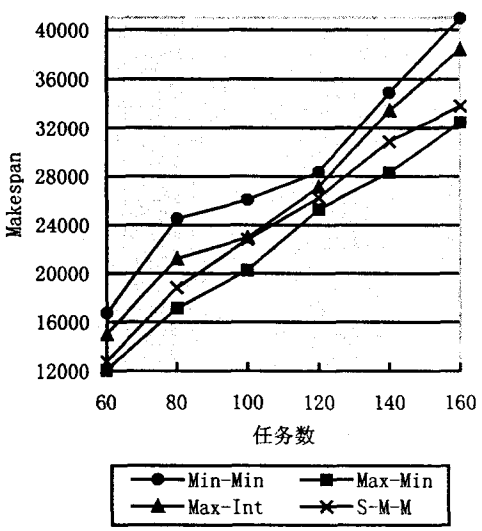

图 2 任务数与完成时间关系图

图 3 描述了被调度的任务集合中任务的数目保持 60 不变时,网格系统中主机数目不同的情况下各种算法的性能表现。当网格系统中的主机数量越来越多时,更多的任务可以在不同的主机上并行执行,这样会减少 Makespan 时间;但是当主机数量多到一定程度时,可以看出每个算法的 Makespan 都趋于一个定值。因为系统中的主机达到一定数目后 Makespan 的值就近似于任务集合中期望执行时间最大的那个任务在它被映射到的那个主机上的运行时间。从图中可以看出 Max－Min 算法在任务数目相同主机数目不断增加时 Makespan 的减少趋势是最明显的,因为在这个算法中大的任务总是先被映射,这样它就可以和更多的小任务并行执行。

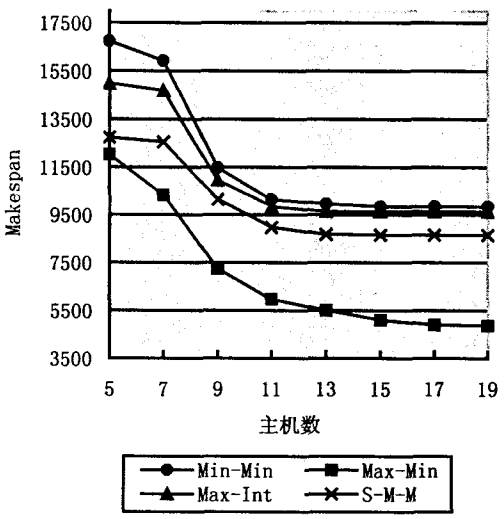

图 3 主机数与完成时间关系图

4 结束语

实验结果证明在同构非均一的系统中 Max－Min 算法在 Makespan 上优于其他算法,但是在保持系统负载均衡上却是 Max－Int 算法最优。Max－Int 算法的特点是可以指定介于最小最早完成时间和最大最早完成时间的任何任务给相应的异构机器,在不同的计算环境都具有良好的适应性,而且它可以实现机器的最优负载平衡,它适合用在对负载平衡要求严格的系统中。文中改进的 S－M－M 算法在追求尽量小的 Makespan 和保持系统负载均衡上都有比较优秀的表现,它很好地解决了 Max－Min 算法造成的系统负载不均衡问题和 Max－Int 算法导致的 Makespan 较大的问题,对传统的几个算法实现了有效的改进,得到了比较显著的性能提升。另外,当被调度的独立任务集合中短任务比长任务多很多时 Max－Min 算法的优势最为明显,因为利用 Max－Min 算法可以实现长任务与多个短任务的并发执行。而在全异构的网格系统中,Min－Min 算法要优于 Max－Min 算法。但是随着异构环境中同类节点的增加,Max－Min 算法的性能逐

(下转第 12 页)

点到目的节点的路径。

每个节点周期性地发送广播蚂蚁，广播蚂蚁不留下信息素，相当于传统路由算法中的 hello 报文。当网络拓扑发生变化时，由于离开网络的节点将无法把广播蚂蚁返回，则发出广播蚂蚁的节点认为该节点已经离开此路径。这时此节点将发送路径维护蚂蚁来通知活动节点的前序列节点查找该节点的路由表以寻找另一条次优路径传输数据。如果该级前序列节点都找不到目的节点的路径，则继续从上一级前序列节点查找。如果上述过程未找到到达目的节点的路径，则源节点再次发起一个路由请求。由于维护过程采用了简单高效的消息传递蚂蚁进行操作，在网络快速变化的情况下仍能够取得较好的路由效果。

5 结束语

在分析移动自适应网特点的基础上，提出了一种基于蚁群算法的路由算法。该算法结合了先应式路由和按需式路由的两方面优点，具有快速的收敛性，整个的路由过程中具有自适应性、自治性、交互性等特点，路由维护简单，占用网络资源小，能显著提高移动自适应网性能，具有很好的发展前景。

参考文献：

[1] David R, Ignas G N. Ad Hoc networking in future wireless communications[J]. Computer Communications, 2003, 26(1): 36－40.

[2] 鲍 晶. 无线 Ad Hoc 网络技术探讨[J]. 计算机技术与发展, 2006, 16(7): 201－203.

[3] Kone V, Nandi S. QoS Constrained Adaptive Routing Protocol For Mobile Adhoc Networks[C]//In: Proceedings of the 9th Information Technology International Conference(ICIT'06). Mumbai, India: IEEE Computer Society Press, 2006: 40－45.

[4] 段海滨. 蚁群算法原理及其应用[M]. 北京：科学出版社, 2005.

[5] Kwang M S, Weng H S. Ant colony optimization for routing and load－balancing: survey and new directions[J]. IEEE Transactions on Systems, Man and Cybernetics: Part A, 2003, 33(5): 560－572.

[6] 孙艳歌，刘 明，许芷岩. Ad Hoc 网络中基于双向收敛蚁群算法的 QoS 路由算法[J]. 微电子学与计算机, 2006, 23(10): 1－3.

[7] Abuzanat H, Trouillet B, Toguyeni A. Routing Fairness Model for QoS Optimization in Wireless Network[C]//In: Proceedings of the Second International Conference on Sensor Technologies and Applications(SENSORCOMM 2008). Cap Esterel, France: IEEE Computer Society Press, 2008: 776－781.

[8] 潘达儒，袁艳波. 一种基于 AntNet 改进的 QoS 路由算法[J]. 小型微型计算机系统, 2006, 27(7): 1169－1174.

[9] 刘卫彪，张修如，朱光辉. 一种基于节点位置信息的 Ad Hoc 网络路由算法[J]. 计算机技术与发展, 2007, 17(10): 158－161.

[10] 王合义，丁建立，唐万生. 基于蚁群优化的路由算法[J]. 计算机应用, 2008, 28(1): 7－8.

(上接第 8 页)

渐接近 Min－Min 算法，甚至有可能超过 Min－Min 算法[9]。文中所讨论的这些算法各有各的特点，没有绝对的最优或最差，它们适用于各种网格系统环境和系统需求。

参考文献：

[1] 罗秉安，张立臣. 网格技术及其应用[J]. 微机发展(现更名：计算机技术与发展), 2002, 12(6): 2－4.

[2] 秦金磊，朱有产，李玉凯. 基于网格计算的关键技术研究[J]. 计算机技术与发展, 2006, 16(11): 71－73.

[3] 桂小林. 网格技术导论[M]. 北京：北京邮电大学出版社, 2005.

[4] 崔玉宝，李建义，薛桂香. 一种改进的启发式网格任务调度算法[J]. 微型电脑应用, 2006(5): 6－7.

[5] Maheswaran M, Ali S, Siegel H J, et al. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems [C]//8th Heterogeneous Computing Workshop. Puerto Rico: IEEE Press, 1999: 30－44.

[6] Maheswaran M, Ali S, Siegel H J, et al. A Comparison of Dynamic Strategies for Mapping a Class of Independent Tasks onto Heterogeneous Computing Systems[R]. School of Electrical and Computer Engineering, Purdue University, in preparation, 1999.

[7] Wu Min You, Shu Wei, Zhang Hong. Segmented Min－Min: A Static Mapping Algorithm for Meta－tasks on Heterogeneous Computing Systems[EB/OL]. [2004－07－10]. http://csdl2. computer. org/dl/proceedings/hcw/2000/0556/00/05560375.pdf.

[8] Hamscher V, Schwiegelshohn U, Streit A. Evaluation of job－scheduling strategies for grid computing[C]//Proceedings of 7th Int'l Conf on High Performance Computing. Berlin, Heidelberg: Springer－Verlag, 2000: 191－202.

[9] 陈宇寒. 网格计算技术研究[J]. 计算机技术与发展, 2008, 18(5): 52－54.