文章编号: 1001-9081(2006)01-0061-04

基于优先级和优化完成时间的网格调度算法

何 岩[1], 李肯立[1], 石岩然[2], 刘晓玲[1], 王 颖[1]
(1. 湖南大学 计算机与通信学院, 湖南 长沙 410082;
2. 南京工业大学 理学院, 江苏 南京 210009)
(heheyan@126.com)

摘 要: 网格由大量的异构资源组成, 具有复杂性、动态性和自治性特点。高效的网格调度算法可以充分利用网格系统资源, 提高网格处理应用程序的能力。Min-min算法是一个简单、快速、有效的调度算法, 但由于总是先分配小任务而不能确保负载平衡。文中首先对网格系统中任务的数据传输和执行进行分析, 计算并优化Min-min算法的任务完成时间, 再根据任务需求赋予任务优先级, 通过优先级安排任务调度, 提高算法负载平衡能力, 最后在上述分析基础上提出 POTE Min-min(Priority and Overlap Transmission and Execution Min-min)调度算法。

关键词: 网格; Min-min算法; 完成时间; 优先级
中图分类号: TP393 文献标识码: A

Scheduling algorithm based on the priority and improved completion time in grid

HE Yan[1], LI Ken-li[1], SHI Ku-i ran[2], LIU Xiao-ling[1], WANG Ying[1]
(1. School of Computer and Communications, Hunan University, Changsha Hunan 410082, China;
2. School of Sciences, Nanjing University of Technology, Nanjing Jiangsu 210009, China)

Abstract: Grid system consists of a wide variety of geographically distributed resources and these resources are heterogeneous, geographically distributed and dynamically available. High efficient scheduling algorithm would be able to increase throughput, maximize system utilization, and fulfill economical system and user constraints. Min-min algorithm is a simple and fast algorithm, and able to deliver good performance, but with the drawback of limitation of load balance. In this paper the transmission and execution of jobs were analyzed first. Then the completion time of the Min-min algorithm was computed and improved based on the schedule model. Furthermore, in order to improve the load balance of the Min-min algorithm, priority was assigned to the jobs according to different schedule limitations, and jobs were scheduled based on these priorities. Finally the POTE Min-min (Priority and Overlap Transmission and Execution Min-min) algorithm was proposed based on the analysis.

Key words: Grid; Min-min algorithm; completion time; priority

0 引言

网格[1]从广义上说是一个集成的计算和资源环境。作为一个分布式计算平台, 网格能够解决科学、工程和经济中的大规模计算问题和数据密集型问题, 能够实现地理上广泛分布的高性能计算资源、信息资源、应用系统、服务系统、组织、人员等各种资源的共享与聚合[2]。网格为计算机和网络之间提供了标准和中间件, 允许个人、研究机构、公司和组织相互之间充分的共享资源, 动态地将空闲计算资源提供给其他远端用户[3]。由于各种资源、管理机制、用户和应用程序间存在大规模的异构性, 网格资源的高效管理和任务的有效调度已成为网格研究的重要内容之一[2]。

调度问题一般可分为两个基本步骤[4]: 首先在空间上对计算和数据进行分配, 包括选取给定任务所需的资源组合、将任务交给这些资源执行、分配相关的数据和计算等; 然后在时间上为计算和通信进行排序。虽然网格调度可以借鉴传统高性能计算中的调度策略和技术, 但因为任何一个网格调度器都无法对所有的网格资源进行管理, 而只能是一定范围内的网格资源, 使得对网格任务的调度更加复杂。另外, 网格资源的动态变化性也加大了对网格资源调度的难度。

网格调度目前多采用启发式调度算法。Min-min[5-7]算法选取每个任务的最小完成时间, 再从所有最小完成时间中选取最小的完成时间进行任务和计算资源匹配。Max-min[5,6]算法在得到每个任务的最小完成时间后, 选取最大的完成时间进行任务和计算资源匹配。Suffrage[5,8]算法计算每个任务的最小完成时间和次小完成时间的差值, 选取所有任务中差值最小的任务和计算资源。SA[6]算法将传统的模拟退火算法应用到网格调度中, 采用迭代技术得到一个可能被

收稿日期: 2005-07-14 修订日期: 2005-09-07
基金项目: 国家自然科学基金资助项目(60273075); 教育部重点项目(105128); 计高技[2000]2034号
作者简介: 何岩(1979-), 女, 吉林人, 硕士研究生, 主要研究方向: 并行处理、网格计算; 李肯立(1971-), 男, 湖南人, 副教授, 博士, 主要研究方向: 并行处理、网格计算; 石岩然(1971-), 男, 湖南人, 博士研究生, 主要研究方向: 运筹学与调度算法; 刘晓玲(1978-), 女, 湖南人, 硕士研究生, 主要研究方向: 并行处理; 王颖(1974-), 男, 湖南人, 硕士研究生, 主要研究方向: 并行处理.

接受的任务 资源匹配对。AppLes[9]基于有效的协同数据定位和适应性调度提出 Xsuffrage算法。Nimrod[7]基于经济预算网格模型，提出根据运行时间限制和经济预算限制的调度算法。这些算法基于不同调度模型、从不同角度研究了调度问题。在众多算法中，Min-min算法是一个简单、快速、有效的算法，但由于该算法总是先分配小任务而不能确保负载平衡。针对 Min-min算法的缺陷，Segmented Min-min[10]将所有任务根据其预期完成时间分为 N 段，从完成时间最长的任务所在段开始按照 Min-min 算法调度。QoS Guided Min-min[11, 12]在调度过程中考虑了任务对计算节点数据处理能力的需求，对需要高数据传输能力计算节点的任务先进行调度。上述文献虽然从某一角度改进了 Min-min算法，但都没有对任务完成时间的计算进行深入分析，且没能全面考虑任务自身需求对调度的影响。

基于这一事实，本文提出一种 POTE Min-min算法。该算法首先根据网格系统中与任务匹配的计算节点上没有存储所需的数据，需要进行数据传输的特点，将任务的执行分为数据传输和计算两个部分，通过对 Min-min 调度算法的任务完成时间进行计算和优化，达到降低任务完成时间、提高网格系统的任务吞吐率的目的。继而，算法采用优先级反映任务需求对调度的影响，将任务按照优先级划分为多个子集合，对各子集合中的任务按照优化完成时间的 Min-min算法进行调度，达到提高算法负载平衡能力的目的。

1 调度算法基本框架和调度模型

1.1 调度算法基本框架

本文沿用文献[5]的概念，将调度算法称为 schedule( )。网格的动态性特点要求任务与资源的匹配关系随着资源和任务的变化及时调整，schedule( )必须被重复调用。schedule( )发生的时间称为调度事件，假设每个调度事件发生时，已知：(1)网格当前可用数据资源节点数、计算资源节点数、网络带宽、计算资源的处理能力等；(2)输入文件的大小和所在位置；(3)运行中的任务和文件传输列表。调度事件的发生通过对资源性能的评估进行预测。

schedule( )的基本框架如下：

```
schedule( ){
    compute the next scheduling event
    create a schedule list
    foreach unscheduled tasks
        compute an estimate of its completion time
    select a task/host pair and add it into the schedule list
    until each host has been assigned enough work and all task have
    been scheduled
    schedule the tasks according to the schedule list
}
```

第(1)行计算下次调度事件，在计算中需要考虑网格资源的变化增加或减少调度事件的发生频率。调度事件的发生频率对网格性能有很大的影响，虽然高调度频率能够得到好的适应性，对于不稳定的网格环境也是必须的，但频率越高，调度开销越大。文献[5]给出一种计算调度事件发生频率的方法，但相对简单，我们将在以后对此进行深入研究。第(2)行初始化调度列表。第(3)行为每个任务计算预期完成时间，然后在第(4)行根据不同的规则选择任务 节点对，将该任务和节点加入调度列表中，重复(3)和(4)直到所有任务已经分配。第(6)行按照调度表进行任务调度。目前已有的各种调度算法主要针对步骤(3) ~(5)进行研究。

1.2 调度模型

任务集 $T = \{T_i,\ i = 1, 2, \cdots, n\}$ 由 n 个相互独立的任务组成[6, 13]，任务之间的相互独立表示在任务之间没有数据通信或数据依赖关系。假设任务的输入数据构成输入文件，输出数据构成输出文件，一个输入文件可以同时提供给一个或多个任务，而一个任务只有一个输出文件，在任务调度之前输入文件和输出文件的大小是已知的。

网格系统由 k 个可用数据存储节点和 m 个可用计算节点构成[3]，如图 1所示。数据存储节点可以是存储设备、分布数据库等。计算节点可以是单个处理器的工作站、超级计算机、集群等，各计算节点的计算能力和内部通信能力可能不同。为简单起见，将分布在一个局域网内的数据库用能力相当的存储设备代替，集群由单个的处理能力相当的处理器代替。假设一个任务所需数据都存储在一个数据存储节点中，数据存储节点之间没有数据通信，则网格系统可以构成一个全相连的二分图(如图 2所示)，相同集合内部的节点之间互不相连，不同集合中的任意两个节点都是相连的。在这种网格调度模型中，每个数据存储节点都可以将数据传输到任意一个计算节点上，各数据存储节点和计算节点之间的带宽是不同的。

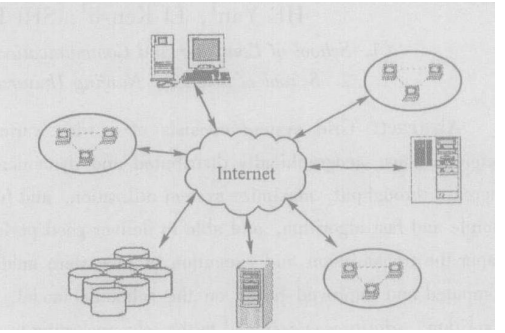

图 1 网格系统

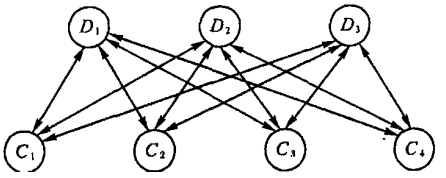

图 2 网格系统抽象模型

上述模型的形式定义如下：

k 个可用数据存储节点由 $D_1, D_2, \cdots, D_k$ 表示，m 个可用计算节点由 $C_1, C_2, \cdots, C_m$ 表示，$L_{ij},\ i = 1, 2, \cdots, k,\ j = 1, 2, \cdots, m$ 表示 D_i 与 C_j 之间的连接，带宽为 BW_{ij}。

假设所有数据存储节点可以同时与所有计算节点进行数据传输；数据存储节点与计算节点之间的数据传输是双向的，且可以同时进行；每个计算节点都有一个足够大的存储器以暂存任务所需的输入文件和任务产生的输出文件。

2 改进的 Min-min调度算法

本节首先对上述模型中任务完成时间进行分析，给出 OTE Min-min算法，再通过对任务优先级分析提出 POTE Min-min算法。

2.1 任务完成时间

在第 1节的调度模型中，假设所有的数据存储节点对所有的计算节点都是同时可用的，一个存储节点可以将该节点

上的数据同时传输给多个计算节点, 忽略数据在存储节点上的查找时间, 所以在下面的分析中, 只考虑数据传输和任务执行时间。

为描述方便, 采用如下符号 (其中 $i=1,2,\cdots n$, $j=1,2,\cdots m$, $l=1,2,\cdots k$):

CT_{ij}: 任务 T_i 在计算节点 C_j 上的完成时间。任务的完成时间包括任务等待时间、任务输入文件的传输时间、任务的执行时间和任务输出文件的传输时间。

ET_{ij}: 任务 T_i 在计算节点 C_j 上的执行时间。

$Dinput_i$: 任务 T_i 输入文件的大小。

$Dapp_i$: 任务 T_i 应用程序代码的大小。

$Doutput_i$: 任务 T_i 输出文件的大小。

BW_{lj}: 任务 T_i 输入文件存储节点 D_l 与计算节点 C_j 之间的带宽。

$TransStartTime_{ij}$: 任务 T_i 所需数据向计算节点 C_j 传输数据的开始时间。

FTT_{ij}: 任务 T_i 在 C_j 上运行输入数据传输完成时间。

T_{i-1}: 同分配在 C_j 且运行在 T_i 之前的任务。

在上述符号约定基础上, 任务 T_i 在计算节点 C_j 上的完成时间为:

$$CT_{ij}=FTT_{ij}+ET_{ij} \tag{1}$$

$$FTT_{ij}=TransStartTime_{ij}+\frac{Dinput_i+Dapp_i+Doutput_i}{BW_{lj}}=CT_{i-1,j}+\frac{Dinput_i+Dapp_i+Doutput_i}{BW_{lj}} \tag{2}$$

在 Minmin算法中, 选取的最小完成时间任务 i计算节点匹配对为:

$$(T_s,C_s)=\min_{\substack{0<i\leqslant n\\0<j\leqslant m}}\{CT_{ij}\} \tag{3}$$

因此基于完成时间的 Minmin算法可描述如下:

```
Minmin{
(1)   while T is not empty
(2)     foreach task Ti
(3)       based on formula (1) and (2), compute the completion
            time CTij
(4)       (Ti, Ci(1)) = ∑ min{CTij}   (0<j≤m)
(5)     endfor
(6)     (Ts, Cs) = ∑ min{CTi,j(1)}   (0<i≤n)
(7)     T = T - {Ts}
(8)   endwhile
}
```

该算法的一次执行对当前所有未调度的任务进行分配(第 1行)。在执行过程中, 对于 T中每一个任务(第 2行), 通过(1)式和(2)式计算其在所有可用计算节点上的完成时间(第 3行), 再通过(3)式取得该任务在所有节点上的最小完成时间(第 4行), 构成该任务的匹配对 $(T_i, T_{i(1)})$。第 6行在所有任务的最小匹配对中根据公式(3)选取最小值对应的任务作为本次调度的任务, 将其从 T中删除(第 7行)。

2.2　OTEMinmin算法

Minmin映射任务的顺序如下: 假设 T_a 为第一个通过 Minmin映射到计算节点集中的任务, C_a 是完成这个任务时间最短的计算节点, 将任务 T_a 映射到 C_a 后改变 C_a 对于其他未分配任务的可用状态, 将 C_a 的下次可用时间增加 CT_{ai}, 再进行下一最小完成时间任务 节点匹配对的查找。由于在任务分配中, 计算能力高的计算节点可能被分配多个任务, 而任务完成时间由数据传输时间和执行时间两部分构成, 且各数据存储节点和计算节点的连接互不影响, 所以可以对分配在同一计算节点多个任务的数据传输时间和执行时间进行重叠, 以减少所有任务的总完成时间。即在 T_{i-1} 数据传输完成后, 可以马上进行 T_i 的数据传输, 而无需等待 T_{i-1} 任务的执行和结果输出。由于数据存储节点和计算节点之间的数据传输是双向的, 且任务之间相互独立, 所以 T_{i-1} 计算结果的输出对 T_i 数据输入和执行没有影响。则任务 Ti的传输完成时间如下:

$$FTT_{ij}=TransStartTime_{ij}+\frac{D_{input}+D_{app}}{BW_{kj}}=FTT_{i-1,j}+\frac{D_{input}+D_{app}}{BW_{kj}} \tag{4}$$

将任务 T_i 的数据传输时间与任务 T_{i-1} 的执行时间重叠后, 任务 T_i 必须满足以下两个条件才能开始执行: (1) T_{i-1} 任务执行完成; (2) T_i 的数据全部传输完成。所以任务 T_i 的完成时间为:

$$CT_{ij}=\max\{FTT_{ij},ET_{i-1,j}\}+ET_{ij}+\frac{D_{output}}{BW_{kj}} \tag{5}$$

由于任务 T_i 的执行必须满足两个条件, 所以可能会出现与任务 T_i 匹配的计算节点等待 T_i 输入数据的传输的情况, 从而增加了总任务的完成时间。为了减少任务的等待时间, 在根据(3)式、(4)式和(5)式得到任务调度表后, 再对同一计算节点上的任务采取局部调整, 将执行时间最长的任务安排在最早进行。因为分配在同一计算节点的任务, 无论执行顺序如何, 总执行时间和总数据传输时间是不变的, 将执行时间最长的任务提前执行则可以在上一任务执行时, 以最高概率将下一任务所需的数据传输到计算节点, 从而减少下一任务执行的等待时间。

根据上述分析, OTEMinmin算法描述如下:

```
OTEMinmin{
Step 1:  initialize
  schedule list is empty
  TransStartTime_ij = 0 (0< i≤ n, 0< j≤ m)
Step 2:  construct the schedule list
  while T is not empty
    for each task Ti
      based on formula (1) and (2), compute the completion time
        CTij
      (Ti, Ci(1)) = ∑ min{CTij}   (0<j≤m)
    endfor
    (Ts, Cs) = ∑ min{CTi,j(1)}   (0<i≤n)
    fill (Ts, Cs) into the schedule list
    T = T - {Ts}
    TransStartTime_is = FTTss
  endwhile
Step 3:  adjust the schedule list
  foreach Cj
    sort the tasks in decreasing order by execution time ETij
    reschedule the tasks by the order
    compute the completion time of each task
  endfor
}
```

该算法分三步进行, 首先初始化调度列表和各任务在各节点开始传输数据的时间。在第 2步骤中通过公式(4)和(5)计算任务完成时间, 再根据公式(3)选取最佳任务 节点

匹配对，将其加入调度列表中，然后修改对应节点上所有任务的开始传输数据时间，重复操作直到所有待调度任务都已被分配。步骤 3对于所有节点上分配的任务进行内部调整，将任务根据任务执行时间的降序次序进行调度，重新计算各任务完成时间。

2.3 POTE Min-min算法

任务自身需求对调度有重要影响。例如，某些任务因为在计算过程中需要进行大量数据交换而对计算节点内部网络传输能力有较高要求，而另一些任务不需要计算节点的高传输能力，当网格系统中可用高传输能力计算节点较少时，在调度中如果先将不需要高传输能力的任务分配给高传输能力的计算节点，则需要高传输能力的任务必须等待，从而降低了系统的执行效力。另外，如果大部分任务预期完成时间很短而少数任务预期完成时间相对较长，OTE Min-min算法会先分配短任务，而当长任务运行时其他节点都处于闲置状态，不能提供较好的负载平衡。

POTE Min-min算法在OTE Min-min算法的基础上通过赋予任务优先级的方法解决 OTE Min-min算法中存在的问题。任务优先级可以根据任务对计算节点数据传输能力的要求、任务大小、任务花费、任务完成时间限制等不同需求进行确定。设定任务优先级后，将所有任务根据优先级的高低划分为多个子集合，优先级高的子集合中任务先于优先级低的子集合中任务进行调度，子集合内部的任务按照 OTE Min-min算法进行调度。

POTE Min-min算法描述如下：

```
POTE Min-min {
(1)   compute the priority of each task based on some requirement
(2)   sort the tasks into a list in decreasing order of their priority
(3)   partition the tasks evenly into q parts and each part compose a
        sub aggregate T_i
(4)   for each T_i
(5)     perform OTE Min-min algorithm
(6)     endfor
}
```

算法中第(1)行首先根据任务需求赋予任务优先级，然后在第(2)行中将所有任务根据优先级降序排列，构成任务列表。第(3)行将任务列表中的任务分为 q个子集合，(4)~(6)对于所有子集合中的任务按照 OTE Min-min算法进行调度。

子集合数目 q对于算法的整体性能有很大影响，子集合数目越多，越能细化任务优先级，有利于优先级高的任务尽早执行，但反过来，子集合数目的增加会丧失 Min-min算法的优点。q值由不同任务需求决定。当以任务 QoS作为任务优先级时，采用与文献[11]类似的方法将任务分为高 QoS需求和低 QoS需求两个子集合。在以任务大小作为任务优先级时，根据文献[12]的分析，选取合适的 q值为 4或 5。其他情况下 q值的确定将在以后进行深入研究。

3 性能分析与比较

采用网格模拟工具 GridSim 对新算法进行评价[7]。GridSim由澳大利亚墨尔本大学开发，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，达到控制网格资源的使用的目的。GridSim 在 SimJava的基础上开发，提供了丰富的函数库以支持模拟网格环境中的异构资源、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，通过消息事件进行通信。模拟结束后，可以调用库函数来收集各种模拟的统计数据。

我们模拟了一些与文献[2]类似的不同特性的时分共享和空间共享的资源，如 Compaq公司的 AlphaServer ES40、Sun公司的 SunNetra 20等。模拟中 PE表示处理器的执行能力，PE以 SPEC CPU(INT)2000为基准值。GridSim 将任务封装为 Gridlet，Gridlet的参数有任务长度(以 MI为单位)、任务输入输出文件大小(以字节为单位)，任务长度表示任务在 100M IPS处理器上运行所需的时间。

测试中计算节点数是存储节点数 3倍，计算节点数分别为 50、100和 150，任务数分别为 50、100和 150，分别取 5次测量结果作均值。表 1和表 2分别为根据 Min-min算法和 OTEM 算法进行任务调度的完成时间(makespan)。

表 1 Min-min算法 makespan

资源数	任务数		
	50	100	150
50	3 770	6 662	9 902
100	2 631	4 650	6 609
150	1 365	2 354	3 380

表 2 OTE Min-min算法 makespan

资源数	任务数		
	50	100	150
50	3 653	5 833	8 215
100	2 594	4 475	6 077
150	1 348	2 340	3 233

从表中可以看出，当任务数小于或与计算节点数相近时，OTE Min-min算法性能与 Min-min算法接近，这是因为对于所有任务有足够的计算节点，当计算节点之间计算能力相近时，每个计算节点上分配一个任务，此时 OTE Min-min算法性能与 Min-min算法近似。但当任务数与计算节点数之比较高时，OTE Min-min算法性能比 Min-min算法有较大提高，此时一个计算节点被分配了多个任务，OTE Min-min算法将分配在同一计算节点上任务的传输时间与执行时间重叠，从而减少任务的总完成时间。在理想情况下，任意任务的执行都可以在计算节点空闲后马上开始，则性能提高率为传输时间在任务完成时间中所占百分比。

我们以任务的 QoS划分任务优先级测试 POTE Min-min算法。将 QoS需求大的任务构成高优先级子集合，其余任务构成优先级低的子集合。在计算节点数为 100、存储节点数为 30、任务数为 150的测试条件下，分别对 QoS需求高的任务占总任务 80%、50%和 20%三种情况进行测试，POTE Min-min算法结果如表 3。

表 3 三种情况下 OTE Min-min和 POTE Min-min的 makespan

	OTE Min-min	POTE Min-min	Improvement
80%	5 825	5 289	9.21%
50%	6 105	5 485	10.11%
20%	6 304	6 238	1.05%

从表中可以看出，根据 QoS划分子集合并在各子集合内部使用 OTE Min-min算法减少了任务的总完成时间，进一步改善了算法性能。当总任务中 QoS任务占多数时，总完成时间减少 9.21%，当 QoS任务与非 QoS任务数量相当时，总完成时间减少 10%左右，而当 QoS任务较少时，性能改进约为 1%。

(下转第 69页)

对任务迁移率等不同方面进行测试比较。模拟的网格环境由 20个集群组成 4个局域,初始信任关系值均设为 1,每个集群上有各自的数据存储系统,每个集群的 CPU速率(计算能力)在 500MHz到 800MHz之间随机给定,局域内部集群之间的通信带宽设为 100Mbps,局域之间的通信带宽设为 1Mbps。对每一次试验都进行了 100次仿真,试验依次使用 100、150和 200个随机产生的任务,每个任务随机从某个集群上提交到网格系统,其 QoS需求随机给定,且必须存取一个数据源来获得输入数据,每个任务的运行时间已知。每一次仿真集群的出错个数设定为集群数的 20%(即 20×20% = 4个),对任务迁移采用第二种方法且不考虑迁移开销,最后计算这 100次仿真的平均值作为结果。实验结果如图 4所示。

图 4(a)显示对任务 QoS需求进行考虑调度的 TrustMin-min算法比不考虑任务 QoS需求的 Minmin算法执行效果好,随着任务的增加,TrustMin-min算法的 makespan增加速度比 Minmin算法稍慢。图 4(b)显示随着仿真次数的增加,信任机制的建立,TrustMin-min算法的任务迁移率逐渐减小,而 Minmin算法的任务迁移率则基本保持不变。

4　结语

网格计算中的任务调度问题是一个非常困难的 NP完全问题。本文提出了一种基于信任机制的动态任务调度模型,并使用改进的基于信任机制的 TrustMin-min算法进行任务调度。与 Minmin相比,它可以更合理的对任务进行调度,能够尽可能地避免将任务调度到不稳定的局域或资源节点,加强了网格计算的有效性。因此,基于信任机制的调度模型和策略是一种比较有效的任务调度解决方案。

参考文献:

[1] BU GY,XU ZW. A Grid System Theoretical Model[A]. Proc of HPCAsia 2001[C]. 2001.

[2] LUO JZ,JI P,WANG XZ,*et al*. Resource management and task scheduling in grid computing[A]. CSCWD 2004[C]. 2004. 431-436.

[3] BRAUN TD,SIEGEL H J,BECK N. *et al*. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems[J]. Journal of Parallel and Distributed Computing,2001,61(6):810-837.

[4] CHEN HT,MAHESWARAN M. Distributed dynamic scheduling of composite tasks on grid computing system[A]. Parallel and Distributed Processing Symposium,Proceedings International,IPDPS 2002[C]. 2002. 88-97.

[5] MIN R,MAHESWARAN M. Scheduling co-reservations with priorities in grid computing system[A]. Cluster Computing and the Grid 2nd IEEE ACM International Symposium CCGRID 2002[C]. 2002. 250-251.

[6] MDS documentation[EB/OL]. http://www.globus.org/mds.

[7] RICH W. Dynamically forecasting network performance using the network weather service[J]. Journal of Cluster Computing,1998,1(1):119-132.

[8] 李东升,李春江,肖依,等. 数据网格环境下一种动态自适应的副本定位方法[J]. 计算机研究与发展,2003,40(12):1775-1780.

[9] WU M,SUN XH. A general self-adaptive task scheduling system for non-dedicated heterogeneous computing[A]. Cluster Computing,2003. Proceedings 2003 IEEE International Conference on 2003[C]. 2003. 354-361.

[10] RITCHIE G,LEVINE J. A Fast,Effective Local Search for Scheduling Independent Jobs in Heterogeneous Computing Environments[A]. Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group[C]. PLANSIG 2003[C]. 2003.

(上接第 64页)

4　结语

本文将网格环境中任务的执行分成数据传输和计算两个部分,对任务完成时间进行详细的分析,给出减少任务完成时间的方法,并根据任务的需求指定任务优先级,然后通过优先级对任务进行调度的算法,该算法降低了任务的总完成时间,能够提供较好的负载平衡。今后的工作将进一步研究调度事件的发生频率对网格性能的影响和根据任务的执行价格决定任务调度的策略。

参考文献:

[1] FOSTER I,KESSELMAN C. The Grid:Blueprint for a Future Computing Infrastructure[M]. Morgan Kaufmann Publishers,USA,1999.

[2] BUYYA R,MURSHED M. A Deadline and Budget Constrained Cost-Time Optimisation Algorithm for Scheduling Task Farming Applications on Global Grids[R]. Technical Report,Monash University,March 2002.

[3] VISWANATHAN S,VEERAVALLI B. Design and Analysis of a Dynamic Scheduling Strategy with Resource Estimation for Large Scale Grid Systems[A]. Proceedings of the Fifth IEEE ACM International Workshop on Grid Computing[C]. GRID'04[C]. 2004.

[4] 都志辉,陈渝,刘鹏. 网格计算[M]. 北京:清华大学出版社,2002.

[5] CASANOVA H,LEGRAND A,ZAGORODNOV D,*et al*. Heuristics for scheduling parameter sweep applications in Grid environments[A]. Proc. of the 9th Heterogeneous Computing Workshop[C]. HCW'2000[C]. Cancun,Mexico,2000. 349-363.

[6] MORENO R. Job scheduling and Resource Management Techniques in Dynamic Grid Environments[A]. 1st European Across Grids Conference[C]. 2003.

[7] BUYYA R,ABRAMSON D,GIDDY J. Nimrod/G:An Architecture for a Resource Management and Scheduling System in a Global Computational Grid[A]. Proceedings of the HPC ASIA'2000[C]. 2000.

[8] YARKHAN A,DONGARRA J. Experiments with Scheduling Using Simulated Annealing in a Grid Environment[A]. Workshop on Grid Computing[C]. 2002.

[9] CASANOVA H,OBERTELLI G,BERMAN F,*et al*. The AppLeS Parameter Sweep Template:User-Level Middleware for the Grid[A]. the super computing conference[C]. 2000.

[10] WU MY,SHU W,ZHANG H. Segmented min-min:a static mapping algorithm for meta-tasks on heterogeneous computing systems[A]. Heterogeneous Computing Workshop,[C]. HCW 2000[C]. Proceedings 9th[C]. 2000. 375-385.

[11] HE XS,SUN XH,VON LASZEWSKI G. QoS Guided Min-Min Heuristic for Grid Task Scheduling[J]. The Journal of Computer Science and Technology,2003,18(4):442-451.

[12] MUSUNOORI S[B],ELIASSEN F. Richard Staehli,Simula Research Laboratory. QoS-Aware Component Architecture Support for Grid[A]. Proceedings of the 13th IEEE International Workshops on Enabling Technologies:Infrastructure for Collaborative Enterprises[C]. WET ICE'04[C]. 2004. 277-282.

[13] ROSENBERG AL,YURKEWYCH M. Guidelines for Scheduling Some Common Computation-Dags for Internet-Based Computing[J]. IEEE Trans. Computers,2005,54(4):428-438.