文章编号:1000-582X(2010)10-102-08

网格负载均衡策略及其蚁群优化算法

陈乙雄[1],吴中福[1],朱郑州[2]
(1. 重庆大学 计算机学院,重庆 400044;2. 北京大学 信息科学技术学院,北京 100871)

摘　要:以重庆大学 CampusGrid 建设和加入 ChinaGrid 的发展规划为背景,研究了多网格环境中出现共用节点(即同时为多个网格系统服务的节点)时资源利用率下降问题,并针对该问题提出了以提高资源利用率为优化目标的负载均衡算法。主要分为问题模型建立、算法设计、以及实验评估 3 个部分。提出的算法能较好解决该问题,并考虑了网络通信开销对算法执行效果的影响。实验表明,提出的算法能有效防止网格中出现共用节点时资源利用率的下降,并对网格动态变化的特性具有较强的适应能力。

关键词:网格;负载均衡;蚁群优化;任务调度
中图分类号:TP393　　**文献标志码**:A

Load balancing strategy and ant optimization algorithm for grids

CHEN Yi-xiong[1], *WU Zhong-fu*[1], *ZHU Zheng-zhou*[2]
(1. College of Computer Science, Chongqing University, Chongqing 400044, P. R. China;
2. School of Information Science and Technology, Beijing University, Beijing 100871, P. R. China)

Abstract: In view of the Campus Grid construction, which is also a crucial part of ChinaGrid project, the performance decline for grid scheduling algorithms when non-dedicated nodes emerge in multi-grid environment is studied. A load balancing algorithm to optimize resource usage rate is proposed. The paper involves three parts: problem modeling, algorithm design, and experiment evaluation. The experimental results show that the proposed algorithm is effective for solving the problem of resource usage rate decline under the discussed grid circumstance.

Key words: grid computing; load balance; ant colony optimization; task scheduling

近年来,在高性能计算领域,网格逐渐成为超级计算机以外的另一种选择。由于网格系统中的各节点通常在地理位置上比较分散,而且具有平台异构、跨管理域等特点,因此如何将网格任务在这些节点上进行合理的分配并保证执行效率成为网格研究的 1 个重要方面[1]。

1　网格及其负载均衡问题

调度算法是网格资源分配的核心。然而,由于网格动态变化的特性,即使采用了比较先进的调度算法,网格任务在各个节点上的部署仍有可能出现分布不均的情况,从而导致系统资源利用率下降。

收稿日期:2010-04-12
基金项目:国家科技支撑计划基金资助项目(2006BAH02A 24-6);重庆市自然科学基金资助项目(CSTC2008BB2183);中国博士后科学基金资助项目(20080440699)。
作者简介:陈乙雄(1977-),男,重庆大学博士,主要从事调度理论和网格计算方向研究,(Tel)023-66533877;(E-mail)chenyx@cqu.edu.cn。
吴中福(联系人),男,重庆大学教授,博士生导师,(Tel)023-65112008;(E-mail)wzf@cqu.edu.cn。

这是因为多数算法执行生成的 1 个优化了的调度方案是以当前网格资源的状态信息作为输入的。但从任务按照调度方案发送到具体网格节点到真正提交执行这段时间,任务队列的任何改变都会导致实际的任务执行时间延后或提前(如作业取消)。此外,任务预计执行的时间也可能会因为不同的估计方法而与实际情况有一定误差。最终造成的结果就是某些节点上的任务很快执行完毕而处于空闲状态,而另一些节点则可能堆积了过多的任务,从而使系统整体的资源利用率降低。为了使每个节点都能充分利用,就需要使任务在各个节点的分布保持均匀,即网格中的负载均衡问题。需要强调的是,论文讨论的节点负载实质上是指任务所需要的执行时间而非存储空间。

在网格环境下,有多种原因可造成节点负载情况的改变,从而导致负载不均衡现象:在硬件方面,可能会因为计算节点部分处理机停止工作而使得执行任务的速度降低,从而导致执行时间延长,可反过来视为负载变大;软件方面,工作在非预留方式下的节点会因为任何某个调度范围之外的任务而增加额外的执行时间,从而导致负载变化。这些负载的变化信息是来不及且不可能反馈给该调度模块并让其对已经分配到各节点任务重新做调度安排的,因而负载不均的情况则有可能出现。当然,节点负载情况也会因核心调度算法的不同而不同。例如负载不均衡的情况在某些时候可以由调度算法在二次调度时加以修正。但这种修正在某些情况下并不及时,尤其是当任务调度是成批进行,且每个批次任务数较大时。如果调度范围之外的任务出现数目较多,发生频率较大,例如在多网格环境下[2],网格中出现若干共用节点(即同时属于 2 个以上网格的计算节点)的情形[3],则调度算法的执行效果就会受到较明显的影响。

负载均衡的研究成为除调度算法外提高网格资源利用的另一个方面。多数调度算法是从用户的角度出发,强调用户作业的响应时间和服务质量[4-6],而负载均衡则是从系统的角度来考虑全局的任务分配格局,提高资源的利用率。

负载均衡也可看做系统在任务调度后的对资源分配的一种再调整,其具体实现的方法主要有集中式和分布式 2 种。集中式方法是通过在系统中设置 1 个专门的负载均衡模块,由它收集所有节点的负载信息,进而做出哪些节点的任务需要迁移、以及迁移到哪个节点的策略。这种方法的优点在于掌握了足够的节点负载信息,可以在全局范围内进行任务的迁移和负载平衡。但该方法最大的缺陷就在于通信开销过大,特别是对于网格而言,通常系统的规模较大,且各个节点属于不同的管理域,不同链路的数据通信时延差别也很大,使得集中式的方法缺乏可扩展性。相比来说,分布式方法则更适合网格平台的诸多特点。这类方法允许任务迁移各自在较小的局部范围内进行,可以避免过多的通信开销,且易于扩展。

2　算法的基本思想

参照文献[7]的思路,笔者将负载均衡分成决策和任务迁移两个阶段来实施。决策机构采用一定的策略来决定系统在何种条件下将会触发任务的迁移;而任务迁移机制则按照算法的优化目标,将任务在节点之间进行转移,使节点负载重新达到均衡状态。

2.1　决策机制

对于任何 1 个任务迁移,或多或少都会产生一定的通信开销,尤其在网格环境下,有时候这种开销会很大,使得这种迁移失去意义。因此,在网格中的各个节点之间进行任务迁移,以求达到负载均衡之前,应当有一定的策略来确保进行任务迁移所付出的代价与获得的性能提升相比是可以接受的,否则宁可放弃这种迁移。

笔者设计了一种基于投票的决策机制。即在逻辑上笔者设定一个中心计票器,而每个计算节点可以根据其邻域节点的负载情况来判定自己是否负载过重或过轻,进而做出相应的投票决定。中心计票器收集到这些“选票”后则可以根据一定的规则来判断当前系统是否“值得”进入任务的迁移阶段。通过这种方式,一方面负载信息在节点和邻域节点之间的传输时延可以控制在一个可以接受的范围内;另一方面 “选票”可以用很少的比特数进行编码,在更大网络范围中传输的通信开销远小于直接传输节点负载信息。这就有效地避免了以往集中式方法中,频繁的负载信息传递导致通信开销过大的问题;同时由于以间接的选票信息代替了直接的负载信息并在更大范围内传递,就有利于在更大的范围实现负载均衡。

此外,笔者特别允许共用节点以更高频率进行投票,因为这些节点的负载变化情况往往较一般节点更大,使得需要调整负载的频率也更高。

2.2　任务迁移机制

在确定了要将某些节点上的任务进行迁移后,具体如何将任务在候选节点上重新分配则需要在任务迁移阶段完成。该过程也就是系统进行负载均衡的过程,即将一些负荷过重节点上的任务迁移到候选节点上,如图 1 所示。

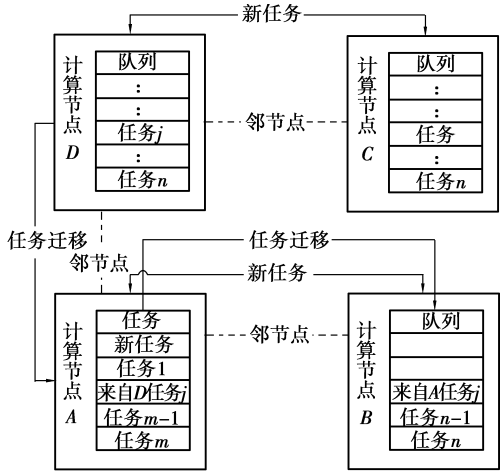

图 1 任务迁移示意图

由于该问题属于 NP 问题，且允许算法执行的时间在网格中通常很短，因此在采用启发式算法来寻求较优解的同时，还需要尽量提高算法的收敛速度。

笔者采用蚁群算法来进行完成解的搜索。蚁群算法的思想来源于对自然界蚂蚁的行为进行模拟，通过蚂蚁群体合作行为以求获得较优的问题解[8]。同时，为了缩短搜索时间，使用了一种基于任务粒度的预分配方法，并以该方法的分配结果为依据，将初始信息素分布到图上，以达到加快蚂蚁搜索收敛速度的目的。

3 Alba 算法

笔者将提出的算法简称为 Alba（ACO-based load balancing algorithm），在建立系统模型基础上，介绍算法的实现过程（分投票决策阶段和任务分配阶段）。表 1 为涉及的主要数学符号：

表 1 主要变量表

变量名称	描述
$M1, M2, \cdots Mm(t)$	t 时刻的 $m(t)$ 个计算节点
$T1, T2 \cdots Tn(t)$	t 时刻的 $n(t)$ 个待分配的任务
u	最低投票数
TT_j	节点 j 的传输开销
$D(t)[m,m]$	t 时刻节点间传输速度
$G(T_j)$	任务 j 的粒度
K	蚂蚁的数量
b	信息素挥发速率
Tl	算法执行时间上限

3.1 系统模型

笔者根据文献[9]介绍的方法，用下列三元组来描述网格的任务分配模型

$$\alpha \mid \beta \mid \gamma = Rm, H_3 \mid a_j, d_j, \text{batch} - M \mid f_o。(1)$$

其中 α, β, γ 分别表示机器环境、任务性质以及目标函数。

3.1.1 机器环境

在网格中，各计算节点平台通常可以是异构的，但在运行网格作业时是并行方式，这里笔者用 RM (related machine)表示。每个计算节点可以包含一个或多个 CPU，而每个 CPU 也可以有不同的处理速度（以 MIPS 为单位）。网络拓扑允许动态变化，t 时刻节点之间的传输速率由矩阵 $\boldsymbol{D}(t)[m,m]$给出。在节点中的任务分配是以一种层次结构传递的，参照文献[10]的方法，笔者用 H_3 来定义该结构

$$H_3 = GQ_1\{LQ_1, LQ_2, \cdots\cdots, LQ_n\}[FQ_1, FQ_2, \cdots\cdots, FQ_m], \quad (2)$$

其中：GQ 为全局队列；LQ 为本地队列；FQ 表示先进先出队列。

3.1.2 任务性质定义

网格系统中的任务可以不定期的到达，到达时刻由 a_j 表示。同时，每个任务的属性，包括提交时间、运行要求、到期时间 d_j 等均可在任务实体的参数中加以描述。需要指出的是，任务的粒度 $G(T_j)$ 将在算法的任务预分配过程中作为判定依据。$G(T_j)$的值并不是任务 T_j 所占用的存储空间大小，而是其在标定机器（固定 MIPS）上所反映出的执行长度。任务的分配以 batch-M 方式，即成批进行调度分配，每个批次的大小为 M。

3.1.3 目标函数

由于网格环境的复杂性，根据不同的应用需求往往存在不同的优化目标[10-11]。从用户角度来讲，通常只关心一个作业（由若干任务组成）完成的快慢，即响应时间；而从系统角度而言，系统的吞吐率、资源利用率常常作为衡量的主要指标。在论文中，笔者着重强调算法在保证作业响应时间的同时，尽可能提高资源利用率。并给出了以下目标函数公式

$$f_o = \frac{\max\{(TT_j + \text{WET}_j) \mid j = 1 \ldots M\}}{RU}。(3)$$

在上面的公式中，分子指的是第一个任务到达至最后一个任务完成的整个时间跨度。同时，这个时间值也和所有节点的运转时间的最大值相等。而每个节点的运转时间包括任务发送、结果返回（TT_j）、和任务等待与执行所消耗的时间（WET_j）。（即 Wall Clock 时间而非 CPU 时间）。RU 为资源利用率，是所有节点任务执行时间和所有节点运转时间（CPU 执行和空闲时间之和）的比值，即

$$RU = \frac{\sum_{j=1}^{M} R_j}{\sum_{j=1}^{M} (R_j + I_j)}。\tag{4}$$

3.2　系统结构

根据文献[12]的分类方法,可以把网格系统的结构划分为集中式、分布式、以及层次结构。由于集中式结构不利于系统的扩展,而分布式结构常常不容易收集足够的状态信息来获得 1 个较优的解,故笔者采用层次结构定义系统框架。这样既有利于信息的收集,也比较容易扩展。该结构的定义如图 2 所示。

在图 2 中,笔者假定所有的任务队列都有足够的空间来存储等待运行的任务,即论文没有讨论队列任务溢出的情况。对于共用节点,来自其它网格的任务可以随时出现,从而影响节点的负载情况。笔者假定网络状况发生突变的几率很小,即当前的网络环境是确定的,矩阵的值保持不变。

在实际运行过程中,每个节点允许和它的邻域节点集中的节点交换负载信息。这里笔者引入了邻域集的概念。邻域集是指某个节点附近,传输单位数据的时延小于一定阈值的那些节点所组成的集合。把负载信息的交换限制在邻域集中可以有效地降低频繁的信息交换所产生的通信开销。

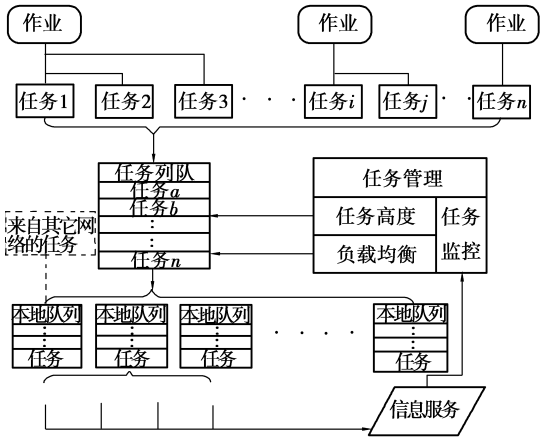

图 2　任务调度与执行结构

3.3　投票决策机制

笔者采用了一种类似于投票的策略作为负载均衡的触发规则。根据邻域集的概念,以共用节点为“选民”,因为这些节点往往会因为外来任务而导致负载不均衡的情形发生。每个共用节点均可以获得其周围节点的负载信息,并且这些信息可以按照一定频率更新以确保准确性。根据这些信息,每个节点都可以向投票决策模块发出 1 张“选票”来表示支持或者反对系统进行 1 次任务分配的均衡化。规则如下

规则 1:

$$v = \frac{1}{M}\sum Y(M_j), Y(M_j) = \begin{cases} 1, o(M_j) < u_1 \ or \ o(M_j) > u_2; \\ 0, u_1 \leqslant o(M_j) \leqslant u_2。 \end{cases}\tag{5}$$

规则 2:

$$S = \begin{cases} 0, v \leqslant rx; \\ 1, v > r。 \end{cases}\tag{6}$$

即在某个节点的邻域中,如果有超过 $r\%$的节点其负载 $o(M_j)$过轻或过重,u_1 的取值通常可以在 0～60(负载相对太轻);u_2 的取值通常在 140(负载相对过重),则该节点投支持票,反之投反对票。而投票决策模块最终会根据选票数量,按照规则 2 来确定是否需要启动负载均衡过程。此外,某些情况下还可以给不同节点的选票加上一定的权重,如某些节点的 CPU 时间更贵,其选票权重就可以设大一些。

3.4　基于任务粒度预分配

当前面的布尔变量 S 为真时,系统便进入了任务迁移过程。使用了一种基于粒度的预分配方法,并将其结果生成为初始信息素,则可以大大加快蚂蚁的搜索速度。这里,任务粒度的大小并非其指令条数或占用的存储空间,而是标定机器下任务的执行长度,因而往往循环语句较多的任务会具有较大的粒度。

预分配方法的过程就是将任务按照粒度的大小从高到低排序,而节点则按照负载的多少按由轻到重排序,然后尽量将任务粒度大的任务分配到负载最轻的节点,并在每次更新后不断重复这个过程直到所有任务分配完毕。

上述过程可以由图 3 所示实例加以说明。

假设有 6 个任务需要分配到 5 各节点机上,则

第一步:分别将任务和节点机按照粒度和负载情况排序,如图 3 所示;

第二步:将当前粒度最大的任务 4 分配到负载最轻节点上,并将其标识为已分配;

第三步:更新节点的负载信息,并重新排序;

第四步:若还存在未分配的任务则回到第二步,反之预分配结束。

需要指出的是,在预分配过程中并没有考虑通信开销的问题。这是因为频繁的计算每一种分配的通信开销会导致预分配过程变慢,从而影响算法的

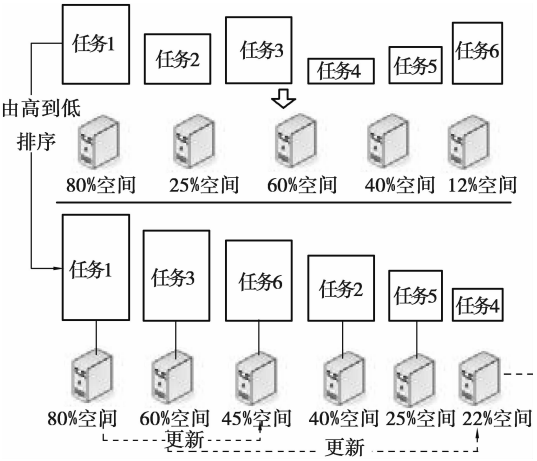

图 3 按任务粒度分配示意图

执行效率。该问题可以留到蚁群优化过程中，让蚂蚁在相应的路径上通过信息素的数量来避免将任务分派到传输时延大的节点上去。

3.5 蚁群优化

在任务预分配的基础上，笔者采用了蚁群算法对问题的解进行了优化。借鉴文献[13]蚁群优化方法的思路，笔者将优化过程分为了以下 4 个步骤进行。

第一步：构造搜索图。

首先构造 1 个图，将问题的求解转化为搜索图中的 1 条最优路径，如图 4。

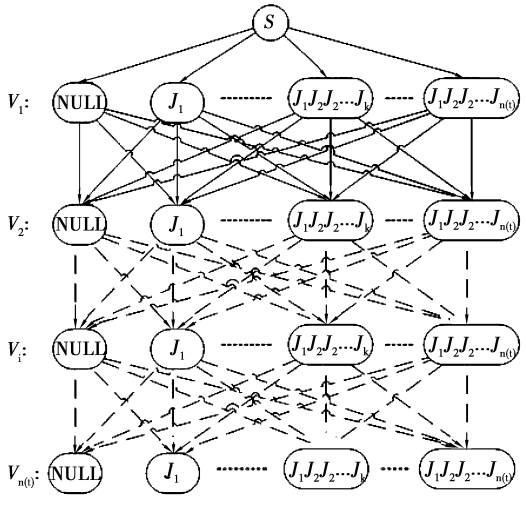

图 4 蚁群优化搜索图

图 4 的每行节点代表了任务集合的所有可能组合(包括空集)。图节点的行数与节点机数目相同。增加 1 个初始节点 S，并且每行节点均发出有向边指向下一行的每个节点。

第二步：放置信息素。

如果图中无信息素，初期收敛速度较慢，故可以将预分配的结果作为初始信息素，并以高斯公式分布到图中，则可以大大加快蚁群优化过程

$$\mathrm{Gauss}(x,\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}\mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}。\tag{7}$$

这里，x 是节点到当前最佳路径节点的汉明距离，变量 μ 和 σ 可以分别用来控制信息素放置的中心和扩散程度，在初始状态 $\mu=0,\sigma=1$。

第三步：蚂蚁行进。

算法执行时，在图中放置若干只蚂蚁，每只蚂蚁按照下面的规则前进

1)每只蚂蚁都从初始节点 S 发，由上至下行进。在蚂蚁行进过程中，根据后续节点的信息素浓度来确定移动到该节点的可能性。若 τ 表示某个节点上信息素的数量，则蚂蚁 α 从 i 移动到 j 的可能性为

$$P_j(\alpha,i)=\frac{\tau(v_{i,j})}{\sum_{1\leqslant k\leqslant m_i}\tau(v_{i,k})}。\tag{8}$$

2)蚂蚁在前进过程中还应遵循以下约束。

$$S_{iq}\cap S_{jr}=\phi(1\leqslant q<r\leqslant m);\tag{9}$$

$$S_{1p}\cup\cdots\cup S_{iq}\cup\cdots\cup S_{jr}\cup\cdots\cup S_{mt}=\{J_1,J_2,\cdots,J_{n(t)}\}。$$

这里，$S_{1p},\cdots,S_{iq},\cdots,S_{jr},\cdots,S_{mt}$ 为 1 条有效路径上节点所代表的任务子集。通过该公式就可以保证蚂蚁不会行进到 1 个包含有已被分配任务的子集节点上。

根据蚁群算法，蚂蚁在找到一条可行路径的同时会在其经过的道路上放置一定数量的信息素。为了使整个搜索朝着更优的解推进，信息素的分布应该依据当前的最优路径进行更新。即找到一条可行的路径后，将其对应的目标函数值 X_{new} 与当前的最优值 X 进行比较，如果 $X_{\mathrm{new}}<X$ 则令 $X=X_{\mathrm{new}}$，反之保持 X 的值不变(如图 5 所示)

值得注意的是，在蚂蚁搜索最佳路径的早期，有可能会找到一个局部次优解而使得信息素在相应路径上堆积过快，从而吸引更多的蚂蚁并放置更多的信息素，最后算法过早收敛于局部次优解。为了防止这种过快收敛情况的发生，可允许信息素按照一定的比率 b 挥发，以增大蚂蚁寻找更大范围内最优解的可能性

$$\mu_{\mathrm{new}}=(1-b)\mu_{\mathrm{old}};$$
$$\sigma_{\mathrm{new}}=\begin{cases}(1+b)\sigma_{\mathrm{old}},(1+\mathrm{b})\sigma_{\mathrm{old}}<\sigma_{\max};\\ \sigma_{\max},\mathrm{otherwise}。\end{cases}\tag{10}$$

整个搜索过程会不断重复直到满足一定的终止条件，通常网格中允许这类算法执行的时间比较有限，这里可以简单的定义 1 个算法执行的期限 Tl 为终止变量。

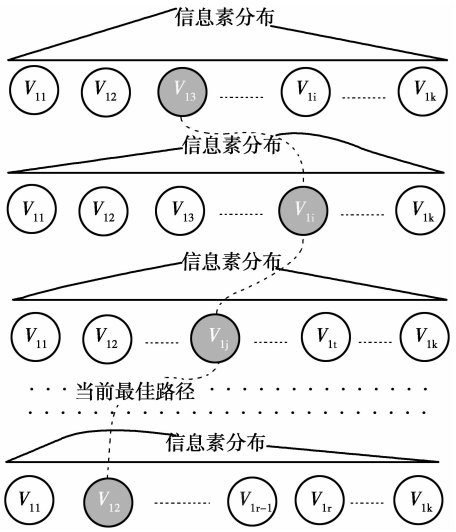

图 5　激素更新示意图

归结起来，上述过程也可以由下面的伪代码加以描述

Procedure ACO_Optimization

变量初始化：

在预先分配过程后得到的路径上放置信息素；

While(执行时间$<T_l$)do

蚂蚁搜索；

当前解$=\min\{f_{obj}(P_k)\mid k=1,2,\cdots K\}$；//即所有蚂蚁得到的路径中最佳的一条

IF $X_{new}<X$　　THEN　　$X=X_{new}$；

信息素更新；END。

4　实验与分析

4.1　实验环境

由于网格平台规模较大且跨管理域，因此算法的评估测试主要以仿真实验[14-15]为主。具体笔者采用了基于 GRIDSIM 的网格模拟器，并针对共用节点的情况对数据集进行了部分修改。使用了 6 组不同的数据集，每个数据集中均定义了机器环境(包括 CPU 数目，运行速率等)和任务性质(包括到达时间，到期时间，任务大小，运行要求等属性)。

笔者在共用节点上以随机的方式提交了一定数量的任务，绕过了调度模块直接提交来观察对已有任务运行情况的影响，并和未出现共用节点的情况做了对比。为了不失一般性，笔者测试了 3 种调度算法，包括 1 个基本调度策略 FIFO(first in first out)，1 个高级调度策略 EDFBF(earliest deadline first back filling)，以及一个使用了启发式算法(禁忌搜索)进行优化的调度方法。

4.2　实验结果分析

首先，笔者使用了 6 个不同的数据集对上述 3 种算法在没有共用节点的条件下进行了测试，测试结果如表 2。

上述结果表明在不存在共用节点的条件下，随着采用的调度策略的不同，先进的调度方法对任务执行的效率有显著的提高。随后，笔者将参与测试的节点的 1 部分(40%)设置为共用节点，再使用相同的调度策略和数据集重新进行了测试，且为了提高数据可信度，每次运行重复 10 遍获取 1 个平均值作为最后的测试数据。结果见表 3。

表 2　初始测试结果

数据集	执行时间			任务完成时间			资源利用率		
	FIFO	FB	TB	FIFO	EDF	TB	FIFO	EDF	TB
1	0.11	37.85	20.575	4923.71	4643.105	4226.13	92.345	93.605	93.345
2	0.1	24.11	19.11	4376.925	4219.77	3808.495	91.825	93.415	90.825
3	0.11	12.09	16.33	3831.63	3542.73	3327.735	91.065	93.125	88.025
4	0.115	26.065	19.03	4671.65	4374.15	3939.65	91.785	93.725	93.755
5	0.105	16.855	17.295	4208.705	3928.625	3592.775	91.33	93.605	89.22
6	0.12	17.645	19.295	4360.66	4008.07	3787.905	91.675	92.795	88.775

表 3 目标环境测试结果

数据集	执行时间			任务完成时间			资源利用率		
	FIFO	FB	TB	FIFO	EDF	TB	FIFO	EDF	TB
1	0.105	38.055	22.295	5142.760	4723.715	4689.905	92.330	93.220	86.220
2	0.110	24.030	21.030	4987.080	4450.120	4210.102	91.025	93.375	83.375
3	0.100	12.650	18.110	4670.102	3747.105	3765.225	90.885	93.220	81.025
4	0.120	26.295	20.995	5084.270	4600.495	4401.665	91.220	93.755	85.955
5	0.102	17.110	19.030	4831.990	4115.255	4002.385	91.755	93.825	82.330
6	0.110	17.975	21.395	4715.619	4199.025	4173.550	91.345	92.345	80.995

通过上表不难发现，当环境中的共用节点数目逐渐增多时，平均而言任务执行的完成时间会延长（有个别情况这种影响不明显，而另一些则有显著变化），同时资源的利用率也会出现下降。这证明共用节点对任务调度确实存在影响，并可导致执行效率降低。为此，笔者再次测试了相同的数据，分别将参与测试的节点的 20%，40%，60%，80%设置为共用节点集并集成了论文提出的算法。同一比例共用节点情况下用不同数据集测试取平均值。测试的各项主要指标见图 6～图 9。

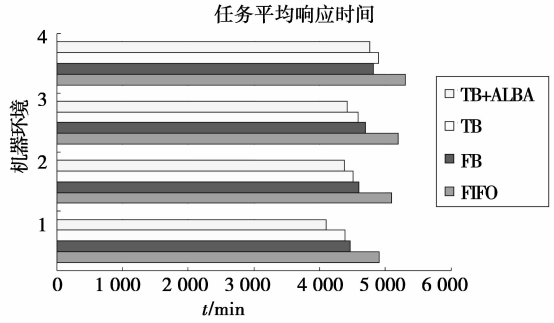

图 6 任务完成时间测试结果

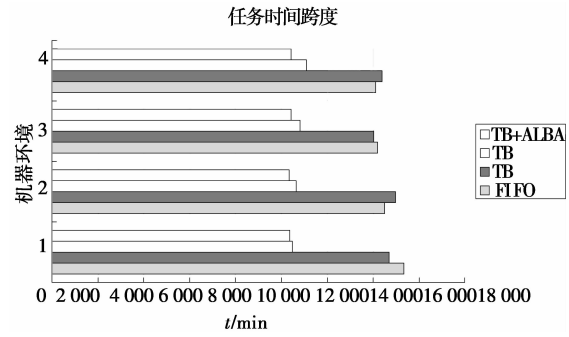

图 7 时间跨度测试结果

上述结果表明，提出的算法在环境中出现不同数目的共用节点条件下，均能有效地防止任务执行的主要性能指标降低情况的出现，特别对资源利用率有较好的保证。原因在于研究算法能有效将某些

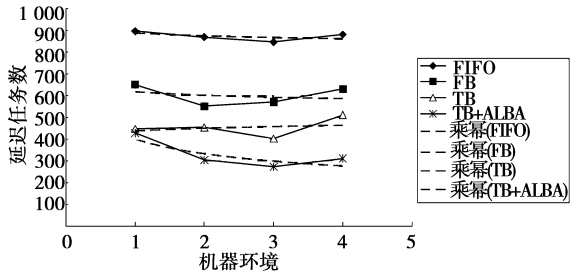

图 8 任务延迟测试结果

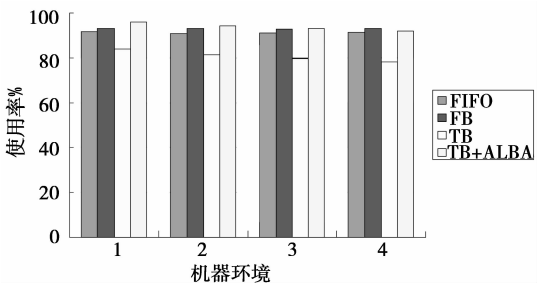

图 9 资源使用率测试结果

负载过重节点上的任务迁移到较为空闲的节点，使各个节点的任务分配比较均衡，因而机器空闲时间大大减小，从而提高了利用率。此外，笔者用不同的任务批次和任务数，对 ALBA 在共用节点环境下（共用节点数 60%）进行了测试，结果如图 10。

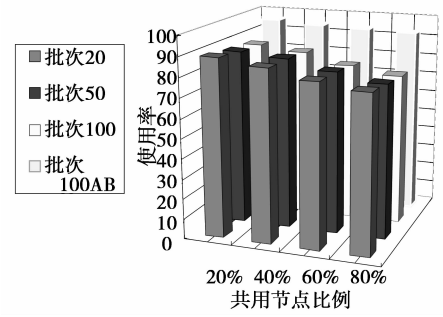

图 10 不同批次下资源使用率测试结果

结果表明，随着任务批次的加大，任务执行效率受共用节点的影响也越大，而 ALBA 被触发的几率也大大提高。这是因为当任务批次较小时，前一批任务因共用节点导致的分布不佳情况能够较快反馈到调度模块而在后一批任务调度时加以修正，而任务批次较大时，上述情况的影响无法及时纠正，此时 ALBA 算法的作用则更加明显。

5 结　语

研究了网格中的负载均衡问题，并特别针对系统中出现共用节点时，任务调度算法执行效率下降的问题，提出一种基于蚁群优化的负载均衡算法 ALBA，算法考虑了共用节点对任务分配的影响和特殊性，并采用了一种类似于投票的决策机制，较好地处理了任务迁移和通信开销之间的矛盾。在算法的优化上，使用了基于任务粒度的预分配机制，使得优化过程的收敛速度得以提高。

参考文献：

[1] FOSTER I, MORGAN K. The grid: blueprint for a new computing infrastructure 2nd edition[M]. USA. Elsevier, 2004.

[2] YANG C T, HU W J, LAI K C. A resource broker with cross grid information services on computational multi-grid environments [J]. Lecture Notes in Computer Science, 2009. 20-31.

[3] 万军洲，李腊元. 非专用分布式系统中任务调度的性能预测模型的研究[J]. 武汉理工大学学报，2004，28(4).
WAN JUN-ZHOU, LI NA-YUAN. A performance prediction model for task scheduling in a non-dedicated system [J]. Journal of Wuhan University of Technology, 2004, 28(4):72-80.

[4] 李春林，郑辉. 网格计算中基于 QoS 的资源调度优化模型[J]. 武汉理工大学学报，2008，32(2):58-62.
LI CHUN-LIN, ZHENG HUI. QoS based resource scheduling optimization policy in computational grid [J]. Journal of WuHan University of Technology, 2008, 32(2).

[5] LI C L, LI L Y. A system-centric scheduling policy for optimizing objectives of application and resource in grid computing[J]. Computers and Industrial Engineering, 2009, 57(10):1052-1061.

[6] 刘海迪，杨裔，马生峰，等. 基于分层遗传算法的网格任务调度策略[J]. 计算机研究与发展，2008，45 (1)：35-39.
LIU HAI-DI, YANG YI, MA SHENG-FENG, et al. A grid task scheduling strategy based on multi－level genetic algorithm[J]. Journal of Computer Research and Development, 2008, 45 (Sup1)：35-39.

[7] ALBERT Y, ZOMAYA, Y T. Observations on using genetic algorithms for dynamic load-balancing [J]. IEEE Transactions on Parallel and Distributed Systems, 2001,12(9): 899-911.

[8] MARCO D, THOMAS S. Ant colony optimization[M]. USA: the MIT Press, 2004:57-72.

[9] PETER B. Scheduling algorithms (5th ed) [M]. Germany: Springer-Verlag, 2007:12-29.

[10] PAVELl F, LUDEK M, HANA R. Model of grid scheduling problem [J]. Grid and Autonomic Computing, 2005(2): 17-24.

[11] LI C L, XIU Z J, LI L Y. Resource scheduling with conflicting objectives in grid environments: Model and evaluation [J]. Journal of Network and Computer Applications, 2009, 3(25): 760-769.

[12] MAOYHEN L, MARK B. The grid: core technologies[M]. England: John Wiley & Sons Ltd., 2005.

[13] PETER K, JURIJ S, KLEMEN O, et al. The differential ant-stigmergy algorithm: an experimental evaluation and a real-world application [J]. Evolutionary Computation, 2007,9 :157-164.

[14] ANTHONZ S, UROS C, SRIKUMAR V, et al. A toolkit for modelling and simulating data grids: an extension to gridSim [J]. Concurrency and Computation: Practice and Experience, 2007, 11 : 532-626.

[15] XHAFA F, BAROLLI L, et al. A static benchmarking for grid scheduling problems [C]// Proceedings of International Conference on Advanced Information Networking and Applications. [S. L.]: IEEE, 2009, AINA:170-175.

(编辑　侯　湘)