费用-时间优化的网格有向无环图调度算法

陈宏伟[1], 王汝传[1,2]
(1. 南京邮电大学计算机科学与技术系, 江苏南京 210003; 2 南京大学计算机软件新技术国家重点实验室, 江苏南京 210093)

摘 要: 网格环境下, 基于时间限制和费用最小的有向无环图(DAG)调度算法运用经济规律把网格用户中的任务映射到网格资源中运行, 弥补了 Buyya R 提出的算法中未考虑任务运行的优先关系. 该算法有两个关键技术: DAG 中有效路径的提取能够定位任务何时在资源上运行; DAG 中在作业运行时间尽可能允许的情况下, 把任务映射到价格便宜的资源上运行. 通过仿真实例, 论证了该算法的优越性.

关键词: 网格调度; 有向无环图; 费用-时间优化
中图分类号: TP393.02 文献标识码: A 文章编号: 0372-2112 (2005) 08-1375-06

A Grid DAG Scheduling Algorithm for Cost-Time Optimization

CHEN Hong-wei[1], WANG Ru-chuan[1,2]
(1. *Department of Computer Science and Technology, Nanjing University of Posts and Telecommunications, Nanjing, Jiangsu 210003, China;*
2. *State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu 210093, China*)

Abstract: The Directed Acyclic Graph(DAG) scheduling algorithm, based on time-constrained and minimum-cost in grid environment, applies economic law to map the running tasks of grid users into grid resources. The proposed algorithm overcomes the deficiency of Buyya R's algorithm, which is not concerned with priority relationship between running tasks. It is highlighted in two key techniques: one is that the extraction of valid paths from DAG can effectively locate tasks in specific resources during specific periods; the other one is that tasks should be mapped into cheap resources so that they can be run as much as possible with the restriction of job-running time. A simulative example based on the algorithm is also provided to analyse the characteristics of the algorithm.

Key words: grid scheduling; directed acyclic graph; cost-time optimization

1 引言

网格计算(Grid Computing)是一个分布式和并行计算的支持平台, 是一种无缝、集成计算和协同环境. 它可以作为虚拟的整体使用在地理上分散的计算资源. 网格把整个 Internet 整合成一台巨大的超级计算机, 实现计算资源、存储资源、数据资源、信息资源、知识资源、专家资源的全面共享和协同. Ian Foster 把网格描述为"在动态变化的多个虚拟组织间共享资源和协同解决问题"[1]. 一个简单的网格模型主要有网格用户、资源中介(Resource Broker)、网格资源三类实体. 网格用户是资源的消费者, 网格资源是资源的提供者, 而资源中介的功能是如何把网格用户提交的作业合理地配置到网格资源上. 资源中介具有调度功能. 由于作业是由一些具有不可分解性的任务组成, 因此资源中介的主要功能是如何把作业中的任务合理有效地映射到资源上, 并使得性能最优.

由于资源提供者提供的是一种服务, 它不可能无偿地为网格用户提供服务. 因此资源中介就需要运用经济规律把网格用户中的作业配置到网格资源中. 在网格环境下, 目前比较常见的经济模型有: 商品市场模型(Commodity Market Model)、牌价模型(Posted Price Model)、议价模型(Bargaining Model)、合同网模型(Tendering/Contract-Net Model)、拍卖模型(Auction Model)[2-3]. 对网格用户而言, 他需要得到物美价廉的服务, 也就是用户提交的作业在尽可能短的时间内, 以尽可能低的价格, 获得尽可能高安全可靠的服务. Ernemann C[4] 主要给出了不同访问和价格策略, 并依据响应和等待时间最小化和利用率等指标来评价算法的性能. Moreno R[5] 主要扩展网格资源信息服务模式来处理网格经济相关信息, 并结合性能和经济信息对不同的调度算法进行了仿真和评价. 然而 Ernemann C 和 Moreno R 提出的算法和模型不符合现实世界中的情况. R Buyya[6] 提出一个基于限期和预算的费用-时间优化的算法(A Deadline and Budget Constrained Cost-Time Optimization Algorithm), 该算法把经济规律运用到网格资源调度中去, 特别是考虑到时间、费用、资源性能等因素, 比较符合现实世界的情况. 该算法依据时间、费用、时间变体等三种优化策略来支持

收稿日期: 2005-02-02; 修回日期: 2005-04-28
基金项目: 国家自然科学基金(No. 60173037, No. 70271050); 江苏省自然科学基金和江苏省自然科学基金预研项目(No. BK2004218); 江苏省高技术研究计划(No. BG2004004); 江苏省计算机信息处理技术重点实验室基金(No. kjs04)

时间限制和费用预算,该算法在 Nimrod 上运行,并通过 Gridsim 进行了仿真验证. Feng H L[7] 等对 R Buyya 的算法进行了改进,并假设网格用户的作业有 M 个相互独立的任务,可发到 N 个网格资源中,并假设的基础上进行了建模和仿真. 然而,R Buyya 提出算法和 Feng H L 的改进算法未考虑作业分解成任务集后,任务与任务之间可能存在着优先运行的关系. 如何解决任务之间的优先关系呢? 有向无环图 *DAG*[8~10] (Directed Acyclic Graph) 是一个比较有效的方法. 下面,论文将讨论费用-时间优化的 *DAG* 算法.

2 费用-时间优化的 *DAG* 调度算法

2.1 算法基本思想

本论文基于费用-时间优化的网格调度问题主要解决在作业运行时间限期下,尽可能使作业运行费用最少. 该问题描述如下:已知用户作业 $J=\{t_1, t_2, \cdots, t_N\}$,其中 t_i 表示作业 J 的第 i 个任务. 作业 J 的 $DAG=(V, E)$, $V=\{v_1, v_2, \cdots, v_N\}$, $E=\{e_1, e_2, \cdots, e_X\}$. 任务 t_i 与顶点 v_i 之间是一一对应的关系, $e_k=(v_i, v_j)$ 可表达为有向边 $v_i \rightarrow v_j$, 它表示任务 t_i 必须在任务 t_j 之前运行,否则任务 t_j 不能运行. 已知网格资源集合 $R=\{r_1, r_2, \cdots, r_M\}$. $U(r)$ 表示运行资源 r 的单价, $P(r)$ 表示资源 r 的计算能力. $Q(t)$ 表示任务 t 的计算量, $C(t, r)$ 表示在资源 r 上运行任务 t 的费用, $D(t, r)$ 在资源 r 上运行任务 t 的结束时间点. 假设作业 J 的运行时间期限为 *Deadline*,费用为 *Cost*,则问题求解目标是:

$$\text{Min}(Cost)=\sum_{i=1}^{N} C(t_i, rs_i)$$

$$\forall\, D(t_i, rs_i) \leqslant Deadline,\ rs_i \in \{r_1, r_2, \cdots, r_M\} \qquad (1)$$

由于 *DAG* 中可能存在着多条从起点到终点的路径,通常,路径上任务计算量越大,则运行时间就越长,就越有可能接近作业的时间期限. 此时,就越有可能确定任务在指定资源的开始运行时间和结束运行时间. 这样,不同任务在同一个资源运行时间上的冲突就可避免了. 而且在 *DAG* 的从起点到终点的路径中尽可能地把最大的任务,给费用最便宜的资源去运行. 这就是本论文基于费用-时间优化的网格调度的核心思想. 下面将分别阐述该算法的两个关键技术.

2.2 *DAG* 中有效路径的提取

定义 1 路径(Path)是指在 *DAG* 中从 $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_n$ 的通路. 其中 v_0 属于起点集合,即入度为 0 的顶点; v_n 属于终点集合,即出度为 0 的顶点;且 $(v_0, v_1, \cdots, v_n)$ 都不相同.

定义 2 有效路径(Valid Path)是指在 *DAG* 中路径 $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_n$,不存在着一条通路 $v_i \rightarrow \cdots \rightarrow v_j$ 使得 $v_0 \rightarrow v_1 \rightarrow \cdots v_i \rightarrow \cdots \rightarrow v_j \rightarrow \cdots \rightarrow v_n$ 形成一条新的路径. 其中 $\{v_0, v_1, \cdots, v_n\} \cap \{v_i, \cdots, v_j\} = \varnothing$.

定义 3 无效路径(Invalid Path)是指在 *DAG* 中路径 $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_n$,存在着一条通路 $v_i \rightarrow \cdots \rightarrow v_j$ 使得 $v_0 \rightarrow v_1 \rightarrow \cdots v_i \rightarrow \cdots \rightarrow v_j \rightarrow \cdots \rightarrow v_n$ 形成一条新的路径. 其中 $\{v_0, v_1, \cdots, v_n\} \cap \{v_i, \cdots, v_j\} = \varnothing$.

定义 4 无向环是指在 *DAG* 中,若所有有向边 $v_i \rightarrow v_j$ 变为无向边 (v_i, v_j),此时有向无环图 *DAG* 变为无向图. 若无向图中存在着通路 $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_n \rightarrow v_0$, 且 $(v_0, v_1, \cdots, v_n)$ 都不相同,则该通路在对应的有向无环图 *DAG* 中称为无向环.

如图 1 所示,该 DAG 中存在如下路径: $1 \rightarrow 2 \rightarrow 4$、$1 \rightarrow 2 \rightarrow 3 \rightarrow 4$、$1 \rightarrow 2 \rightarrow 3 \rightarrow 5$、$1 \rightarrow 3 \rightarrow 4$、$1 \rightarrow 3 \rightarrow 5$、$1 \rightarrow 2 \rightarrow 3 \rightarrow 6$、$1 \rightarrow 3 \rightarrow 6$、$1 \rightarrow 6$. 其中,依据定义 3, $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$、$1 \rightarrow 2 \rightarrow 3 \rightarrow 5$、$1 \rightarrow 2 \rightarrow 3 \rightarrow 6$ 是有效路径. 由于有效路径能够覆盖掉无效路径中的所有信息,则在 *DAG* 中无效路径的获取只会增加不必要的工作量. 由于无向环是造成无效路径产生的原因,但并不是所有的无向环都会造成无效路径的产生,因此必须剪除造成无向环的多余的边.

可采用如下算法剪除无向环中多余的边: 在图 2 中,可创建如下有效路径树,从路径树的根 Root 开始,把所有 *DAG* 中的起点作为 Root 的儿子(child),根据图 2 可知, Root 只有一个儿子 1. 此时获取所有以起点 1 为入度的结点作为起点 1 的儿子. 此时有结点 2、3、6. 依次类推,结点 2 有儿子 3 和 4. 结点 3 搜索其父亲(Parent) 2 的兄弟(Brother)中是否有和它编号相同的结点. 结点 2 的兄弟有结点 3 和 6. 此时,应当删除结点 2 的兄弟结点 3. 通过这种方法,可以去掉所有造成无效路径的多余的边了. 以下给出获取有效路径集合的算法:

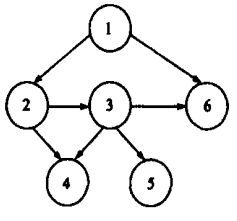

图 1 *DAG* 示例图

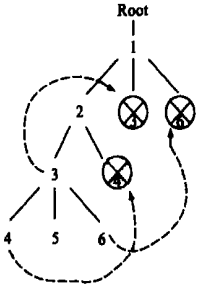

图 2 有效路径树——*DAG* 中有效路径的提取

```
void CreateDAGPath( )
{
    NodeSet S, Z;//DAG 中起点集合 S 和终点集合 Z
    Node Root, V;//DAG 中顶点的根 Root 和顶点 V
    //Root 把 S 中所有顶点作为自己的孩子
    Root.AddChildren(S);
    While(S≠Null)
    {
    V= S.getNode();//从 S 中获取一个顶点 V;
    S= S-{V};//在集合 S 中删除顶点 V;
    CreatePath(V);//创建有效路径树
    RecordPath(V);//获取有效路径集合
      }
    }
    void CreatePath(Node V)
    //为 DAG 创建有效路径树
    {
    NodeSet TmpSet;
    Node TmpV, TmpV2;
    //若 V 中所有孩子都创建过路径
```

```
    if( V. BrothersVisited( ) = = TRUE)
    {
    V. Visited( ) = TRUE;//表示结点 V 已经创建过
    //表示结点 V 的父亲也已经创建好路径了
    V. getParent( ) . Visited( ) = TRUE;
    //若递归到路径的起点, 则退出
    If( V. getParent( ) = = Root. getChild( ) ) return;
    //当递归到终点时, 就为顶点 V 父亲的兄弟创建路径
    CreatePath( V. getParent( ) . getBrother( ) );
}
//若 V 属于终点集合 Z 中的元素, 且 V 中不是所有兄弟都创建了路径
    else if( V∈ Z && V. BrothersVisited( ) = = FALSE)
    {
    V. Visited( ) = TRUE;//表示结点 V 已经创建过
    //当递归到终点时, 就为顶点 V 的未创建路径的兄弟创建路径
    CreatePath( V. getBrother( ) );
}
else
{
    //把顶点 V 中所有后继顶点作为 V 的孩子
    TmpSet = V. getSuccessor( );
    V. AddChildren( TmpSet);
    While( TmpSet≠Null)
    {
    TmpV2 = V. getChild( );//从 V 中获取一个孩子;
    TmpSet = TmpSet - { V};
    TmpV = V;
    //遍历向上搜索是否存在无向环
    While( TmpV≠Root. ofChild( ) )
    {
    //如果 V 和祖辈的某个兄弟相同, 则打删除标志
    If( TmpV2 = = TmpV. ofBrother( ) )
    {
    TmpV. getParent( ) . Delete( TmpV);
    break;
    }
    TmpV = TmpV. getParent( );
    If( TmpV = = Root. getChild( ) ) break;
    }
    }
    CreatePath( V. getChild( ) );
    }
}
void RecordPath( Node V)
//获取有效路径集合
{
    Path P;//有效路径数据结构
    PathSet PS; //有效路径集合
    if( V∈ Z && V. BrothersRecorded( ) = = TRUE)
{
    V. Recorded( ) = TRUE;
    PS. AddPath( P + { V} );
    V. getParent( ) . Recorded( ) = TRUE;
    If( V. getParent( ) = = Root. getChild( ) ) return;
    P = P - { V. getParent( )};
    RecordPath( V. getParent( ) . getBrother( ) );
}
else if( V∈ Z && V. BrothersRecorded( ) = = FALSE)
{
    V. Recorded( ) = = TRUE;
    PS. AddPath( P + { V} );
    RecordPath( V. getBrother( ) );
}
else
{
    P = P + { V};
    RecordPath( V. getChild( ) );
    }
}
```

2.3 *DAG* 中任务与资源之间的映射

DAG 中任务与资源映射需要用到如图 3 所示的资源预留表和任务与资源映射表. 由于在网格环境下, 一个资源可能预留给许多个不同用户作业中的任务运行. 如图 3 所示, 网格资源 *R*1 就预留给任务 Task *a*、*b*、…、*z* 运行. 设置资源预留表的目的就是为了使映射到资源上的任务不和将运行在资源上的其他任务相互冲突. 而设置任务与资源映射表的目的是: 在算法的某条有效路径上, 一旦作业中的某个任务映射到网格环境中的某个资源上, 则在其他有效路径上就可参照任务与资源映射表, 并避免再次重新映射该任务到其他资源上了.

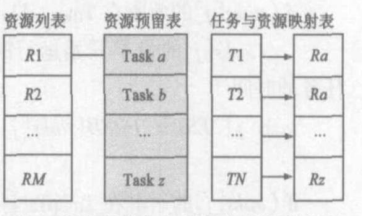

图 3　资源预留表和任务与资源映射表

DAG 中任务与资源映射的算法的思想大致如下: (1) 对网格中的资源按照单价递增进行排序, 若单价相同, 则按照计算能力递减进行排序; (2) 给有效路径按照计算量递减进行排序, 并依次选取有效路径; (3) 在有效路径上, 依次为作业上任务按照在时间限制尽可能允许的情况下, 选取合适的资源. 若该路径上存在一些前面有效路径中已经完成映射的任务时, 则在该路径上为任务选取资源时, 应当满足一些限制性的要求. 若选取的资源超过了作业的时间限制, 则更新该有效路径上已选取的运行最慢的资源; (4) 若某条有效路径上的任务无法找到合适资源满足作业的时间限制要求, 则该算法回溯到前面的有效路径上更新一些任务与资源的映射关系; (5) 若仍然无法完成任务与资源的映射, 则运行失败; 否则继续上述过程, 直到完成所有任务与资源之间的映

射. 下面给出 *DAG* 中任务与资源映射的具体算法:

```
bool CreateDAGMap( )
{ ResourceTable ResTable; //资源预留表 ResTable
MapTable TmapTable; //任务与资源映射表 TmapTable
Path PA[ W]; //有效路径数组 PA
//其中 W 表示DAG 中有效路径的个数
网格资源按照单价递增进行排序;
if( 资源单价相同)
    按照单价相同资源的计算能力递减进行排序;
for( i = 0; i< W; i++ )
    计算每条有效路径的计算量;
给有效路径按照计算量递减进行排序;
for( i = 0; i< W; i++ )
{
    X= PA[ W- 1]. getNodeNum; //获取有效路径上的任务数
     Y= 该路径上有 Y 个任务已经完成资源映射;
     //X- Y 表示该路径上尚未完成资源映射的任务
     for( j= 0; j< X- Y; j++ )
     { // tasks 表示该路径上尚未完成资源映射任务集
     //EST 表示任务的最早开始时间
     //LFT 表示任务的最迟结束时间
     if ( tasks[ j] 是 DAG 的起点)
       tasks[ j]. EST= Job. EST;
     if ( tasks[ j] 是 DAG 的终点)
       tasks[ j]. LFT= Job. LFT;
     if ( tasks[ j] 的前驱在 TmapTable 中存在)
     { //tasks[ j] 的最早开始运行时间为其前驱中最迟结束任务的时间
     tasks[ j]. EST= PRED( tasks[ j]). LFT;
     }
     if ( tasks[ j] 的后继在 TmapTable 中存在)
     { //tasks[ j] 的最迟结束任务运行时间为其前驱中最早开始任务的时间;
     tasks[ j]. LFT= SUCC( tasks[ j]). EST;
     }
     if ( 选取费用尽可能低的资源 rs && 该资源 rs 满足 tasks[ j]. EST 和 tasks[ j]. LFT 的限制)
     { //把任务 tasks[ j] 映射到资源 rs 上
       Map( tasks[ j], rs);
       tasks[ j]. Mapped= TRUE;
       if ( 作业中所有任务映射到资源上)
       return TRUE; //映射成功, 退出
     }
     else
     {
     while( tasks[ j]. Mapped= = FALSE)
     {
     if ( 已经回溯到最长有效路径 && 无法找到满足 tasks[ j] 的资源)
       return FALSE; //无法映射, 退出
     else if ( 路径无法找到满足 tasks[ j] 的资源)
     { //回溯取消上条路径上计算最慢的资源 ri 和 ni 在该路径上最大的任务 ti 之间的映射
       UnMap( ti, ri);
       计算出需要满足任务 task[ j] 和 ti 运行的计算资源的条件;
       }
       else
       { //取消该路径上计算最慢的资源 ri 和 ni 在该路径上最大的任务 ti 之间的映射;
       UnMap( ti, ri);
       计算出需要满足任务 task[ j] 和 ti 运行的计算资源的条件;
       }
       if( 可找到满足计算能力条件的资源 rs, rt)
       {
       Map ( ti, rs);
       Map ( tasks[ j], rt);
       tasks[ j]. Mapped= TRUE;
       更新映射对 TmapTable 造成的影响;
       }
       }
       }
       }
     }
}
```

3 仿真结果

假设在网格环境下计算资源如表 1 所示:

表 **1** 网格环境下计算资源表

资源	单价(10^{-6} ¥/MI)	运算能力(10^3MIPS)
R1	0.9	2.5
R2	0.8	2.0
R3	0.6	1.0
R4	0.9	3.0
R5	1.2	50

注: 单价单位为 10^{-6} ¥/MI, 表示资源运行 100 万条 CPU 指令需要获取 10^{-6}元人民币的服务费; 运算能力单位为 10^3MIPS, 表示资源的计算能力为 10^3* 100 万条指令/秒.

有两个作业 *J*1 和 *J*2, 其 *DAG* 图分别如图 4、图 5 所示. 从图 4 中, 可得知作业 *J*1 的有效路径为 $1^{\rightarrow}2^{\rightarrow}3^{\rightarrow}4^{\rightarrow}7$、$1^{\rightarrow}2^{\rightarrow}3^{\rightarrow}5^{\rightarrow}7$、$6^{\rightarrow}3^{\rightarrow}4^{\rightarrow}7$、$6^{\rightarrow}3^{\rightarrow}5^{\rightarrow}7$. 从图 5 中, 可得知作业 *J*2 的有效路径为 $a^{\rightarrow}d^{\rightarrow}c^{\rightarrow}b$、$a^{\rightarrow}d^{\rightarrow}e$. 假设作业 *J*1 和 *J*2 的运行时间限制为 5* 10^4 秒, 并且遵循先来先服务(FCFS) 的原则, 即网格调度中心先处理 *J*1 的任务和资源之间的映射, 然后在处理 *J*2 的. 作业 *J*1 和 *J*2 任务运行的甘特图如图 6 所示:

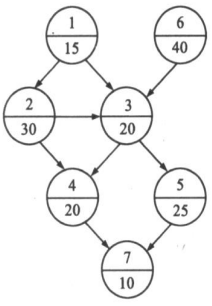

图 4　作业 $J1$ 的 DAG 图

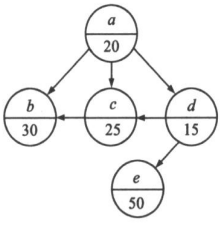

图 5　作业 $J2$ 的 DAG 图

注: 圆圈中上半部分是作业中任务的编号, 下半部分是任务的运算量. 任务的运算量单位为 10^6MI, 表示任务的运算量为 $10^6\times100$ 万条指令.

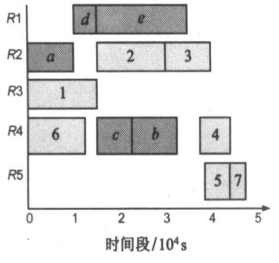

图 6　作业 $J1$ 和 $J2$ 在时间限制为 5×10^4 秒的甘特图

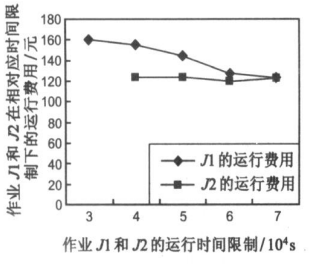

图 7　作业 $J1$ 和 $J2$ 在不同运行时间限制时的运行费用

图 7 比较作业 $J1$ 和 $J2$ 在不同运行时间限制时的运行费用. 其中作业 $J2$ 在时间限制为 $3*10^4$ 秒时无解, 在图中没有表示其运行费用. 图 7 中可知作业 $J1$ 随着运行时间限制的增加, 其运行费用逐渐降低. 然而, 作业 $J2$ 随着运行时间限制的增加, 其运行费用并没有什么明显变化. 由于 $J1$ 和 $J2$ 之间遵循先来先服务(FCFS)的原则, 随着运行时间限制的增加, $J1$ 抢先占用了运行费用低廉的计算资源, $J2$ 就只能使用相对昂贵一点的计算资源. 但是运行时间限制的增加会使得 $J2$ 选取尽可能便宜一点的计算资源, 可以弥补 $J2$ 的运行费用一些损失. 因此这两个因素的相互作用, $J2$ 运行费用没有出现明显的变化.

图 8 比较作业 $J1$ 和 $J2$ 在不同运行时间限制时的运行时间. 其中作业 $J2$ 在时间限制为 $3*10^4$ 秒时无解, 在图中没有表示其运行时间. 图 8 中可知作业 $J1$ 随着运行时间限制的增加, 其运行时间随着运行费用的逐渐降低而增加. 然而, 作业 $J2$ 的运行时间随着运行时间限制的增加出现波动. 由于, 由于 $J1$ 选取资源的时候尽可能选取便宜的计算资源, 而且选取的资源存在资源预留的情况, 从而造成 $J2$ 选取资源时候存在许多时间片段的情况. 而且, 任务运行的不可分解性决定了 $J2$ 有时无法利用便宜资源中的时间片段. 因此, 当 $J2$ 能够利用比较便宜的计算资源的时候, 其运行时间较接近限制的时间, 反之亦然.

图 9 比较在运行作业 $J1$ 和 $J2$ 不同运行时间限制下, 不同资源的运行时间. 其中作业 $J2$ 在时间限制为 $3*10^4$ 秒时无解, 在图中没有表示其资源的运行时间. 从图 9 中, 可知计算资源 $R2$ 和 $R4$ 的运行时间较多, 这是因为 $R2$ 和 $R4$ 的性价比较高, 计算性能最高的、最昂贵的资源 $R5$ 主要用在时间限制要求较高的场合下; 而计算性能最低、最便宜的资源 $R3$ 主要用在时间限制要求较低的场合下. 资源 $R1$ 主要运行在作业无法使用 $R4$ 的一些场合下.

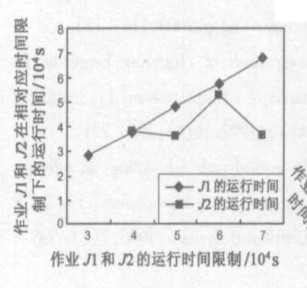

图 8　作业 $J1$ 和 $J2$ 在不同运行时间限制时的运行时间

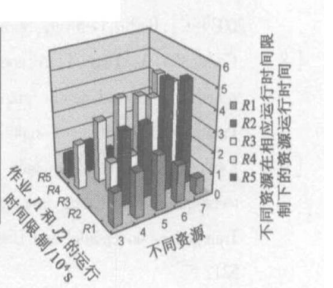

图 9　在作业 $J1$ 和 $J2$ 不同运行时间限制下的资源运行时间

4　结束语

网格环境下基于时间限制的费用最小的 DAG 调度算法运用经济规律把网格用户中的任务映射到网格资源中运行, 弥补了 Buyya 提出的算法未考虑任务之间相互运行的优先关系的不足. 该算法有两个关键技术: DAG 中有效路径的提取能够有效地定位任务何时在资源上运行; DAG 中在作业运行时间尽可能允许的情况下, 尽可能地把任务映射到价格便宜的资源上运行. 但是, 该算法也有不足之处: 为了使得作业运行费用最小, 造成资源中存在很多的时间碎片, 使得作业的任务有时候很难有效地利用资源. 本算法有些地方需要进一步完善: 当作业的任务之间传输量很大的时候, 需要考虑任务之间的通信费用和通信时间. 把这种情况考虑到算法中是非常复杂的, 这需要进一步深入的研究.

参考文献:

[1] Foster I, Kesselman C, Tuecke S. The anatomy of the grid: Enabling scalable virtual organizations[J]. International Journal of High Performance computing Applications, 2001, 15(3): 200-222.

[2] Buyya R, Abramson D, Giddy J, Stockinger H. Economic models for resource management and scheduling in grid computing[J]. Journal of Concurrency: Practice and Experience, 2002, 14(13): 1507-1542.

[3] Buyya R, Vazhkudai S. Compute power market: Towards a market-oriented grid[A]. Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid[C]. Los Alamitos, CA, USA: IEEE Comput. Soc, 2001. 574-581.

[4] Ernemann C, Hamscher V, Yahyapour R. Economic scheduling in grid computing[A]. Job Scheduling Strategies for Parallel Processing. 8th International Workshop, JSSPP 2002[C]. Berlin, Germany: Springer Verlag, 2002. 128-152.

[5] Moreno R, Alonsr-Conde A B. Job scheduling and resource management techniques in economic grid environments[A]. Grid Computing. First European Across Grids Conference[C]. Berlin, Germany: Springer Verlag, 2003. 25-32.

[6] Buyya R, Murshed M. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing[J]. Concurrency and Computation Practice & Experience, 2003, 14(13-15): 1175-1220.

[7] Feng H L, et al. A deadline and budget constrained cost-time optimization algorithm for scheduling dependent tasks in grid computing[A]. Grid and Cooperative Computing. Second International Workshop (GCC 2003)[C]. Berlin, Germany: Springer-Verlag, 2003. 113-120.

[8] Gerasoulis A, Yang T. A comparison of clustering heuristics for scheduling directed acyclic graphs of multiprocessors[J]. Journal of Parallel and Distributed Computing, 1992, 16(4): 276-291.

[9] Kwok Y K, Amad I. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors[J]. IEEE Transactions on Parallel and Distributed Systems, 1996, 7(5): 506-521.

[10] Malloy B A, Lloyd E L, Soffa M L. Scheduling DAG's for asynchronous multiprocessor execution[J]. IEEE Transactions on Parallel and Distributed Systems, 1994, 5(5): 498-508.

作者简介:

陈宏伟 男, 1975 年生于湖北武汉, 南京邮电学院通信与信息系统专业博士研究生, 主要研究方向为网格技术、p2p 技术、移动代理和信息安全等.

王汝传 男, 1943 年生于安徽合肥, 南京邮电学院教授、博士生导师, 主要研究方向是计算机软件、计算机网络、信息安全、移动代理和虚拟现实技术等. E-mail: wangrc@njupt.edu.cn.