第33卷 第24期 计 算 机 工 程 2007年12月
Vol.33 No.24 Computer Engineering December 2007
·软件技术与数据库· 文章编号：1000—3428(2007)24—0037—04 文献标识码：A 中图分类号：TP311

网格计算环境下费用-时间优化 i-DAG 调度算法

高承实，付江柳，戴 青
(解放军信息工程大学电子技术学院，郑州 450004)

摘 要：由于现有算法在网格环境下不能很好地解决资源有偿服务和满足用户的 QoS 需求间的问题，该文通过综合考虑作业中任务之间的优先关系，给出费用-时间优化的 i-DAG 调度算法，在保证作业时间期限的条件下，利用所求的最大路径，最大程度上将任务集中映射并映射到较便宜的资源上，减少了作业的计算开销和通信开销。实验仿真证明了算法的优越性。

关键词：网格计算；资源调度；最大路径

Cost-time Optimization i-DAG Schedule Algorithm for Grid Computing

GAO Cheng-shi, FU Jiang-liu, DAI Qing
(Institute of Electronic Technology, PLA University of Information Engineering, Chengzhou 450004)

【Abstract】 In the current grid environment, the existing schedule algorithm can not solve the issues of paying for using the resources and satisfying the QOS need of the customer. Considering priority in subtasks of the job, an cost-time optimization i-DAG schedule algorithm is presented, which can solve the existing issues well. The sets of the maximum path are gained from the algorithm, with the restriction of job running time, the subtasks are concentrative mapped into cheap resources so that the algorithm can save computing cost and communications cost of the job. A simulative example based on the algorithm is also provided to prove that the algorithm has good characteristics.

【Key words】 grid computing; resource schedule; maximum path

网格的使用模式是用户通过向网格系统提交作业来共享网格资源，作业由一些不可分割的任务组成，网格资源调度程序按照某种策略将这些任务分配给合适的资源[1]。在实际应用中，一方面，大量的资源并不是无偿使用的，要吸引资源提供者加入网格，就必须保证他们的利益。另一方面，用户希望在自己的约束条件下(完成时间、服务质量等)，尽可能开销小而又能得到高质量的服务。高效调度算法或策略可以充分利用网格系统的处理能力，满足用户要求。现有的解决方案是将传统操作系统的遗传算法、蚂蚁算法等[2-3]调度算法直接移植到网格的资源管理中，不能很好地解决资源有偿服务和满足用户的QoS需求间存在的问题。文献[4]提出的基于限期和预算费用-时间优化算法，将经济规律运用到网格资源调度中，综合考虑时间、费用和资源性能等因素来满足用户的QoS需求(完成期限和预算)，在一定程度上解决了现有问题，但该算法假设完成作业的各任务之间相互独立，所以不具有普遍性。文献[5]提出了任务间存在优先关系的费用-时间优化DAG算法，利用有效路径找出各任务的时间约束条件，实现资源和任务的映射。但当多个作业同时需要调度时，存在开销较大的问题。因此，本文给出了更好解决费用-时间优化的i-DAG调度算法。

1 费用-时间优化的DAG调度算法[5]

问题描述：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。$DAG=(V,T,E)$ 表示作业，$V=\{v_1,v_2,\cdots,v_n\}$，V 是任务集，顶点 v_i 与任务 t_i 相对应；E 是边的集合，$E=\{e_1,e_2,\cdots,e_x\}$，$e_k=(v_i,v_j)$ 表示有向边 $v_i\rightarrow v_j$，它表示任务 v_i 必须在任务 v_j 之前运行；$T=\{\tau_1,\tau_2,\cdots,\tau_n\}$，$\tau_i$ 是每个任务 v_i ($v_i\in V$)的计算量。已知网格资源集合 $R=\{r_1,r_2,\cdots,r_m\}$，$U(r_i)$ 表示资源 r_i 的单价，$p(r)$ 表示资源的计算能力，$c(r,v)$ 表示任务 v 在资源 r 上运行的费用。$F(r,v)$ 在资源 r 上运行任务 v 的结束时间。设作业 J 时间限制为 *deadline*，即所有任务必须在 *deadline* 内完成，总费用为 *cost*，问题的求解目标是

$$\min(\cos t)=\sum_{i,j=1}^{n}c(r_j,v_i)\quad \forall F(r_j,v_i)\le Deadline, r_j\in\{r_1,\cdots,r_m\}$$

DAG 算法弥补了文献[4]算法的缺陷，但其提取的每条有效路径含有大量重复节点，在取消某任务节点与资源的映射时，会造成多条路径上的任务节点重新映射。当有多个作业同时调度时，先来的小作业抢占了便宜的资源，后到的大作业只能使用较昂贵的资源，没有有效地减少运行开销。

2 费用-时间优化 *i*-DAG 调度算法

针对 DAG 算法存在的问题，在 DAG 从起点到终点的路径中，任务计算量之和最大的路径完成的时间最有可能接近作业的时间限制。可以先为这条路径上的任务分配合适的资源，以此确定其他任务的开始和结束时间。在分配资源时，尽量为任务量大的任务分配便宜的资源，并尽量将路径上的任务在较便宜的资源上集中运行，以减少作业的计算开销和通信开销。

基金项目：国家“973”计划基金资助项目(TG1999035801)；国家自然科学基金资助项目(6053012)
作者简介：高承实(1973－)，男，博士研究生，主研方向：信息安全；付江柳，硕士研究生；戴 青，教授
收稿日期：2006-12-25 **E-mail：**gjians@sina.com

2.1 求 DAG 中的最大路径集合

对 DAG 进行变形，使在 DAG 中入度为 0 的节点(即起点)和出度为 0 的节点(即终点)唯一：若 DAG 有两个或两个以上的起点，在 DAG 中加入一个虚拟任务节点 v_{root} ($\tau_{root}=0$)，使之成为所有起点的父亲；若有两个或两个以上的节终，在 DAG 中加入一个虚拟任务节点 v_{end} ($\tau_{end}=0$)，使之成为所有终点的孩子。在 DAG 图中给出以下 4 个定义：

定义 1 在 DAG 中在 $v_0 \to \cdots v_n$ (v_0 是入度为 0 的点，v_n 为出度为 0 的点，$v_0,\cdots,v_n$ 都不相同)所有通路中 $\sum\tau$ 最大的通路，称之为最大路径。

定义 2 节点 v_i 的级值 $l(v_i)$ [2]：

$$l(v_i)=\begin{cases}0 & parent(v_i)=\varnothing \quad parent(v_i)\text{为空}\\ 1+\max(l(v_j)) & \text{否则，}v_j\in parent(v_i)\end{cases}$$

定义 3 节点 v_i 的最优值函数 $f(v_i)$ =从 v_i 到终点的最大路径上的 $\sum\tau$。

定义 4 节点 v_i 的转移函数 $u(v_i)$ =从 v_i 到终点的最大路径上的 v_i 的孩子。

如果由起点 v_0 经过 v_j 和 v_k 到达终点 v_n 是一条最大路径，则由 v_j 出发经过 v_k 到达终点 v_n 的这条子路线必定也是从 v_j 到终点 v_n 的最大路径。利用节点的最优值函数和转移函数寻找最大路径，根据节点的级值对 DAG 进行变形，使级值相同的节点位于同一层。从 DAG 中级值最大的节点开始，由后向前递推，求出各节点的最优值函数，最后求得 v_0 的最优值函数。再利用转移函数反推，可得到最大路径节点集合。再求 DAG 中另一条次最大路径，重复直到所有的节点都存储到相应的集合中。以下给出获取所有最大路径集合的算法：

```
void CreatePath(PathSet S) //创建最大路径
{
  Path tempP;
  Node tempV;
  if(!S.IsEmpty())//不是第一次求最大路径
  {
graph.SetZero(S.currentpath); //把上一次求得的最大路径节点的
//计算开销置零
  }
  //第一次求最大路径
  else
  {//从起点开始求
        tempV = graph.root;
        while(!tempV.IsEnd())
        {
              S.AddCurrentPath(tempV);
              tempV = tempV.Transfer();
        }
        S.AddCurrentPath(tempV);//最后把终点加入
  }
}
void BestValue(Node V)//最优值函数
{//如果是终点，其最优值就是该节点的计算开销
  if(V.IsEnd())
  {
        V.bestvalue = V.computecost;
        V.transfer = V;
  }
  //如果不是终点，要先计算其子节点的最优值
  else
  {
        V.bestvalue = V.computecost + V.maxofchild();
        V.transfer = V.GetMaxNodeOfChild();
  }
}
void RecordPath(PathSet S)
{
  Path result[num];
  Path tempPath;
  int i = 0;
  tempPath = S.GetFirstPath();
  while(!tempPath.IsEmpty())
  {
        tempPath.SortByLevel();//按级值递增排列
        tempPath.DeleteZero();//去掉零点
         按零点将集合分成 k 部分;
         将 这 k 部 分 依 次 放 入 result[i],
result[i+1],…,result[i+k-1]
        i+k;
  tempPath = S.GetNextPath();
  }
}
```

2.2 任务与资源映射

假设资源的单价是资源计算能力的单调非递减函数，即计算能力越强的资源单价越高。DAG 中任务与资源映射需要用到资源映射表，每个资源都有一张映射表，每个表说明资源在哪个时间段内分配给哪个任务。每个任务只能在一个资源上运行。

算法的大致思想如下：对网格中满足作业时间限制的空闲资源按单价递增进行排序，单价相同则按计算能力递增排序；为最大路径算法中输出的任务集合$s_1,s_2,\cdots,s_k$依次分配资源：在满足时间限制的资源中寻找可执行完集合中的全部任务，且开销较小的资源，若存在，将集合上的任务分配，尽量将相关联的任务在一个资源上运行以减少通信开销。若不存在，则对路径上的任务依次按照时间限制，选取尽可能便宜的资源，当任务无法找到满足时间限制条件时，更新集合中已完成的运行最慢的任务和资源的映射关系。若仍无法完成任务与资源的映射，则映射失败。具体算法下：

```
bool CreateDAGMap()
{
MapTable maptable;//任务与资源映射表
Path S[w];//最大路径数组 S，其中，w 表示 S 中元素的个数
Resource r[m];//资源数组，其中，m 表示资源的个数
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(int i=0; i<w; i++)
{
S[i].deadline=S[i].last.LFT-S[i].first.EST;
S[i].total =集合中所有节点的计算量总和;
for(int j=0; j<m; j++)
  {
```

```
r[j].leisure =p(r[j])×r[j]的空闲时间;
            if(r[j]].leisure >= S[i].total)
          {
              S[i]集合中的所有任务在 r[j]上分配成功;
                break;
}
      }
if(i>0 && S[i]只有一个任务)
      {
            if(S[i].v 是 DAG 的起点)
            S[i].v.EST=Job.EST;
            if(s[i].v 是 DAG 的终点)
          S[i].v.LFT=Job.LFT;
        S[i].v.EST = PRED(S[i].v).LFT;
          S[i].v.LFT = SUCC(S[i].v).EST;
for(j=0; j<m; j++)
          {
            if(r[j]满足 S[i]的时间限制)
            {
                S[i]在 r[j]上分配成功;
                break;
              }
           }
return FALSE;// 分配失败
}
        else
        {
              for(int k=0; k<num; k++)//num 为 S[i]的任务数
              {
                    for(j=0; j<m; j++)
                    {
  if(r[j]满足时间限制 S[i].last.LFT- S[i].v[k].EST)
把 r[j]分配给 S[i].v[k];
                              break;
                    }
}
}
do
{
                    Unmap(S[i].v[l], S[i].v[l].r);
                  计算任务 S[i].v[k]和 S[i].v[l]的时间限制;
                  if(找到满足时间限制的资源)
                  {
将 S[i].v[k]  和 S[i].v[l]分配;
更新映射表中的记录;
Break;
}
}until(无法找到满足 S[i].v[k]的资源)
                return FALSE; //分配失败
              }
        }
  }
  return TRUE; //全部分配成功
}
```

3 实验仿真

已知作业J1(图 1)和作业J2(图 2)，假设：(1)运行时间限制为 5×10^4 s，且图 1、图 2 中的节点上半部分是作业中任务的编号，下半部分是任务的计算量，单位为 10^6 MI，运算能力单位为 10^3 MI/s；(2)在网格环境下计算资源如表 1 所示。

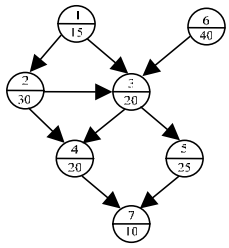


图 1　作业 J1 的 DAG

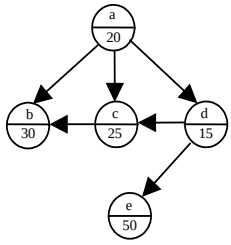


图 2　作业 J2 的 DAG

表 1　网格环境下计算资源表

资源	单价/$\times10^6$ MI	运算能力/$\times10^3$ MI•s^{-1}
R1	0.9	2.5
R2	0.8	2.0
R3	0.6	1.0
R4	0.9	3.0
R5	1.2	50

将两个作业的 DAG 根据级值变形，相同级值的节点位于同一层，变形后如图 3 所示。其中，*root* 和 *end* 节点为虚拟节点，它们的计算量 τ_{root} 和 τ_{end} 为 0，使 DAG 中起点和终点唯一。

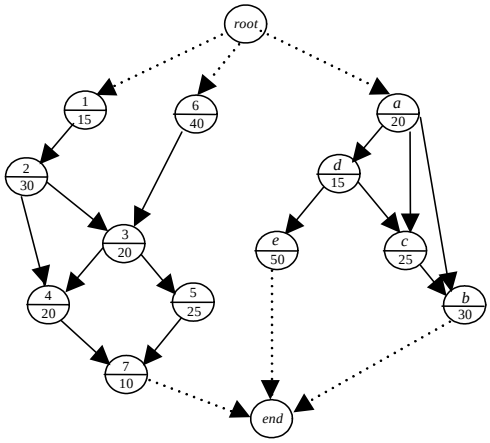


图 3　两个作业的 DAG 根据级值的变形

图 3 中的最大路径集合为S_1={1, 2, 3, 5, 7}，S_2={*a*, *d*, *c*, *b*}，S_3={6}，S_4={4}，S_5={*e*}。根据上述算法，作业J1 和作业J2 运行的甘特图如图 4 所示，在 5×10^4 s的时间限制内有合适的资源能够执行完两条最大路径上的所有任务，任务运行相对集中，同时资源的时间碎片相应减少。

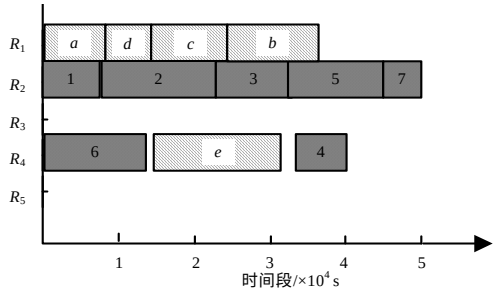


图 4　作业 J1，J2 在每个资源上的分配情况

当无法完成映射时，可以按照先来先服务的原则去除后到的资源，重新映射。图 5 是作业 J1 和 J2 在不同时间限制时计算开销的比较，可知，时间限制较小时，第一个能够满足运行最大路径上所有任务的资源常常是偏贵的资源，任务能够集中运行，但计算费用较高。随着时间限制的增加，最

大路径集合中任务计算量大的集合最先找到合适且便宜的资源，后面集合中的任务只能使用较昂贵的资源。时间限制越大，便宜资源的空闲计算能力越强，作业的计算费用越便宜且逐渐趋于平稳。

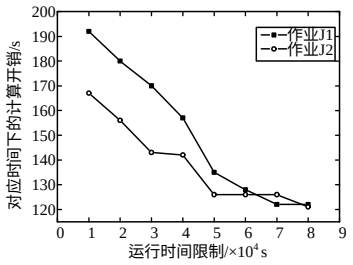


图 5　作业 J1, J2 在不同时间限制下对应的不同计算费用

图 6 是作业 J1 和 J2 在不同运行时间限制下，不同资源运行时间的比较。

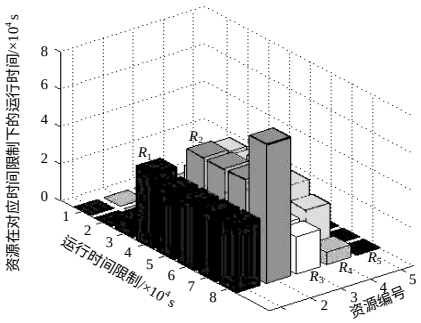


图 6　不同资源在不同时间限制下对应的使用时间

在时间限制较小的情况下，任务多集中在计算能力大、价格高的资源上运行。随着时间限制的增大，便宜资源上运行任务的时间逐渐增加。

4　结束语

针对任务之间存在优先关系的作业与资源的映射，本文提出了费用-时间优化 *i*-DAG 调度算法。该算法先求出最大路径集合，再实现任务与资源的映射，将最大的任务映射到较偏宜的资源，减少了作业的计算开销。同时尽量保证相关联的任务在较少的资源上运行，对于任务间数据传输频繁的作业，节省了通信开销。但该算法为了保证多个任务在尽量少的资源上运行，在时间限制较小的情况下很难充分利用价格便宜的资源，同时在运行任务的资源上存在一些时间碎片，需要进一步完善。

参考文献

[1] 查　礼, 徐志伟, 林国璋, 等. 基于 Simgrid 的网格任务调度模拟[J]. 计算机工程与应用, 2003, 39(14): 91-93.

[2] 林剑柠, 吴慧中. 基于遗传算法的网格资源调度算法[J]. 计算机研究与发展, 2004, 41(12): 2190-2194.

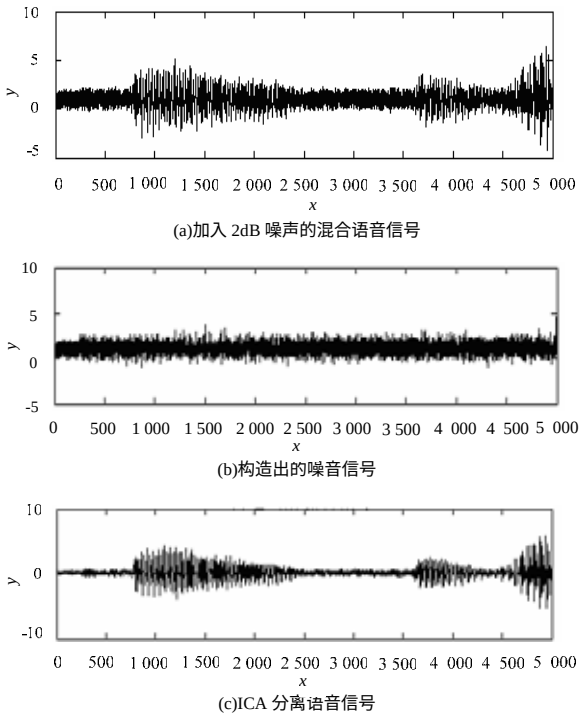
[3] 许智宏, 孙济洲. 用蚂蚁算法进行网格任务调度的研究[J]. 计算机应用, 2005, 25(10): 2236-2237.

[4] Buyya R, Murshed M, Abramson D. A Deadline and Budget Constrained Cost-time Optimize Algorithm for Scheduling Parameter Sweep Application on the Grids[EB/OL]. (2001-12-07). http://www. buyya.com/gridsim.

[5] 陈宏伟, 王汝传. 费用-时间优化的网格有向无环图调度算法[J]. 电子学报, 2005, 33(8): 1375-1381.

(上接第 36 页)

进行 FASTICA 分离，得到 ICA 去噪语音信号。仿真结果如图 2 所示。



(a)加入 2dB 噪声的混合语音信号

(b)构造出的噪音信号

(c)ICA 分离语音信号

图 2　基于 ICA 的单通道语音降噪仿真

笔者做了不同信噪比的语音增强实验。实验结果如表 1 所示。信噪比的定义为 $SNR = 10 \lg (P_S / P_R)$ (dB)。其中， P_S 为信号的功率， P_R 为噪声的功率。

表 1　不同信噪比的语音增强实验对比

输入/输出信号	信噪比/dB		
输入信号	2	5	10
ICA 去噪信号	6	8	12

4　结束语

针对独立分量分析难以应用到单通道信号去噪的问题，本文提出了一种通过构造一路噪声信号进行独立分量分析的方法，实验结果表明，通过构造一路合适的噪声信号，可以较好地解决独立分量分析不能用于信号消噪的问题，获得比较好的消噪结果，提高了输入信号的信噪比。

参考文献

[1] Comon P. Independent Component Analysis——A New Concept[J]. Signal Processing, 1994, 36(3): 287-314.

[2] Hyvarinen A, Oja E. Independent Component Analysis: Algorithm and Applications[J]. Neural Networks, 2000, 13(4): 411-430.

[3] Hyvarinen A. Survey on Independent Component Analysis[J]. Neural Computing Surveys, 1999, 2(1): 94-128.

[4] 彭　煊, 刘金福, 王炳锡. 基于独立分量分析的语音增强[J]. 信号处理, 2002, 18(5): 477-479.

[5] Hyvarinen A, Oja E. A Fast Fixed-point Algorithm for Independent Component Analysis[J]. Nerural Computation, 1997, 9(7): 1483-1492.

[6] 刘开文, 何培宇, 张　玲. 基于短时综合叠接相加法的语音盲信号分离研究[J]. 四川大学学报, 2003, 40(6): 1071-1074.