基于效益函数的网格任务调度算法*)

游新冬[1] 常桂然[2] 陈学耀[3] 田翠华[1]
(东北大学信息科学与工程学 沈阳 110004)[1] (东北大学计算中心 沈阳 110004)[2]
(东软集团网络安全事业部 沈阳 110058)[3]

摘 要 在动态、异构、分布广泛的网格环境中,对资源的调度是一个非常复杂而重要且具有挑战性的问题。本文针对网格环境中的动态性特点,特别是用户 QoS 要求的动态变化性,提出了一种基于效益函数的网格任务调度算法,并采用 GridSim 模拟器分别对该调度算法和模拟器自带的代价最优和时间最优的网格任务调度算法进行模拟。实验的结果表明:该调度算法更能体现用户对 QoS 要求的动态变化;在系统完成相同数量的网格任务时,消耗相同时间的情况下,该调度算法在代价上优于基于时间优化的调度算法;而花费相同预算的情况下,在时间上优于基于代价优化的调度算法。

关键词 网格计算,任务调度,效益函数,服务质量

Scheduling Algorithm in Grids Based on Benefit

YOU Xin-Dong[1] CHANG Gui-Ran[2] CHEN Xue-Yao[3] TIAN Cui-Hua[1]
(School of Information Science & Engineering, Northeastern University, Shenyang 110004)[1]
(Computing Center, Northeastern University, Shenyang 110004)[2]
(Network Security Division, Neusoft Co. Ltd. Shenyang 110058)[3]

Abstract Scheduling grid tasks onto the dynamic, heterogeneous and widely distributed grid environments is a complex and challenging task. According to dynamic characteristic in grids, especial for the dynamic QoS request by the users, a scheduling algorithm based on benefit is present in this paper. Running the algorithm onto the simulator GridSim and comparing it to the algorithm based on deadline and the algorithm based on budget implies that the algorithm based on benefit take advantageous over the two algorithms of the simulator in some degree. Because of adapting to the dynamic request of QoS by the users, the algorithm not only achieves better performance in budget than the algorithm based on deadline when they finished the number of gridlets is same and spent equal deadline, but also in deadline than the algorithm based on budget on the assumption that they finished the same set of gridlets and consumed the equal quantity of budget.

Keywords Grid computing, Task scheduling, Benefit function, QoS

1 引言

网格[1]是在动态、异构、广泛分布的资源中实现计算资源、存储资源、通信资源、软件资源、信息资源、知识资源的全面共享以及协同计算。

任务的调度是根据任务和资源的特性采用不同的算法将任务映射到相应的资源上执行的过程。网格环境中的动态性、异构性以及资源的分布性,使得对资源的调度十分复杂且具有挑战性。传统的集中式调度算法在很大程度上不能适应网格环境的要求。从经济学的角度出发,对网格中的任务进行调度是一种新的思维方式,并取得了良好的效果[2]。Rajkumar Buyya 提出了基于代价优化和基于时间优化的调度算法,并设计了 GridSim 模拟器,对其提出的调度算法进行模拟。在该模拟系统使用资源是有价的,即不同计算能力的资源成本是不同的,用户只能给出相应的价格才能使用相应的资源。用户在提交的任务中设有参数 Deadline 和 Budget。Deadline 参数表明任务必须在该期限之内完成,如果没有资源能够满足该条件,则放弃任务的执行;Budget 参数表明执行任务所付出的价格总和不能超过该预算,否则放弃未完成任务的执行。另一方面,用户可以根据不同的 QoS 要求设置调度的参数 OPTIMIZATION_COST 或 OPTIMIZATION_TIME,而使系统分别采用基于代价优化和时间优化的调度算法。这两种调度算法能够在一定程度上满足用户的 QoS 要求:基于时间优化的调度算法总能在时间性能上优于基于代价优化的调度算法,而基于代价优化的调度算法则总能在预算方面的开销小于基于时间优化的调度算法,使得时间紧迫的任务可以通过增加预算来换取执行时间。反之,任务不紧急的情况下,用户可以开销少量的预算来完成任务。这是网格计算中管理资源的有效手段[2]。然而,在调度并执行用户的所有任务过程中,用户的 QoS 要求是静态的,要么是基于时间优化,要么是基于代价优化。

本文主要的创新点在于:用户的 QoS 要求是随着调度过程中时间和预算的消耗而变化的。用户在提交作业的同时,不是单单设置了 Budget 和 Deadline 这两个参数,而是根据这

*)高等学校教育博士点专项基金(20030145017)。游新冬 博士研究生;常桂然 教授、博士生导师。

两个参数构造了基于 Budget 和 Deadline 的效益函数[3]。基于 Budget 的效益函数中的效益值随着预算的消耗而变化。而基于 Deadline 的效益函数中的效益值随着时间的消耗而改变，系统在调度过程中，根据调度时刻效益值的不同而采用不同的调度算法。该调度算法反映了用户 QoS 要求的动态变化性，并保证系统对任务的调度能够使用户获得最大的“效益”，从而在充分满足用户 QoS 要求的同时提高了系统资源的利用率。

2 基于效益函数的调度算法

2.1 基于时间或代价优化的调度算法

时间最优算法：时间最优算法的出发点是尽量快地在预算范围内完成任务，基本思想是对每个资源考虑到以往分配的任务和完成率，估算一个任务的完成时间，再依据完成时间对资源按升序排序，再从队列中依次取出资源。如果该资源的成本小于或等于该任务的预算，则分配该任务给这个资源。重复以上步骤，直到所有任务分配完毕。该算法的描述如下：

1)对每个资源预测并建立任务消耗率，或者测量和推测可用资源份额，并考虑到处理以往任务的时间；

2)对每个资源而言，如果有在前一次调度中分配到的任务但还没有执行，且资源可用性方面又有了变化，把合适数量的任务移至未分配任务列表。这样，基于最新的资源可用性信息所做的调整有助于更新调度；

3)对未完成任务列表中的任务重复以下步骤：

- 从中选取一个任务；
- 创建一个资源群，包含可用的资源(即那些处理价格小于等于剩下的任务预算的资源)；
- 对资源群中的每个资源，计算预测任务完成时间，考虑到以前分配的任务和任务完成率和资源份额可用性；
- 对资源群中的资源按照任务完成时间的升序排序；
- 如果未分配任务列表中的任务的预测完成时间小于资源群中第一个资源的完成期限，则分配该任务给这个资源。

代价最优算法：尽量节俭地在完成期限内完成任务，基本思想是首先给资源按价格升序排序，对队列中的每个资源，在不超过完成期限的范围内分配尽可能多的任务。该算法的描述如下：

1)对每个资源进行负载分析，来建立任务消耗率，或者测量和推测可用资源份额；

2)对于每个资源，基于它的任务消耗率或可用资源份额，预测并建立该资源在完成期限前可以处理的任务数；

3)对于每个排好序的资源：

- 如果当前分配给该资源的任务数少于预测的该资源可完成的任务数，则从未分配任务队列中，或者基于任务状态和可行性从最昂贵的机器任务队列中，取出来更多的任务分配给它，并且保证任务的预算足以满足要求；
- 如果该资源已分配到的资源数超过了它在完成期限前所能完成的任务数，将那些多余的任务移到未分配任务队列。

2.2 效益函数的构建

$$\beta=\begin{cases}a, & t<bD\\ a-c(t-bD), & t\geqslant bD\end{cases} \qquad (1)$$

构建效益函数的主要目的：利用该函数动态地刻画随着时间或预算的消耗采用用户所能获取的“效益”的变化，使得系统在调度任务时有一个动态的依据或标准——最大化用户的“效益”。用户根据自己的需求定义各种不同的效益函数，最简单的情况是：基于时间的效益函数与任务完成时间成反比，而基于代价的效益函数与预算的消耗成反比。图 1 给出了几种常见的效益函数的图形。本文主要采用 1(c)的形式，式(1)是数学表达式，其中 a,b,c 是用户定义的常数。

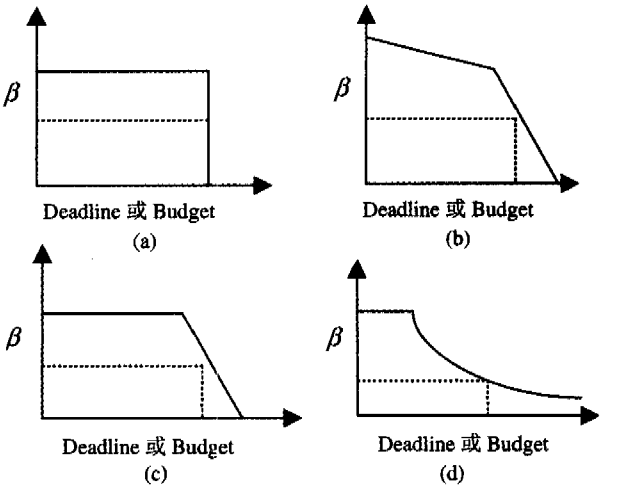

图 1 效益函数

2.3 提出的调度算法

本文采用的调度算法就是基于上述构建的基于 Deadline 或 Budget 的效益函数，在用户的 QoS 动态变化的过程中最大化用户的“效益”。用户在提交实验(包含多个任务)时，设置了效益函数所需的参数 Deadline 和 Budget，构建基于 Deadline 和基于 Budget 的效益函数。随着时间的消耗，基于 Deadline 的效益函数的效益值(*D*_ Benefit)动态地改变；而随着预算的消耗，基于 Budget 的效益函数的值(*B*_ Benefit)是动态变化的。系统就是根据这两个效益值的不同而采用不同的调度算法：若 *D*_ Benefit 大于或等于 *B*_ Benedit，则系统在调度过程中尽量减少时间的开销(即采用基于时间优化的调度算法)；反之，系统在调度时考虑的是最小化预算的开销(即采用基于代价优化的调度算法)。具体调度算法的代码片断如下：

```
scheduleWithBFC_BenefitOptimisation()
{ int scheduled=0; // 此次实验完成对 Gridlets 的调度个数
  Collections. sort(brokerResourceList_, new OrderCostTime());//
     将资源按照资源成本的升序排序
  BrokerResource br;
  LinkedList BrokerResourcesToUse=new LinkedList();
  LinkedList BRGridletTimeList=new LinkedList();//用于基于时
     间优化时的资源列表
  LinkedList BRGridletCostList=new LinkedList();//用于基于代价
     优化时的资源列表
  for(int j=0; j < glTempList. size(); j++)//对队列中未完成的
     Gridlet 进行调度
  { DFunction df=new DFunction(deadline, time_elapse, dpa, dpb,
     dpc);
  BFunction bf=new BFunction(budget, budget_expent, bpa, bpb,
     bpc);
  double Dbenefit=df. getBenefit(); double Bbenefit=bf. getBenefit
     (); //获取两种效益函数的效益值
  if(Dbenefit>=Bbenefit)//采用基于时间优化的调度算法(也是
     Min-Min 调度算法)
  { Gridlet gl=(Gridlet)glTempList. get(j);
    for(int k=0; k < brokerResourceList_. size(); k++)//将资
      源成本小于任务预算的资源放入一个列表
      { br =(BrokerResource)brokerResourceList_. get(k);
        if(br. getExpectedProcessingCost(gl) <= remainingBud-
          getForGridlet(gl))
          { BrokerResourcesToUse. add((BrokerResource)br); }}
    if(BrokerResourcesToUse. size()> 0)
```

```
{ for(int k=0; k < BrokerResourcesToUse.size(); k++)//
    将列表中的资源按完成任务的实际排序
    { BRGridletTimeList.add( new BRGridletProcessingTime(
        (BrokerResource) BrokerResourcesToUse.get(k),
        gl)); }
      Collections.sort(BRGridletTimeList,
      new OrderBRGridletProcessingTime());
      boolean gridlet_assigned_flag=false;
      for(int k=0; k < BRGridletTimeList.size() && !
      gridlet_assigned_flag; k++)
      //将没有分配的任务分派到排好序中的资源列表中第一
      个满足条件的资源上执行
      {   BrokerResource myBR = ((BRGridletProcessing-
          Time)
          BRGridletTimeList.get(k)).br;
          double Gl_completion_time=GridSim.clock()+
        ((BRGridletProcessingTime) BRGridletTimeList.get
        (k)).gridlet_processing_time;
        if( Gl_completion_time <=experiment_.getDeadli-
        neTime())
        {  gl.setGridletStatus(Gridlet.READY);
        l.setResourceParameter(myBR.resource.getResour
              ceID(),myBR.resource.getCostPerSec());
              glUnfinishedList_.remove(gl);
              myBR.glList.add(gl);
              scheduled++;
              gridlet_assigned_flag=true;
        }
    }
  }
  //时间和预算的变化
  int id=gl.getResourceID();
  time_elapse+=gl.getFinishTime()-gl.getSubmissionTime
      (id);
  budget_expent+=gl.getGridletLength() * gl.getCostPerSec
      ();
      }
  else // 如果 df<bf 则采用基于代价优化的调度算法
 {具体的代价优化算法见前文,这里略}
}
return scheduled;
```

3 模拟实验

该模拟实验是在网格模拟器 GridSim[4] 上运行的,与该模拟器中由 Rajkumar Buyya 提出的基于时间优化和基于代价优化的调度算法相比较。

3.1 模拟实验条件

为了对比算法的优劣性,在模拟实验中分别采用时间最优算法、代价最优算法以及本文提出的效益最优算法。运行该模拟实验的机器配置情况如下:Inter(R)Celeron(R)CPU 2.40GHz,504 Memory,Windows 2003 Server,Java JDK1.4.2。因考虑到单用户和多用户的条件下系统进行调度的情况大致相同,目前只是对单个用户的情况进行模拟。该用户提交的实验中有 10 个 Gridlets,并在模拟系统中创建了 6 个异构的计算资源。固定用户的期限 Deadline 为 250 个单位时间,Budget 从 36000 开始,步长为 500,一直到 40000。两种效益函数的大致关系如图 2。

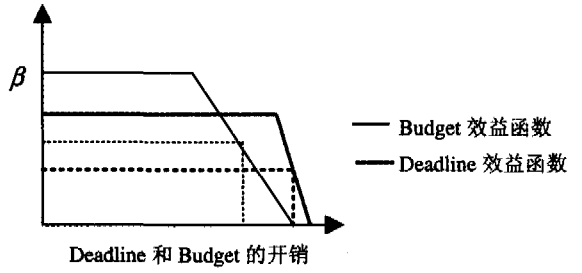

图 2 两种效益函数的大致关系图

3.2 模拟实验的结果及分析

从预算的消耗、时间的花费以及 Gridlets 的完成数量 3 个方面来衡量算法的性能。

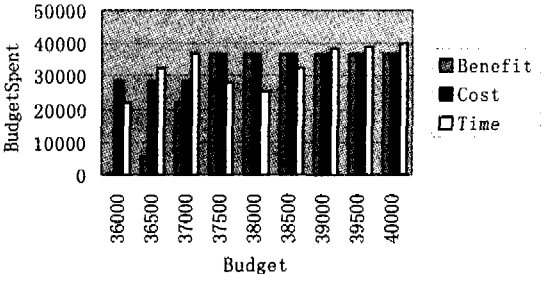

图 3 三种调度算法的实际花费预算

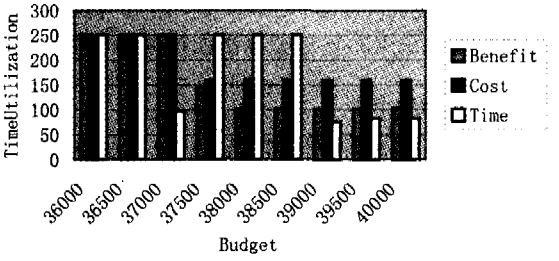

图 4 三种调度算法实际花费的时间

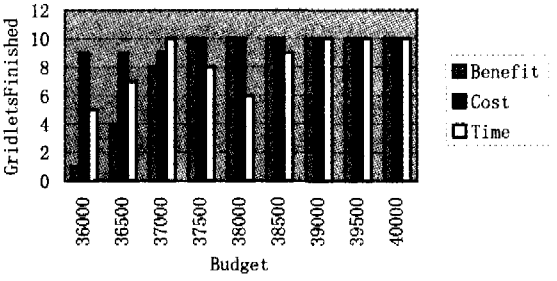

图 5 三种调度算法实际完成的任务数

从上面模拟实验的结果图可以知道:从整体上而言,代价最优的调度算法在预算的消耗上小于时间最优的调度算法,如图 3。时间最优算法在时间上的开销总是小于代价最优算法,如图 4。而基于效益函数的调度算法则因为能够在系统资源的动态变化过程中考虑到用户 QOS 要求的动态变化,充分地利用系统的资源,从而达到:在完成相同数量的 Gridlet 网格任务时,消耗相同时间的情况下,该调度算法在代价上优于基于时间优化的调度算法;而花费相同预算的情况下,在时间上优于基于代价优化的调度算法。

4 其它相关工作

网格系统中的任务调度算法主要有:DFPLTF,Suffrage-C,Min-Min,Max-Min 和 WQ 等,DFPLTF[5](Dynamic FPLTF)是针对动态的网格环境修改 FPLTF[6](Fastest Processor to Largest Task First)而成的调度算法。FPLTF 是静态的调度算法,在异构的并行环境中取得良好的时间性能。DFPLTF 对大任务设置比较高的优先级,如果同时有多个处理器可以提供给任务时,则从中随机选择一个处理器来执行。它需要对处理器速度和任务长度进行预测,计算出每个处理器完成某一任务的时间,以此为依据进行调度。Suffrage-C[7] 在调度任务时主要根据每个任务的 Suffrage 值:任务的 Suffrage 值随着网格中计算资源的改变而改变。该调度算法主要的思想就是:最先调度那些 Suffrage 值下降最快的任务。Suffrage-C 同样需要预测处理器的速度和任务的长度。文

[8]提出了 Min-Min 和 Max-Min 的任务调度算法，在 Min-Min 调度算法中小任务优先，并把任务分派到能最快完成该任务的处理器上执行；Max-Min 调度算法则考虑先调度大任务，在执行大任务的过程中，可以并行完成多个小任务。这两种调度算法都是基于处理器的时间，是静态的，在某一段时间是不变的，并不适合动态的网格环境。另外，还有 WQ[9](Work Queue)调度算法，主要的思想就是处理速度快的处理器总是比处理速度慢的处理器完成更多的任务，它最初是针对同构的并行系统提出来的一种调度算法，并且也需要对处理器的处理速度和任务长度大小进行预测。而针对网格的动态环境，以及上述几种调度算法的不足，文[10]提出了一种近似的任务调度算法 RR，该近似算法能在保证 n 个任务在 m 个处理器上的完成时间在 $1+\frac{m(\log_e(m-1)+1)}{n}$ 之内，并且不需要对处理器的速度和任务的大小进行任何的预测，在网格这样一个动态的环境中有非常重要的作用。然而，在上述的调度算法中，衡量算法的主要标准是给定条件下完成所有任务的时间，即 makespan，而没有考虑到用户的 QoS 要求。文[2]在考虑用户进行任务调度时的 QoS 的基础上，提出了时间最优和代价最优的调度算法，实验表明这两种调度算法能够在一定程度满足用户的 QoS 要求，本文提出的调度算法在文[2]的基础上，针对用户 QoS 的要求的动态变化性提出了效益最优的调度算法。

结论和展望 针对极具动态性的网格环境，充分考虑到用户 QoS 要求的动态变化性，提出了一种基于效益函数的网格任务调度算法，构建了基于 Deadline 和基于 Budget 的效益函数，用户的动态 QoS 要求主要通过这两个效益函数来体现。由于考虑了资源和用户 QoS 的动态变化，效益最优的调度算法能够充分地利用系统资源，提高资源的利用率，最终达到提高系统性能的目的。在 Gridsim 模拟器上构建的模拟实验结果，表明该算法相比于时间最优和代价最优的调度算法在预算和时间的开销上都有更好的性能。

目前的模拟实验考虑的还是单用户、多任务的情况，构建的效益函数也仅仅考虑了图 1(c)的情况。为了使模拟实验更加符合网格环境，下一步的工作是在多用户、多任务环境下构建更能够体现用户 QoS 要求的效益函数，来进一步评价效益最优这个调度算法的性能。

参 考 文 献

1 Foster I, Kesselman C. The Grid2: Blueprint for a New Computing Infrastructure [M], Morgan Kaufman, 2003

2 Buyya R. Economic-based Distributed Resource Management and Scheduling for Grid Computing [D]. Melbourne, Australia: Monash University, 2002

3 丁箐，陈国良，顾钧. 计算网格环境下一个统一的资源映射策略[J]. 软件学报，2002，13(7)：1303～1308

4 Buyya R, Murshed M. GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing. The Journal of Concurrency and Computation: Practice and Experience, 2002, 14(13-15)

5 Paranhos D, Cirne W, Brasileiro F. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In: International Conference on Parallel and Distributed Computing (Euro-Par), Lecture Notes in Computer Science, 2003, 2790: 169～180

6 Menasc'e D A, Saha D, Porto S C D S, et al. Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. Journal of Parallel and Distributed Computing, 1995, 28: 1～18

7 Casanova H, Legrand A, Zagorodnov D, et al. Heuristics for scheduling parameter sweep applications in grid environments. In: 9th Heterogeneous Computing Workshop (HCW), 2000. 349～363

8 Ibarra O H, Kim C E. Heuristic algorithms forscheduling independent tasks on nonidentical processors. Journal of the ACM, 1977, 24(2): 280～289

9 Graham R L. Bounds for certain multiprocessing anomalies. Bell System Technical Journal, 1966, 45: 1563～1581

10 Fujimoto N, Hagihara K. A Comparison Among Grid Scheduling Algorithms for Independent Coarse-Grained Tasks. In: SAINT 2004 Workshop on High Performance Grid Computing and Networking, IEEE Press, January 26-30, 2004. 674～680

(上接第 84 页)

移动节点(MN)和它的当前路由器之间的双向可到达的确认机制，即路由器可以知道节点是否继续可达，同时移动节点(MN)也可以随时知道当前路由器是否继续可达。如果移动节点(MN)检测到当前路由器不再可用，它就会去请求另外一台路由器，因面很好地解决了黑洞问题。

结论 经分析车载计算机系统的功能，结合移动 IPv6 和移动 IPv4 的优缺点，可以得出以下结论：移动 IPv6 基本协议的设计目标是保障终端能以一个固定 IP 地址连到任何链路上，并在切换链路时保持已有通信不被阻断，移动 IPv6 很好地解决了三角路由问题，具有更高效的路由效率，减少了分组延迟以及 HA 的工作负担。在 QOS、安全保障和互操作性等方面都比 IPv4 有明显的优点，只有基于 IPv6 的车载计算机系统才能满足未来对车辆驾驶性能、舒适性、安全性的需要，因此，对基于 IPv6 的移动车载计算机系统进入深入研究将是一个重要的应用研究领域。

参 考 文 献

1 Johnson D, Perkins C, Arkko J. Mobility Support in IPv6. IETF draft, draft—left—mobileip—ipv6— 21. Txt. 2003

2 周树清，宋伟. IPv6 在移动通信中的应用. 山东通信技术，2005(3)

3 Arkko J, et al. Using IP-sec to Protect Mobile IPv6 Signaling between Mobile Nodes and Home Agents [Z]. IETF Internet draft, 2002

4 IETF RFC3588-2003. Diameter Base Protocol

5 彭雪海，马琳，张宏科. 移动 IPv6 关键协议技术的研究. 2005. 2

6 蔡一兵，王春峰，孙利民，李忠诚. 网络移动支持研究. 计算机科学，2005(7)

7 LIU Chuda, LIU Yi, SUN Haitao, et al. QOS Provisioning in Mobile IPv6. Journal of Changsha University of Electric Power (Natural Science), November 2004

8 Johnson D, Perkins C, Arkko J. RFC3775 Mobility Support in IPv6. June 2004