基于关键作业动态预测的P2P网格系统工作流调度算法*

狄盛[1+]，王卓立[1]

[1](香港大学 计算机科学与技术系，香港)

Task Scheduling based on Dynamic Critical Task Estimation in P2P Grid Workflow*

Sheng Di[1+], Cho-Li Wang[1]

[1](Department of Computer Science and Technology, The University of Hong Kong, Hong Kong, China)

+ Corresponding author: Phn: 00852-62340776, Fax: 00852-25497908, E-mail: sdi@cs.hku.hk, http://www.cs.hku.hk

Abstract: One challenging research problem is how to cooperate and schedule the workflows submitted by different users, in a fully decentralized P2P computing environment composed by autonomous heterogeneous desktop computers, obtaining as high resource utilization as possible and optimized task execution efficiency. This paper first analyzed several new difficulties of the workflow in P2P computing environment, and then proposed a distributed task scheduling algorithm in terms of dynamic estimation on the critical task of workflows. Such algorithm runs on every participating node and schedules local tasks by leveraging the best-response dynamics on the run-time estimation. Via simulation, we validate that this algorithm owns high approximated optimized average throughput, finish-time, and execution efficiency of workflow in the competitive P2P computing systems.

Key words: Task Scheduling, Workflow, P2P Grid System, Probability Theory

摘 要: 在由自治异构的桌面计算机构成的全分布式 P2P 计算环境下，一个挑战性问题是如何协调和调度不同用户异步提交的工作流，从而达到尽可能高的资源利用率和最优化的作业执行效率。本文深入分析了工作流在P2P 计算环境中需要面临的新难点，并提出了一种基于对工作流关键作业动态预测的分布式调度算法。该算法独立的运行于每个参与节点上，通过运行时的动态估计以最佳响应动态的方式调度本地的作业。通过仿真测试，证明了该算法在竞争的 P2P 计算环境下具有近似最优的平均吞吐量、平均完成时间和平均执行效率。

关键词: 作业调度；工作流； P2P 网格系统；并行计算

1 引言

随着 P2P 技术的飞速发展和网格技术的日趋成熟，基于 P2P 的网格系统已经显示出其强大的计算潜力并成为不可替代的计算模式。目前典型的 P2P 网格系统包括 Condor-Flock P2P [1], Alchemi [2], Harmony [3],

* This research is supported by China 863 grant 2006AA01A111 and Hong Kong RGC grant HKU7176/06E.

作者简介: **狄盛**(1981－),男,山东济南人,博士生,主要研究领域为 P2P 网格系统下的作业调度及负载均衡;**王卓立**(1961－),男,副教授,博士生导师,主要研究领域为 P2P 网格系统,普适计算,基于 transaction 的共享内存并行模型,多核计算。

Entropia [4], P2PGrid[5], OurGrid[6], 等。在 P2P 网格系统中，每个参与节点既是资源的提供者，也是资源的请求者，因此和传统的计算网格[7]相比，工作流作业调度面临着更多更复杂的问题。

首先，每一个工作流(Workflow)中的作业之间往往会存在控制或数据依赖关系，从而形成多条执行路径。该依赖关系可以表示成一个有向无环图(DAG)，其中结点[1]代表作业，有向边表示作业之间的控制依赖或数据依赖。并且称从初始作业(第一个提交的作业)到终止作业(最后一个完成的作业)之间的可行的链接为该工作流的一条路径。其中，把决定该工作流最终完成时间的路径称为“关键路径”(*Critical Path*)，而该路径上的作业则称为“关键作业” (*Critical Task*)。在 P2P 网格系统中，因为同一个工作流的不同作业可能被调度在多个独立的异构资源上，因此如何判断和估计关键作业是一个难点。本文通过深入分析关键作业的特点以此做出最佳响应动态预测，并提出一种 P2P 计算环境下的动态调度算法对不同优先级的作业做出优化的调度策略。

其次，P2P 网格系统需要更加灵活的动态资源发现机制。P2P 环境中，任何资源节点在任何时刻都可能强行的加入或离开系统，从而导致不稳定的系统状态。一个健壮有效的分布式信息收集策略是各节点高效自治的进行作业调度的先决步骤。由于资源的高度动态性，我们采用轻量级的 *gossip protocol* [8,9]完成资源信息的聚合。在 *gossip protocol* 中，每个节点周期性的随机更换少量 (往往是 $log(n)$,其中 n 为系统规模)的节点作为其动态邻居节点并交换状态信息,从而让每个节点的实时状态信息可以像病毒一样扩散通知给其他节点。

最后，我们根据 Waxman Model 通过 Brite 生成器构造了一个模拟的互联网结构，并利用 PeerSim 工具实现 P2P 异步事件仿真，从而对 P2P 网格环境下基于工作流的关键作业预测调度算法作了全面的测试和分析。

2 相关工作介绍

SwinDeW [10] 和 SwinDeW-S [11]是建构在 Web Service 之上的基于 P2P 的工作流作业调度系统。该系统旨在协调广域分布的 Web 服务，将所有服务分成为多个存在数据依赖关系的虚拟组，然后分别在组内部进行作业调度。当前驱组的作业完成后，其工作将转移至后续组的某个节点上继续执行，从而实现基于 P2P 模式的工作流协作调度。但是，该系统没有对关键作业的选择做出细致的分析和预测。

一般的网格工作流主要采取集中的启发式作业调度算法，大致分为两类：成组调度算法(*Clustering algorithm*)和列表调度算法(*List scheduling algorithm*)。成组调度算法类似于上文的SwinDeW的调度策略，在此不再赘述。典型的列表调度算法包括HEFT算法和CPOP算法[12]等。列表调度算法包含三个步骤：(1) 为DAG中的结点和边赋rank值；(2) 将rank值降序排列成一个作业列表；(3)按照高者优先规则选择作业并根据最早完成时间分配资源。HEFT算法和CPOP算法的区别在于前者的排序根据upward rank排序而后者根据总rank (upward rank + downward rank) 进行排序。这种集中式的作业调度算法不适合P2P环境下的工作流调度，主要因为工作流中的独立作业会被分配到异步节点上进行自治的调度，从而无法精确计算该作业的rank以及判断该作业是否是关键作业。而且，不同作业之间的资源竞争对关键作业的调度策略必然会造成一定影响，从而导致作业调度的不稳定性。此外，HEFT/CPOP算法属于静态算法，即所有工作流和作业的调度策略必须在执行之前事先决定，相比之下，本文的调度算法基于P2P环境通过异构工作流中异步的执行点位置进行动态调度。

3 P2P 网格系统中工作流关键作业的动态预测

3.1 P2P网格系统的工作流模型

首先定义一个工作流为一个 DAG 图，并且将图中的结点称为作业，结点间的有向边称为依赖。因此，任何两个作业 A 和 B 之间的关系可以分为三种：A 依赖[2]B (表示为 B→A)，B 依赖 A (表示为 A→B)，A 与 B 不相关 (表示为 A ↮ B)。A 依赖 B 表示作业 A 的执行具有时序上的限制，即 A 需要在 B 执行结束后才能开始执行，此时 B 称为 A 的前驱，A 称为 B 的后继。任何一个工作流(DAG 图)可以有一个或多个起始结点和

1 本文的“结点”指 DAG 图中的作业，区别于 P2P 系统中的计算“节点”。

2 这里的“依赖”指的是直接依赖，并非间接依赖。

一个或多个终止结点。如果给每个起始结点加一个依赖结点 S，则 S 可以没看作是该工作流的唯一的起始结点。同理，也可以构造一个唯一的终止结点。因此，在本文提到的工作流模型中只有唯一的起始结点和终止结点。每个结点和每个依赖都有一个权值对应。结点的权值表示作业的计算量（可用浮点计算量来衡量）；依赖的权值则表示从一个作业所在的执行节点到其后续作业所在节点的网络带宽，以此来估计数据传输消耗的时间。从第一个提交的作业到最后一个完成的作业之间的可行的链接称为该工作流的一条路径，并且称并行执行的互不依赖的作业链接为分支(或并行分支)，即从一个作业汇聚点执行到下一个作业汇聚点的一条作业链接。在 DAG 图中，正在执行的作业称为执行节点，正在等待被调度的结点称为调度结点或调度点。如图 1 所示，S 和 E 结点分别表示初始作业和终止作业，则 S→A→D→E 是一条路径，而 S→A→D 是一个分支。从一个汇聚作业点(如图 1 中的 S)执行到下一个汇聚点(如图 1 中的 D)的所有分支构成一个分支组，称该分支组执行时间最长的那个分支为关键分支，其对应的执行时间称为该分支组的 *makespan*。

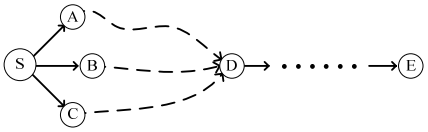

图 1 工作流中的路径和分支

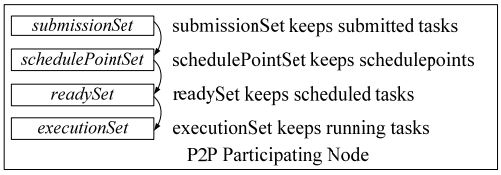

图 2 资源节点的四个作业集合

P2P 网格系统的工作流模型中，每个计算节点本地保存四个作业集合(如图 2 所示)：*submissionSet*、*schedulePointSet*、*readySet* 和 *executionSet*。当工作流提交到某个计算节点时，其作业被暂存到 *submissionSet*，其调度点存入 *schedulePointSet*；使用 3.2 节设计的算法，*schedulePointSet* 中的作业将被周期的检查和调度到本地或邻近的节点的 *readySet* 中准备执行，该步骤也称为作业调度；每当一个作业执行完毕时，对应计算节点的 *readySet* 将收到一个执行信号并根据作业的关键性优先级别(即紧迫程度)选择一个作业放入 *executionSet* 进行执行。任何节点存在两种状态：*空闲状态*和*非空闲状态*。节点的空闲状态表示在该节点下一次作业调度之前所有的 *readySet* 和 *executionSet* 里的作业将提前执行完毕使得该 CPU 存在一段时间的空闲状态。非空闲状态则是指该节点下一次周期作业调度之前仍然存在未执行完毕的作业。

本文旨在降低 P2P 网格系统中的工作流平均完成时间和提高工作流平均执行效率。不失一般性，设该系统有 n 个节点，分别表示为 p_i (i=1,2,…,n)。每个节点的计算能力用 c_i 表示，其上提交 n_i 个工作流 f_{ij} (j=1,2,…,n_i)。定义工作流 f_{ij} 的作业集合为 $T(f_{ij})$，作业 $t_k^{(ij)}$表示为 $T(f_{ij})$的第 k (=1,2,…,$|T(f_{ij})|$)个作业，并且 $t_k^{*(ij)}$表示工作流 f_{ij} 的关键作业。作业 $t_k^{(ij)}$的期望执行时间 $et(t_k^{(ij)})$定义为利用平均带宽传输数据和平均计算能力执行作业，从开始执行到执行完毕的运行时间；作业 $t_k^{(ij)}$的实际完成时间 $ct(t_k^{(ij)})$记为实际系统中从作业提交到执行完毕的时间；工作流 f_{ij} 的实际完成时间 $ct(f_{ij})$记为从初始作业提交到终止作业执行完毕的时间。不是一般性，设每个工作流存在唯一的一条关键路径，则工作流 f_{ij} 的效率 $e(f_{ij})$定义为 $\frac{1}{ct(f_{ij})}\sum_{t_k^{*(ij)}\in T(f_{ij})} et(t_k^{*(ij)})$。因此，P2P 网格系统的工作流平均完成时间 *ACT* 和工作流平均执行效率 *AE* 分别表示为等式(1)和等式(2)。

$$ACT=\frac{1}{\sum_{i=1}^{n} n_i}\sum_{i=1}^{n}\sum_{j=1}^{n_i} ct(f_{ij}) \tag{1}$$

$$AE=\frac{1}{\sum_{i=1}^{n} n_i}\sum_{i=1}^{n}\sum_{j=1}^{n_i} e(f_{ij}) \tag{2}$$

3.2 基于工作流关键作业动态预测的调度算法——*CTE-P2PHEFT*

我们融合 HEFT 算法[12]和关键作业预测策略(3.3 节)并结合 P2P 资源发现机制，设计了基于关键作业预测的 P2P-HEFT 算法(*CTE-P2PHEFT*)。算法 1 给出了该算法的伪代码，该算法自治的运行在每个 P2P 计算节

点上。第 2 行利用 *gossip protocol* 聚合空闲节点状态信息并存入本地的空闲节点集合。每个节点时刻接收外界提交或迁入的作业。第 4~11 行，用来对本地作业进行动态预测判断其是否是关键作业并计算相应分支和分支组的 *makespan*。然后，对本地的关键作业进行调度 (第 12～22 行); 调度完所有的关键作业后如果仍然存在空闲节点，则调度非关键作业(第 23～31 行)。第 15 行根据最快完成时间选择作业 $task_j$，从而可以最小化由于逻辑依赖引起的后继作业等待时间。第 23 行根据(*gms–bms*)小者优先原则选择相对紧迫的作业进行执行。

算法 1: P2P 网格系统中基于关键作业的动态调度算法 (设该算法运行在节点 p_s 上)

输入: p_s 上的本地 *schedulePointSet*(p_s); 通过 *gossip protocol* 在 p_s 上聚合的空闲节点集合 *RS*(p_s); 提交或调度到 p_s 的工作流的 DAG 集合{DAG(f_{ij})| $t_k^{(ij)}$(p_s)提交或调度在 p_s; DAG(f_{ij})表示工作流 f_{ij} 的依赖关系图}。

输出: 从集合 *T*(p_s)中选择作业调度到 *RS*(p_s)的节点执行，使得 ACT 和 AE 尽可能最大化。

```
1.  while(true) do
2.    Gossip protocol: 随机选取[log(n)]个节点作为邻居节点并交换/转发状态信息;
3.    T*(p_s)=∅; T¯*( p_s)=∅; /*令 T*(f_si)和 T¯*(p_s)分别表示 p_s 还未调度的关键作业集合和非关键作业集合, i=1,2,…,n_s*/
4.    for (each non-scheduled task_j in schedulePointSet(p_s)) do
5.      根据 3.3 节的工作流关键作业预测算法(算法 2)判断 task_j 是否是关键作业, 返回值为{task_j ,C, bms, gms};
        /*其中 C 表示 task_j 是否是关键作业, bms 表示 task_j 所在分支的 makespan, gms 表示 task_j 所在分支组的 makespan*/
6.      if(task_j 是关键作业) then
7.        T*(p_s) ← {task_j ,TRUE, bms, gms}; /*此时 bms=gms*/
8.      else
9.        T¯*(p_s) ← {task_j ,FALSE, bms, gms};
10.     end if
11.   end for
12.   if (RS(p_s)≠∅&& T*(p_s) ≠∅) then
13.     根据 RS(p_s)为 T*(p_s)中的每个作业 task_j 计算最快完成时间：即 min(FT(task_j,*)); /* FT(task_j,*)由下文的公式(4)计算*/
14.   end if
15.   for(T*(p_s)中的作业 task_j) do /*根据完成时间从小至大的顺序选择作业和对应的计算节点 p_k*/
16.     if (RS(p_s)≠∅) then
17.       将 task_j 所在分支组的信息通知节点 p_k 并更新节点 p_k 的状态信息; /*让 p_k 之后也能动态估计 task_j 的关键性程度*/
18.       将作业 task_j 移至 p_k 的 readySet 准备执行;
19.     else
20.       break;
21.     end if
22.   end for
23.   for(T¯*(p_s)中的作业 task_j) do /*根据(gms–bms)从小至大的次序选择, 越紧迫的作业执行优先级越高*/
24.     if (RS(p_s)≠∅) then
25.       从 RS(p_s)选择使得 task_j 最早完成的节点 p_k 作为执行节点并更新节点 p_k 状态信息;
26.       将 task_j 所在分支组的信息通知节点 p_k; /*让 p_k 之后也能动态估计 task_j 的关键性程度*/
27.       将作业 task_j 移至 p_k 的 readySet 准备执行;
28.     else
29.       break;
30.     end if
31.   end for
32.   sleep(a short time period);
33. end while
```

3.3 P2P网格系统的工作流关键作业预测方法

传统网格系统中的工作流关键路径及关键作业的判断是基于完全信息的假设[13]，即集中控制的作业调度策略。当服务器收集到所有节点的状态信息和需要调度的工作流请求以后进行统一调度。$ST(t_k^{(ij)},p_h)$和$FT(t_k^{(ij)},p_h)$分别定义为作业 $t_k^{(ij)}$在节点 p_h 上的开始时间和结束时间。如果 $t_{k1}^{(ij)} \rightarrow t_{k2}^{(ij)}$，则 $t_{k1}^{(ij)} \in Pre(t_{k2}^{(ij)})$并且 $t_{k2}^{(ij)} \in Post(t_{k1}^{(ij)})$，其中 $Pre(t_{k2}^{(ij)})$表示作业 $t_{k2}^{(ij)}$的所有前驱作业的集合，$Post(t_{k1}^{(ij)})$表示作业 $t_{k1}^{(ij)}$的所有后继作

业的集合。因此，对于任何作业 $t_k^{(ij)}$，其开始时间和结束时间可以根据等式(3)和等式(4)进行估计，其中 $R(t_k^{(ij)},p_h)$ 表示节点 p_h 相对于作业 $t_k^{(ij)}$处于空闲状态所需要的等待时间，$cost(t_{k'}^{(ij)}, t_k^{(ij)})$表示作业 $t_{k'}^{(ij)}$传送数据到 $t_k^{(ij)}$所消耗的时间。$cost(t_{k'}^{(ij)}, t_k^{(ij)})$可以近似估计为 $datasize(t_{k'}^{(ij)}, t_k^{(ij)})/bandwidth(p_{h'}, p_h)$，其中 $bandwidth(p_{h'}, p_h)$表示两节点($p_{h'}$和 p_h)间的网络带宽，$datasize(t_{k'}^{(ij)}, t_k^{(ij)})$表示作业 $t_k^{(ij)}$对作业 $t_{k'}^{(ij)}$所依赖的数据大小。然而，由于 P2P 环境中作业调度的相互影响，$R(t_k^{(ij)},p_h)$是很难准确获取的，为简化问题，在算法设计时，令 $R(t_k^{(ij)}, p_h)=0$。

$$ST(t_k^{(ij)}, p_h) = \max(R(t_k^{(ij)}, p_h), \max_{t_{k'}^{(ij)} \in Pre(t_k^{(ij)})} (FT(t_{k'}^{(ij)}, p_{h'}) + cost(t_{k'}^{(ij)}, t_k^{(ij)}))) \quad (3)$$

$$FT(t_k^{(ij)}, p_h) = ST(t_k^{(ij)}, p_h) + et(t_k^{(ij)}) \quad (4)$$

利用等式(3)和等式(4)的递归性，可以根据工作流的当前执行作业的位置和 DAG 估计出经过各条路径到达终止作业 $t_{end}^{(ij)}$的完成时间，即 $FT(t_{end}^{(ij)}, node)$。注意，相同的作业在不同的资源节点上执行会获得不同的执行时间，因此在判断该作业是否是关键作业的时候，需要考虑资源的异构性。也就是说，一方面，当判断某本地作业的关键性程度(即是否是关键作业和它对 *makespan* 的影响)时，需要通过事先预测其调度节点来估计执行时间。另一方面，一个工作流内无关联部分(即并行)的作业在其调度到的节点上是异步执行的。由于每个节点只收集其局部范围的状态信息，因此工作流中某分支作业的执行状态对于另一个并行分支的作业来说可能是未知的，从而为判断本地作业是否是关键作业带来困难。也就是说，还需要通过预测其他并行分支的作业所调度获得的资源的计算能力来估计该并行分支的执行时间。最终，通过比较本地分支和并行分支的完成时间来估计本地作业的关键性。因此，不管是本地分支的作业还是其他分支的作业，对于未知的作业调度资源，我们根据 *aggregation gossip protocol* [14]所聚合的统计平均值作为对该未知资源的估计。图 3 给出了一个具体的例子：p_i 根据通过 *gossip protocol* 收集的信息对 DAG 中结点和边的权值进行估计。算法 2 给出了该关键作业预测方法的为代码。该算法用到的 makespan(·)是个递归函数，当用到资源节点或带宽信息时，首先在 *TaskInfoSet* 查看有无收到确定的消息，如果没有再使用平均值进行估计。

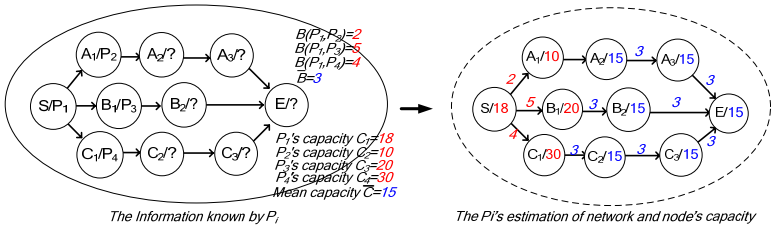

图 3: 每个节点对网络状态和其他节点的性能估计

算法 2:工作流关键作业预测算法(设该算法运行在节点 p_s 上)

输入: 本地作业的静态信息 *TaskInfoSet*，包括所在工作流的 DAG、作业的执行时间等；*schedulePointSet*

输出: 为 *schedulePointSet* 中的每个作业计算{$task_j$,*C*, *bms*, *gms*} (其中，C 表示 $task_j$ 是否是关键作业，*bms* 表示作业 $task_j$ 所在分支路径的 *makespan*，*gms* 表示作业 $task_j$ 所在分支组路径的 *makespan*)

1. **for**(each *task* in p_s's schedulePointSet) **do**
2. 根据公式(3)和(4)计算 bms: *makespan*(*task*);
3. 根据公式(3)和(4)计算 gms: 即遍历 *task* 所在分支组的每个调度点 $task^*$，并计算最大的 *makespan*($task^*$);
4. **if** (*bms*==*gms*) **then**
5. 构造{$task_j$, *TRUE*, *bms*, *gms*}
6. **else**
7. 构造{$task_j$, *FALSE*, *bms*, *gms*}
8. **end if**
9. **end for**

3.4 算法时间复杂度分析

算法 1 的时间复杂度=*MAX*(第 5 行算法 2 的复杂度，第 13 行遍历计算关键作业最快完成时间的复杂度)。算法 2 的时间复杂度=$\boldsymbol{O}(|schedulePointSet(p_s)|\cdot L(p_s))$，其中 $L(p_s)$表示 p_s 上所有工作流的 makespan 的最大

值。算法 1 第 13 行遍历计算关键调度点作业最快完成时间的时间复杂度=$\boldsymbol{O}(|T^*(p_s)|\cdot|RS(p_s)|)+\boldsymbol{O}(|T^*(p_s)|^3)$。该复杂度可以推导如下：第 1 轮计算每个关键调度点作业的最快完成时间需要遍历每个关键调度点和 $RS(p_s)$中的每个空闲节点以选出并调度完成时间最快的作业(记该作业为 $task^1$)，因此需要$|T^*(p_s)|\cdot|RS(p_s)|$；然而，在第 2 轮循环不需要重新遍历所有作业和节点，只需要重新计算剩余的$|T^*(p_s)|-1$ 个关键作业在 $task^1$ 所调度到的节点上执行的完成时间，从而消耗的复杂度为 $1\cdot(|T^*(p_s)|-1)$；类似的第三轮循环的时间复杂度为 $2\cdot(|T^*(p_s)|-2)$；以此类推。从而，第 1 轮至第$|T^*(p_s)|$轮的计算时间=$|T^*(p_s)|\cdot|RS(p_s)|+1\cdot(|T^*(p_s)|-1)+2\cdot(|T^*(p_s)|-2)+\cdots\cdots+(|T^*(p_s)|-1)\cdot 1 = |T^*(p_s)|\cdot|RS(p_s)|+|T^*(p_s)|\sum_{i=1}^{|T^*(p_s)|-1} i-\sum_{i=1}^{|T^*(p_s)|-1} i^2 = |T^*(p_s)|\cdot(|RS(p_s)|+(|T^*(p_s)|^2-1)/6)$，即时间复杂度为 $\boldsymbol{O}(|T^*(p_s)|\cdot|RS(p_s)|)+\boldsymbol{O}(|T^*(p_s)|^3)$。

因此，算法 1 的总体时间复杂度= $MAX(|schedulePointSet(p_s)|\cdot L(p_s), \boldsymbol{O}(|T^*(p_s)|\cdot|RS(p_s)|)+\boldsymbol{O}(|T^*(p_s)|^3))$。

4 性能评估

4.1 P2P网格计算系统仿真

本文中的测试环境使用 Brite 拓扑构造工具[15]根据 *Waxman* 模型[16]构造了一个仿真的物理 Internet，然后，在该网络上利用 Peersim 工具[17]生成异步事件来仿真节点之间的消息传递和工作流作业的执行。

我们假设每个节点在系统初始时刻接收到一个工作流，且每个工作流的 DAG 依赖关系是如图 4 所示的几种模式之一。当工作流的作业个数不超过 4 时，分别根据(a) (b) (c)的三种结构构造工作流；当作业数超过 4 时，则随机选择图(d)中的两种模式之一进行构造。测试时间为 36 小时，每个计算节点的动态调度周期为 15 分钟，*gossip protocol* 中的 TTL=4，每个节点的直接邻居为 $log(n)$。另外，每个节点的执行环境设置为独占性非抢占式资源，即任何时刻的任何节点资源只能执行一个作业。其他的环境配置参数可参考表 1。

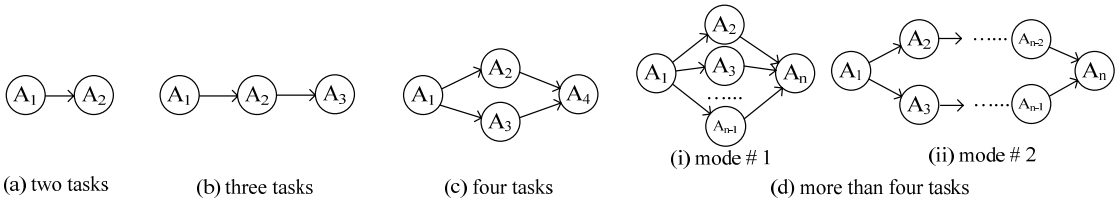

图 4: P2P 网格计算系统仿真中工作流的模式

表 1: P2P 网格计算系统仿真环境配置参数

参数	数值	参数	数值	参数	数值
节点数	1000	每工作流作业数	2~10	作业计算量	100~10000 Gflopoints
作业载荷数据	10~1000 M	网络带宽	0.1~10Mb/s	节点计算能力	1 ,2,4,8,16 Gflops

根据 H. Topcuoglu 等人的实验[13]，*HEFT* 算法在工作流的平均 makespan、平均加速比和平均执行效率等都表现出卓越的性能，并远远优于其他的静态调度算法，包括遗传(*GA*)算法、动态分级调度(*DLS*)算法、*CPOP* 算法、映射启发(*MH*)算法、时间等级最小化(*LMT*)算法等。因此，*HEFT* 算法可以被视为全局集中式工作流静态调度的近似最优算法。

我们将本文设计的 P2P 工作流动态调度算法和 HEFT 算法[12]的三种不同版本进行比较。第一种，Global-Heuristic HEFT 表示全局信息的静态调度，即上文所述的近似最优算法；其他两种(RLP2PHEFT 和 PFP2PHEFT)均为 HEFT 算法的 P2P 版本：它们分布式的运行于每个 P2P 节点上并在系统启动之前将提交到本地节点的所有工作流利用局部收集的资源信息执行局部范围内的 HEFT 调度算法。RLP2PHEFT 是根据独立节点的实际状况——即节点在调度时完全独立于其他节点的调度状态；PFP2PHEFT 表示完美的理想状况——即节点在调度时可以立刻获得其他节点执行 HEFT 调度算法对节点状态的更新。

与 RLP2PHEFT 和 PFP2PHEFT 相比，本文提出的 CTE-P2PHEFT 算法的特点是在实际情况中动态分析和预测工作流调度点的关键性，即调度点是否是关键作业以及该作业的紧迫程度。此外，RLP2PHEFT 和

PFP2PHEFT 都存在两个嵌套步骤：(1)选择作业(2)为该作业选择最优节点；而 CTE-P2PHEFT 算法对应的策略(算法 1 的第 13 行)则是选择作业和空闲节点的最优组合，从而最大限度的降低其他作业的等待时间。

4.2 测试结果

图 5 展示了 1000 个计算节点的仿真环境下四种不同调度算法的吞吐量。其中，由于 Global-Heuristic HEFT 是全局范围的近似最优算法，它显示出最大的吞吐量；HEFT 算法的两种 P2P 版本的吞吐量非常低，执行完所有的工作流大约需要 18 个小时；而 CTE-P2PHEFT 算法和全局最优算法大概保持近似最优，而且明显的优于其他两种 P2P 工作流调度算法。

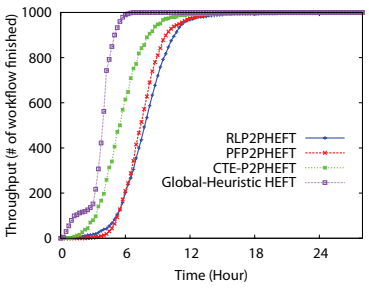

图 5: P2P 工作流吞吐量

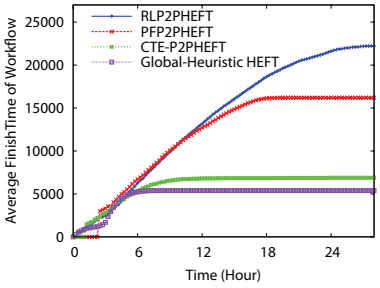

图 6: P2P 工作流平均完成时间

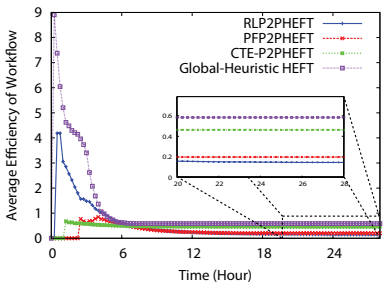

图 7: P2P 工作流平均执行效率

图 6 和图 7 分别展示了该测试中工作流的平均完成时间(即公式(1))和平均执行效率(即公式(2))。其中 Global-Heuristic HEFT 依旧显示出最优的性能：最低的工作流平均完成时间和最高的工作流平均执行效率。RL/PFP2PHEFT 则显示出快速拟线性增长的工作流平均完成时间，相比之下，本文的 CTE-P2PHEFT 算法的平均工作流完成时间则相对于 Global-Heuristic HEFT 保持近似最优(approximated optimal)。注意到，对于 HEFT 算法，图 7 中的初始时间段有一个超标的工作流平均执行效率(即>1)，这是由于计算平均执行效率时，作业 $t_k^{(ij)}$的期望执行时间 $et(t_k^{(ij)})$是利用平均带宽传输数据和节点的平均计算能力求出的，因此，在初始阶段，HEFT 算法的贪婪性质会使得作业优先调度到高速节点上从而导致执行效率获得暂时的超标，然而随着高速节点的饱和，执行效率会急剧下降并受制于不均衡的负载状态。

表 2 显示了在不同系统规模(n)下的 CTE-P2PHEFT 算法中平均每个节点收集到的空闲节点的个数。由此可见，在 TTL=4 和节点邻居个数=log(n)的 *gossip protocol* 下， P2P 节点的空闲资源集合大小成对数增长，从而具有理想的可扩展性。

表 2: P2P 网格系统节点的根据 *gossip protocol* 聚合的平均空闲资源大小

系统规模(节点个数)	100	200	400	600	800	1000	1200	1400	1600	1800	2000
平均每节点收集的空闲节点个数	14	19	19	23	23	23	27	27	27	27	27

5 结束语

本文针对 P2P 网格系统中工作流调度策略进行了分析和优化，并提出了一种基于关键作业动态预测的工作流调度算法——*CTE-P2PHEFT*。该算法结合使用 *gossip protocol* 的 P2P 资源发现机制和异构最早完成时间(HEFT)策略来动态预测工作流当前调度作业的关键程度，从而降低因控制依赖/数据依赖造成的资源空闲率。通过仿真测试，我们证明了 *CTE-P2PHEFT* 优于其他的 P2P 工作流调度算法，其执行效率达到了近似最优。

References:

[1] Butt RA, Zhang RM, Hu CY. A self-organizng flock of condors. Journal of the Journal of Parallel and Distributed Computing. 2003, 66(1):145-161.

[2] Luther A, Buyya R, Ranjan R, Venugopal S. Alchemi: A .NET-based Grid Computing Framework and its Integration into Global Grids. Technical Report, GRIDS-TR-2003-8, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, 2003.

[3] Naik KV, Sivasubramanian S, Bantz D, Krishnan S. Harmony: a desktop grid for delivering enterprise computations. In: 4th International Workshop on Grid Computing (Grid 2003), 2003.

[4] Chien A, Calder B, Elbert S, Bhatia K. Entropia: architecture and performance of an enterprise desktop grid system. Journal of Parallel and Distributed Computing, 2003, 63(5):597-610.

[5] Cao J, Liu BF, Xu ZC. P2pgrid: integrating p2p networks into the grid environment: Research articles. Chichester, UK, John Wiley and Sons Ltd, 2007. 1023–1046.

[6] Andrade N, Cirne W, Brasileiro F, Roisenberg P. Ourgrid: An approach to easily assemble grids with equitable resource sharing. In: Job Scheduling Strategies for Parallel Processing, 2003. 61–86.

[7] Foster I, Kesselman C, The GRID 2: Blueprint for a New Computing Infrastructure. Second Edition, Morgan Kaufmann, 2004.

[8] Allavena A, Demers A, Hopcroft EJ. Correctness of a gossip based membership protocol. In*:* Proceedings of the 24th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing (PODC 2005), New York, USA, 2005. 292–301.

[9] Boyd S, Ghosh A, Prabhakar B, Shah D. Gossip algorithms: design, analysis and applications. In: 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM2005), 2005. 1653–1664.

[10] Yan J, Yang Y, Raikundali KG, SwinDeW-a p2p-based decentralized workflow management system, IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 2006. 36(6): 922-935.

[11] Shen J, Yan J, Yang Y, SwinDeW-S: Extending P2P Workflow Systems for Adaptive Composite Web Services, In: proc. of Australian Software Engineering Conference (ASWEC2006), 2006. 61-69.

[12] Topcuoglu H, Hariri S, Wu M. Performance Effective and Low-complexity Task Scheduling for Heterogeneous Computing. IEEE Transactions on Parallel and Distributed Systems, 2002. 13(3): 260-274.

[13] Cao HJ, Jin H, Wu XX, Wu S, Shi XH. DAGMap: Efficient scheduling for DAG grid workflow job. In: 9th IEEE/ACM International Conference on Grid Computing (Grid2008), 2008. 17-24.

[14] Jelasity M, Montresor A, Babaoglu O, Gossip-Based Aggregation in Large Dynamic Networks. ACM Transaction on Computer Systems, 2005. 23(3): 219-252.

[15] Peersim simulator: http://peersim.sourceforge.net.

[16] Naldi M. Connectivity of Waxman topology models. Journal of Computer Communications, 2005. 29(1): 24-31.

[17] Brite topology generator: http://cs-pub.bu.edu/brite/.