DOI:10.13196/j.cims.2016.02.018

基于任务执行截止期限的有向无环图实时调度方法

许荣斌[1,2],刘 鑫[1],杨壮壮[1],郭 星[1,2],谢 莹[3+],吴建国[1]
(1.安徽大学 计算机科学与技术学院,安徽 合肥 230061;
2.安徽大学 信息保障技术协同创新中心,安徽 合肥 230061;
3.安徽大学 计算机教学部,安徽 合肥 230061)

摘 要:为了将任务映射到多个资源中运行,以提高任务的执行效率进而有效管理和组织一个业务流程。在传统有向无环图模型的基础上,首先给出一种新的模型,将调度任务分解成若干串行和可并行调度部分;然后在新的有向无环图模型基础上提取任务调度的一条有效主路径,按照有效主路径将任务并行展开;随后基于任务的执行完成时间和执行截止期限提出一种拉伸调度策略,使任务在满足执行截止期限的同时,尽可能减少对处理器资源的占用,使其他任务能有效地进行调度。通过仿真实例证明所提方法具有很好的适用性。

关键词:有向无环图;实时调度;执行截止期限;有效主路径;并行调度;拉伸调度
中图分类号:TP311 **文献标识码**:A

Real-time DAG scheduling method based on deadline of tasks

XU Rongbin[1,2], *LIU Xin*[1], *YANG Zhuangzhuang*[1], *GUO Xing*[1,2], *XIE Ying*[3+], *WU Jianguo*[1]
(1. School of Computer Science and Technology, Anhui University, Hefei 230601, China;
2. Co-Innovation Center for Information Supply & Assurance Technology, Anhui University, Hefei 230601, China;
3. Computer Studies Department, Anhui University, Hefei 230601, China)

Abstract: To map tasks into multiple resources for improving the execution efficiency, and to manage a business process effectively, a novel Directed Acyclic Graph (DAG) model was given based on the traditional DAG model, which divided the tasks into serial and parallel scheduling parts. A valid main path for task scheduling was extracted to make the task be expanded into parallel mode based on the novel DAG model. Based on completion time and deadline of the task, a stretch scheduling strategy was put forward to meet the deadline of tasks and reduce the occupation of resources, which made other tasks could be scheduled effectively. A simulation example was provided to prove the excellent feasibility of the proposed method.

Keywords: directed acyclic graph; real-time scheduling; execution deadline; valid main path; parallel scheduling; stretch scheduling

1 问题的提出

随着经济全球化进程的逐渐加速,当今电子商务快速发展,通常需要在有限时间内处理大量的业务流程,有效管理和组织一个业务流程变得更加重要[1]。常见的一种业务流程是实例密集型商业流程,指企业和顾客之间、供应商和其他相关合作伙伴之间的交互,以及参与商务活动中的各方为完成其交易而共同协作的过程。例如:利用移动电子商务、电子资金转账和电子数据交换等

收稿日期:2015-11-21。Received 21 Nov. 2015.
基金项目:国家973计划资助项目(2015CB351705);国家自然科学基金资助项目(61472001,61300169,61300042);教育部人文社会科学研究青年基金资助项目(14YJCZH169);安徽省自然科学基金资助项目(1608085MF130)。**Foundation items**: Project supported by the National Basic Research Program, China (No. 2015CB351705), the National Natural Science Foundation, China (No. 61472001, 61300169, 61300042), the Youth Fund for Humanities and Social Science of MOE, China (No. 14YJCZH169), and the Natural Science Foundation of Anhui Province, China (No. 1608085MF130).

技术，在电子系统上进行产品或服务的买卖[2]；商业银行每天处理的大量支票和转账业务[3]；企业需要考虑工作效率、工作任务之间的时序依赖性等因素，将众多工作任务分配给合适员工的过程[4]。这些商业应用通常都很复杂，并且关注于内部流程，目的是节约成本、提高效率和生产率[5]。这些商业应用通常都包含若干个有序或部分有序的过程，通过考虑这些复杂的实例密集型商务流程，确保商业任务按时完成，即在有时间限制的范围内完成所有任务的调度，对所有任务的运行状况进行有效地监控，以提高任务分配的效率和可靠性，是商业应用的一个很重要的标准[6]。

证券交易包含大量交易信息，是一个典型的商业工作流程。每一个交易信息都是一个相对较小的工作流实例，每个实例都包含一些交易的多个步骤。证券交易系统是非常庞大而复杂的应用，其交易过程遵循特定的规则和时间要求，主要过程如图1所示，包括以下步骤：投资者通过身份认证、确认账户资金、通过交易系统提出交易委托、满足委托证券交易条件时确认交易生效、将交易结果记录到证券公司的数据库中、产生交易数据、计算证券变化情况、对证券进行交割、通过证券交易所抵消投资者之间证券买卖的数量、计算产生的交易费用、计算资金变化情况、对证券买卖双方的账户设置收支平衡、注册股票变化情况、计算资金的变更、将数据整理存档。

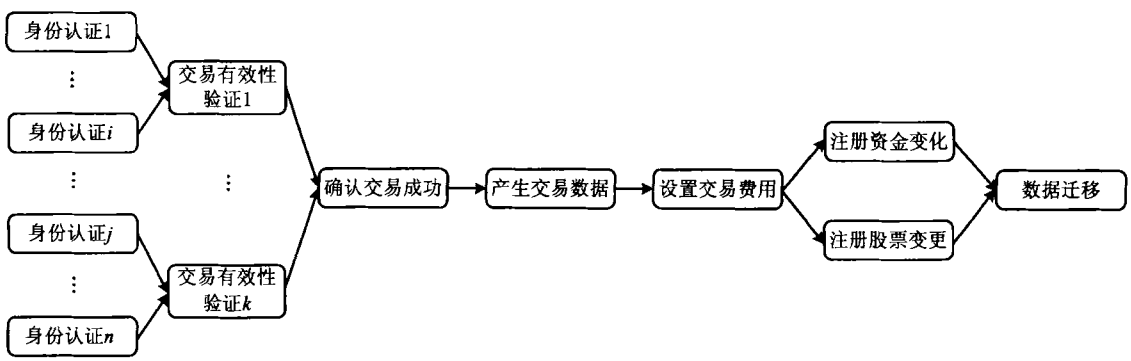

图1 证券交易流程简图

证券交易市场中存在大量交易活动，每条交易活动又包括如图1所示的有时序限制的若干子活动，这些子活动中部分按顺序执行、部分可并行执行。当前，最常见的商业流程建模方式是使用有向无环图(Directed Acyclic Graph, DAG)，研究诸多任务以及每个任务多个子活动间的相互关系[7]。证券交易过程中的实时性要求很高，因此可利用DAG模型对其任务进行实时调度，这对于大量顺序和并行共存的任务来说是一个很大的挑战。因为对于具有时序限制的任务调度来说，不仅需要关注调度结果的正确性，还需要考虑任务的执行完成时间和执行截止期限、任务的并发执行等多种因素。本文将深入研究有多个串行和并行子活动的DAG任务实时调度，考虑任务各自的时序限制，给出一种有效的DAG任务调度模型，从而提取DAG任务的有效主路径，将任务按有效主路径并行展开，根据任务的执行截止期限与其子活动的时序限制，提出DAG任务的拉伸调度策略，使多个任务调度时各自都可以满足时序限制的要求。

2 相关工作

调度过程中，追求任务的最优映射已被证明是一个NP完全问题[8-9]。文献[8]提出一种模块分配问题，指出如果通信图是一个部分 k 叉树或有 k 个参数的完全树，则可在多项式时间范围内找到一个最小成本的分配策略。文献[9]提出了最短树算法，用以任意连接分布式系统中任意数量的处理器，最小化执行和通讯费用的总和。该算法使用动态规划的方法，可以在 $O(m^2n)$ 时间范围内将 n 个任务映射至 m 个处理器中执行。以上算法均关注降低调度任务的执行和通信费用。

对于商业流程来说，任务与任务间可能存在优先级或执行顺序的关系，通常可以使用DAG模拟商业流程的任务调度[7]。因此，DAG任务的调度算法在业界越来越被重视，已有许多学者对该问题进行了深入研究[10-12]。文献[10]提出费用与时间优化的网格DAG调度算法，该算法能够有效地确定任务在资源上运行的时间，而且在时间允许的情况下能够尽可能将任务映射到价格便宜的资源上运行

以降低总体运行费用,然而该算法所使用的资源中存在较多时间碎片;文献[11]设计了 DAG 工作流中带有通信开销的工作流费用优化模型,使其适用于真实的云环境;文献[12]在分层算法的基础上,考虑截止期限约束,用 DAG 图描述工作流费用优化问题。

关于 DAG 任务在多处理器系统的实时调度问题,文献[13]指出具有单调截止期限的调度适合于拉伸调度算法,而最早截止期限优先的条件更适合于并行调度算法;文献[14]描述了多核处理器中,为了解决 folk-join 任务模型中建立并行程序范式,例如 OpenMP 出现的隐式截止期限问题,提出对一个任务进行拉伸变换,从而有效避免任务执行时间超出截止时限的问题,但该算法出现了缓存污染和同步延迟等问题;文献[15]提出一种改进的单调截止期限调度(Improved Deadline Monotonic Scheduling, IDMS)算法,通过迭代计算,使用精确的调度试验来确定可调度过程,保证在截止期限内完成所有任务;文献[16]针对比较稀疏的实时任务调度,给出有限制抢占式的最早截止期限优先(Limited Preemptive Earliest Deadline First, LPEDF)调度算法,可以很好地应用于多处理器平台。

基于时间限制和费用最小的 DAG 调度算法中,文献[10]所提算法的关键技术是采用 DAG 中提取的有效路径能够确定任务在资源上运行的时间,但该技术选出的有效路径较多,给资源在任务上的分配带来许多不便。针对已有算法存在的缺陷,本文在多条有效路径的基础上,基于每个子活动出入度的大小提取有效主路径,作为任务合理调度的依据,然后选取与主路径中活动并行执行的其他活动进行并行处理。本文基于有效主路径的并行调度,结合任务的执行截止期限,进一步提出更为合理和规范的拉伸调度算法,该算法既可保证有效处理所有活动,又能使 DAG 任务调度满足时序要求。

3　调度任务模型

下面给出业务流程管理中的相关概念,针对业务流程建模常用的 DAG 任务提出一种新的 DAG 可调度模型,将 DAG 任务分解成若干串行和并行部分。

3.1　相关概念

(1)工作流　工作流的概念起源于生产组织,指"业务过程的部分或整体在计算机应用环境下的自动化",其主要目标是通过合理地调用和分配有关信息及人力资源来协调业务过程中的各个活动,以促使业务目标的高效实现。从用户的角度看,工作流提供了对复杂应用的抽象定义、灵活配置和自动化运行,进而提高了服务质量;从服务提供者的角度看,工作流提供了任务的自动调度、资源的优化和管理,进而压缩了运行成本[17]。目前,工作流已经成为实现业务过程自动化的核心技术[18]。

(2)业务流程　业务流程是跨越事件和地点的有序工作活动,有起始点和终止点,有明确的输入和输出,是一系列结构化的可测量活动的集合,也是一组能够一起为客户创造价值的相互关联的活动进程。流程包括输入的资源、活动、活动的相互作用、输出的结果、客户和价值六个要素。

(3)业务流程管理　业务流程管理是一套达成企业各种业务环节整合的全面管理模式,是一种从整体上对组织的各个方面(流程、资源等)进行统筹规划以满足客户需求的方法。

(4)过程定义　过程定义是业务过程的形式化描述,用来支持系统建模和运行过程的自动化。过程可分解为一系列子过程和活动,包括描述过程起始、终止的活动关系网络,以及一些关于个体行为的信息,即构成过程的活动以及各活动的关系、组织成员的角色、应用中的数据结构等。

(5)活动　活动是业务过程的一个执行阶段,由执行者完成,执行者可以是人、软件系统或二者的集合。活动可以按照执行顺序分为若干子活动,并可按照各子活动之间的关系将所有子活动分为串行和并行部分。子活动是过程执行中可被工作调度的最小工作单元。

(6)过程实例　过程实例是一个工作流过程的具体执行。在过程实例的执行中,工作流引擎负责解释对应的过程定义,动态生成活动实例,并根据过程定义中的规则控制协调这些活动实例之间的执行顺序,同时完成活动之间的数据传递。

3.2　DAG 任务分解模型

设有一个任务集合 τ,有 n 个可并发执行的 DAG 任务,所有任务记为$\{\tau_1,\tau_2,\cdots,\tau_n\}$,有 m 个可使用的并行处理器。通常每个 DAG 任务都含有若干个子活动,记为$\{\tau_{i,j}\mid 1\leqslant i\leqslant n,1\leqslant j\leqslant n_i\}$,表示第 i 个任务的第 j 个子活动。为方便叙述,本文将子活动称为活动,并与传统结构中的节点为同一概念,n_i 表示任务中活动的总数。同一任务的多个活动通常

都具有一定时序依赖性，执行存在先后顺序关系。

本文针对 DAG 任务的特性，将 DAG 任务 i 中的活动分成若干串行和可并行执行部分，记为 $\{\tau_i:(((C_i^1/P_i^1),\cdots,P_i^{k-1},C_i^k,P_i^{k+1},\cdots,C_i^{j-1},P_i^j,C_i^{j+1},\cdots,(C_i^{s_i}/P_i^{s_i})),n_i,m_i,x_{i,j},y_{i,j},T_i,D_i,Max_i,Min_i),1\leqslant j\leqslant s_i\}$。其中：$C_i$ 表示 DAG 任务 τ_i 活动中的串行部分；P_i 表示并行部分；通常情况下 DAG 任务的开始和结束部分既可为串行部分也可为并行部分，因此 (C_i^1/P_i^1) 和 $(C_i^{s_i}/P_i^{s_i})$ 为任务 τ_i 的第一部分和最后部分活动是串行或者并行的；s_i 表示任务 τ_i 被分成串行和并行部分的总数；n_i 为任务 i 中的活动总数；m_i 为活动中的最大并行数量（也为可使用的并行处理器数量）；$x_{i,j}$ 为任务 τ_i 中第 j 个可并行部分的活动数量；在实际的 DAG 任务调度过程中，考虑任务执行完成时间和截止期限等多种因素，并不是所有并行部分都会并行处理，因此用 $y_{i,j}$ 表示任务 τ_i 中第 j 个可并行部分实际使用的处理器数量。以上参数有如下关系：$n_i\geqslant m_i\geqslant x_{i,j}\geqslant y_{i,j}$，$1\leqslant j\leqslant s_i$，且 m_i 为 $x_{i,j}$ 的最大值，即 $m_i=\max(x_{i,j})$，$1\leqslant j\leqslant s_i$。

因为 DAG 模型中既可能存在单个开始和结束活动，也可能存在多个开始和结束活动，所以 τ_i 中的开始和结束节点既可为串行的单个节点，也可为并行的多个节点。若 DAG 的所有节点都为串行结构，则 $s_i=1$，此时可以只使用一个处理器执行任务 τ_i；若 DAG 的所有节点都为并行结构，同样 $s_i=1$，此时可以使用 m_i 个处理器执行任务 τ_i。这两种情况都表明 τ_i 中只含有单一的串行或并行结构。通常情况下，DAG 任务都包含若干个串行和并行部分，即 $1\leqslant s_i\leqslant n_i$。设 T_i 为任务 τ_i 的执行完成时间，D_i 为 τ_i 的执行截止期限，可以根据 τ_i 执行时的不同需求得到不同的调度结果。当所有子任务都被拉伸成顺序执行时 τ_i 最大，记为 Max_i；当所有活动都被并行执行时 T_i 最小，记为 Min_i。

4 DAG 任务调度策略

下面首先提出一种提取 DAG 任务有效主路径的方法，根据如图 2 所示的 DAG 任务，给出提取有效主路径的示例与伪码；然后选取与主路径子活动并行的其他子活动进行并行处理，在有效主路径并行调度的基础上采用拉伸调度策略，使任务在满足执行截止期限的同时，尽可能减少对处理器资源的占用，并最后给出拉伸调度的示例与伪码。

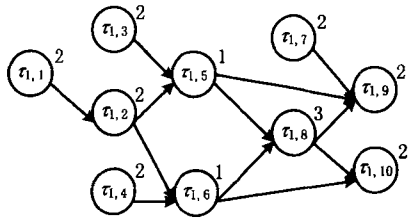

图2 DAG模型实例

4.1 提取有效主路径

首先给出提取有效路径的通用模型，然后针对图 2 所示的 DAG 任务，结合所有活动的出入度提取该任务的有效主路径，最后给出提取有效主路径的伪码。

4.1.1 有效主路径提取策略

如第 3 章所述，DAG 任务由若干个子活动构成，所有子活动又存在一定的串行和并行结构。每个任务有相应的执行截止期限 D_i 以及最大和最小执行完成时间，选取合适的路径作为顺序执行的主体，将其他子活动与主体部分并行执行，既可影响最后的调度结果，又可影响多处理器的并发执行状况。因此通过有效主路径提取策略，从 DAG 任务中既能提取一条代表性的路径，又能有效处理完成所有子任务，从而使 DAG 任务的调度满足时序的要求。

DAG 模型中用有向图连接所有活动，表示活动之间存在一定时序关系，连接两个子活动的边称为有向边。在一条有向边中，前序节点称为父亲节点，后序节点称为孩子节点。DAG 中的每个活动都存在若干父亲节点和孩子节点，对于任一活动，其父亲节点的个数为该活动的入度，其孩子节点的个数为该活动的出度，用出入度 $De_{\tau_{i,j}}$ 描述任一活动 $\tau_{i,j}$ 的父亲节点与孩子节点的个数总和。对于任一 DAG 任务，存在一个或多个活动，只有孩子节点而没有父亲节点，其入度为 0，称为起始活动，记为 $\tau_{i,\text{start}}$；对于任一 DAG 任务，存在一个或多个活动，只有父亲节点而没有孩子节点，其出度为 0，称为终止活动，记为 $\tau_{i,\text{end}}$。通常情况下，DAG 中几条连续边即可称为路径，但其并不一定代表 DAG 模型的主要部分，因此本文定义了一组与有效路径相关的概念。

定义 1 有效路径。指 DAG 任务中一条包含起始活动和终止活动的连续有向边的路径，记为 VPath。

定义 2 路径长度。指 DAG 任务中任意一条有效路径包含的活动总数，记为 PLength。

定义 3 最大路径长度。指所有有效路径中路

径长度最大的值,记为 MPLength。

定义 4　有效主路径。找出所有有效路径后,选取其中一条路径作为调度过程中的主串行路径,记为 $VMPath$,将剩余的节点作为与该路径并行的节点。有效主路径将考虑选取包含最多活动的路径,即选取长度 $PLength$ 最大的路径。因此,若有效路径中有包含关系的路径,则可将被包含的路径从有效路径中除去。

定义 5　可选择有效主路径。在 DAG 模型中,经常会出现多条路径长度 $PLength$ 最大的有效主路径,称这些路径为可选择有效主路径,记为 $OVMPath$,随后通过某种方式选取一条有效主路径。

对于所有可选择有效主路径,首先找出这些路径中包含的节点,记为 $Node_i:\{\}$,当某条路径已不再是可选择主路径时,将该路径中其他路径不包含的节点从 $Node_i$ 中移除;其次在可选择有效主路径的基础上计算所有活动的出入度,找出出入度最高的活动。因为在实际的 DAG 模型中某一活动可能存在较多父亲节点或孩子节点,使其出入度较大,但该情况不能真实反映 DAG 模型的主路径,所以本文方法找出的可选择有效主路径将可以较好地避免该问题。若某个 DAG 任务中只有一个活动的出入度最高,则找出包含该活动的所有可选择有效主路径,将该活动从 $Node_i$ 中移出,然后依次找出剩余 $Node_i$ 中出入度最高的活动,确定出一条有效主路径。若存在多个出入度同样最高的活动,则需考虑多个活动之间的关系,使有效主路径包括尽可能多的出入度高的活动。主要存在如下两种情况:当某个出入度最高的活动是所有可选择有效主路径中共有的节点时,先选择该活动;当某两个或多个出入度最高的活动是并行结构,即多个活动是兄弟节点时,在可选择有效路径中分别计算出各个活动的所有父亲节点与孩子节点的出入度之和 $De_{\tau_{i,j}Pa\&Ch}$,选择该值最大的活动作为有效主路径中的活动,依次取出所有出入度高的节点,直至最后选出一条有效主路径。

4.1.2　有效主路径提取实例

如图 2 所示的 DAG 任务 τ_1 中有 10 个子活动 $\{\tau_{1,1},\tau_{1,2},\cdots,\tau_{1,10}\}$,执行时间为{2,2,2,2,1,1,2,3,2,2},执行完成时间期限 $D_1=14$。有 4 个活动的入度为 0,属于起始活动,该集合记为 $\{\tau_{1,start}:(\tau_{1,1},\tau_{1,3},\tau_{1,4},\tau_{1,7})\}$;另有 2 个活动的出度为 0,属于终止活动,该集合记为 $\{\tau_{1,end}:(\tau_{1,9},\tau_{1,10})\}$。

依据定义 1 可以找出图 2 中的所有有效路径。$VPath_1:\tau_{1,1}\rightarrow\tau_{1,2}\rightarrow\tau_{1,5}\rightarrow\tau_{1,8}\rightarrow\tau_{1,9}$,$Vpath_2:\tau_{1,1}\rightarrow\tau_{1,2}\rightarrow\tau_{1,5}\rightarrow\tau_{1,8}\rightarrow\tau_{1,10}$,$VPath_3:\tau_{1,1}\rightarrow\tau_{1,2}\rightarrow\tau_{1,5}\rightarrow\tau_{1,9}$,$VPath_4:\tau_{1,1}\rightarrow\tau_{1,2}\rightarrow\tau_{1,6}\rightarrow\tau_{1,8}\rightarrow\tau_{1,9}$,$VPath_5:\tau_{1,1}\rightarrow\tau_{1,2}\rightarrow\tau_{1,6}\rightarrow\tau_{1,8}\rightarrow\tau_{1,10}$,$VPath_6:\tau_{1,1}\rightarrow\tau_{1,2}\rightarrow\tau_{1,6}\rightarrow\tau_{1,10}$,$Vpath_7:\tau_{1,3}\rightarrow\tau_{1,5}\rightarrow\tau_{1,8}\rightarrow\tau_{1,9}$,$VPath_8:\tau_{1,3}\rightarrow\tau_{1,5}\rightarrow\tau_{1,8}\rightarrow\tau_{1,10}$,$VPath_9:\tau_{1,3}\rightarrow\tau_{1,5}\rightarrow\tau_{1,9}$,$VPath_{10}:\tau_{1,4}\rightarrow\tau_{1,6}\rightarrow\tau_{1,8}\rightarrow\tau_{1,9}$,$VPath_{11}:\tau_{1,4}\rightarrow\tau_{1,6}\rightarrow\tau_{1,8}\rightarrow\tau_{1,10}$,$VPath_{12}:\tau_{1,7}\rightarrow\tau_{1,9}$。

如图 2 所示,路径 $VPath_3:\tau_{1,1}\rightarrow\tau_{1,2}\rightarrow\tau_{1,5}\rightarrow\tau_{1,9}$ 是路径 $VPath_1:\tau_{1,1}\rightarrow\tau_{1,2}\rightarrow\tau_{1,5}\rightarrow\tau_{1,8}\rightarrow\tau_{1,9}$ 的子集,取二者包含节点数较多的路径 $VPath_1$ 作为一条可选择的有效主路径;同样 $VPath_5$ 包含了 $VPath_6$ 的所有节点,取 $VPath_5$ 作为一条可选择的有效主路径。$VPath_1$ 和 $VPath_5$ 的路径长度为 5,可知路径长度为 5 是图 2 所有有效路径中的最大路径长度,$MPLength=5$ 的可选择有效主路径有 $VPath_1$,$VPath_2$,$VPath_4$ 和 $VPath_5$。

图 2 中,所有 10 个活动出入度之和 $De_{\tau_{i,j}}$ 分别为{1,3,1,1,4,4,1,4,3,2}。上述 4 条可选择有效主路径包含的节点有 $Node_1:\{\tau_{1,1},\tau_{1,2},\tau_{1,5},\tau_{1,6},\tau_{1,8},\tau_{1,9},\tau_{1,10}\}$,对应的出入度为{1,3,4,4,4,3,2},可知出入度最高的活动为 $\tau_{1,5}$,$\tau_{1,6}$ 和 $\tau_{1,8}$。接着考虑这三个活动之间的关系,因为 $\tau_{1,5}$ 和 $\tau_{1,6}$ 是 $\tau_{1,8}$ 的父亲节点,所有可选择有效主路径都包含 $\tau_{1,8}$,所以将 $\tau_{1,8}$ 加入有效主路径 $VMPath$,并将 $\tau_{1,8}$ 从 $Node_1$ 中移出。$\tau_{1,5}$ 和 $\tau_{1,6}$ 为兄弟节点,计算出 $De_{\tau_{1,5}Pa\&Ch}$ 和 $De_{\tau_{1,6}Pa\&Ch}$ 分别为 10 和 9,因此选择 $\tau_{1,5}$ 加入有效主路径 $VMPath$,并将 $\tau_{1,5}$ 和 $\tau_{1,6}$ 从 $Node_1$ 中移出。然后找出包含 $\tau_{1,5}$ 和 $\tau_{1,8}$ 的路径分别为 $path_1$ 和 $path_2$,这两条路径中都包含 $\{\tau_{1,1},\tau_{1,2},\tau_{1,5},\tau_{1,8}\}$,二者的区别是最后的终止活动 $\tau_{1,9}$ 和 $\tau_{1,10}$,因此只需比较这两个节点的出入度即可得到最后结果。$\tau_{1,9}$ 的出入度为 3,$\tau_{1,10}$ 的出入度为 2,因此选择出入度较高的活动 $\tau_{1,9}$ 加入主路径,最后选择的有效主路径 $VMPath$ 为 $path_1:\tau_{1,1}\rightarrow\tau_{1,2}\rightarrow\tau_{1,5}\rightarrow\tau_{1,8}\rightarrow\tau_{1,9}$。

4.1.3　算法伪码

有效主路径提取算法的伪码如图 3 所示。

4.2　基于任务执行截止期限的拉伸调度方案

不同的活动分配方式有不同的调度结果,根据任务的执行截止期限,可以选择部分活动并行处理,

```
Node d;                                  //节点d
Int InDegree,OutDegree;                  //出度和入度
CreatValidPath()                         //生成有效路径
{
   i=j=0;
   入度为0的节点数组s[];
   出度为0的节点数组f[];
   遍历DAG中的所有节点;
   If (d.InDegree==0)                    //入度为0的节点
         s[i]=d,i=i+1;
   Else If (d.OutDergee==0)              //出度为0的节点
         f[j]=d,j=j+1;
   End
   找出使s[]中的每个节点到达f[]中每个节点的路径VPath;
   找出n条有效路径VPath[n];
   Return VPath[n];
}
CreatOptionalValidPath()      //生成可选择有效路径
{
    CreatValidPath();         //找出n条有效路径
    For i=1:n
        计算每条有效路径VP[i]中的节点个数num[i];
        比较num[]中所有的值,选出最大值M;
    End for
    取出有效路径中节点个数为M的路径;
    找出m条可选择有效路径OVPath[m];
    Return OVPath[m];
}
CreateValidMainPath()      //生成有效主路径
{
    计算OVPath[]中所有路径包含节点的出入度之和;
    d.sumDegree=d.InDegree+d.OutDegree;
    找出出入度之和最大的节点;
    If 存在两个或多个节点的出入度之和都是最大 Then
        先找出所有或大多数路径都共有的节点,选入有效主路径中;
    If 两个或多个节点是兄弟节点 Then
        选择这两个或多个节点所有父亲节点与子节点的出入度之和
较大的节点,作为有效主路径中的节点;
    依次比较,直到最后一个节点,找出有效主路径VMPath;
    Return VMPath;
}
```

图3 有效主路径提取算法伪码

以得出不同的调度方案。下面首先按照前述 DAG 模型计算出不同调度方案下 DAG 任务的执行完成时间;然后在选取有效主路径的基础上,根据不同 DAG 任务的调度需求,给出并行任务的拉伸调度方案;最后给出一个具体任务的拉伸调度结果,并给出拉伸调度算法的伪码。

4.2.1 DAG 任务执行完成时间计算

不同调度方案有不同的调度结果,表现出各种方案的弹性和可调度性。根据 3.2 节所述的 DAG 任务模型,可以计算出任务 τ_i 的执行完成时间,为串行部分与并行部分执行时间的总和:

$$T_i = \sum_{s_i=1}^{m_i}\Big(\sum C_i^{s_i} + \big(\max_{y_{i,j}} P_i^{s_i} + \sum_{k=1}^{x_{i,j}-y_{i,j}} P_i^{s_i}\big)\Big), 1 \leqslant j \leqslant n_i。 \quad (1)$$

DAG 的实际执行情况中,串行部分只能按顺序执行,并行部分可根据处理器的使用情况与总的执行截止期限 D_i 的要求,选择部分并行执行、部分拉伸执行。因此,可将 T_i 分为两部分:①串行部分,指所有串行部分的执行时间总和 $\sum(C_i^{s_i})$;②并行部分,该部分可选择个活动并行处理,执行时间为个活动的最大值,剩余个活动沿主路径拉伸的执行时间为,因此并行部分总的执行时间为。

由式(1)可以求出 T_i 的最大执行和最小执行完成时间,分别如式(2)和式(3)所示:

$$Max_i = \sum_{s_i=1}^{m_i}\Big(\sum C_i^{s_i} + \sum_{k=1}^{x_{i,j}} P_i^{s_i}\Big), \quad (2)$$

即当所有可并行活动都被拉伸处理、只使用一个处理器时所求得的执行完成时间。其中 $\sum_{k=1}^{x_{i,j}} P_i^{s_i}$ 表示 $x_{i,j}$ 个并行活动拉伸后的执行完成时间总和,此时 $y_{i,j}=0$。

$$Min_i = \sum_{s_i=1}^{m_i}\Big(\sum C_i^{s_i} + \max_{x_{i,j}} P_i^{s_i}\Big), \quad (3)$$

即当所有可并行任务都被并行处理、使用 $x_{i,j}$ 个处理器时求得的执行完成时间。其中 $\max_{x_{i,j}} P_i^{s_i}$ 表示 $x_{i,j}$ 个并行活动并行执行后的完成时间最大值,此时 $x_{i,j}=y_{i,j}$。

最大执行和最小执行完成时间各有特点,当活动没有时间要求时可选择拉伸处理,以减少对处理器的占用;当活动需要在最短时间内完成时可选择并行处理,以使用尽可能多的处理器完成调度。实际 DAG 任务中,当任务 τ_i 的执行截止期限 $D_i > Max_i$ 时,无论 τ_i 是并行还是串行处理,都可以在要求的时间范围内完成调度,因此只需使用一个处理器即可完成任务 τ_i;当 $D_i < Min_i$ 时,τ_i 不可能在要求的时间范围内完成调度,即 τ_i 为不可调度的任务。大多数情况下,当 $Min_i \leqslant D_i \leqslant Max_i$ 时,可根据调度执行情况选择部分子活动并行处理。因此,找出一种合理的并行与拉伸调度方式,既可有效利用各处理器、使资源利用最大化,又可使调度执行完成时间尽可能接近执行截止期限 D_i,减少对处理器的占用。

4.2.2 基于有效主路径的拉伸策略

根据 4.1 节提取的有效主路径,可以求得主路径中所有活动的执行完成时间 η_i,由 4.2.1 节分析可知 $\eta_i = Min_i$。当已知任务的执行截止期限 D_i 后,任务 τ_i 的 η_i 相对于 D_i 通常有一些时间冗余,记为 L_i,

$$L_i = D_i - \eta_i。\tag{4}$$

确定有效路径后,所有可并行部分的执行时间总和定义为 P_i,

$$P_i = \sum_{s_i=1}^{m_i}(\max_{\tau_{i,j}} P_i^{s_i})。\tag{5}$$

针对存在的冗余时间 L_i,并不是任意将并行部分进行拉伸,而是通过求出并行部分可拉伸的比例,将部分并行活动进行拉伸,因此定义拉伸因子为 f_i,

$$f_i = \frac{L_i}{P_i}。\tag{6}$$

每一个可并行部分通过拉伸后调度,使总的任务执行时间 T_i 接近或等于 D_i,因此每个并行部分拉伸执行后也有其相应的执行完成时间,定义为 d_i^s,

$$d_i^s = P_i^s \times (f_i + 1), 1 \leqslant s \leqslant s_i。\tag{7}$$

由式(4)、式(5)和式(7)可以求出每个并行部分可使用的时间冗余,即

$$L_i^s = d_i^s - P_i^s, 1 \leqslant s \leqslant s_i。\tag{8}$$

存在以下约束条件:

(1)拉伸后的执行时间应限制在 d_i^s 内,$T_i^{s,1} \leqslant d_i^s$。

(2)根据 L_i^s 的大小可知,不是所有并行部分的活动都可以被拉伸,因此计算出不需要进行拉伸的并行活动的个数为

$$(m_i - \lfloor f_i \rfloor - 2)。\tag{10}$$

(3)拉伸划分后,某些活动可能被切分,计算活动被拉伸部分的时间为 $(f_i - \lfloor f_i \rfloor) \times P_i^s$,活动被拉伸后冗余时间 R_i^s 为 $(\lfloor f_i \rfloor + 1 - f_i) \times P_i^s$。

4.2.3　拉伸调度实例

采用图 2 所示的 DAG 任务,依据 4.1.2 节提取的有效主路径,按照 3.2 节的 DAG 划分模型将图 2 的任务划分成串行与并行部分:$\{\tau_1: P_1^1(\tau_{1,1}, \tau_{1,3}, \tau_{1,4}, \tau_{1,7}), C_1^2(\tau_{1,2}), P_1^3(\tau_{1,5}, \tau_{1,6}), C_1^4(\tau_{1,8}), P_1^5(\tau_{1,9}, \tau_{1,10})\}$。

依据有效主路径 $\tau_{1,1} \rightarrow \tau_{1,2} \rightarrow \tau_{1,5} \rightarrow \tau_{1,8} \rightarrow \tau_{1,9}$ 完全并行执行,可知 $\eta_1 = 10$,同样 $Min_1 = 10$。由于 $D_1 = 14$,若将任务全部展开串行处理,即 $Max_1 = 19$,满足条件 $Min_1 \leqslant D_1 \leqslant Max_1$,则任务 τ_1 适合拉伸调度,可得 $L_1 = D_1 - \eta_1$,$L_1 = 4$,如图 4 所示。

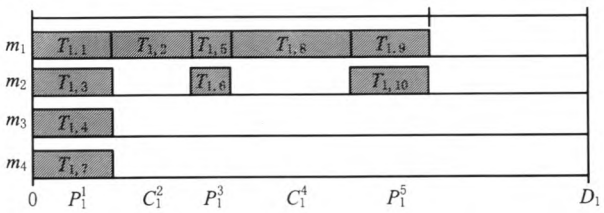

图4　依据有效主路径节点并行调度

由式(5)和式(6)知 $P_1 = P_1^1 + P_1^3 + P_1^5 = 2 + 1 + 2 = 5$,$f_1 = \frac{L_1}{P_1} = \frac{4}{5}$,可将冗余时间平均分配给每个并行部分,求得:

$$d_1^1 = P_1^1 \times (1 + f_1) = 2 \times \left(1 + \frac{4}{5}\right) = 3.6, L_1^1 = d_1^1 - P_1^1 = 1.6, R_1^1 = \left(\left\lfloor \frac{4}{5} \right\rfloor + 1 - \frac{4}{5}\right) \times P_1^1 = 0.4;$$

$$d_1^3 = P_1^3 \times (1 + f_1) = 1 \times \left(1 + \frac{4}{5}\right) = 1.8, L_1^3 = d_1^3 - P_1^3 = 0.8, R_1^3 = \left(\left\lfloor \frac{4}{5} \right\rfloor + 1 - \frac{4}{5}\right) \times P_1^3 = 0.2;$$

$$d_1^5 = P_1^5 \times (1 + f_1) = 2 \times \left(1 + \frac{4}{5}\right) = 3.6, L_1^5 = d_1^5 - P_1^5 = 1.6, R_1^5 = \left(\left\lfloor \frac{4}{5} \right\rfloor + 1 - \frac{4}{5}\right) \times P_1^5 = 0.4。$$

拉伸后可以得出在满足执行截止期限 D_1 情况下的拉伸调度策略,如图 5 所示。

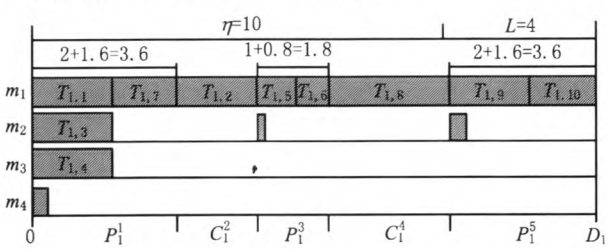

图5　依据有效主路径与执行时间期限拉伸调度

根据求出的拉伸因子 f_1,将冗余时间平均分给任务的并行部分,从而将并行任务拉伸至处理器 m_1 执行,而串行部分不做任何改变,使处理器 m_1 的执行在执行截止期限 D_1 时完成。从图 5 可见,活动 $\tau_{1,7}$,$\tau_{1,6}$ 和 $\tau_{1,10}$ 已被部分拉伸至 τ_1 的主路径所在的处理器 m_1 中,减少了任务对其他处理器的占用。

4.2.4　算法伪码

基于有效主路径的拉伸算法伪码如图 6 所示。

5　实验结果

下面首先通过一组实例来验证基于有效主路径拉伸算法的有效性,然后对比已有改进的单调截止期限调度(Improved Deadline Monotonic Scheduling, IDMS)[15]和最早截止期限优先(Limited Preemptive Earliest Deadline First, LPEDF)[16]算法使用不同数量的处理器执行所得的成功率。

```
Int VPath[];                    //定义有效主路径数组
Int cd[];                       //定义拉伸调度后的路径数组
Int L[];                        //定义冗余时间数组
Int D[];                        //定义执行时间期限数组
Int P[];                        //定义并行部分执行时间数组
Float f[];                      //定义拉伸因子数组

StretchScheduling
{
    If 实例所有子活动的执行时间≤截至期限 Then
        任务按顺序执行;
        For m=1:n do
            将并行段子活动插入VPath;
            以第一个处理器形成顺序结构执行活动
        End for
    else
        求出任务的最短执行时间η
        L=D-η;   //根据任务最短执行时间和时间期限求出冗余时间
        f=L/P;  //求出冗余时间对每一部分并行段的比例f
        计算拉伸后在每一个并行段中并行活动的数目q;
        For k=1:n do
            If k=1 or k>q Then
                将该处理器上的子任务插入VPath中;
            Else
                If k<q Then
                        //构造一个新的并行段
                在该处理器上子活动的执行时间范围为[P, (1+f)P];
                Else
                        //对该虚拟机上的子活动进行切分拉伸
                    计算切分到主路径VPath的子活动的执行时间;
                    计算切分后剩余子活动的执行时间;
                End If
            End If
        End For
    End If
}
```

图6 拉伸调度算法伪码

5.1 实例验证

首先在 4.2.3 节给出的 DAG 任务 τ_1 的基础上，加上其他两个 DAG 任务，每个任务都有各自的执行截止期限。任务 τ_2 和 τ_3 如图 7 所示。

任务 τ_2 只有一个串行活动，该活动的执行时间是 13 个时间单位，其执行时间期限 $D_2=14$；任务 τ_3 包含 1 个串行活动和 2 个并行活动，整个任务的执行时间是 3 个时间单位，其执行截止期限 $D_3=3$，表示任务 3 是实时调度任务，一旦开始执行就不能停止，直至处理完成该任务。所有 3 个任务总的执行截止期限 $D=14$。

在没有采用拉伸调度算法前，尽可能按并行方式进行任务调度，3 个任务的调度将出现如图 8 所示的结果。

图 8 中，基于任务 τ_1 的并行调度方式尽可能早地执行任务 τ_2 和 τ_3。因为任务 τ_2 只能在 m_3 或 m_4 中执行，其最早开始执行时间为 2，执行时间为 13，所以任务 τ_2 的最早完成时间为 15，超出了 3 个任务总的执行截止期限 D。任务 3 的串行部分 C_3^1 可选择在 m_2 或 m_4 中执行，然而在调度完任务 τ_1 和任务 τ_2 后，已有三个处理器被占用，并行部分 P_3^2 只能选择其中一个处理器顺序执行，因此任务 τ_3 的最早完成时间为 4，超出了其执行截止期限 D_3。

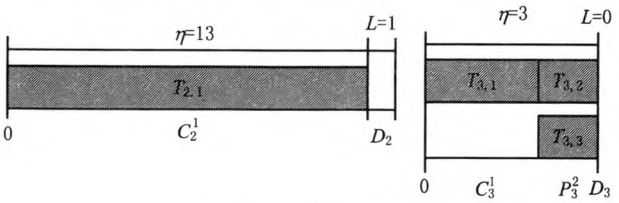

图7 DAG任务 τ_2 和任务 τ_3

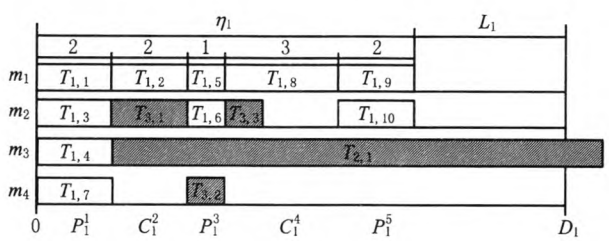

图8 多任务并行调度

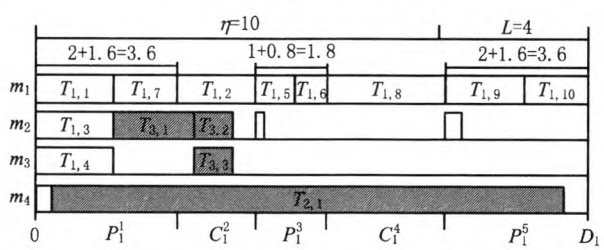

图9 多任务拉伸调度

通过将任务 τ_1 拉伸调度(如图 9)，任务 τ_2 可以在 m_4 中执行，其最早开始执行时间为 0.4，执行时间为 13，因此任务 τ_2 的最早完成时间为 13.4，满足其执行截止期限 D_1 和总的执行截止期限 D。任务 τ_3 的串行部分 C_3^1 可选择在 m_2 或 m_3 中执行，并行部分 P_3^2 可选择在 m_2 和 m_3 中并行执行，因此任务 3 的最早完成时间为 3，满足其执行截止期限 D_3。

综上所述，采用并行方式进行调度可能出现一个任务占用较多处理器资源的情况，使其他任务的执行超出其执行截止期限，并且在执行后期有较多的处理器处于闲置状态，从而使所有任务总的完成时间超出执行截止期限；拉伸调度会在满足总的执行完成期限的同时，尽可能减少对处理器的占用，使其他任务能有效地进行调度。

5.2 大规模拉伸仿真调度验证

使用 YARTISS[19] 仿真器产生仿真数据集。YARTISS 是一个用于实时调度系统验证的新型仿真工具，目的是方便用户将自己开发的算法与实时调度中已有文献的经典算法做比较。该工具使用

Java 开发,由一组易于使用的模块化工具设计而成,不需要解压、编辑甚至重编译即可添加新模块,具有较好的可移植性和普适性,可以用于大规模仿真实验。该仿真器不仅可以测试单处理器调度算法,还可以测试多处理器调度算法,如最早截止期限优先算法。

YARTISS 主要有核心引擎模块、业务模块、框架模块和视图模块四大模块。该工具继承了很多已有调度算法,有大量数据模型并包含并行任务,用户可以很容易使用该工具实施自己的算法。本实验总共随机产生 10 000 个数据集,每个数据集包含的 DAG 任务为 2～10 范围内的随机值,数据集中每个任务的每个活动限定在一定的长度区间内。每个数据集中的任务包含 1～8 个并行活动,表示任务的并行化程度。当并行活动较多时,任务的活动总数会增加,调度任务的结构相对复杂。

本文所提方法整体上比 LPEDF 和 IDMS 的调度均有较大提高。当一个数据集内多个任务所包含的并行活动数量相对较少时,任务比较简单,以串行为主,因此三种调度方法的效果均比较高。然而,当一个数据集内多个任务所包含的并行活动数量较多时,普通的基于单个任务截止时间优先的算法会有较大的局限性,使一个数据集内的其他任务超出截止期限或所有任务的执行总体超出截止期限,导致调度失败。如图 10 所示,当所有任务只有两个并行活动时,所有调度策略都获得 100%的成功率;当最多有 8 个并行活动时,本文所提策略具有明显优势。

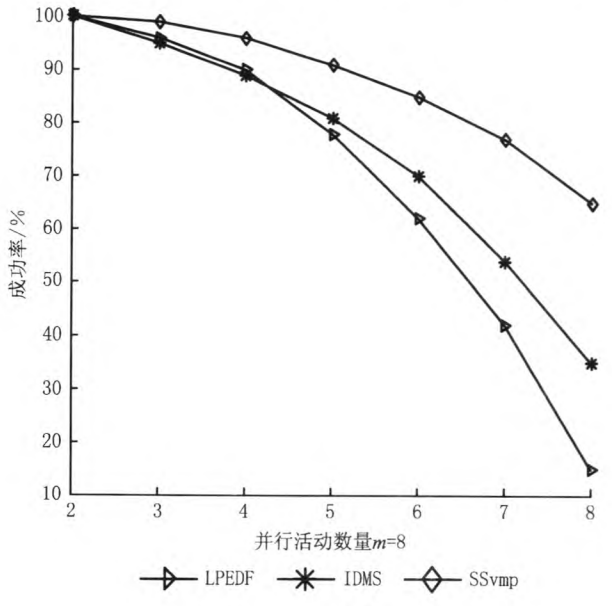

图10　调度成功率仿真结果

6　结束语

本文在传统 DAG 模型的基础上,首先给出一种新的 DAG 模型,将 DAG 调度任务分成若干串行和并行调度部分,然后根据活动的出入度提取 DAG 任务调度的一条有效主路径,给任务的合理调度带来了依据,最后选取与主路径中活动并行的其他活动进行并行处理。基于任务的执行完成时间和执行截止期限,本文又提出一种有效的拉伸调度策略,令任务在满足执行截止期限的同时尽可能减少对处理器资源的占用,使其他任务能有效地进行调度,并能有效地处理完成所有活动。通过仿真实例,证明了本文所提方法的适用性。

在本文基础上,下一步将主要开展以下几方面的研究:①在有效主路径中考虑任务执行时间因素,提高并行任务中执行时间最大的活动所占的权值;②在拉伸调度算法中优先考虑多个并行的任务和较短的任务;③在拉伸调度过程中加入适当的参数,尽可能将任务进行整体拉伸,在满足任务截止期限完成和减少对处理器占用的基础上,更加有效地提高拉伸调度的普适性。

参考文献:

[1] LIU X, WANG D, YUAN D, et al. Throughput based temporal verification for monitoring large batch of parallel processes[C]//Proceedings of the 2014 International Conference on Software and System Process. New York, N. Y., USA: ACM, 2014.

[2] HANDE P V, GHOSH D. A comparative study on factors shaping buying behaviour on B2B and B2C E-commerce platforms in India[J]. EXCEL International Journal of Multidisciplinary Management Studies, 2015, 5(3): 1-10.

[3] LIU X, YUAN D, ZHANG G, et al. Workflow systems in the cloud[M]//The Design of Cloud Workflow Systems. Berlin, Germany: Springer-Verlag, 2012: 1-11.

[4] VAN DEN B J, BELIËN J, DE BRUECKER P, et al. Personnel scheduling: a literature review[J]. European Journal of Operational Research, 2013, 226(3): 367-385.

[5] LIU K, JIN H, CHEN J, et al. A compromised-time-cost scheduling algorithm in SwinDeW-C for instance-intensive cost-constrained workflows on cloud computing platform[J]. International Journal of High Performance Computing Applications, 2010, 24(4): 445-456.

[6] KONG Y, ZHANG M, YE D. A negotiation-based method for task allocation with time constraints in open grid environments[M]. Concurrency and Computation: Practice and Experience, 2015, 27(3): 735-761.

[7] VASILECAS O, SAVICKAS T, LEBEDYS E. Directed acyclic graph extraction from event logs[M]//Information and Software Technology. Berlin, Germany: Springer-Verlag, 2014:172-181.

[8] FERNÁNDEZ-BACA D. Allocating modules to processors in a distributed system[J]. IEEE Transactions on Software Engineering, 1989, 15(11):1427-1436.

[9] BOKHARI S H. A shortest tree algorithm for optimal assignments across space and time in a distributed processor system [J]. IEEE Transactions on Software Engineering, 1981, 7(6): 583-589.

[10] CHEN Hongwei, WANG Ruchuan. A grid DAG scheduling algorithm for cost-time optimization[J]. Chinese Journal of Electronics, 2005, 33(8):1375-1380(in Chinese).[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):1375-1380.]

[11] GUO He, CHEN Zheng, YU Yulong, et al. A communication aware DAG workflow cost optimization model and algorithm[J]. Journal of Computer Research and Development, 2015, 52(6):1400-1408(in Chinese).[郭 禾,陈 征,于玉龙,等. 带通信开销的DAG工作流费用优化模型与算法[J]. 计算机研究与发展,2015,52(6):1400-1408.]

[12] YUAN Yingchun, LI Xiaoping, WANG Qian, et al. Time optimization heuristics for scheduling budget-constrained grid workflows[J]. Journal of Computer Research and Development, 2015, 46(2):194-201(in Chinese).[苑迎春,李小平,王茜,等. 成本约束的网格工作流时间优化方法[J]. 计算机研究与发展, 2015,46(2):194-201.]

[13] QAMHIEH M, GEORGE L, MIDONNET S. A stretching algorithm for parallel real-time DAG tasks on multiprocessor systems[C]//Proceedings of the 22nd International Conference on Real-Time Networks and Systems. New York, N. Y., USA: ACM, 2014.

[14] LAKSHMANAN K, KATO S, RAJKUMAR R. Scheduling parallel real-time tasks on multi-core processors[C]//Proceedings of the IEEE 31st Real-Time Systems Symposium. Washington, D. C., USA: IEEE, 2010.

[15] BEHERA H, PANDA S S, CHAKRABORTY J. Improved deadline monotonic scheduling with dynamic and intelligent time slice for real-time systems[J]. International Journal of Information Technology Convergence and Services, 2012, 2 (2):11-21.

[16] THEKKILAKATTIL A, BARUAH S, DOBRIN R, et al. The global limited preemptive earliest deadline first feasibility of sporadic real-time tasks[C]//Proceedings of the 26th Euromicro Conference on Real-Time Systems. Washington, D. C., USA: IEEE, 2014.

[17] CHAI Xuezhi, CAO Jian. Cloud computing oriented workflow technology[J]. Journal of Chinese Computer Systems, 2012, 33(1):90-95(in Chinese).[柴学智,曹 健. 面向云计算的工作流技术[J]. 小型微型计算机系统,2012,33(1):90-95.]

[18] YUAN Gang, SUN Ruizhi, XIANG Yong, et al. Semantic validation of workflow based condition-restricted and its applications[J]. Journal of Chinese Computer Systems, 2013, 34 (9):2051-2055(in Chinese).[袁 钢,孙瑞志,向 勇,等. 基于条件约束的工作流语义验证方法及应用[J]. 小型微型计算机系统,2013,34(9):2051-2055.]

[19] CHANDARLI Y, FAUBERTEAU F, MASSON D, et al. Yartiss: a tool to visualize, test, compare and evaluate real-time scheduling algorithms[C]//Proceedings of the 3rd International Workshopon Analysis Tools and Methodologies for Embedded and Real-time Systems. Villeurbanne, Lyon, France: CCSD, 2012:21-26.

作者简介:

许荣斌(1981—),男,安徽黄山人,讲师,研究方向:工作流、移动计算等,E-mail:xurb_910@ahu.edu.cn;

刘 鑫(1996—),男,江西赣州人,本科生,研究方向:工作流调度;

杨壮壮(1994—),男,安徽宣城人,本科生,研究方向:业务流程处理;

郭 星(1983—),男,安徽庐江人,讲师,研究方向:移动计算,服务计算等;

+谢 莹(1981—),女,安徽合肥人,副教授,研究方向:机器学习、云计算等,通信作者,E-mail:xieyingahu@126.com;

吴建国(1954—),男,安徽合肥人,教授,研究方向:嵌入式系统、模式识别等。