基于蚁群的工作流任务分配算法研究

崔 璐,毋 涛
(西安工程大学 计算机科学学院,陕西 西安 710600)

摘 要:影响工作流系统性能的因素不仅有工作流执行者的经验、兴趣与能力,还有执行者的当前任务负载,尤其在实例密集的情况下,通常会出现负载失衡或过载的现象,导致工作流效率与流程系统性能降低。针对这一问题,首先考虑流程关键任务对执行者负载的影响,将工作流任务结构定义成有向无环图 DAG 模型,使用拓扑序列确定关键路径与关键任务,在关键任务与任务负载之间建立联系。在此基础上,考虑执行者的预测负载,对执行者的负载进行量化与等级区间划分。随后给出一个基于蚁群的、依据关键任务与负载区间进行任务分配以保证负载均衡的任务分配算法(ACO-CT)。通过对比 HEFT 算法、Round_Robin 算法,表明该算法可在兼顾负载均衡的基础上提升流程效率,并且具有较好的收敛性。

关键词:工作流;负载均衡;关键路径;任务分配;蚁群
中图分类号:TP301 **文献标识码**:A **文章编号**:1673-629X(2021)05-0102-06
doi:10.3969/j.issn.1673-629X.2021.05.018

Research on Workflow Task Allocation Algorithm Based on Ant Colony

CUI Lu, WU Tao
(School of Computer Science, Xi'an Polytechnic University, Xi'an 710600, China)

Abstract: The factors that affect the performance of workflow system are not only the experience, interest and ability of workflow executor, but also the current task load of the executor. Especially in the case of dense instances, the load imbalance or overload often occurs, which leads to the decrease of workflow efficiency and process system performance. In order to solve this problem, firstly considering the impact of process critical tasks on the performer load, the workflow task structure is defined as the DAG model of directed acyclic graph, and the key path and key task are determined by topological sequence, and the relationship between the key task and task load is established. On this basis, considering the executor's predicted load, the executor's load is divided into grades and intervals. Then a task allocation algorithm based on ant colony (ACO-CT) is proposed, which is based on key tasks and load intervals to ensure load balance. The comparison of HEFT algorithm and Round_Robin algorithm shows that the proposed algorithm can improve the process efficiency on the basis of load balancing with better convergence.

Key words: workflow; load balancing; critical path; task allocation; ant colony

0 引 言

工作流就是业务流程的自动化[1],在此过程中,多个参与者之间按照一定的过程规则自动进行文档、信息或任务传递,以实现预期的业务目标,这也是过程、事件、资源的有机结合。随着工作流技术的发展,工作流管理系统被应用于各种领域,工作流管理系统的性能也成为人们关注的焦点。不同的任务分配策略对工作流管理系统的性能有很大的影响,因此需要制定良好的任务分配策略,选择工作流引擎调度系统中合适的资源来操作任务。工作流系统中的资源,根据适用领域的区别,可分为设备资源、人力资源、应用程序或者网络资源等[2],其中人力资源一般指拥有一定经验或专业知识的任务执行者。实际中随着业务流程规模的增大,往往存在多个流程实例同时到达的情景,当工作流系统频繁地把任务分配给能力值或经验值高的执行者,此类执行者任务列表中待处理任务会不断增多,负载过重,反而不能及时处理完所分配的任务。且执行时间相近但重要性不同的任务,对执行者的负载影响也不同。当任务列表中有多个重要任务时,执行者通常会优先执行重要任务而无暇顾及不太重要的任务。这些都是业务流程执行中的重要影响因素,当工作流整体负载不均衡时,会导致工作流系统响应时间

收稿日期:2020-07-15 **修回日期**:2020-11-19
基金项目:陕西省科技成果转移与推广计划项目(2019CGXNG-018)
作者简介:崔 璐(1996-),女,硕士,研究方向为信息系统设计与软件开发;毋 涛,博士,副教授,研究方向为计算机智能信息处理。

过长、资源利用率不高、工作流环境稳定性遭到破坏等问题。

文献[3]提出了一种新的以优化执行时间和处理成本为目标的工作流调度算法,旨在隐式评估适合 VM 执行的实例范围,以避免会导致截止时间违规的过高投入。文献[4]利用三角模糊数将执行者的专业技能、成员协作度、任务完成质量这类定义模糊的影响因素进行量化处理,再对各个影响因素分配合适的权重因子,最后选取综合分数高的候选者分配任务。但这些研究都没有考虑在工作流的实际应用中,当实例密集到达时执行者负载对流程的影响。文献[5]给出结合协作相容与负载均衡的多目标联合优化任务分配算法,提高流程执行效率,却缺少任务重要性对任务负载的影响分析。文献[6]从任务和用户的属性出发,提出一种兼顾用户经验值和任务负载的任务分配算法,并针对任务的重要程度给任务负载加影响因子,但是执行者根据主观定义任务的重要程度,不能客观反映任务重要性差异与流程执行时间长短的关系。文献[7]针对网格环境下工作流调度问题,提出了一种基于遗传算法和蚁群算法相结合的混合机制,优化模型的最大完成时间和成本。将蚁群这种自适应算法应用于工作流调度中,提升流程的实用性与效率。综上所述,为了提高流程的性能,该文提出基于蚁群算法的兼顾关键任务和工作负载的任务分配算法,不仅考虑了执行者的当前工作负载,还考虑了关键任务重要性对工作负载的影响。

1　任务分配策略与模型

1.1　主要思想

任务分配的目标,是根据执行者们的预测负载所在分区和任务列表中待执行的关键任务数量,来选择负载相对较小的执行者执行任务,以平衡实例密集时执行者的负载压力,同时让关键任务减少等待时间,尽可能并行。该文先通过执行者的负载状况,将执行者动态分成轻、中、重三个负载分区,然后将轻负载分区中的执行者作为候选集合,然后从集合中选择待执行关键任务最少的执行者,对其进行任务分配。

1.2　关键任务的量化

1.2.1　关键任务确定

工作流的应用中,企业通常关心的是:(1)在实现业务预期目标的前提下,完成整个工作流最少需要多少时间?(2)哪些任务是影响工作流进度的关键?

可以采用带权值的有向无环图 DAG 来表示工作流[8],由于流程中存在并行路径,所以完成流程的最短时间是从开始点到完成点的持续时间最长路径的长度。有向无环图中,路径长度最长的路径叫做关键路径[9]。关键路径上的活动叫做关键活动,因此可以依据有向无环图的关键活动,得到工作流中影响流程进度的关键任务。

以图 1 所示的简单的领料流程为例,求取关键任务。

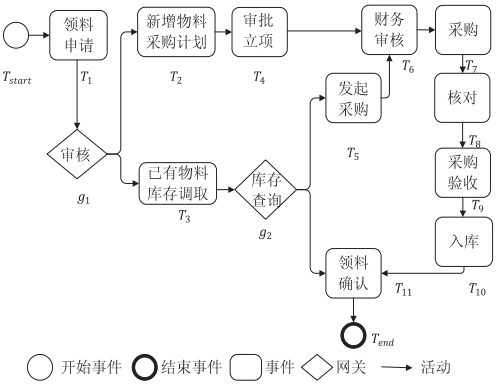


图 1　简单的领料流程

对领料流程进行建模:$G=(V,E)$,其中 V 由一系列的顶点 $\{v_1(T_{\text{start}}),v_2(T_1),\cdots,v_{|V|}(T_{\text{end}})\}$ 组成,表示流程中的顶点以及对应事件,E 由一系列有向边 $\{e_j^i|(v_i,v_j)\in E\}$ 组成,表示事件的依赖关系,即流程中的活动,每条边 e_i 包含一个权值 c_i,表示该活动需要消耗的时间。再将网关从流程图中去掉,得到有向无环图,如图 2 所示。

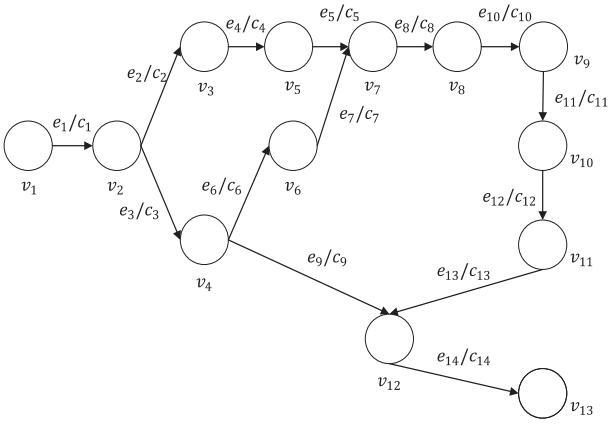


图 2　图 1 流程图对应的有向无环图

这样就把问题转换为求图 2 所示带权值的有向无环图 G 的关键路径 CP(critical path)。当关键任务执行时间延长或缩短时,关键路径的执行时间即流程的总执行时间也相应延长或缩短,所以关键任务的分配方式与执行效率对提高流程效率起到了重要作用。在计算关键路径时,可以参考流程日志中的历史流程数据获得每个任务的平均执行时间作为标准。

1.2.2　关键任务量化

在分配任务时,关键任务量由执行者任务列表中待执行的关键任务数量来确定。执行者拥有的关键任务数量越多,对流程的完成时间影响越大,这是因为实际中任务的重要程度与工作负载成正比例关系。$u_i\in$

U 的关键任务量 $E_{cp}(u_i)$ 是当前执行者任务列表中待执行的关键任务数量,由于它是动态变化的,所以该文取归一化 u_i 的相对关键任务量 $\tilde{E}_{cp}(u_i)$:

$$\tilde{E}_{cp}(u_i)=\frac{E_{cp}(u_i)-\mathrm{MIN}(E_{cp}(u_i))}{\mathrm{MAX}(E_{cp}(u_i))-\mathrm{MIN}(E_{cp}(u_i))} \quad (1)$$

其中, $\mathrm{MAX}(E_{cp}(u_i))$ 是当前执行者中关键任务量最大值, $\mathrm{MIN}(E_{cp}(u_i))$ 是当前执行者中关键任务量最小值。$E_{cp}(u_i)$ 值越大,执行者 u_i 的相对关键任务量越大,在执行者负载相近时,获得任务的概率越小。

1.3 负载量化

任务执行者的当前工作负载是完成自己工作列表中所有任务所需要的时间。分配任务时该文假设同一任务的可选执行者执行本任务的能力相似,即执行同样任务,不同执行者的时间相同。则可以定义执行者 u_i 的当前工作负载为 $W_{cur}(u_i)$:

$$W_{cur}(u_i)=\sum_{T_k\in \mathrm{TA}_k} n_k\omega_k \quad (2)$$

其中, TA_k 是任务执行者 u_i 的当前任务列表集合,列表中每个任务 T_k 的数量为 n_k ,完成时间为 ω_k 。设执行者 u_i 的当前工作负载为 $W_{cur}(u_i)$,若现将任务 T_k 分配给 u_i ,则 u_i 的预测负载 $W_{pred}(u_i)$ 为:

$$W_{pred}(u_i)=W_{cur}(u_i)+\omega_k \quad (3)$$

令 $\mathrm{UA}_k=\{u_i\}$ 表示能够执行任务 T_k 的执行者们的集合,设集合中执行者 u_i 的预测负载为 $W_{pred}(u_i)$,则通过归一化处理得到 u_i 的相对预测负载 $\tilde{W}_{pred}(u_i)$ 为:

$$\tilde{W}_{pred}(u_i)=\frac{W_{pred}(u_i)}{\sum_{u_i\in \mathrm{UA}_k} W_{pred}(u_i)} \quad (4)$$

据此,设共有 n 个执行者,完成工作流中所有任务的总负载 W_{total} 为:

$$W_{total}=\sum_{i=1}^{n} W_{pred}(u_i) \quad (5)$$

令 $\bar{W}=\frac{1}{n}\sum_{i=1}^{n} W_{cur}(u_i)$ 为执行者的平均工作负载,则所有执行者的负载均衡情况可以用标准差来表示:

$$\sigma=\sqrt{\sum_{i=1}^{n}\frac{[W_{cur}(u_i)-\bar{W}]^2}{n}} \quad (6)$$

依据该文的主要思想,结合关键任务与负载的分配模型为:

$$\mathrm{MAX}(\tilde{E}_{cp}(u_i)^{-1},W_{total}^{-1}) \quad (7)$$

即流程中任务分配的执行者相对关键任务量与总负载都尽可能少。

2 基于蚁群算法的任务分配策略

蚁群算法(ant colony system)是一种用来寻找优化路径的概率型算法,由 Marco Dorigo 于 1992 年在他的博士论文中提出[10]。算法模拟蚂蚁的觅食行为,在蚂蚁寻找食物的过程中不断释放被称为信息素的物质,蚂蚁的栖息地到食物源的路径越短,该路径上通过的蚂蚁的数量就越多,信息素就越强,从而指引蚂蚁的行为[11]。即蚁群之间通过信息素的浓度变化进行信息交流和相互协作,浓度越高的路径选择的概率越大[12]。基于此该文提出一种基于 ACO 的任务分配策略,除此之外还使用关键路径算法从历史流程数据中确定关键任务节点集合,在此基础上考虑负载均衡通过联合优化策略给出初始解并计算流程总负载。

2.1 基于关键路径的流程关键任务确定

将工作流进行形式化描述后得到带全权值的有向无环图 G ,先通过拓扑排序得到流程任务的拓扑序列 topoSort[ ],基于此与历史流程数据确定流程的关键路径,从而得到流程关键任务集合 CT。

2.2 联合优化策略

目标是在保持执行者负载相对平衡的基础上,选择相对关键任务量较小的执行者,以减少关键任务堆积对执行者与流程的影响。

CTLB 算法的执行流程为:

(1)当分配流程某一实例中的一个任务 T_k 时,对可执行 T_k 任务的执行者集合 UA_k ,使用公式(1)~公式(3)计算集合中每个执行者的预测负载和相对预测负载,依据相对预测负载的值将执行者放入对应的负载分区。负载分区分为轻负载集合 $W_L=\{u_i\mid \tilde{W}_{pred}(u_i)\in[0,\varepsilon_L)\}$ 、中负载集合 $W_M=\{u_i\mid \tilde{W}_{pred}(u_i)\in[\varepsilon_L,\varepsilon_M)\}$ 、重负载集合 $W_H=\{u_i\mid \tilde{W}_{pred}(u_i)\in[\varepsilon_M,1)\}$,其中 $\varepsilon_L<\varepsilon_M$ 且都是可由系统更改的区间阈值。

(2)判断待分配任务是否是关键任务,即是否在集合 CT 中,若是关键任务,通过公式(1)在 W_L 与 W_M 中选择 $\tilde{E}_{cp}(u_i)$ 较小的执行者,若不是关键任务则选择轻负载中预测负载值较小的执行者。

(3)循环(1)和(2)直至所有流程中所有任务分配完成。

2.3 ACO-CT 算法

根据蚁群算法的基本原理,可以了解到算法主要依靠启发式信息构建和信息素浓度更新来求取可行解。本节主要分为三部分,包括算法初始化、状态转移规则和信息素更新规则[13]。

2.3.1 *算法初始化*

利用 ACO-CT 算法进行任务分配的目的是寻找 Time_{min} 时的解。解的形式是一个执行者编号组成的数组,每个执行者对应流程中一个事件的分配结果,且

这个结果在考虑了关键任务影响与负载均衡的同时保证流程执行时间尽可能短。

初始化时,数组各元素为 0,调用 CTLB 得到 TAS 为一组初始解,计算得到 WL_Total。若经过一次迭代后,得到的总负载优于 WL_Total,则更新总负载与执行序列解。任务 T_k 与执行者 u_i 之间的信息素矩阵 τ 初始化为:

$$\tau(k,i)=\frac{1}{n} \tag{8}$$

其中,n 为工作流中执行者的数量。并采用自适应蚁群算法 MMAS 对基本蚁群算法的改进,将 $\tau(k,i)$ 限定在 $[\tau_{\min},\tau_{\max}]$ 之间,以避免算法陷入局部最优。

2.3.2　状态转移规则

算法采用概率选择的方法构建可行解,将起始任务的执行者随机分配给不同的蚂蚁,完成该任务后,蚂蚁按照状态转移概率对下一个任务选择合适的执行者,不断重复这个过程直至执行完所有任务。状态转移概率主要由信息素浓度和启发式信息确定[14],而本算法启发式信息的构建目标是引导蚂蚁往列表中关键任务较少的执行者移动,即相对关键任务量越少,执行者被选择的概率越大。启发式信息构建如下:

$$\eta(k,i)=\tilde{E}_{\mathrm{cp}}(u_i)^{-1} \tag{9}$$

则算法的状态转移规则为:

$$P_m(k,i)=\begin{cases}\dfrac{[\tau(k,i)]^{\alpha}\times[\eta(k,i)]^{\beta}}{\sum[\tau(k,i)]^{\alpha}\times[\eta(k,i)]^{\beta}}, & i\in \mathrm{UA}_k\\ 0 & \text{其他}\end{cases} \tag{10}$$

表示蚂蚁从可选执行者列表 UA_k 中选择执行者 u_i 的概率。其中 α 是信息启发式因子,β 是期望启发式因子,利用因子控制信息素和启发式信息的相对影响程度。

2.3.3　信息素更新规则

对于信息素更新规则,选择改进后的全局更新规则,即不再对所有蚂蚁进行更新,而是对每次迭代中最优的蚂蚁进行更新[15]。当一次迭代中每个蚂蚁都完成工作流的任务分配,并得到一个执行者序列与流程总负载,选择最优的流程总负载与初始解的流程总负载比较,对较优的那个信息素进行更新,更新规则如下:

$$\begin{cases}\tau(k,i)\leftarrow(1-\rho)\times\tau(k,i)+\rho\times\Delta\tau^{\mathrm{gb}}(k,i)\\ \Delta\tau^{\mathrm{gb}}(k,i)=\begin{cases}\dfrac{W_{\mathrm{pred}}(u_i)}{\mathrm{WL\_Total}_{\mathrm{gb}}}, & (k,i)\in \mathrm{TAS}_{\mathrm{gb}}\\ 0 & \text{其他}\end{cases}\end{cases} \tag{11}$$

其中,ρ 是信息挥发因子,则 $1-\rho$ 是残留因子,$\mathrm{WL\_Total}_{\mathrm{gb}}$ 是全局最优蚂蚁的工作流总负载,$\mathrm{TAS}_{\mathrm{gb}}$ 是最优蚂蚁的任务分配的执行者序列。

2.3.4　ACO-CT 算法执行流程

基于以上三部分,ACO-CT 的算法执行流程可概述为:

(1)流程初始化,先建立关键路径模型,得到流程的关键任务合集,并在此过程中得到每个任务的前驱、后继任务集合。

(2)在步骤(1)的基础上,通过 CTLB 联合优化算法得到一组初始解暂时作为最优解 $\mathrm{TAS}_{\mathrm{gb}}$,并计算初始解的流程总负载暂时作为最优解 $\mathrm{WL\_Total}_{\mathrm{gb}}$。

(3)初始化任务与执行者的信息素矩阵、算法迭代次数、蚁群规模、信息挥发因子、信息启发式因子和期望启发式因子。

(4)在一次迭代中,每只蚂蚁根据公式(10)的状态转移概率选择任务的执行者,将执行者序列保存下来,计算每个蚂蚁的流程总负载,并与 $\mathrm{WL\_Total}_{\mathrm{gb}}$ 的值相比较,大于则舍去,小于则更新 $\mathrm{WL\_Total}_{\mathrm{gb}}$,并将该解的执行者序列赋值给 $\mathrm{TAS}_{\mathrm{gb}}$,成为新的最优解。直到一次迭代完成。

(5)对 $\mathrm{TAS}_{\mathrm{gb}}$ 使用公式(11)更新信息素。

(6)重复步骤(4)和(5),直至迭代全部完成。

ACO-CT 算法伪代码如下:

输入:Tesk set: $T=\{\{T_k\}\}$; Executor set: $U=\{u_i\}$;

输出:Ant path with optimal execution sequence $\mathrm{TAS}_{\mathrm{gb}}$

(1) Initialization ant number :Ants; Number of iterations:Max_Itera,

(2) Parameters:Rho,Alpha,Beta

(3) Workflow mode and Critical task set {CT} :according to Get_TopoSort and Get_CTasks

(4) Use CTLB algorithm to generate a set of initial solutions as the $\mathrm{TAS}_{\mathrm{gb}}$,and calculate the total workload $\mathrm{WL\_Total}_{\mathrm{gb}}$ according to CAL_WLT

(5) FOR each $pher(k,i)=p_{\mathrm{Init}}$ according to formula (8)

(6) ENDFOR

(7) FOR m=1 TO Max_Itera DO

(8) FOR t=1 TO Ants DO

(9) FOR each $T_k\in T$ DO

(10) $\mathrm{TAS}_t(k)=0$; /* Assignment of ant t to task T_k

(11) ENDFOR

(12) FOR each $T_k\in T$ DO

(13) IF T_k is the first task in workflow

(14) Randomly choose $u_i\in \mathrm{UA}_k$ as the executor for ant t

(15) $\mathrm{TAS}_t(k)=u_i$

(16) ELSE choose an executor $u_i\in \mathrm{UA}_k$ according to formula (10)

(17) $\mathrm{TAS}_t(k)=u_i$

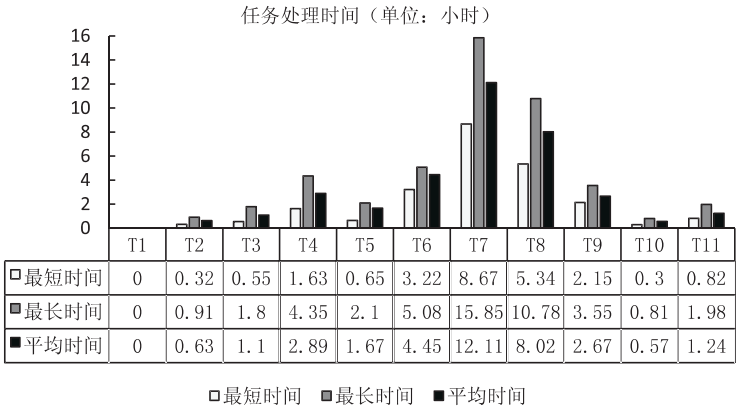
(18) ENDIF

(19) ENDFOR

(20) ENDFOR
(21) FOR t = 1 TO Ants DO
(22) Calculate WL_Total_t according to algorithm CAL_WLT
(23) IF($WL\_Total_t < WL\_Total_{gb}$)
(24) Update the optimal ant to t
(25) $TAS_{gb} \leftarrow TAS_t$
(26) $WL\_Total_{gb} \leftarrow WL\_Total_t$
(27) ENDIF
(28) ENDFOR
(29) FOR each $T_k \in T$ DO
(30) Update pheromone matrix $pher(k,i)$ using the TAS_{gb} according to formula(11)
(31) ENDFOR
(32) ENDFOR

3 仿真实验

本节通过仿真实验检验 ACO-CT 算法解决任务分配问题的可行性。实验从 3 方面评估算法的性能：(1)不同实例规模下 3 种任务分配算法的完成时间；(2)3 种任务分配算法下任务执行者的负载均衡情况；(3)ACO-CT 算法的收敛性。仿真采用图 1 简单领料流程的工作流模型。根据历史流程数据得到各个任务的处理时间，如图 3 所示。



	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
□最短时间	0	0.32	0.55	1.63	0.65	3.22	8.67	5.34	2.15	0.3	0.82
■最长时间	0	0.91	1.8	4.35	2.1	5.08	15.85	10.78	3.55	0.81	1.98
■平均时间	0	0.63	1.1	2.89	1.67	4.45	12.11	8.02	2.67	0.57	1.24

图 3 领料流程各任务时间

历史流程数据通过算法可以得出此流程的关键任务集 $CT = \{T_1, T_2, T_4, T_6, T_7, T_8, T_9, T_{10}, T_{11}\}$，实验中，设置负载区间阈值 ε_L、ε_M 分别为 0.34 和 0.67。在工作流实例不同到达概率的情况下，分别用 ACO-CT、HEFT、Round_Robin 算法进行仿真实验。对每一个实例到达率，对每种算法进行 50 次仿真，并用同样的实例在不同算法上仿真，实验结果采用 50 次仿真的平均值进行分析。

结合领料的实例，平均完成时间比较如图 4 所示。

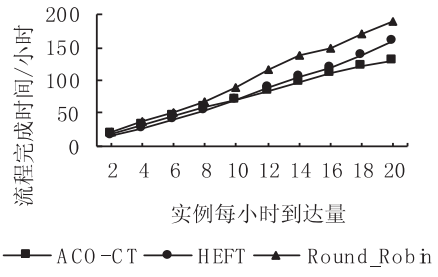


图 4 不同实例规模下的工作流完成时间

从图 4 可以看出，ACO-CT 算法在实例规模不断增加的情况下结果都比较好，尤其在实例规模较大时表现更好，这是因为 ACO-CT 兼顾负载均衡与关键任务量，一定程度上缩短了关键任务过多时对执行者的影响。HEFT 仅仅考虑了期望任务负载，忽略了多实例同时到达时负载不均的影响，但在实际的工作中，一般工作流系统应用于企业时实例规模都会较大，所以 ACO-CT 考虑更加全面。

现分析 3 种任务分配算法下任务执行者的负载情况。实验采用流程实例规模为 8 时，各个执行者的任务负载情况。

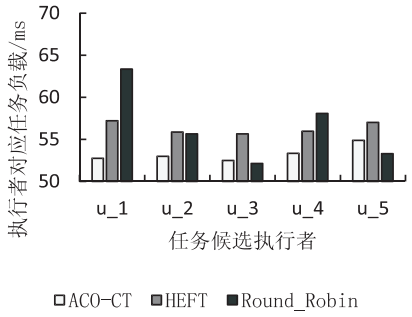


图 5 实例规模为 8 时各个执行者任务负载

从图 5 可以看出，执行者在多流程实例且每个执行者都可以执行多个任务的情况下，Round_Robin 算法执行者任务负载差距较为明显，而 ACO-CT 与 HEFT 算法可以使执行者负载相对平衡。HEFT 算法的思路很简单，就是将所有任务都分配能够使它最早完成的执行者，也可以一定程度上让负载相对均衡，然而 ACO-CT 算法因为提前对可选执行者进行负载分区，并在一定分区内继续考虑关键任务量的影响，所以不仅考虑到了执行者之间的负载均衡，还减少了整体

完成时间。

由于 ACO-CT 算法是一个启发式算法,所以对算法的收敛性进行检验。图 6 是在实例规模为 10 时的最优序列完成时间。随着迭代次数的增加,ACO-CT 算法的最优序列的完成时间逐渐减小,且本算法引入了 MMAS 的思想进行改进,避免了过早收敛的现象。

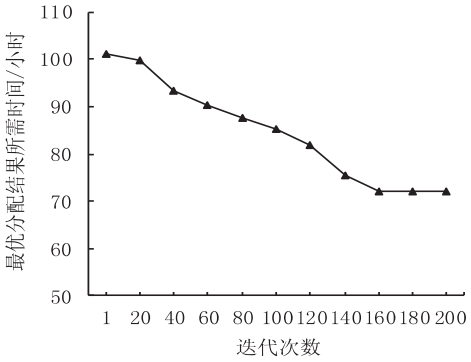


图 6　ACO-CT 算法收敛性

从图 6 可以看出,在迭代 160 次左右时,算法就收敛了,就收敛速度来看,不存在过早收敛的现象。

4　结束语

该文研究了基于蚁群的工作流负载平衡任务分配算法,通过对工作流中执行者关键任务量对流程性能的影响以及执行者之间负载均衡进行建模,采用蚁群算法实现模型,并通过实验验证了算法的有效性。然而,文中的关键任务的确认是基于历史流程数据的,忽略了实际运行过程中任务执行时间的动态变化,下一步将考虑动态关键路径的研究;此外,对于带环的工作流,需要进一步处理,例如等效替换等,将带环的工作流抽象成一个任务,具体方案还需进一步研究。

参考文献:

[1] WorkflowManagement Coalition. Workflow management coalition terminology & glossary[EB/OL]. [2010-03-01]. http://www. wfmc. org/standards/docs/TC - 1011 _ term _ glossa-ry_v3. pdf.

[2] FARD H M, PRODAN R, FAHRINGER T. Multi-objective list scheduling of workflow applications in distributed computing infrastructures[J]. Journal of Parallel & Distributed Computing, 2014, 74(3): 2152-2165.

[3] MBOULA J E N, KAMLA V C, DJAMEGNI C T. Cost-time trade-off efficient workflow scheduling in cloud[J]. Simulation Modelling Practice and Theory, 2020, 103: 102107.

[4] SHEN M, TZENG G H, LIU D R. Multi-criteria task assignment in workflow management systems[C]//Proceedings of the 36th annual Hawaii international conference on system sciences. Big Island, HI, USA: IEEE, 2003.

[5] 胡海洋,姬朝配,胡　华,等. 基于协作相容性的工作流任务分配优化方法[J]. 计算机研究与发展,2017,54(4):872-885.

[6] 姜劲松,杨　波,缪志敏,等. 基于任务和用户属性的工作流任务分配算法[J]. 计算机仿真,2015,32(12):222-225.

[7] SATHISH K, REDDY R M. Workflow scheduling in grid computing environment using a hybrid GAACO approach[J]. Journal of the Institution of Engineers (India): Series B, 2006, 98(1): 121-128.

[8] 孙　婷,肖创柏,张雅琴,等. 基于关键路径前瞻的工作流调度算法[J]. 北京工业大学学报,2018,44(8):1136-1144.

[9] 刘雨潇,王　毅,袁　磊,等. 基于期限约束与关键路径的云工作流调度[J]. 计算机工程,2018,44(8):30-37.

[10] 查英华,杨静丽. 改进蚁群算法在云计算任务调度中的应用[J]. 计算机工程与设计,2013,34(5):1716-1719.

[11] 刁兴春,刘　艺,曹建军,等. 多目标蚁群优化研究综述[J]. 计算机科学,2017,44(10):7-13.

[12] 何长杰,白治江. 云环境下基于改进蚁群算法的任务调度[J]. 计算机技术与发展,2018,28(12):13-16.

[13] 彭武良,王成恩. 一种求解资源受限项目调度问题的蚁群算法[J]. 系统仿真学报,2009,21(7):1974-1978.

[14] 文一凭,刘建勋,陈志刚. 面向实例方面处理的工作流动态调度优化方法[J]. 软件学报,2015,26(3):574-583.

[15] 吕　龙,胡海洋,李忠金,等. 基于蚁群算法的工作流系统优化任务分配[J]. 计算机集成制造系统,2018,24(7):1723-1735.