基于分层优化的资源受限系统任务拆分调度方法

韦智勇[1] 杨晓武[2]

[1](南宁职业技术学院 广西 南宁 530008)

[2](中南大学机电工程学院 湖南 长沙 410083)

摘 要 为挖掘资源受限系统的服务潜力,提出一种考虑任务拆分执行方式的调度方法。分析资源调配中所需遵循的约束条件,并以任务满足率为优化目标建立规划模型。设计问题求解框架,提出多层优化算法结构。其中,改进粒子群算法被应用于决策层问题求解,可通过种群进化实现对问题解空间的快速搜索。将冲突规避策略和任务拆分规则应用于逻辑层,能够根据资源稀缺程度进行任务切割,并根据需求分布挑选低重叠度时段进行分配。在仿真实验中,对该方法进行组合测试,分析结果验证了该方法的有效性。

关键词 资源受限系统 调度模型 任务拆分 粒子群

中图分类号 TP312 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2019.06.048

TASK SPLITTING SCHEDULING METHOD OF RESOURCE-CONSTRAINED SYSTEM BASED ON HIERARCHICAL OPTIMIZATION

Wei Zhiyong[1] Yang Xiaowu[2]

[1](*Nanning College for Vocational Technology*, *Nanning 530008*, *Guangxi*, *China*)

[2](*School of Mechanical and Electrical Engineering*, *Central South University*, *Changsha 410083*, *Hunan*, *China*)

Abstract In order to tap the service potential of resource-constrained system, we proposed a scheduling method considering task splitting execution. We analyzed the constraints to be followed in resource allocation, and established a programming model with the objective of optimizing the task satisfaction rate. A problem solving framework was designed and a multi-layer optimization algorithm structure was proposed. The improved particle swarm optimization (PSO) was applied to solve the decision-making layer problem, and the fast search of the solution space could be realized by population evolution. Conflict avoidance strategies and task splitting rules were applied to the logic layer, which could cut tasks according to the degree of resource scarcity, and selected low overlap periods for scheduling tasks according to the demand distribution. In the simulation experiment, the proposed algorithm is tested in combination, and the analysis results verify the effectiveness of the proposed method.

Keywords Resource-constrained system Scheduling model Task splitting Particle swarm

0 引 言

当前,资源受限系统[1-3]调度问题广泛存在于企业生产和社会实践中,对其研究的重要性和必要性不言而喻。一般而言,资源受限系统调度问题是以若干任务需求为处置对象,以消耗有限的服务资源为代价,在满足一定使用规则前提下,寻找能够促使特定指标得以优化的可行方案[4-5]。通常情况下,资源受限系统调度问题属于 NP-hard 问题,具有明显的复杂性特征。许多学者致力于为其寻找高效求解算法,通过调整任务编排方案以最大化资源使用效益。在此过程中,启发式算法和群智能算法应用普遍。

近年来针对资源受限系统调度问题的研究较多,

收稿日期:2018-10-16。广西高等学校千名中青年骨干教师培育计划资助项目;广西科学研究与技术开发计划-重点研发计划项目(桂科 AB18126023)。**韦智勇**,高工,主研领域:任务规划与调度,数据分析与处理 。**杨晓武**,讲师。

例如,张忆文等[6]以并行处理器的节能调度策略为研究背景,在考虑任务执行时间与 CPU 处理速率为非线性关系的情况下,建立任务调度模型,并结合动态电压调节技术和功耗管理技术设计出低能耗资源调配算法。何杰光等[7]针对资源受限系统调度问题提出一种动态多样性群智能算法,主要思路是采用两点交叉算子和变异算子进化个体,基于精英保留策略优化种群,算法能够在全局搜索和局部优化之间进行平衡。GABREL 等[8-9]对成像卫星观测任务调度问题开展研究,通过抽象约束条件建立出任务规划模型,并采用加权有向无环图方法进行求解。Arnaout 等[10]分析了无联系并行机的调度策略,将蚁群算法 ACO(Ant colony optimization)应用于问题求解,通过两阶段问题转换,获取最短任务完成时长的优化方案。在上述研究中,共性的基本假设之一是任务不可拆分,即每个任务若可被安排执行,只能一次完成,执行过程不能停顿。但是实际应用中,对长耗时任务进行切割以提升满足概率是被允许的。此外,任务切割还可以更好地利用调度过程中产生的碎片化资源,提升资源利用率。

本文针对任务可拆分条件下的资源调度方法开展研究,探讨采用何种切割方式提升任务满足概率。为保证调度效果,采用改进粒子群算法 IPSO(Improve particle swarm optimization)[11-13]进行全局搜索。为降低资源使用的冲突程度,在执行时段优选时引入冲突规避策略 CAS(Confliction avoid strategy)。文中给出了冲突集的概念,构建了冲突度模型。通过对比测试,对文中所提出方法的有效性进行了验证。

1　问题分析

1.1　调度过程描述

对于一个完备的资源调度系统[14-15],应至少包含若干服务节点(服务资源)和任务需求(服务对象),调度过程可从以下角度进行描述:

(1) 从服务资源的角度描述。在初始时刻,各服务节点均为空闲状态。调度开始后,规划器实时获取各服务节点的繁忙状态,根据一定规则挑选任务指派给服务节点。获取任务的服务资源被占用一定时长,完成服务后再转入空闲状态。

(2) 从服务对象的角度描述。在调度开始前,不同用户提交的资源使用申请汇聚形成待调度任务队列。可满足的任务汇入等待服务任务队列,到达预定服务时限后转入服务队列接收处置,服务完毕后再转入完成服务任务队列。在此过程中,可对部分任务进行拆分,通过化整为零方式执行,提升任务满足可能性。

结合实际应用背景,资源调度过程通常追寻特定优化目标。从服务资源的角度考量,资源利用率最大化是追求的重要指标。从服务对象的角度考量,任务满足率一般作为评价标准,这两项指标存在关联性,即高任务满足率往往对应着高资源利用率。但对于资源受限系统而言,通常存在服务资源稀缺的情况,即系统的服务能力仅能满足部分而非全部服务需求,这就要求设计合理的资源调配方法,使有限的服务资源得到高效利用。采用任务拆分方式,能够有效利用破碎化的资源时段,促使原本不可满足的任务得以执行。在此过程中,如何选择拆分对象,以何种粒度拆分是必须解决的问题。

1.2　服务资源描述

考虑资源受限系统服务节点异构,具体表现为状态准备时间、任务处置速率、状态恢复时间存在差异。采用五元组对服务节点进行形式化描述,表示为:

$$NS = <Node, TW, SP, TP, TR> \tag{1}$$

$Node = \{node_1, node_2, \cdots, node_n\}$:服务资源集合,$node_i$为所有资源中的第 i 个服务节点,n 为服务节点的数量;

$TW = \{tw_1, tw_2, \cdots, tw_n\}$:可用时间窗口集合,$tw_i = [twb_i, twe_i]$为 $node_i$的可用时段,twb_i和 twe_i分别为 tw_i的起始时间和结束时间;

$SP = \{sp_1, sp_2, \cdots, sp_n\}$:任务处理速率集合,$sp_i$为 $node_i$的任务处置速率;

$TP = \{tp_1, tp_2, \cdots, tp_n\}$:状态准备时间集合,$tp_i$为 $node_i$开始任务前所需的状态准备时间;

$TR = \{tr_1, tr_2, \cdots, tr_n\}$:状态恢复时间集合,$tr_i$为 $node_i$完成任务后,再开展下一次任务前所需的状态恢复时间。

1.3　任务需求描述

对用户提交各种请求进行规范化处理,形成标准化待任务需求。本文考虑的任务需求均为相互独立的无关联任务,采用五元组对服务节点进行形式化描述,表示为:

$$TS = <Task, TA, TD, DL, PD> \tag{2}$$

$Task = \{task_1, task_2, \cdots, task_m\}$:任务需求集合,$task_i$为第 i 个任务,m 为任务数量;

$TA = \{ta_1, ta_2, \cdots, ta_m\}$:任务允许执行最早时间集合,$ta_i$表示 $task_i$允许被执行的最早时刻;

$TD = \{td_1, td_2, \cdots, td_m\}$:任务执行截止期集合,$td_i$表示 $task_i$的执行截止期;

$DL=\{dl_1,dl_2,\cdots,dl_m\}$:任务数据量集合,$dl_i$表示$task_i$的指令长度;

$P=\{p_1,p_2,\cdots,p_m\}$:任务优先级集合,p_i表示$task_i$的优先级系数。

2 模型构建

任务规划的结果是完成服务资源与任务需求匹配,形成合理的规划方案,而构建资源调度模型是实现该目标的前提条件,其主要由决策变量集,优化目标集,约束条件集等要素构成。为便于后续描述,表1给出部分符号说明。

表1 部分符号说明

符号	定义
$stask_{i,j}$	$task_i$分割后形成的第j个子任务
dn_i	$task_i$分割后形成的子任务数量
$tsb_{i,j}$	$stask_{i,j}$的有效服务开始时间
$tse_{i,j}$	$stask_{i,j}$的有效服务结束时间
$tvb_{i,j}$	$stask_{i,j}$的资源占用开始时间
$tve_{i,j}$	$stask_{i,j}$的资源占用结束时间
$tsl_{i,j}$	$stask_{i,j}$的有效服务时间长度
$tvl_{i,j}$	$stask_{i,j}$的资源占用时间长度

决策变量以三元组$<X,Y,ETW>$描述,其中,决策变量$X=[x_{i,j}^k]$用于标识任务指派方案,明确任务与资源的映射关系,赋值方法如下:

$$\begin{cases} x_{i,j}^k=1 & stask_{i,j}\text{ is allocated to }node_k \\ x_{i,j}^k=0 & \text{otherwise} \end{cases} \tag{3}$$

式中:$x_{i,j}^k=1$表示$stask_{i,j}$被分配给$node_k$执行;$x_{i,j}^k=0$表示$stask_{i,j}$未分配给$node_k$执行。

决策变量$Y=[y_i]$用于标识任务的可执行性,明确任务是否分配资源,赋值方法如下:

$$\begin{cases} y_i=1 & \sum_{j=1}^{dn_i}\sum_{k=1}^{n}x_{i,j}^k\geqslant 1 \\ y_i=0 & \text{otherwise} \end{cases} \tag{4}$$

式中:$y_i=1$表示$task_i$被分配资源(安排执行);$y_i=0$表示$task_i$未被分配资源(拒绝执行)。

决策变量$ETW=[<tvb_{i,j},tve_{i,j}>]$用于标识任务的有效服务时段。对于$stask_{i,j}$,其资源占用时段与有效服务时段存在以下关系:

$$\begin{cases} tsb_{i,j}=tvb_{i,j}-tp_k \\ tse_{i,j}=tve_{i,j}+tc_k \end{cases} \tag{5}$$

在筹划任务规划方案阶段,所需遵循的主要约束条件如下:

约束1:如果任务被安排执行($y_i=1$),对其累计有效服务时长不低于所需最少保障时间。

$$\sum_{j=1}^{dn_i}\sum_{k=1}^{n}tvl_{i,j}^k sp_k\geqslant dl_i \tag{6}$$

约束2:任务的执行时段应处于其允许执行窗口中,占用资源时段应处于其所分配服务节点的可用时间窗口内。

$$\begin{cases} [tvb_{i,j}^k,tve_{i,j}^k]\subseteq[ta_i,td_i] \\ [tsb_{i,j}^k,tse_{i,j}^k]\subseteq tw_i \end{cases} \tag{7}$$

约束3:若任务被分解执行,其子任务不能被并行处理。

$$[tsb_{i,j}^k,tse_{i,j}^k]\cap[tsb_{i,j'}^k,tse_{i,j'}^k]=\varnothing\quad j\neq j' \tag{8}$$

约束4:资源具有独占性,即每个服务节点同一时刻最多处理一个任务(子任务)。

$$[tsb_{i,j}^k,tse_{i,j}^k]\cap[tsb_{i',j'}^k,tse_{i',j'}^k]=\varnothing\quad i\neq i',j\neq j' \tag{9}$$

约束5:若安排任务执行($y_i=1$),其子任务应当全部得到满足。

$$\sum_{j=1}^{dn_i}\sum_{k=1}^{n}x_{i,j}^k=dn_i \tag{10}$$

本文考虑以任务满足率作为主要优化目标,以资源利用率为次要优化目标,表示为:

$$\begin{cases} GT(X)=\max\limits_{X\in Q_1}\left\{\dfrac{\sum_{i=1}^{m}\sum_{j=1}^{dn_i}\sum_{k=1}^{n}x_{i,j}^k tvl_{i,j}}{\sum_{i=1}^{m}(twe_i-twb_i)}\right\} \\ GR(Y)=\min\limits_{Y\in Q_2}\left\{\dfrac{\sum_{i=1}^{m}y_i p_i}{\sum_{i=1}^{m}p_i}\right\} \end{cases} \tag{11}$$

式中:Q_1、Q_2为X和Y的可行域;$GT(X)$为可用服务资源的利用率;$GR(Y)$为考虑优先级后的加权任务满足率。

3 算法设计

3.1 基本结构

本文在算法设计中引入了分层优化的算法结构,这是基于两点考虑:一是本文研究的问题既要考虑传统的任务资源匹配问题,还要考虑任务切分问题,所考虑的约束存在关联性,这增加了问题求解难度。采用分层优化的方法可以化繁为简,分而治之,更容易工程实现。二是从求解效率上来看,任务资源匹配问题和任务切分问题存在强耦合关系,若采用整体优化方式,搜索解空间较大。采用分层优化方式可逐层消减解空间规模,有利于有化解的快速产生。整体算法结构包

括主控层、决策层和逻辑层,如图 1 所示。

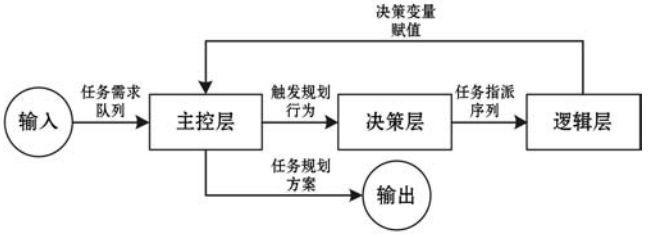


图 1　求解算法的主要构成

主控层负责掌控算法状态,控制迭代过程;决策层用于选择搜索方向,产生任务指派顺序;逻辑层负责挑选拆分任务,解算有效服务时段,具体生成任务规划方案。算法的基本流程如下:

Step 1　主控算法接收任务需求队列,则触发规划活动;

Step 2　决策层采用 IPSO 算法优化任务指派序列,并将结果传递给逻辑层;

Step 3　逻辑层完成约束消解,对决策变量 X、Y、ETW 赋值,并传递给主控层;

Step 4　主控层评估规划结果,若决定继续迭代,转入 Step 2;否则,直接输出规划方案,算法结束。

3.2　IPSO 算法

经典 PSO 算法的基本思想是通过驱动粒子以位移的方式搜索问题解空间,而个体根据种群间的交互信息确定移动方向和速度。其中,每个粒子均表示决策层的一个方案,种群挑选个体最佳位置作为群体最优位置。

3.2.1　粒子编码规则

设种群由 SN 个粒子构成,其中 par_i 为种群中的第 i 个粒子,其在第 t 次进化时的位置向量为:

$$Pos_i^t=(pos_{i,1}^t,pos_{i,2}^t,\cdots,pos_{i,m}^t) \tag{12}$$

式中:$pos_{i,j}^t\in[0,1]$ 为 Pos_i^t 的第 j 个位置分量,其数值大小用于表征任务的优先调度顺序。

例如,随机生成一个粒子 par_i,其位置向量及表示的任务指派序列为:

Code: $Pos_i^t=(0.27,0.04,0.09,0.82,0.69)$

Decode: $Plan_i^t=(task_2,task_3,task_1,task_5,task_4)$ (13)

3.2.2　位置更新方式

首先给出种群的适应度函数,用于评估粒子所在位置的优劣,如下所示:

$$f=\frac{1}{1+\exp[-k_1GT(X)-k_2GR(Y)]} \tag{14}$$

在第 t 次进化时,设 $TPos_i^t$ 为 par_i 已获知的历史最优位置,$GPos^t$ 为整个种群已获知的历史最优位置,则第 t 次进化时 par_i 的位置更新为:

$$\begin{cases}V_i^t=wV_i^{t-1}+a_1rand_1(TPos_i^t-Pos_i^{t-1})+\\ \qquad a_2rand_2(GPos_i^t-Pos_i^{t-1})\\ Pos_i^t=Pos_i^{t-1}+V_i^t\end{cases} \tag{15}$$

式中:a_1 和 a_2 为学习因子;$rand_1$ 和 $rand_2$ 为(0,1)区间上的随机数。此外,严格限定 Pos_i^t 的赋值范围,若 $\max(Pos_i^t)>1$ 或 $\min(Pos_i^t)<0$,则令:

$$pos_{i,j}^t=\frac{pos_{i,j}^t-\min\limits_j(pos_{i,j}^t)}{\max\limits_j(pos_{i,j}^t)-\min\limits_j(pos_{i,j}^t)+0.01} \tag{16}$$

3.2.3　基于云模型的变异策略

为避免种群陷入早熟,引入变异操作。首先借鉴云模型产生变异概率,然后根据个体适应度挑选变异对象。

设 Pos_i^t 对应的适应度为 FV_i^t,$TPos_i^t$ 适应度为 TFV_i^t,$GPos^t$ 适应度为 GFV^t。算法中采用变异方法为:

Step 1　构建云模型,用于计算 Pos_i^t 的变异概率 pc_i^t,如下所示:

$$\begin{cases}Ex=\dfrac{1}{m}\sum\limits_{i=1}^{m}FV_i^{t-1}\\ En=rand\times(GFV^{t-1}-Ex)^2+\\ \qquad(1-rand)\times(TFV_i^{t-1}-Ex)^2+0.01\\ pc_i^t=\exp\left(-\dfrac{(FV_i^t-GFV^{t-1})^2}{2En}\right)\end{cases} \tag{16}$$

Step 2　产生随机数 $rand$,若 $rand>pc_i^t$,则确定对 Pos_i^t 进行变异;否则,保持 Pos_i^t 不变;

Step 3　计算 Pos_i^t 与 $GPos^t$ 各位置分量的距离,如下所示:

$$d_{i,j}^t=|pos_{i,j}^t-gpos_j^t|\quad j=1,2,\cdots,m \tag{17}$$

Step 4　对 Pos_i^t 的部分位置分量进行更新:

$$\begin{cases}d_{i,k}^t=\max\limits_j(d_{i,j}^t)\\ pos_{i,k}^t\leftarrow gpos_k^t\end{cases} \tag{18}$$

3.3　CAS 策略

针对决策层传递的任务指派序列,利用 CAS 策略进行解析,生成具体的资源调配方案。在此过程中,需根据任务状态构建若干集合:

WSTSet:等待规划任务集,用于存储尚未参与调度的(子)任务队列;

WETSet:等待执行任务集,用于存储已分配资源但未执行的(子)任务队列;

RETSet:无法满足任务集,用于储存已参与调度但未满足的(子)任务队列;

FETSet:完成执行任务集,用于储存执行完毕的(子)任务队列。

在上述集合中，*WSTSet* 是规划系统的处置对象，可满足的任务移至 *WETSet* 中，无法满足的任务移至 *RETSet* 中，而 *FETSet* 中的元素对对规划过程无影响。

定义 1 对于 $task_i \in WSTSet$ 和给定时间窗口 *TS*，称 $task_i$ 在 *TS* 中具有需求冲突，记为 $Conf(TS, task_i) = 1$，若满足以下关系：

$$TS \cap [ta_i, td_i] \neq \varnothing \tag{19}$$

设 $task_i \in WSTSet$ 在 $node_k$ 上的空闲时间窗口集为 $FTWS_i^k$，需求冲突集为 $HDCS_i^k$。其中，$task_i$ 在 $ftw_{i,j}^k \in FTWS_i^k$ 上需求冲突度记为 $hdc_{i,j}^k$。同时，考虑到任务模板本身需占用一定的资源，若空闲时间窗口长度低于任务模板时长，则该窗口不具备可用性。

定义 2 将长度超过任务模板时长的空闲时间窗口称为有效时间窗口，记为 $VTWS_i^k$，其中 $vtb_{i,j}^k$、$vte_{i,j}^k$ 分别为 $vtw_{i,j}^k \in VTWS_i^k$ 的开始时间和结束时间。

采用 CAS 策略计算 $< VTWS_i^k, HDCS_i^k >$，基本步骤如算法 1 所示。

算法 1 Pseudocode of **CAS**

01: Delete $task_i$ from *WSTSet*;
02: **for** $node_k$ in *Node*
03: $FTWS_i^k \leftarrow [ta_i, td_i]$;
04: **for** $stask_{i',j'}$ in *WETSet*
05: **if** $x_{i',j'}^k = 1$ **then**
06: $FTWS_i^k \leftarrow FTWS_i^k \cap ([tsb_{i',j'}, tse_{i',j'}])^C$;
07: **end if**
08: **end for**
09: **for** $ftw_{i,j}^k$ in $FTWS_i^k$
10: **if** $\text{span}(ftw_{i,j}^k) < tp_k + tr_k$ **then**
11: *Delete* $\text{ftw}_{i,j}^k$ *from* FTWS_i^k;
12: **else**
13: $hdc_{i,j}^k \leftarrow 0$;
14: **for** $task_{i'}$ in *WSTSet*
15: $hdc_{i,j}^k \leftarrow hdc_{i,j}^k + Conf(ftw_{i,j}^k, task_{i'})$;
16: **end for**
17: **end if**
18: **end for**
19: $VTWS_i^k \leftarrow FTWS_i^k$;
20: **end for**
21: **Return** $< VTWS_i^k, HDCS_i^k >$;

决策层传递的任务指派序列存储在 *WSTSet* 中，逻辑层采用 CAS 策略对 $task_i \in WSTSet$ 逐个检测。以 $< VTWS_i^k, HDCS_i^k >$ 的计算结果为输入，基于允许任务切割 PTC(Promise task cutting)规则挑选拆分对象，并为其子任务优选执行资源，具体方法如算法 2 所示。

算法 2 Pseudocode of **PTC**

01: **for** $task_i$ in *WSTSet*
02: Calculate $< VTWS_i^k, HDCS_i^k >_{i=1,2,\cdots,n}$ by 算法 1;
03: Sort $VTW_{i,j}^k$ by $HDCS_{i,j}^k$ in increasing order;
04: $Rem\_len \leftarrow dl_i, dn_i \leftarrow 0, y_i = 0$;
05: **while** $\bigcap_k VTWS_i^k \neq Null \cap \text{Rem\_len} > 0$ **then**
06: Assume $VTW_{i,j'}^{k'}$ is the first member in $\bigcap_k VTWS_i^k$;
07: Let $x_{i,j'}^{k'} = 1$ and delete $VTW_{i,j'}^{k'}$ from $\bigcap_k VTWS_i^k$;
08: Let $dn_i \leftarrow dn_i + 1, tvb_{i,dn_i} \leftarrow vtb_{i,j'}^{k'}, x_{i,j'}^{k'} = 1$;
09: $tve_{i,dn_i} \leftarrow \textbf{min}(vte_{i,j}^{k'}, \text{Rem\_len}/sp_{k'} + tr_{k'} + tp_{k'})$;
10: $tsb_{i,dn_i} = tvb_{i,dn_i} + tr_{k'}, tse_{i,dn_i} = tve_{i,dn_i} + tp_{k'}$;
11: $tsl_{i,dn_i} \leftarrow tse_{i,dn_i} - tsb_{i,dn_i}, tvl_{i,dn_i} \leftarrow tve_{i,dn_i} - tvb_{i,dn_i}$;
12: $Rem\_len \leftarrow Rem\_len - tve_{i,dn_i} + tvb_{i,dn_i}$;
13: **end while**
14: **if** $Rem\_len = 0$ **then**
15: $y_i \leftarrow 1$;
16: **else**
17: $dn_i \leftarrow 0$;
18: $tsl_{i,j} \leftarrow Null, tvl_{i,j} \leftarrow Null, tvb_{i,j} \leftarrow Null, tve_{i,j} \leftarrow Null, tsb_{i,j} \leftarrow Null, tse_{i,j} \leftarrow Null, x_{i,j}^k \leftarrow 0, j = 1, 2, \cdots, dn_i, k = 1, 2, \cdots, m$;
19: **end if**
20: Let $X = [x_{i,j}^k], Y = [y_i], ETW = [< tvb_{i,j}, tve_{i,j} >]$;
21: **Return** $< X, Y, ETW >$;

4 实验分析

本节中模拟生成任务规划场景，对文中所提方法的有效性进行对比分析。

4.1 测试场景设置

批量生成场景数据，调用不同算法进行实验，以统计指标的平均值作为测试结果，场景要素的设置方法如下：

1）整个任务规划活动的周期跨度为 24 h；

2）设置 3 个服务节点，服务能力指标根据参数设置情况随机生成，如表 2 所示。

表 2 服务资源的设置情况

服务节点名称	任务处理速率/Mbps	可用时间窗口累计时长/h	状态准备时间/min	状态恢复时间/min
$node_1$	100	22 ~ 24	5 ~ 10	2 ~ 3
$node_2$	150	20 ~ 22	8 ~ 13	3 ~ 4
$node_3$	200	18 ~ 20	10 ~ 15	4 ~ 5

3）任务数量为 100 ~ 300 个，任务数据量包括 150 ~ 250 min、250 ~ 350 min、350 ~ 450 min 三种类型，任务允许执行最早时间和截止期在规划活动周期内随机生成。

在求解方法设计时，同时嵌入了启发式算法和智

能优化算法。其中,启发式算法用于快速提升解的质量,本文主要测试其有效性。IPSO 算法单次运行时间较长,理论上而言,足够长的迭代次数均能获取较优的可行解,因此本文更关注其搜索效率。采用分步验证方法进行实验,测试内容主要包括以下两项:

(1) 算法的搜索质量分析。主要是 CAS 策略和 PTC 规则的有效性。决策层固定使用 IPSO 算法,逻辑层引入 Greedy 策略和禁止任务切割 FTC(Forbid task cutting)规则,与之前所提算法进行对比。主控层算法根据对比算法的组合方式命名,如表 3 所示。

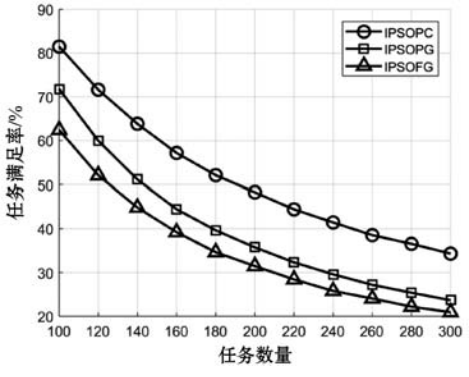
表 3　测试算法的组合方式

主控层	决策层	逻辑层	
		时段优选策略	任务拆分规则
IPSOFG	IPSO	Greedy	FTC
IPSOPG	IPSO	Greedy	PTC
IPSOPC	IPSO	CAS	PTC

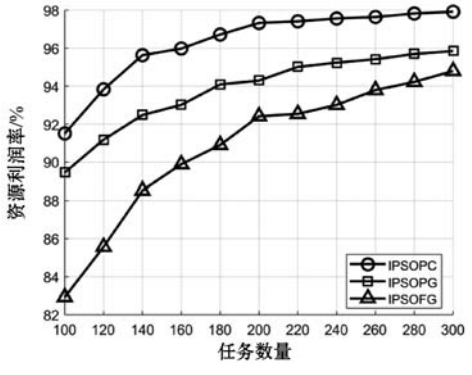
(2) 算法的搜索速率分析。主要是加入 CAS 策略和 PTC 规则后对算法运行时间的增量,以及 IPSO 算法的收敛速率。

4.2　算法的搜索质量分析

在不同任务强度下对比三种组合算法的实验结果,统计指标包括任务满足率和资源利用率,如图 2 所示。



(a) 任务满足率



(b) 资源使用率

图 2　不同任务规模下的调度结果对比

从图 2 可以看出,随着任务强度的增加,任务满足率迅速下降,而资源使用率在快速增长后趋近平稳。系统资源的服务能力经历了由充足到饱和的过程,说明所选测试场景能够较为全面地呈现出多种供需状态,具有一定的代表性。

根据测试结果,在不同测试场景下,IPSOPG 的求解结果均优于 IPSOFG,这是由于 PTC 规则通过任务切割能够使原本无法满足的任务得到执行,由此说明该策略的有效性;IPSOPC 的求解结果均优于 IPSOPG,这是优于在具体指派资源时,CAS 策略能够预见性的规避需求冲突,最大程度地保留后续任务的满足机会,因而求解效果更好。综上可以得出结论,CAS 策略和 PTC 规则对于提升算法搜索质量是有效的。

4.3　算法搜索速率分析

在不同任务强度下测试算法的运算时间,统计结果如图 3 所示。

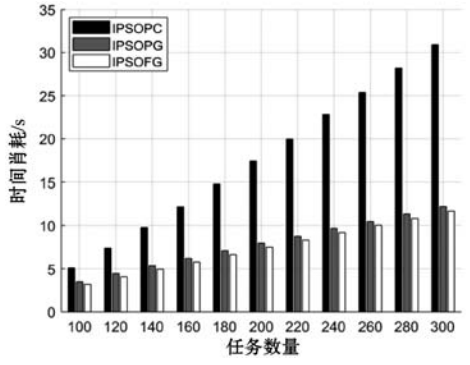


图 3　不同任务规模下的算法耗时对比

从图 3 可以看出,以 IPSOPC 算法的耗时最高,IPSOFG 与 IPSOPG 算法的耗时接近。由此可以说明,采用 FTC 规则对于算法造成的负担不显著,而引入 CAS 策略会明显增加算法的时间复杂度。为进一步分析 CAS 策略和 FTC 规则对算法搜索速率的影响,以 IPSOFG 算法为基准,分析其他两种算法的相对搜索时间增量,如图 4 所示。

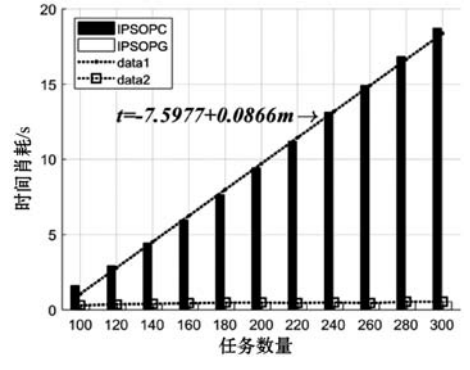


图 4　相对搜索时间增量的变化趋势

在图 4 中,随任务强度的增加,IPSOFG 算法的搜索时间增量稳定处于较低水平,说明 FTC 规则对于算法搜索速率的影响较弱。IPSOPC 算法搜索时间的增

量呈线性变化,本文采用最小二乘法进行拟合,得到的近似关系式为:

$$t = -7.597\ 7 + 0.086\ 6m \tag{20}$$

由此可以说明,CAS 策略虽会延缓算法的搜索速率,但该策略不会导致算法的搜索时间呈爆发式增长,其时间复杂度在可接收范畴。

IPSOPC 为本文提出的主要算法,对算法中的种群适应度演变过程进行统计,以分析算法的搜索效率,如图 5 所示。

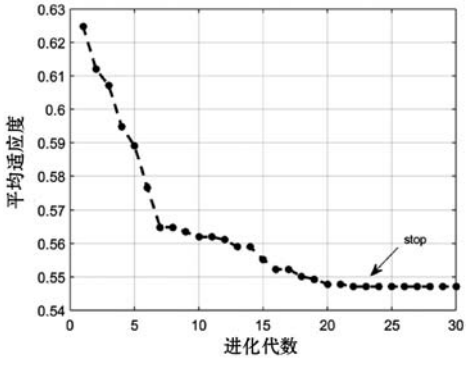


图 5 种群适应度的演变过程

从图 5 可以看出,种群在第 1 ~ 7 代进化明显,在第 7 ~ 23 代进化平稳,在第 23 代后逐渐停止进化。说明 IPSO 算法在迭代早期能够快速搜索解空间,获取问题的初始优化解集。在迭代中期,能够不断提升种群适应度,持续改进优化解的质量。在迭代末期,能够保持种群的稳定,对优化解集进行收敛。基于上述过程,IPSO 算法基本实现了在确保收敛前提下避免过早陷入局部最优的目标,能够以较为高效的搜索速率获得理想结果。

5 结 语

本文针对任务可拆分条件下的资源受限系统调度问题开展研究,以最大化任务满足率和资源利用率为优化目标,建立了约束模型。为降低求解复杂度,在算法设计时,构建了多层求解结构,并分别提出了优化算法。其中,IPSO 算法应用于决策层,该算法是在经典 PSO 算法的基础上加入了部分改进策略,能够避免种群陷入早熟。CAS 策略和 PTC 规则被应用于逻辑层,用于服务资源的指派和任务执行时段的明确。在实验部分,通过测试对文中所提方法的有效性进行了验证。需要指出的是,本文所提算法仍有很大改进空间,后续将根据项目背景作进一步研究。

参 考 文 献

[1] Vanhoucke M. Debels D. The impact of various activity assumptions on the lead time and resource utilization of resource-constrained projects[J]. Computers and Industrial Engineering, 2008, 54(1): 140 - 154.

[2] Vincent V P, Mario V. A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem[J]. European Journal of Operational Research, 2010, 201(2): 409 - 418.

[3] Subramanian A, Garcia M J, Callaway D S, et al. Real-time scheduling of distributed resources[J]. IEEE Transactions on Smart Grid, 2013, 4(4): 2122 - 2130.

[4] Shou Y, Li Y, Lai C. Hybrid particle swarm optimization for preemptive resource-constrained project scheduling[J]. Neurocomputing, 2015, 148(19): 122 - 128.

[5] Kolisch R, Hartmann S. Experimental investigation of heuristics for resource-constrained project scheduling: An update[J]. European J of operational Research, 2006, 174(1): 23 - 37.

[6] 张忆文,王成,郭锐锋. 资源受限周期任务低能耗调度算法[J]. 小型微型计算机系统,2017,38(5):1076 - 1080.

[7] 何杰光, 陈新度, 陈新, 等. 求解资源受限项目调度的动态多样性进化策略[J]. 计算机集成制造系统, 2015, 21(8): 2089 - 2098.

[8] Gabrel V, Vanderpooten D. Enumeration and interactive selection of efficient paths in a multiple criteria graph for scheduling an earth observing satellite[J]. European Journal of Operational Research, 2002, 139(3): 533 - 542.

[9] Gabrel V, Moulet A, Murat C, et al. A new single model and derived algorithms for the satellite shot planning problem using graph theory concepts[J]. Annals of Operations Research, 1997, 69(1): 115 - 134.

[10] Arnaout J P, Rabadi G, Musa R. A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times[J]. Journal of Intelligent Manufacturing, 2010, 21(6): 693 - 701.

[11] Liang J J, Qin A K, Suganthan P N, et al. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions[J]. IEEE Transactions on Evolutionary Computation, 2006, 10(3): 281 - 295.

[12] Zhan Z H, Zhang J, Li Y, et al. Orthogonal learning particle swarm optimization[J]. IEEE Transactions on Evolutionary Computation, 2011, 15(6): 832 - 847.

[13] Leu M S, Yeh M F, Wang S C. Particle swarm optimization with grey evolutionary analysis[J]. Applied Soft Computing, 2013, 13(10): 4047 - 4062.

[14] 方晨,王凌. 资源约束项目调度研究综述[J]. 控制与决策, 2010, 25(5): 641 - 650.

[15] Hartmann S. A self-adapting genetic algorithm for project scheduling under resource constraints[J]. Naval Research Logistics, 2005, 49(5): 433 - 448.