DOI：10.7652/xjtuxb201709017

考虑处理机时间窗口的可分任务调度优化模型

赖俊凡，王宇平，王晓丽
（西安电子科技大学计算机学院，710071，西安）

摘要：针对异构分布式系统下处理机具有时间窗口约束的可分任务调度问题，通过寻找最优的任务分配方案和最优的处理机调度顺序，可以使得任务的完成时间最短。首先，在已有模型上引入处理机时间窗口的概念，使得所建模型更加贴切实际；然后，建立了一个新的考虑处理机时间窗口可分任务调度的非阻塞优化模型，同时设计了一种基于全局优化的遗传算法来求解模型；最后，为了快速、高效地求解模型，所提算法同时对处理任务量和调度顺序进行编码，利用不同的交叉算子来优化调度顺序和任务分配量，设计了合理的修正算子来修正不满足处理机时间窗口的任务分配方案，并且设计了高效的局部搜索算子来加快算法的收敛速度。仿真实验结果表明，在处理机时间窗口约束下，与已有算法相比，所提算法至少提升了20%以上的性能，从而证明了所提算法的正确性和有效性。

关键词：处理机；可分任务调度；时间窗口；遗传算法
中图分类号：TP18 **文献标志码**：A **文章编号**：0253-987X(2017)09-0118-07

An Optimization Model for Divisible-Load Scheduling Considering Processor Time-Window

LAI Junfan, WANG Yuping, WANG Xiaoli
(School of Computer Science and Technology, Xidian University, Xi'an 710071, China)

Abstract: A divisible-load scheduling problem under the constraint of processor time-window for heterogeneous distributed systems is presented, and the make-span is minimized by finding the optimal load partition strategy and the optimal distribution sequence of processors. On the basis of existing research, the concept of time-window is introduced first, which makes the model more practical. Then a novel divisible-load scheduling non-blocking model is proposed considering the time-window. Meanwhile, a genetic algorithm is designed for the proposed model. In order to solve the model quickly and efficiently, the load partition and the distribution sequence of processors are encoded at the same time. Different crossover operators are designed to optimize the load partition and the distribution sequence of processors, and a modification operator is designed to modify the load partition scheme which does not satisfy the constraint of processors' time-window. Moreover, An efficient local search operator to is also designed speed up the convergence rate of the algorithm. Finally, simulation experiments are carried out, and the results show that under the constraint of the processor time-window, the proposed algorithm can improve the performance of the existing algorithms by at least 20%, which proves the correctness and effectiveness of the proposed algorithm.

收稿日期：2017-01-06。 作者简介：赖俊凡(1992—)，男，硕士生；王宇平(通信作者)，男，教授，博士生导师。 基金项目：国家自然科学基金资助项目(61472297，U1404622)。

Keywords：processor；divisible-load scheduling；time-window；genetic algorithm

可分任务是指任务可以被切分为任意数量的若干子任务且子任务间不具有优先级关系[1]。可分任务理论广泛应用于图像与视频处理[2]、数据库检索[3]、基因测序[4]等不同领域。在不同的网络拓扑结构下，基于可分任务调度理论的模型及算法得到了广泛研究，包括总线型网络、树型网络、高斯网格、二进制树形网络和云计算系统[5-9]。文献[10]建立了一种星形网络调度模型，在求得模型紧式最优解的同时，证明了当处理机调度顺序遵循通信链路传输速率递减顺序时，可得到最短任务完成时间。文献[11-12]在星型网络中研究了处理机具有任意释放时间的调度问题，建立了考虑处理机释放时间的调度模型，并且通过算法求得最优解，但是在模型中并未考虑到处理过程必然存在的启动开销和任务调度顺序对任务完成时间的影响，模型不是很精确。文献[13]对异构环境下的任务调度问题进行了研究，有效解决了处理机数目选择、任务划分和调度顺序 3 个问题，但是模型较为简单，实际应用意义不大。文献[14]针对异构环境下处理机具有任意释放时间的调度问题建立了一种新可分任务调度模型，但是该模型中只考虑了释放时间，并没有考虑到实际中处理机下线时间和调度顺序对任务完成时间的影响。文献[15]在文献[14]的基础上加入了处理机调度顺序，但模型仍然不够精确。

综上所述，已有的可分调度模型大多假设所有处理机在任务分配前后都一直保持在线状态，且处理机不能同时传输和计算任务。现实中的处理机大多采用非阻塞模式，可同时传输和处理任务，且在给处理机分配任务前后都有不同的限制，处理机具有不同的释放时间和下线时间。因此，本文定义每台处理机释放时间到下线时间之间的可用时间段为处理机的时间窗口，提出了一种新的考虑处理机时间窗口可分任务调度非阻塞模型，同时设计了一种基于遗传算法的可分任务调度策略，用以寻找最优解。

1　考虑时间窗口的可分任务调度优化模型

1.1　问题描述

异构星型网络的拓扑结构如图 1 所示。图 1 中 $N+1$ 台处理机通过通信链路互连，P_0 为主处理机，$\{P_i \mid i \in \{1,2,\cdots,N\}\}$ 为从处理机集合，L_i 为连接 P_0、P_i 的通信链路。P_0 并不参与计算，仅负责将任务分配给从处理机，首先将归一化总任务 W_{total} 划分为 N 个子任务 $A=\{\alpha_1,\alpha_2,\cdots,\alpha_N\}$，其中 $0 \leqslant \alpha_i \leqslant W_{\text{total}}$，$\sum_{i=1}^{n}\alpha_i = W_{\text{total}}$。$P_0$ 按照传输顺序将子任务 $\alpha_1,\alpha_2,\cdots,\alpha_N$ 依次传输给 $P_{\sigma_1},P_{\sigma_2},\cdots,P_{\sigma_N}$，从而完成计算，其中 $\sigma=(\sigma_1,\sigma_2,\cdots,\sigma_N)$ 为 $1\sim N$ 的一个全排列，子任务 $\alpha_i(i=1,2,\cdots,N)$ 传输给处理机 P_{σ_i} 去处理。P_0 在同一时刻只能给一个处理机传输任务，从处理机可以同时接收和计算任务，并不要求所有处理机都参与计算。假设参与计算的处理机数目为 n，则只需在 $P_{\sigma_1},P_{\sigma_2},\cdots,P_{\sigma_N}$ 中的前 n 个处理机 $P_{\sigma_1},P_{\sigma_2},\cdots,P_{\sigma_n}$ 分配任务 $\alpha_i>0,i=1,2,\cdots,n$，其余处理机不分配任务，即 $\alpha_i=0,i=n+1,n+2,\cdots,N$。

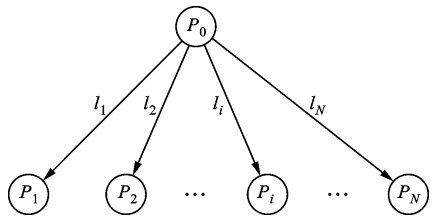

图 1　星型网络的拓扑结构

在异构系统中，有 $\forall i\neq j, w_i\neq w_j$ 且 $g_i\neq g_j$，其中 w_i 为处理机 P_i 与标准处理机计算相同任务的时间之比，g_i 为通信链路 L_i 与标准链路传输相同任务的时间之比。由于考虑了启动开销，所以子任务 α_i 的传输、计算时间分别为 $e_{\sigma_i}+g_{\sigma_i}\alpha_i$、$f_{\sigma_i}+w_{\sigma_i}\alpha_i$，其中 e_{σ_i}、f_{σ_i} 为传输启动开销、计算启动开销。

记处理机 P_i 的释放时间、下线时间分别为 r_i、o_i，其中 $i=1,2,\cdots,N$。图 2 给出了异构星型网络下考虑处理机时间窗口的一种可能的时序调度图，P_0 在 $t=r_{\sigma_1}$ 时开始将子任务 α_1 传输给 P_{σ_1}，记处理机 P_{σ_i} 开始接收任务的时刻为 s_i，所以对于 P_{σ_1} 有 $s_1=r_{\sigma_1}$。P_{σ_i} 的开始时间 s_i 不仅与其释放时间 r_{σ_i} 有关，而且与前一个处理机 $P_{\sigma_{i-1}}$ 的开始时间 s_{i-1} 以及 $P_{\sigma_{i-1}}$ 接收子任务 α_{i-1} 的传输时间 $e_{\sigma_{i-1}}+g_{\sigma_{i-1}}\alpha_{i-1}$ 有关。由图 2 可知 $s_i=\max\{r_{\sigma_i},s_{i-1}+e_{\sigma_{i-1}}+g_{\sigma_{i-1}}\alpha_{i-1}\}$，其中 $i=2,3,\cdots,n$。

给定任务调度顺序 $(P_{\sigma_1},P_{\sigma_2},\cdots,P_{\sigma_N})$ 和任务分配方案 $A=\{\alpha_1,\alpha_2,\cdots,\alpha_N\}$，通过 $n=\text{card}(\{\alpha_i \mid \alpha_i\in A,\alpha_i>0\})$ 计算得到参与计算的处理机数目 n，其中 $\text{card}(X)$ 为计算集合 X 中元素个数的函数。参与计算的 n 个处理机中第 i 个处理机 P_{σ_i} 的任务完成时间 $T_i=s_i+e_{\sigma_i}+f_{\sigma_i}+w_{\sigma_i}\alpha_i,i=1,2,\cdots,n$，可得总任务的完成时间 $T=\max\limits_{1\leqslant i\leqslant n}\{s_i+e_{\sigma_i}+f_{\sigma_i}+w_{\sigma_i}\alpha_i\}$。

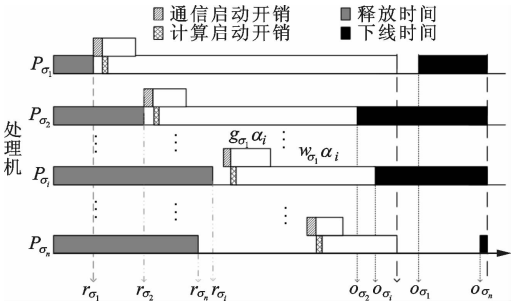

图 2 一种可能的时序调度图

1.2 新的可分任务调度模型

基于前述问题，本文建立了新的考虑处理机时间窗口的可分任务调度模型

$$\min_{A,\sigma} T = \min_{A,\sigma}\{\max_{1\leqslant i\leqslant n}\{s_i + e_{\sigma_i} + f_{\sigma_i} + w_{\sigma_i}\alpha_i\}\}$$

其中 $0\leqslant\alpha_i\leqslant W_{\text{total}}$，$\sum_{i=1}^{n}\alpha_i = W_{\text{total}}$，$i=1,2,\cdots,n$；$\sigma=(\sigma_1,\sigma_2,\cdots,\sigma_N)$，$\sigma_i\in\{1,2,\cdots,N\}$，且 $\forall j,k\in\{1,2,\cdots,N\}$，若 $j\neq k$，则 $\sigma_j\neq\sigma_k$；$s_i+e_{\sigma_i}+f_{\sigma_i}+w_{\sigma_i}\alpha_i\leqslant o_{\sigma_i}$，$n=\text{card}(\{\alpha_i\,|\,\alpha_i\in A,\alpha_i>0\})$，$\text{card}(X)$为计算集合 X 中元素个数的函数，$s_i=r_{\sigma_1}$，$s_i=\max\{r_{\sigma_i},s_{i-1}+e_{\sigma_{i-1}}+g_{\sigma_{i-1}}\alpha_{i-1}\}$。

2 基于全局优化的遗传算法

对于变量 $A=\{\alpha_1,\alpha_2,\cdots,\alpha_N\}$、$\sigma=(\sigma_1,\sigma_2,\cdots,\sigma_N)$，本文用遗传算法来求解可分任务调度优化模型。

2.1 编码方案

本文采用混合编码方案，将个体记为 $I=(\sigma,A)$，其中 $\sigma=(\sigma_1,\sigma_2,\cdots,\sigma_N)$为 $1\sim N$ 的一个全排列，用于表示任务调度顺序；$A=\{\alpha_1,\alpha_2,\cdots,\alpha_N\}$表示任务分配方案，其中 $0\leqslant\alpha_i\leqslant W_{\text{total}}$，$\sum_{i=1}^{n}\alpha_i = W_{\text{total}}$，$i=1,2,\cdots,N$，若 $\alpha_i=0$ 表示处理机 P_{σ_i} 不参与任务计算。

2.2 交叉和变异算子

为了同时优化调度顺序和任务分配方案，本文设计了两种交叉算子。两种交叉算子交替执行，可最大程度增加后代个体多样性。

(1)第一种交叉算子同时进化 σ、A。给定两个父代个体 $I^1=(\sigma^1,A^1)$、$I^2=(\sigma^2,A^2)$，其中 σ^1、σ^2 为参与交叉的个体，通过以下步骤生成两个子代个体 $I^3=(\sigma^3,A^3)$、$I^4=(\sigma^4,A^4)$：①由于调度顺序问题是 TSP 问题的变种，考虑到优化调度顺序的同时需要优化任务分配方案，采用顺序交叉方式进化 σ；②将 σ^i 在匹配区外与 σ^{3-i}在匹配区内相同的数去掉，再将匹配区的数移到最左边得 σ^3、σ^4，若 $p=4$，$q=6$，可得

$$\left.\begin{aligned}\sigma^1 &= (9,8,4,\,|\,5,7,6,\,|\,1,3,2)\\ \sigma^2 &= (8,7,1,\,|\,2,3,9,\,|\,5,4,6)\end{aligned}\right\}$$
$$\Rightarrow\left.\begin{aligned}\sigma^3 &= (5,7,6,x,x,x,8,4,1)\\ \sigma^4 &= (2,3,9,x,x,x,8,1,4)\end{aligned}\right\}$$

③再将 σ^i 的匹配区加到 σ^{3-i}的匹配区后，对应的可以得到 σ^3 和 σ^4，即

$$\left.\begin{aligned}\sigma^3 &= (5,7,6,x,x,x,8,4,1)\\ \sigma^4 &= (2,3,9,x,x,x,8,1,4)\end{aligned}\right\}$$
$$\Rightarrow\left.\begin{aligned}\sigma^3 &= (5,7,6,2,3,9,8,4,1)\\ \sigma^4 &= (2,3,9,5,7,6,8,1,4)\end{aligned}\right\}$$

④采用算术交叉方式优化 A，首先随机生成两个整数 p、q 且满足 $1\leqslant p<q\leqslant N$，将 p、q 之间对应的任务量做算术平均，若 $p=4$，$q=6$，可得 $A^1=(0.1,0.2,0.15,|0.2,0.1,0.05,|0.05,0.05,0.1)$，$A^2=(0.05,0.1,0.05,|0.1,0.2,0.15,|0.2,0.1,0.05)$。将 A^1、A^2 交叉区间的任务量，按照算术平均值计算可得 $A^3=(0.1,0.2,0.15,|0.15,0.15,0.1,|0.05,0.05,0.1)$，$A^4=(0.05,0.1,0.05,|0.15,0.15,0.1,|0.2,0.1,0.05)$。从而得到后代个体 I^3、I^4。

(2)第二种交叉算子是在固定 σ 的基础上进化 A。由于任务量采用实数编码方案，不存在编码冲突问题，故采用两点交叉方式优化 A。随机生成两个整数 p、q 且满足 $1\leqslant p<q\leqslant N$，并将其作为交叉点。交换 A^1、A^2 在 p、q 之间的基因，生成 A^3、A^4，若 $p=4$，$q=6$，则 $A^1=(0.1,0.2,0.15,|0.2,0.1,0.05,|0.05,0.05,0.1)$，$A^2=(0.05,0.1,0.05,|0.1,0.2,0.15,|0.2,0.1,0.05)$。将 A^1、A^2 交叉区间的任务量交换可得 $A^3=(0.1,0.2,0.15,|0.1,0.2,0.15,|0.05,0.05,0.1)$，$A^4=(0.05,0.1,0.05,|0.2,0.1,0.05,|0.2,0.1,0.05)$。从而完成任务量的优化。

(3)变异算子是为了增加生成后代的多样性，所以变异算子中分别交换处理机和处理任务量，即随机生成 p、q、l 和 m 这 4 个整数，且 $1\leqslant p<q\leqslant N$、$1\leqslant l<m\leqslant N$。对于父代个体 $I=(\sigma,A)$，分别交换 σ_p、σ_q 位置的处理机编号，A_l、A_m 位置上的任务量生成子代个体。若 $p=4$，$q=6$，$l=2$，$m=7$，则 $\sigma=(9,8,4,5,7,6,1,3,2)$，$A=(0.1,0.2,0.15,0.2,0.1,0.05,0.05,0.1)$。交换 σ_4、σ_6 和交换 A_2、A_7 可得 $\sigma=(9,8,4,6,7,5,1,3,2)$，$A=(0.1,0.05,0.15,0.2,0.1,0.05,0.2,0.1)$。从而变异算子生成后代个体完成。

2.3 修正算子

不论是种群的初始化，还是通过交叉和变异算

子产生的新个体，都不能保证个体满足模型的全部约束条件，尤其是下线时间的约束条件 $s_i+e_{\sigma_i}+f_{\sigma_i}+w_{\sigma_i}\alpha_i\leqslant o_{\sigma_i}$，$i=1,2,\cdots,N$。因此，对于新生成的个体需要检查其是否满足模型的所有约束条件，同时对不满足约束的个体进行修正。若 P_{σ_i} 完成所分配任务量 α_i 的时间超出了其下线时间，则将超出下线时间的部分调整到相邻的下一台处理机 $P_{\sigma_{i+1}}$ 上执行。若第一轮调整后，$P_{\sigma_{i+1}}$ 的完成时间也超出了其下线时间，则继续将 $P_{\sigma_{i+1}}$ 的冲突部分调整到相邻的下一台处理机 $P_{\sigma_{i+2}}$ 上执行，直至冲突完全消除。修正的过程是一个迭代的过程。

一种不满足模型约束的可分任务调度时序图如图 3 所示，P_{σ_2} 的任务完成时间大于其下线时间 o_{σ_2}，这是因为 P_{σ_2} 分配的任务量过大造成了时间冲突。根据修正算子，应将多余的任务量分配给下一个处理机 P_{σ_3}，修正算子的迭代过程示意图如图 4 所示。由图 4 可知，P_{σ_3} 产生了新的时间冲突，即 P_{σ_3} 的任务完成时间超出了其下线时间，根据修正算子，需要计算分配给 P_{σ_3} 的多余任务量，并将其重新分配给 P_{σ_4}，整个调整过程重复进行，直到消除所有冲突为止。随着任务量的增加，任务完成时间也越来越长，超出其下线时间的处理机也越来越多，修正算子的迭代次数也越来越多。对一个不满足约束的个体进行修正，迭代算子的迭代次数最小是 1，即修正一次后即满足约束。最坏的情况则是 $n-1$，其中 n 为参与计算的处理机数量，即从第一个处理机开始就不满足约束，修正后的下一个也不满足约束，以此类推，一共有 $n-1$ 个处理机不满足约束，则一共需要修正 $n-1$ 次。

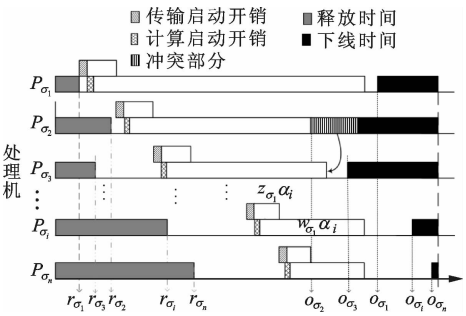

图 3　一种不满足模型约束的可分任务调度时序图

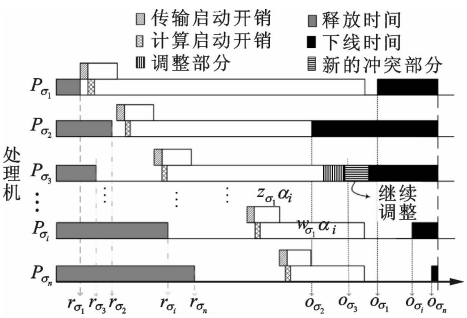

图 4　修正算子的迭代过程示意图

2.4　局部搜索算子

为了加快算法的收敛速度，本文引入了局部搜索算子，该算子通过调整具有最长任务完成时间 $T_{\max}$ 和最短完成时间 $T_{\min}$ 的两个处理机上任务分配量，使得那些满足下线时间约束的处理机能尽可能的同时完成计算，从而减小总的任务完成时间。局部搜索算子的具体执行过程为：找出所有处理机中对应完成时间最大的处理机 $P_{\max}$ 和最小的处理机 $P_{\min}$；计算两个完成时间的差值的一半，即 $\Delta T=(T_{\max}-T_{\min})/2$；将所得的差值时间除以各处理机中计算速率较大的那一个，即 $\Delta\alpha=\Delta T/w_{\max}$，计算出调整任务量 $\Delta\alpha$ 后，将 $P_{\max}$ 的任务量减少 $\Delta\alpha$，将 $P_{\min}$ 对应任务量增大 $\Delta\alpha$。若调整完成后，出现不满足约束情况，调用修正算子进行修正，直到满足约束。

应用局部搜索算子之前和之后的调度时序如图 5、6 所示，可知 P_{σ_1} 的任务完成时间 $T_{\max}$ 最长，而 P_{σ_3} 的任务完成时间 $T_{\min}$ 最短；通过局部搜索算子的调整后，均衡了分配给 P_{σ_1}、P_{σ_3} 的任务量，使得任务完成时间更加接近，从而减小了总任务的完成时间。

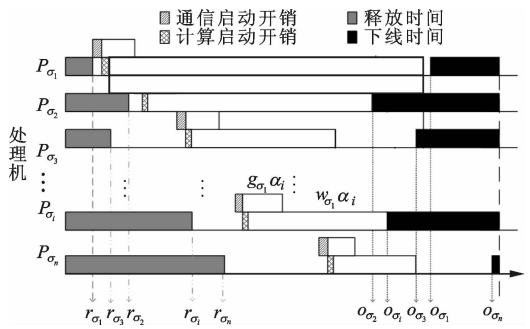

图 5　应用局部搜索算子之前的调度时序图

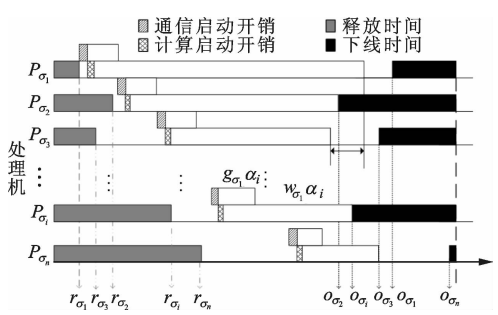

图 6　应用局部搜索算子之后的调度时序图

2.5 可分任务调度遗传算法整体框架

基于编码与解码规则、交叉与变异算子、修正算子和局部搜索算子，给出考虑处理机时间窗口的可分任务调度遗传算法如下：

步骤 1（初始化） 令进化代数 $t=0$，根据编码规则随机生成 S 个个体构成初始种群 $P(t)$，对种群 $P(t)$ 中的每个个体进行解码，计算任务的完成时间并将其倒数作为个体的适应度；

步骤 2（交叉） 通过轮盘赌操作从种群 $P(t)$ 中选择 S 个个体进入交叉池，然后按照交叉概率 p_{cros} 从交叉池中选择父代个体进行交叉，交叉后得到的子代集合记为 $O_1(t)$；

步骤 3（变异） 根据变异概率 p_{mut} 从集合 $O_1(t)$ 中选择父代个体按照变异算子进行变异，变异后得到的子代集合记为 $O_2(t)$；

步骤 4（修正） 对集合 $O_1(t)\bigcup O_2(t)$ 中不满足约束的子代个体进行修正，修正后的个体集合记为 $O_3(t)$；

步骤 5（局部搜索） 对集合 $O_3(t)$ 中的个体进行局部搜索，更新后的个体集合记为 $O_4(t)$；

步骤 6（选择） 从集合 $P(t)\bigcup O_4(t)$ 中选择适应度最大的 E 个个体直接进入下一代种群，通过轮盘赌操作从集合 $P(t)\bigcup O_4(t)$ 中继续选择 $S-E$ 个个体到 $P(t+1)$ 中。

初始化种群时，每个个体初始化的时间复杂度为 $O(n)$，所以初始化 S 个个体的时间复杂度为 $O(nS)$。在迭代过程中，算法的时间复杂度以计算个体适应度的次数来替代，在每一次迭代过程中，交叉算子中会计算 $2Sp_{\text{cros}}$ 次，变异算子会计算 $2Sp_{\text{mut}}$ 次，修正算子中最多计算 $n-1$ 次，局部搜索算子最多计算 $S+2S(p_{\text{cros}}+p_{\text{mut}})$ 次，所以迭代一次的算法复杂度为 $O(nS)$。算法一共迭代 t 次，则总的时间复杂度为 $O(ntS)$。

3 实验与结果分析

本文设计了多组仿真实验来证明所提算法的有效性和高效性，表 1 给出了异构星型网络的相关参数设置。本文所提算法 TW-GA 在实验中所使用到的 $S=100$，$p_{\text{cros}}=0.6$，$p_{\text{mut}}=0.02$，$E=5$，$t=50\ 000$。文献[16]所提穷举算法 EA 要求输入一个固定的任务调度顺序，与传统遗传算法 GA 进行比较，可知本文所提算法的有效性。

在星型网络中，假设所有处理机的下线时间无穷大，也就是只考虑处理机的释放时间。对于每一个测试任务量，本文使用 TW-GA 算法所求得的最优调度顺序作为 EA 算法的输入，来验证 TW-GA 算法所得到的任务完成时间和 EA 算法所得到任务完成时间是否相同。

TW-GA 算法和 EA 算法在相同任务调度顺序下的实验结果对比如表 2 所示。对于每一个任务量，TW-GA、EA 算法在所使用的处理机数目和任务完成时间上是一致的，由 EA 算法的正确性，通过穷举法可以找出最优解。因此，对于考虑处理机释放时间的可分任务调度情况，TW-GA 算法可以找到最优任务分配方案，即可以得到最小的任务完成时间。

在考虑处理机时间窗口的可分任务调度情况下，将 TW-GA 和 EA、GA 算法做了对比实验，其中 EA 在 3 种不同调度顺序 IG、IW 和 IR 下执行。由于 EA、GA 算法没有考虑到下线时间，所以对于超过下线时间的处理机需要重新分配任务量，本文对 EA、GA 算法采用多趟分配方式，将需要重新分配的任务量在下一趟中重新分配给处理机，直到所有任务完成计算。表 3 给出了 3 种算法不同任务量下的任务完成时间。由表 3 可知，TW-GA 算法在不同任务量下的完成时间均小于 EA、GA 算法。

TW-GA 和 EA 算法在不同任务量下任务完成时间的变化趋势如图 7 所示。任务量小于 600 时，TW-GA 算法的性能高出 EA 算法在 3 种调度顺序下的 30%左右。任务量增加到 1 000 时，TW-GA 的任务完成时间相当于降低了 EA 算法完成时间的

表 1 异构星型网络的相关参数设置

处理机	g	w	r	e	f	o	处理机	g	w	r	e	f	o
P_1	0.53	2.90	46.34	7.06	5.80	331.85	P_9	0.99	2.27	34.38	5.89	9.11	1 175.66
P_2	0.77	7.61	53.55	3.02	0.14	397.44	P_{10}	0.98	5.34	17.06	6.95	2.44	1 299345
P_3	0.71	4.14	78.14	8.14	0.45	420.58	P_{11}	0.99	1.57	79.63	1.06	6.76	1 474.28
P_4	0.79	9.62	93.47	8.63	3.74	509.31	P_{12}	0.10	7.99	31.81	5.75	1.03	1 763.75
P_5	0.06	3.64	13.86	8.71	9.50	613.82	P_{13}	0.05	3.82	53.32	2.84	2.96	1 768.40
P_6	0.77	5.92	57.31	5.25	0.54	722.11	P_{14}	0.95	4.01	17.40	3.01	9.80	1 911.18
P_7	0.30	6.48	47.47	4.69	6.23	855.85	P_{15}	0.16	6.47	60.60	2.78	1.63	1 943.74
P_8	0.28	8.25	22.18	2.64	8.30	964.82							

表 2　TW-GA 和 EA 算法在相同任务调度顺序下的实验结果对比

算法	W_{total}	n	T/s	处理机调度顺序
TW-GA	200	12	159.9	5,8,12,7,13,15,1,14,11,9,2,6
EA		12	159.9	5,8,12,7,13,15,1,14,11,9,2,6
TW-GA	400	14	254.651	5,8,12,7,13,15,1,3,2,14,11,9,6,10
EA		14	254.651	5,8,12,7,13,15,1,3,2,14,11,9,6,10
TW-GA	600	15	349.445	5,8,12,7,13,15,1,3,2,6,14,11,9,10,4
EA		15	349.445	5,8,12,7,13,15,1,3,2,6,14,11,9,10,4
TW-GA	800	15	442.97	5,8,12,13,15,7,1,3,2,6,14,11,9,10,4
EA		15	442.97	5,8,12,13,15,7,1,3,2,6,14,11,9,10,4
TW-GA	1 000	15	538.902	5,8,12,13,15,7,1,3,2,6,14,11,9,10,4
EA		15	538.902	5,8,12,13,15,7,1,3,2,6,14,11,9,10,4

70%左右，TW-GA 算法与 EA 算法的性能差距逐渐拉大。这是因为在实际任务调度中，处理计算速率、传输速率或释放时间时，最优调度顺序不是严格递增或递减。

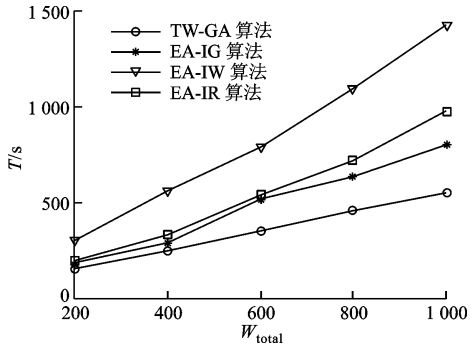

图 7　TW-GA、EA 算法任务完成时间的变化趋势

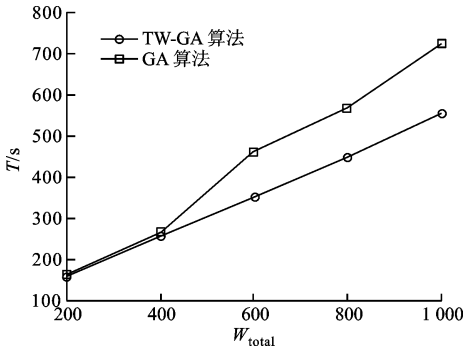

图 8　TW-GA、GA 算法任务完成时间的变化趋势

TW-GA 和 GA 算法任务完成时间的变化趋势如图 8 所示。由图 8 可知，在任务量低于 400 时，由于此时任务量较小，在没有达到处理机下线时间时所有处理机都已完成计算，此时 TW-GA 算法的任务完成时间略微优于 GA 算法，这证明了本文所提遗传算子的优越性。随着任务量的增加，由于任务完成时间大于某些处理机的下线时间，所有 GA 算法需要多趟调度才能完成任务，而 TW-GA 算法只需要单趟调度即可找出最优解，此时两种算法的任务完成时间差距拉大，TW-GA 算法的完成时间相当于降低了 20%的 GA 任务完成时间，并且随着任务量的增大，这个比例也会逐渐增加。

TW-GA 算法通过对处理机顺序和处理任务量同时进行编码，在求解最优任务分配方案的同时对调度顺序进行优化，通过设计出高效的交叉变异算子、修正算子和局部搜索算子，可在找到最优分配方案的同时找到最优调度顺序。由仿真实验可知，随着任务量的增加，处理机的时间窗口对于算法性能的影响也越来越大。

表 3　TW-GA 和 EA、GA 算法完成时间的对比

算法	T/s				
	$W_{total}=200$	$W_{total}=400$	$W_{total}=600$	$W_{total}=800$	$W_{total}=1\ 000$
TW-GA	159.900	254.651	350.827	451.640	557.384
EA-IG	191.610	288.659	517.155	629.508	798.301
EA-IW	304.482	558.191	787.359	1 095.140	1 427.850
EA-IR	196.475	330.915	541.837	715.468	977.349
GA	162.921	265.135	463.383	568.413	725.261

4 结 论

考虑到处理机的实际工作环境，本文提出了处理机时间窗口的概念，并在此基础上提出了一种新的考虑处理机时间窗口的可分任务调度模型去求解最优任务完成时间，同时基于所提模型，设计了基于全局优化的遗传算法来求解模型，算法中设计了多个高效的遗传算子，使得算法可同时求解出最优任务分配方案和最优任务调度顺序。通过设计多组不同的仿真实验验证了所提算法的性能，在相同调度顺序下，所提算法与对比算法取得了完全一致的最优解，证明了所提算法的正确性。在不同任务量下进行调度，比较了算法的最小任务完成时间。结果表明，在处理机时间窗口的约束下，所提算法较对比算法至少提升了 20%的性能，证明了所提算法的有效性和高效性。

参考文献：

[1] ROBERTAZZI T G. Ten reasons to use divisible load theory [J]. Computer, 2003, 36(5): 63-68.

[2] SURESH S, MANI V, OMKAR S N, et al. Parallel video processing using divisible load scheduling paradigm [J]. Journal of Broadcast Engineering, 2005, 10(1): 83-102.

[3] KO K, ROBERTAZZI T G. Signature search time evaluation in flat file databases [J]. IEEE Transactions on Aerospace & Electronic Systems, 2008, 44(2): 493-502.

[4] MIN W H, VEERAVALLI B. Aligning biological sequences on distributed bus networks: a divisible load scheduling approach. [J]. IEEE Transactions on Information Technology in Biomedicine, 2005, 9(4): 489-501.

[5] LIU D, YANG X, CHENG Z. An energy-aware scheduling algorithm for divisible loads in a bus network [J]. Concurrency and Computation Practice and Experience, 2016, 28(5): 1612-1628.

[6] BRANDÃO J S, NORONHA T F, MAURICIO G. C, et al. A biased random-key genetic algorithm for single-round divisible load scheduling [J]. International Transactions in Operational Research, 2015, 22(5): 823-839.

[7] ZHANG Z, ROBERTAZZI T G. Scheduling divisible loads in Gaussian, mesh and torus network of processors [J]. IEEE Transactions on Computers, 2015, 64(11): 3249-3264.

[8] CHEN C Y, CHU C P. Novel methods for divisible load distribution with start-up costs on a complete b-ary tree [J]. IEEE Transactions on Parallel & Distributed Systems, 2015, 26(10): 2836-2848.

[9] SURESH S, HUANG H, KIM H J. Scheduling in compute cloud with multiple data banks using divisible load paradigm [J]. IEEE Transactions on Aerospace & Electronic Systems, 2015, 51(2): 1288-1297.

[10] VEERAVALLI B, GHOSE D, MANI V. Optimal sequencing and arrangement in distributed single-level tree networks with communication delays [J]. IEEE Transactions on Parallel & Distributed Systems, 1994, 5(9): 968-976.

[11] VEERAVALLI B, BARLAS G. Scheduling divisible loads with processor release times and finite size buffer capacity constraints in bus networks [J]. Cluster Computing, 2003, 6(1): 63-74.

[12] HU M, VEERAVALLI B. Requirement-aware strategies with arbitrary processor release times for scheduling multiple divisible loads [J]. IEEE Transactions on Parallel & Distributed Systems, 2011, 22(10): 1697-1704.

[13] 王晓萍，孟坤. 异构分布式系统的可分任务调度算法 [J]. 小型微型计算机系统，2015，36(4)：797-800.
WANG Xiaoping, MENG Kun. New genetic algorithm for divisible load scheduling in heterogenous distributed systems [J]. Journal of Chinese Computer Systems, 2015, 36(4): 797-800.

[14] 王晓丽，王宇平，孟坤. 考虑处理机释放时间的可分任务调度优化模型 [J]. 西安电子科技大学学报，2016，43(1)：55-60.
WANG Xiaoli, WANG Yuping, MENG Kun. Release time aware divisible-load scheduling optimization model [J]. Journal of Xidian University, 2016, 43(1): 55-60.

[15] 王晓丽，王宇平，蔡坤. 考虑释放时间和调度顺序的可分任务调度模型 [J]. 华中科技大学学报(自然科学版)，2015，43(12)：106-111.
WANG Xiaoli, WANG Yuping, CAI Kun. Release time and start-up overhead aware divisible-load scheduling model [J]. Journal of Huazhong University of Science and Technology (Natural Science Edition), 2015, 43(12): 106-111.

[16] CHOI K, ROBERTAZZI T G. An exhaustive approach to release time aware divisible load scheduling [J]. International Journal on Internet & Distributed Computing Systems, 2011, 1(2): 145-151.

(编辑 赵炜)