考虑处理机下线时间的可分任务调度优化模型

王晓丽，王宇平，蔡 坤，赖俊凡

(西安电子科技大学计算机学院 西安 710071)

【摘要】随着科学应用逐渐趋于数据密集型计算，为并行与分布式系统寻求高效的任务调度策略成了研究的热点问题。已有的可分任务调度模型均假设所有处理机都能100%的完成子任务的计算，即处理机在完成任务计算之前一直保持在线状态。实际上，并行与分布式系统中不同处理机的在线时间可能不同。若忽略处理机的在线时间，为其分配的任务量过大，则任务的完成时间可能超出处理机的下线时间，从而造成任务的计算无法按时完成。因此，为处理机分配任务时应充分考虑处理机下线时间的限制。为解决上述问题，该文提出了一种新的考虑处理机下线时间的可分任务调度优化模型，并设计了全局优化遗传算法求解该模型。最后，通过仿真实验结果验证了模型和算法的有效性。

关 键 词 可分任务调度； 遗传算法； 下线时间； 并行与分布式系统

中图分类号 TP393 **文献标志码** A doi:10.3969/j.issn.1001-0548.2017.01.014

Off-Line Time Aware Divisible-Load Scheduling Optimization Model

WANG Xiao-li, WANG Yu-ping, CAI Kun, and LAI Jun-fan

(School of Computer Science and Technology, Xidian University Xi'an 710071)

Abstract As scientific applications become more data intensive, finding an efficient scheduling strategy for massive computing in parallel and distributed systems has drawn increasingly attention. Most existing scheduling models assume that all processors can 100% finish computing, that is, they keep online during the completion of assigned workload fractions. In fact, in the real parallel and distributed environments, different processors have different off-line time. Therefore, off-line time constraints of processors should be taken into account before distributing of the workload fractions; otherwise, some processors may not be able to fish computing their assignments. To solve the above issue, this paper proposes an off-line time aware divisible-load scheduling model and designs an effective global optimization genetic algorithm to solve it. Finally, experimental results illustrate the effectiveness of the proposed model and the efficiency of the proposed algorithm.

Key words divisible-load scheduling; genetic algorithm; off-line time; parallel and distributed systems

随着大数据时代的来临，数据的规模呈爆炸式增长，如何高效快速处理并分析数据成了研究的重点和难点问题。大数据应用问题，如大规模矩阵运算、DNA测序分析、卫星图像处理等，虽然数据规模庞大，但是大都可以抽象为可分任务，即任务可以被划分为任意大小的子任务，子任务间相互独立且没有优先级关系[1]。并行与分布式系统下可分任务调度问题的目标是寻求最优的任务分配方案使得任务的完成时间最短。

针对并行与分布式系统下常见的线型拓扑结构、总线型拓扑结构和树型拓扑结构，已有很多文献对可分任务的最优调度策略进行了研究。文献[2]给出了同构线型网络下最优任务分配方案的紧式耦合解，文献[3]给出了同构树型网络与总线型网络下任务分配方案的渐近解。文献[4]给出了异构星型网络下任务分配方案的紧式耦合解，并且证明了处理机调度顺序在遵循通信速率递减的情况下任务的完成时间最短。对于异构树型网络，文献[5]的研究表明处理机的调度顺序只依赖于其通信速率而非计算速率。然而，这些研究成果都没有考虑处理机的计算启动开销和网络的通信启动开销。文献[6-7]在调度模型中引入了启动开销，分析了总线型网络下启动开销和处理机的调度顺序对任务完成时间的影响，证明了当处理机遵循计算速率递减的顺序时任务的完成时间最短。为了使可分任务调度模型更符合分布式平台的实际网络环境，文献[8]将可分任务调度模型扩展到多源网格环境中，文献[9]研究了云平台下处理机带宽受限的可分任务调度问题，文献

收稿日期：2015－07－13；修回日期：2016－05－31
基金项目：国家自然科学基金(61402350, 61472297, 61572391)；中央高校基本科研业务费专项资金(JB150307)
作者简介：王晓丽(1987－)，女，博士，主要从事并行与分布式系统下的任务调度方面的研究.

[10]将其扩展到混合云计算平台环境中，文献[11-12]将其扩展到无线传感器网络中，文献[13-14]将其扩展到实时环境中。

但是，已有的可分任务调度模型均假设所有处理机都能100%的完成所分配的子任务，即处理机在完成所分配的子任务之前一直保持在线状态[15]。然而在实际的网络环境下这个假设并不成立，不同处理机的下线时间可能不同。若为处理机分配的任务量过大，则任务的完成时间可能超出处理机的下线时间，从而造成任务的计算无法按时完成。若处理机未完成计算就已下线，则为其分配的任务需要等待其他处理机完成计算空闲后，调度到其他仍在线的处理机上重新开始计算，从而导致任务的总完成时间增大。鉴于此，本文提出了一种新的考虑处理机下线时间的可分任务调度优化模型，并且设计了新的遗传算法对模型进行求解。

1　考虑处理机下线时间的可分任务调度优化模型

1.1　问题描述

$N+1$ 台处理机通过星型网络(单层树网络)互连，其中，P_0 是主处理机，$\{P_i \mid i \in \{1,2,\cdots,N\}\}$ 是从处理机，所有处理机的计算速率相同。任务位于主处理机 P_0 上，任务大小记为 W_{total}。处理机 P_i 的下线时间记为 o_i，其中 $i=1,2,\cdots,N$。l_i 是连接 P_0 和 P_i 的通信链路，所有链路的传输速率相同。值得注意的是，并不是所有处理机都必须参与任务的计算，记参与计算的处理机数目为 n ($n \leqslant N$)。主处理机 P_0 负责将总任务划分为 n 个子任务 $A=(\alpha_1,\alpha_2,\cdots,\alpha_n)$，然后按次序依次调度到从处理机 $P_1,P_2,\cdots,P_n$ 上完成并行计算。所有处理机分配的任务量之和应为总任务量，即：

$$\sum_{i=1}^{n}\alpha_i = W_{\text{total}} \tag{1}$$

主处理机 P_0 向从处理机 P_i 传输大小为 α_i 的子任务所需的传输时间为 $C+z\alpha_i$，其中，C为通信启动开销，z 为通讯链路传输单位大小的任务所花费的时间。处理机 P_i 计算子任务 α_i 所需的时间为 $B+w\alpha_i$，其中，B 为计算启动开销，w 为处理机计算单位任务所需的时间。

1.2　可分任务调度优化模型

记处理机 P_i 完成所分配的子任务 α_i 的时间为 T_i，则总任务的完成时间 $T=\max\{T_1,T_2,\cdots,T_n\}$。文献[3]证明了只有当所有处理机同时完成计算时，任务的完成时间最短，否则完全可以将后完成计算的处理机上的部分任务调度到先完成计算的处理机上执行。但是，由于处理机受下线时间的限制，不能保证所有参与计算的处理机同时完成计算。下面分两种情况来讨论处理机的下线时间对任务完成时间的影响。

情况1　$T=\max\{T_1,T_2,\cdots,T_n\} \leqslant o_i$，其中 $i=1,2,\cdots,n$，$T_i = iC + z\sum_{j=1}^{i}\alpha_j + B + w\alpha_i$，即任务的完成时间小于所有处理机的下线时间，即所有处理机均不受下线时间的影响。图1给出了满足该约束的一种可能的任务调度时序图。由于处理机不受下线时间的约束，因此所有处理机同时完成计算时总任务的完成时间最短，由此可得：

$$B+w\alpha_i = C+z\alpha_{i+1}+B+w\alpha_{i+1},\ i=1,2,\cdots,n-1 \tag{2}$$

则 α_{i+1} 可以写成 α_i 的表达式：

$$\alpha_{i+1} = \frac{w}{z+w}\left(\alpha_i - \frac{C}{w}\right),\ i=1,2,\cdots,n-1 \tag{3}$$

记 $q=w/(z+w)$，递推可得：

$$\alpha_i = q^{i-1}\alpha_1 - \frac{q-q^i}{1-q}\frac{C}{w},\quad i=1,2,\cdots,n \tag{4}$$

将式(4)带入式(1)可得：

$$\frac{1-q^n}{1-q}\alpha_1 - \frac{C}{w}\frac{nq-nq^2-q+q^{n+1}}{(1-q)^2} = W_{\text{total}} \tag{5}$$

整理可得 α_i 的解为：

$$\alpha_1 = \frac{1-q}{1-q^n}W_{\text{total}} + \frac{C}{w}\frac{nq-nq^2-q+q^{n+1}}{(1-q)(1-q^n)} \tag{6}$$

由图1可知，总任务的完成时间 T 满足：

$$T=(z+w)\alpha_1+C+B = C+B+(z+w)\times \frac{1-q}{1-q^n}W_{\text{total}} + \frac{C}{w}\frac{nq-nq^2-q+q^{n+1}}{(1-q)(1-q^n)} \tag{7}$$

式中，$q=w/(z+w)$。可以看出，对于情况1，可以直接推导 α_i 的紧式耦合解式，将式(6)带入式(4)可得所有处理机的任务分配方案，进而求得任务的最短完成时间。

情况2　$\exists o_i < T=\max\{T_1,T_2,\cdots,T_n\}$，其中，$i=1,2,\cdots,n$，即存在处理机 P_i 受其下线时间 o_i 的约束，只能完成既定大小的任务。图2给出了满足该约束的一种可能的任务调度时序图。处理机 P_i 的任务完成时间必须满足：

$$T_i = iC + z\sum_{j=1}^{i}\alpha_j + B + w\alpha_i = o_i$$

因此，若处理机 P_i 受其下线时间的约束，其任务分配量为：

$$\alpha_i = (o_i - iC - z\sum_{j=1}^{i}\alpha_j - B)/w,\ i = 1,2,\cdots n \qquad (8)$$

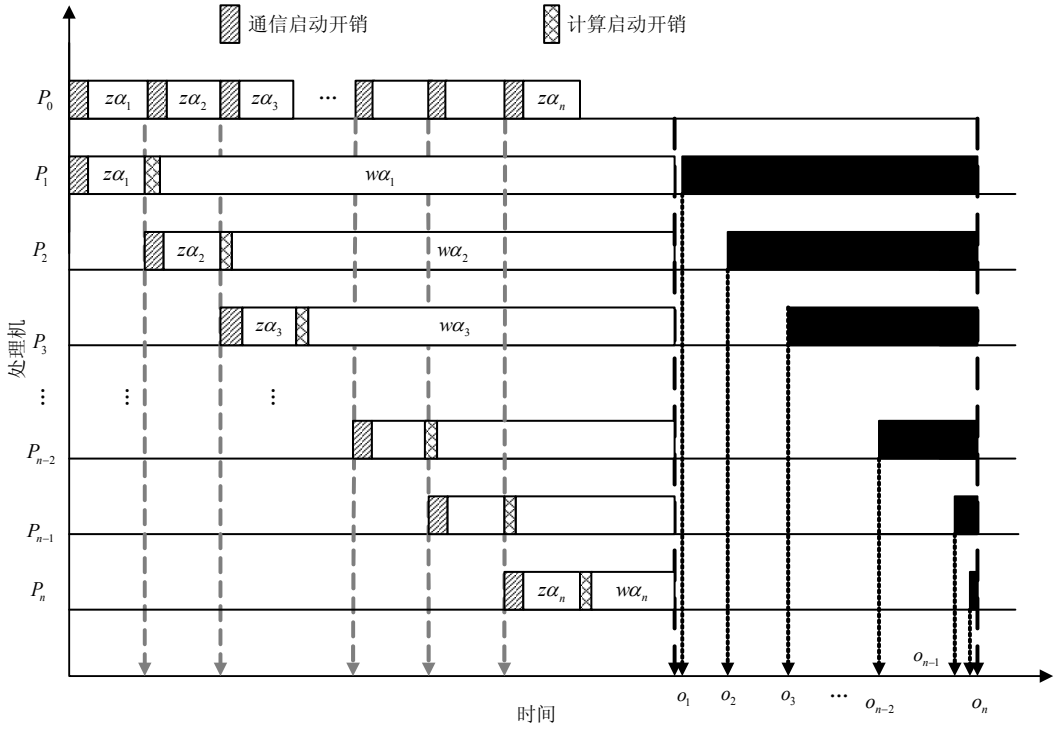

图1 所有处理机不受下线时间的影响

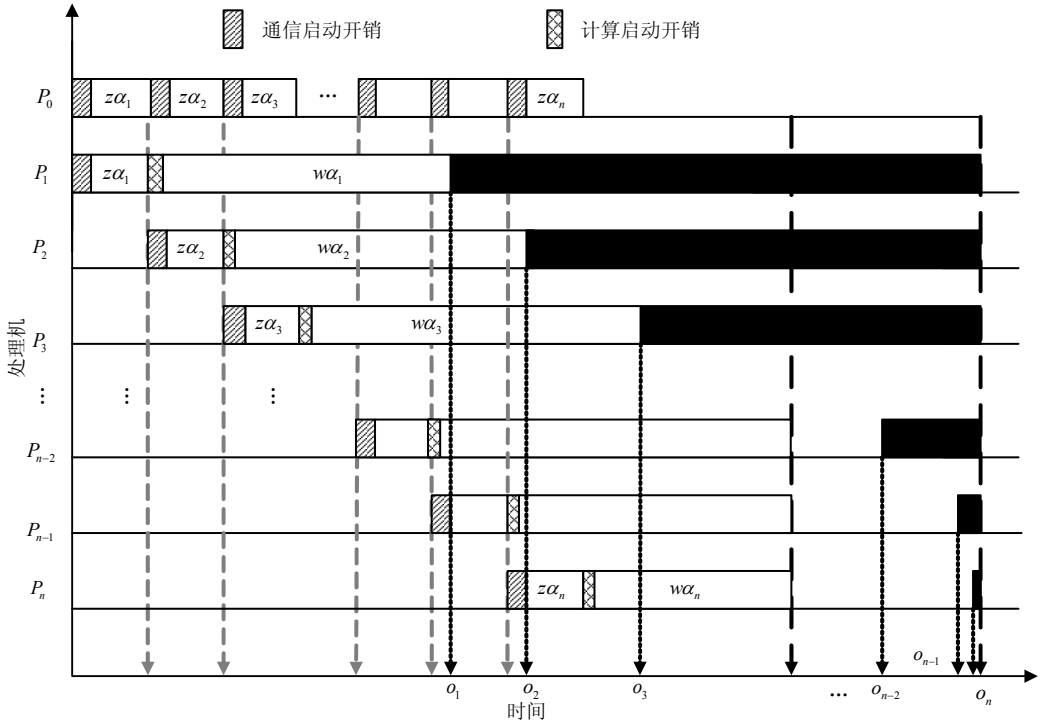

图2 部分处理机受下线时间的影响

与情况1不同的是，对于情况2，无法直接通过计算得到所有处理机的最优任务分配方案，也就无法直接得到总任务的完成时间。这主要是因为可能只有部分处理机受下线时间的约束，因此无法通过式(7)得到所有处理机的最优任务分配方案。鉴于此，本文建立了以下考虑处理机下线时间的可分任务调度优化模型：

$$\min_{n,A} T = \min_{n,A}\left(\max_{1\leqslant i\leqslant n} T_i\right) =$$

$$\min_{n,A}\left(\max_{1\leqslant i\leqslant n}\left(iC + z\sum_{j=1}^{i}\alpha_j + B + w\alpha_i\right)\right)$$

s.t. 1) $0 < n \leqslant N$，其中 N 为处理机总数，n 为参与计算的处理机数目；

2) $0 < \alpha_i \leqslant W_{\text{total}}$，$i = 1,2,\cdots,n$；

3) $\sum_{i=1}^{n}\alpha_i = W_{\text{total}}$；

4) $iC + z\sum_{j=1}^{i}\alpha_j + B + w\alpha_i \leqslant o_i$，$i = 1,2,\cdots,n$。

模型的目标是总任务的完成时间最短，变量是参与计算的最优处理机数目 n 和处理机的最优任务分配方案 $A = (\alpha_1,\alpha_2,\cdots,\alpha_n)$。模型的约束条件1)表示并非所有的处理机都必须参与计算；约束条件2)表示每个处理机所分配的任务量必须非负，且不能超过总任务量 W_{total}；约束条件3)表示所有处理机分配的任务量之和为总任务量 W_{total}；约束条件4)表示所有处理机完成所分配任务的时间必须小于等于其下线时间。

2　全局优化遗传算法

为了求解建立的优化模型，本文设计了新的全局优化遗传算法，包括编码与解码规则、交叉与变异算子、修正算子和局部搜索算子。

2.1　编码与解码

本文采用实数编码方式，个体用 N 维向量 $\boldsymbol{I} = (\alpha_1,\alpha_2,\cdots,\alpha_N)$ 表示，其中，N 表示处理机总数，α_i 表示分配给处理机 P_i 的任务量大小，因此 $0 \leqslant \alpha_i \leqslant W_{\text{total}}$ 且 $\sum_{i=1}^{n}\alpha_i = W_{\text{total}}$。举例说明：设 $N = 5$，任务总量 $W_{\text{total}} = 1$，则一种可能的编码方案为：$\boldsymbol{I} = (0.3,0.3,0.2,0.2,0)$。由该编码方案可知，共有 $n = 4$ 台处理机参与计算，因为 $\alpha_5 = 0$，即处理机 P_5 分配的任务量为零，即该处理机没有参与计算。而其他4台处理机 P_1,P_2,P_3,P_4 所分配的任务量分别为 $0.3,0.3,0.2$ 和 0.2。

对个体解码时，统计个体 $\boldsymbol{I} = (\alpha_1,\alpha_2,\cdots,\alpha_N)$ 中非零元素的个数，并将其作为参与计算的处理机数目 n。值得注意的是，并不是所有符合上述编码方案构成的个体都是有效的个体。若解码后，个体不满足模型的全部约束条件，则需要按照修正算子对个体进行修正，2.3节将给出修正算子的具体算法思想和步骤。

2.2　交叉和变异算子

交叉算子采用三三交叉的方式，即3个父代两两交叉生成3个子代。具体操作步骤如下：首先，按照交叉概率 p_{cros} 从交叉池中选择3个个体作为父代，然后随机生成3个整数 low、mid 和 high 满足 $0 \leqslant \text{low} < \text{mid} < \text{high} \leqslant N$，按照图3所示的方法进行交叉，使得每个子代个体都包含3个父代个体的部分基因。通过三三交叉可以更大程度地增加种群的多样性，较常用的两两交叉而言，可以加快算法收敛于全局最优解，从而提高整个算法的执行效率。

本文采用两点变异的方式。对于个体 $\boldsymbol{I} = (\alpha_1,\alpha_2,\cdots,\alpha_N)$，随机生成两个整数 p 和 q，满足 $0 \leqslant p < q \leqslant N$，互换 α_p 和 α_q 的值，生成新的子代个体。值得注意的是，通过交叉和变异算子所产生的子代不一定满足模型的全部约束条件，此时需要按照修正算子对子代进行修正。

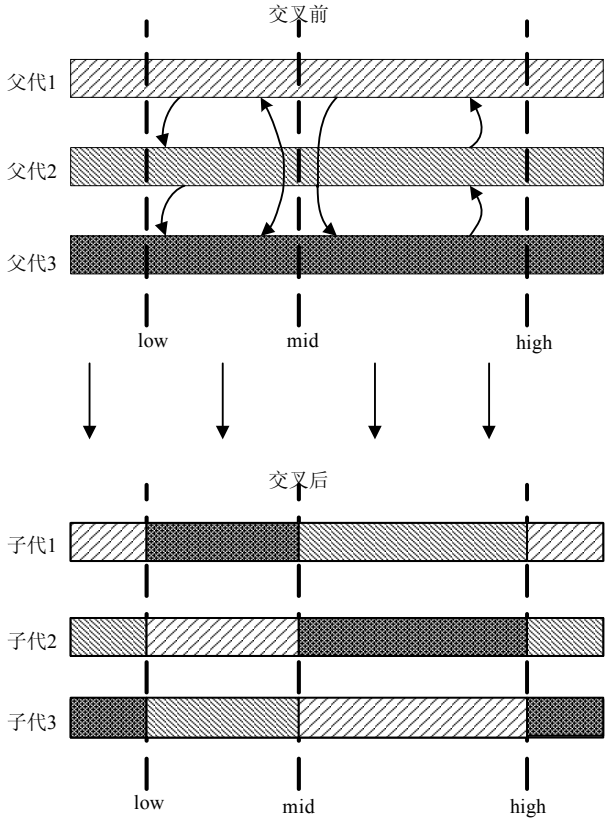

图3　交叉示意图

2.3　修正算子

如前文所述，不论是种群的初始化，还是交叉及变异产生的新个体，都不能保证个体满足模型的全部约束条件，因此需要对个体进行修正。修正算子的主要思想如下：若处理机 P_i 完成所分配的任务量 α_i 的时间超出了其下线时间，则将超出下线时间

的部分(又称为冲突部分)调整到相邻的下一台处理机 P_{i+1} 上执行。若第一轮调整过后，处理机 P_{i+1} 的完成时间也超出了其下线时间，则继续将 P_{i+1} 的冲突部分调整到相邻的下一台处理机 P_{i+2} 上执行，循环此过程，直至冲突完全消除。可见，修正的过程是一个迭代的过程。

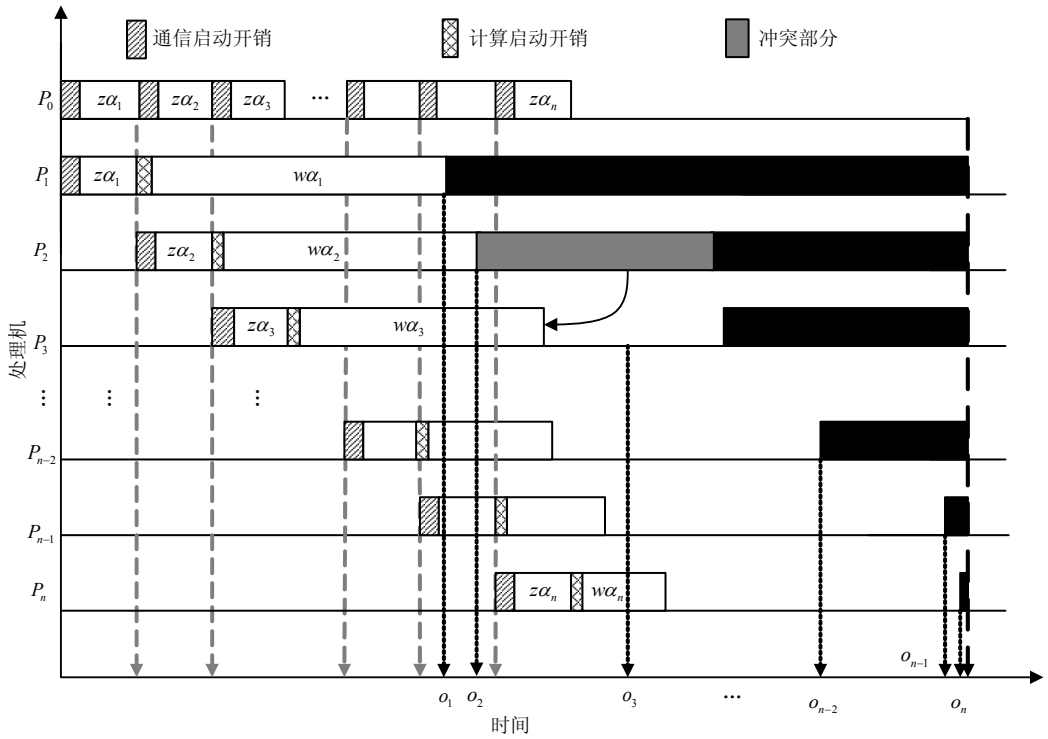

图4 一种不满足模型约束的可分任务调度时序图

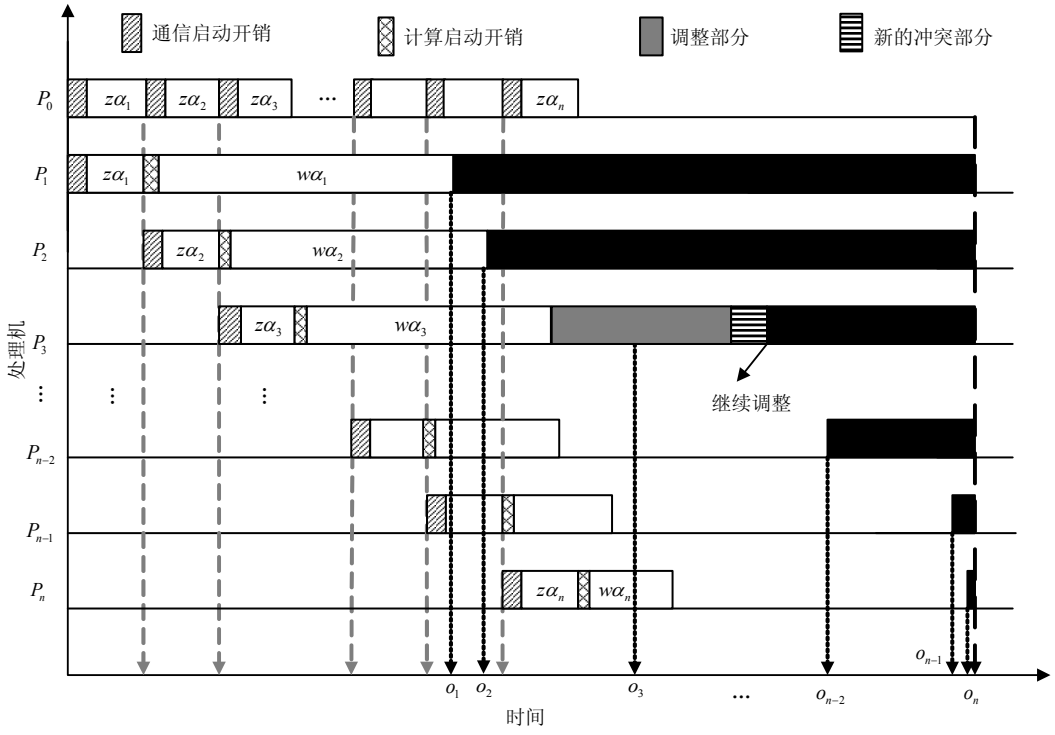

图5 修正算子的迭代过程示意图

图4给出了一种不满足模型约束的可分任务调度时序图。从该图可以看出，处理机 P_2 完成任务 α_2 所需要的时间超出了该处理机的下线时间 o_2，图中灰色部分即为冲突部分。因此，需要对处理机 P_2

的任务量α_2进行调整。首先，将冲突部分对应的任务量分配给处理机P_3，图5给出了调整后的可分任务调度时序图。选择分配给相邻的下一台处理机是考虑到每台处理机的开始时间与其前面的所有处理机的任务分配量直接相关。如果将冲突部分调整到前面的处理机，调整后将推迟后续所有处理机的开始时间，导致新的冲突产生。从图5可以看出，将处理机P_2的冲突部分调整给处理机P_3后，处理机P_3产生了新的冲突部分，因此需要继续迭代修正过程，直至冲突全部消除。

2.4　局部搜索算子

为了增强算法的局部搜索能力，提高算法的收敛速度，本文为遗传算法引入了局部搜索算子。

算子的设计思想如下：由于所有处理机是同构的，因此若不考虑处理机的下线时间，为达到总任务的完成时间最短，调度顺序靠前的处理机其分配的任务量应大于调度顺序靠后的处理机。鉴于此，可以通过置换相邻两台处理机的任务分配量来寻求更优的个体。局部搜索算子的具体执行过程如下：对所有处理机的任务量从后向前逐个进行比较，如相邻两个处理机中，前面处理机的任务完成时间小于其下线时间且后面处理机所分配的任务量大于前面处理机的任务量，那么将两个处理机的任务量进行交换。图6给出了一种可能的局部搜索执行过程。

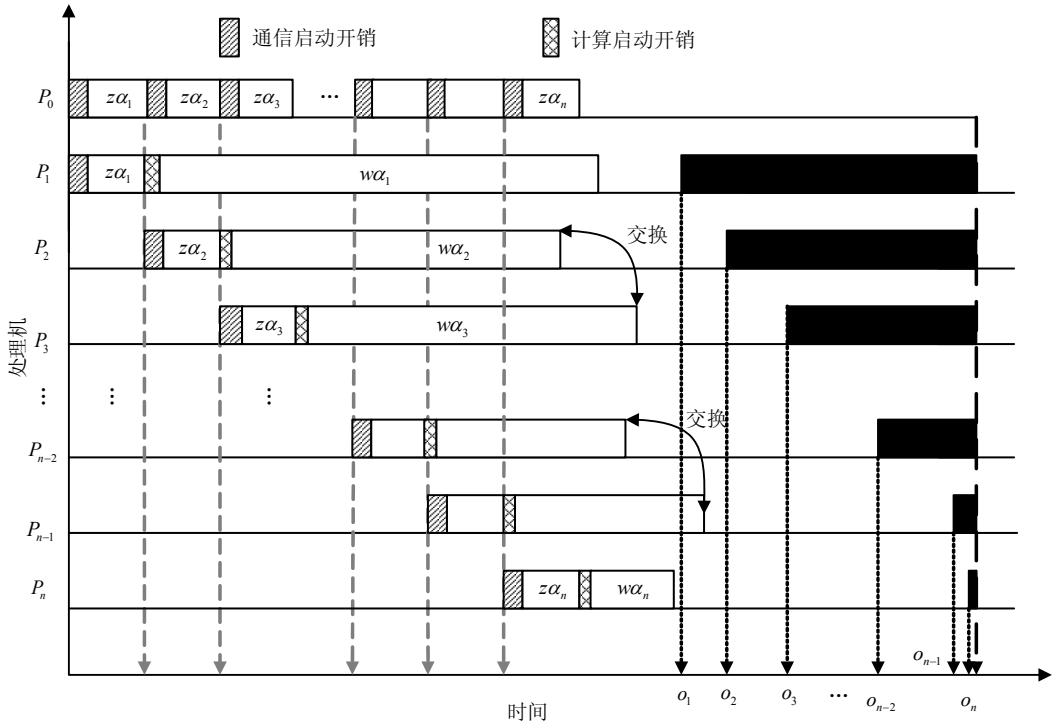

图6　局部搜索算子执行过程示意图

2.5　可分任务调度遗传算法整体框架

基于前文设计的编码与解码规则、交叉与变异算子、修正算子和局部搜索算子，下面给出考虑处理机下线时间的可分任务调度遗传算法整体框架。

算法1　考虑处理机下线时间的可分任务调度遗传算法。

1) (初始化)令进化代数$t=0$，根据编码规则随机生成Popsize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。

2) (交叉)通过轮盘赌操作从种群$P(t)$中选择Popsize个个体进入交叉池，然后按照交叉算子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$O_1(t)$。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$O_2(t)$。

4) (修正)对集合$O_1(t)\cup O_2(t)$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$O_3(t)$。

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$O_4(t)$。

6) (选择)从集合$P(t)\cup O_4(t)$中选择适应度最大的E个个体直接进入下一代种群$P(t+1)$。然后通过轮盘赌操作从集合$P(t)\cup O_4(t)$中继续选择$\text{PopSize}-E$个个体到$P(t+1)$中。令$t=t+1$。

7) (终止条件)若算法达到终止条件，则终止；否则，转至2)。

3 实验与结果分析

为了验证本文提出的算法的有效性，将其与已有算法[15]进行了多组对比实验，其中并行与分布式系统的参数设置如下：$N=20$，$z=0.8$，$w=1.2$，$C=0.005$，$B=0.002$；遗传算法的参数设置如下：种群大小 PopSize=100，交叉概率 $p_{\text{cros}}=0.6$，变异概率 $p_{\text{mut}}=0.02$，精英保留个数 $E=5$，终止条件为进化代数 $t=10\,000$。

实验1 固定处理机的下线时间，对比两个算法在不同任务量情况下求得的任务完成时间。20台处理机的下线时间分别为：

$o_1 \sim o_4$：431.85, 467.44, 570.58, 599.36

$o_5 \sim o_8$：669.31, 813.82, 881.45, 922.11

$o_9 \sim o_{12}$：1 055.85, 1 086.29, 1 164.82, 1 200.71

$o_{13} \sim o_{16}$：1 275.66, 1 299.45, 1 474.28, 1 763.75

$o_{17} \sim o_{20}$：1 768.40, 1 813.13, 1 911.18, 1 943.74

表1给出了不同任务量($W_{\text{total}}=100 \sim 1\,000$)下两种算法对比的实验结果，其中，GA代表本文的全局优化遗传算法，TSA代表文献[15]的传统调度策略。

表1 不同任务量情况下两种算法对比的实验结果

算法	任务量/MB	完成时间/s	算法	任务量/MB	完成时间/s
GA	100	80.09	GA	600	480.11
TSA		80.09	TSA		787.49
GA	200	160.10	GA	700	560.12
TSA		160.10	TSA		918.70
GA	300	240.10	GA	800	640.14
TSA		240.10	TSA		1 197.49
GA	400	320.11	GA	900	720.17
TSA		320.11	TSA		1 384.61
GA	500	400.11	GA	1 000	800.21
TSA		400.11	TSA		1 538.43

从表1的左半部分可以看出，当任务量 $W_{\text{total}}=100 \sim 500$ 时，由于处理机不受下线时间的影响(任务的完成时间小于所有处理机的下线时间)，两个算法求得的任务完成时间相同。从表1的右半部分可以看出，当任务量 $W_{\text{total}}=600 \sim 1\,000$ 时，由于部分处理机受到下线时间的影响(任务的完成时间大于部分处理机的下线时间)，两个算法求得的任务完成时间不同，而且本文算法GA求得的任务完成时间要远小于算法TSA求得的任务完成时间。

为了更直观地反应表1中两种算法的对比结果，图7给出了两种算法的任务完成时间随任务大小的变化趋势。由图7可以看出，随着任务量的增大，两个算法求得的任务完成时间的差距越来越大。这主要是因为本文的算法在任务调度前充分考虑到各个处理机下线时间的限制，为下线时间较早的处理机分配合理的任务量，避免了任务的重新分配和重新计算，从而减少了总任务的完成时间。随着任务量的增大，处理机下线时间的影响越来越明显，因此两个算法求得的任务完成时间的差距就越来越大。

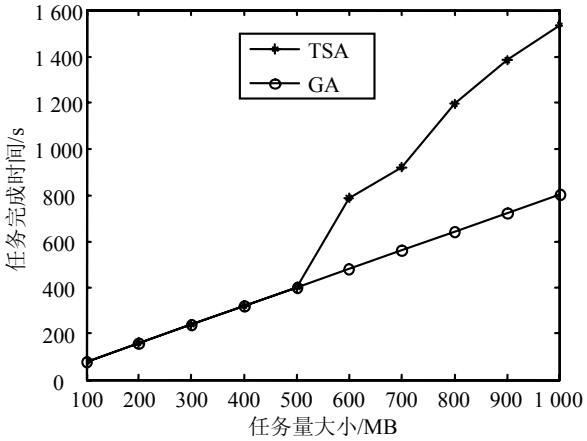

图7 两种算法的任务完成时间随任务大小的变化趋势

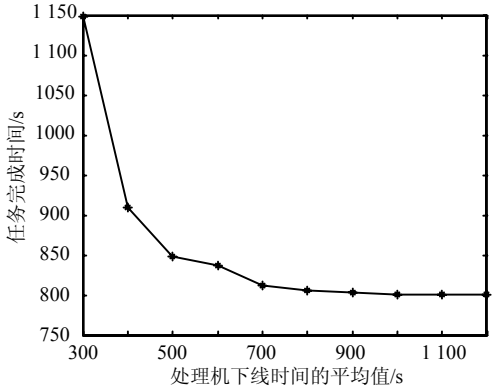

图8 GA的任务完成时间随下线时间平均值的变化趋势

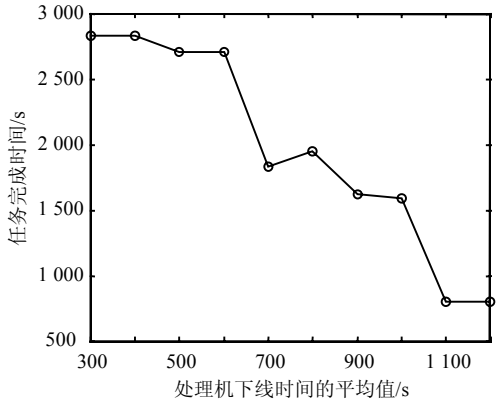

图9 TSA的任务完成时间随下线时间平均值的变化趋势

实验2 固定任务量，对比两个算法在不同下线时间情况下求得的任务完成时间。任务量 $W_{\text{total}}=1\,000$，处理机 $P_1 \sim P_{20}$ 的下线时间 $o_1 \sim o_{20}$ 是

指数分布的随机数。图8和图9分别给出了算法GA和TSA的任务完成时间随处理机下线时间的平均值(300～1 200)的变化趋势。

由图8和图9可以看出，随着处理机下线时间平均值的增大，两个算法求得的任务完成时间趋于平稳，结果趋于一致。这主要是因为针对固定大小的任务，随着下线时间的增大，处理机不再受下线时间的影响，当所有处理机同时完成计算时任务的完成时间最短，因此两个算法求得的任务完成时间相同。当下线时间的平均值较小时，部分处理机开始受到下线时间的影响，两个算法求得的任务完成时间不同且GA求得的任务完成时间要远小于对比算法TSA求得的任务完成时间。而且，针对固定大小的任务，处理机下线时间的平均值越小，对任务完成时间的影响越大，本文提出的算法GA相较于TSA的优势就越明显。

4　结 束 语

本文在考虑实际并行与分布式环境中处理机存在下线时间的基础上，分析了两种不同时间约束下的任务调度过程，提出了一种新的考虑处理机下线时间的可分任务优化调度模型，并设计了高效的全局优化遗传算法对其进行求解。实验结果表明本文提出的算法能够针对不同处理机的下线时间有针对性的为其分配任务，使得总任务的完成时间最短。

参 考 文 献

[1] BHARDWAJ V, GHOSE D, MANI V, et al. Scheduling divisible loads in parallel and distributed systems[M]. Los Alamitos CA: IEEE Computer Society Press, 1996.

[2] MANI V, GHOSE D. Distributed computation in linear networks: Closed-form solutions[J]. IEEE Transactions on Aerospace and Electronic Systems, 1994, 30(2): 471-483.

[3] GHOSE D, MANI V. Distributed computation with communication delays: Asymptotic performance analysis[J]. Journal of Parallel and Distributed Computing, 1994, 23(3): 293-305.

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

[5] KIM H J, JEE G I, LEE J G. Optimal load distribution for tree network processors[J]. IEEE Transactions on Aerospace and Electronic Systems, 1996, 32(2): 607-612.

[6] SURESH S, MANI V, OMKAR S N. The effect of start-up delays in scheduling divisible loads on bus networks: an alternate approach[J]. Computers & Mathematics with Applications, 2003, 46(10): 1545-1557.

[7] VEERAVALLI B, LI X, KO C C. On the influence of start-up costs in scheduling divisible loads on bus networks[J]. IEEE Transactions on Parallel and Distributed Systems, 2000, 11(12): 1288-1305.

[8] MURUGESAN G, CHELLAPPAN C. Multi-source task scheduling in grid computing environment using linear programming[J]. International Journal of Computational Science and Engineering, 2014, 9(1): 80-85.

[9] LIN W, LIANG C, WANG J Z, et al. Bandwidth aware divisible task scheduling for cloud computing[J]. Software: Practice and Experience, 2014, 44(2): 163-174.

[10] HOSEINYFARAHABADY M R, LEE Y C, ZOMAYA A Y. Randomized approximation scheme for resource allocation in hybrid-cloud environment[J]. The Journal of Supercomputing, 2014, 69(2): 576-592.

[11] SHI H, WANG W, KWOK N M, et al. Adaptive indexed divisible load theory for wireless sensor network workload allocation[J]. International Journal of Distributed Sensor Networks, 2013(1): 1-18.

[12] DAI L, SHEN Z, CHEN T, et al. Analysis and modeling of task scheduling in wireless sensor network based on divisible load theory[J]. International Journal of Communication Systems, 2014, 27(5): 721-731.

[13] HU M, VEERAVALLI B. Dynamic scheduling of hybrid real-time tasks on clusters[J]. IEEE Transactions on Computers, 2014, 63(12): 2988-2997.

[14] HU M, VEERAVALLI B. Requirement-aware strategies for scheduling real-time divisible loads on clusters[J]. Journal of Parallel and Distributed Computing, 2013, 73(8): 1083-1091.

[15] ROSAS C, SIKORA A, JORBA J, et al. Improving performance on data-intensive applications using a load balancing methodology based on divisible load theory[J]. International Journal of Parallel Programming, 2013, 42(1): 94-118.

编　辑　漆　蓉