DOI：10.7652/xjtuxb201808013

分布式计算系统下可分任务的周期性多趟调度

朱海[1,2]，王晓丽[2]，马海明[2]
(1.周口师范学院网络工程学院，466001，河南周口；2.西安电子科技大学计算机学院，710071，西安)

摘要：针对已有单趟任务调度模型因无法充分利用分布式平台的并行特性导致系统利用率和任务完成效率较低的问题，提出了一种新的周期性多趟任务调度优化模型。在给定处理机调度顺序的情况下，推导得到了分布式系统最优任务分配方案的解析解；通过分析任务完成时间关于调度趟数和服务器数的变化曲线，设计了一种启发式算法寻求最优的调度趟数和参与计算的服务器数；为了获得最优的服务器调度顺序，提出了一种高效的全局优化进化算法。实验结果表明：与已有调度算法相比，所提算法能够在分布式平台下最小化任务的完成时间，对于小规模和大规模任务，任务完成时间分别降低了至少25%和43%。

关键词：多趟调度；可分任务；全局优化；进化算法
中图分类号：TP338.8 **文献标志码**：A **文章编号**：0253-987X(2018)08-0080-07

A Periodical Multi-Installment Scheduling Model and Algorithm for Divisible Loads in Distributed Computing Systems

ZHU Hai[1,2], WANG Xiaoli[2], MA Haiming[2]
(1. School of Network Engineering, Zhoukou Normal University, Zhoukou, Henan 466001, China;
2. School of Computer Science and Technology, Xidian University, Xi'an 710071, China)

Abstract: A new periodical multi-installment task-scheduling model is proposed to solve the problem that the existing single-installment task-scheduling models could not take full advantage of the parallel characteristics of distributed platforms, which results in low system utilization and inefficiency of task computation. A closed-form of the solution for an optimal load distribution strategy is firstly derived for a given scheduling sequence of servers. Then, a heuristic algorithm to find optimal numbers of installments and servers involved in workload computation is designed by analyzing the function of finish time with respect to the numbers of installments and servers. An efficient global optimization evolutionary algorithm is proposed to obtain an optimal scheduling sequence of servers. Experiment results and comparisons with existing scheduling algorithms show that the proposed algorithms obtain a minimum finish time of tasks under distributed computing systems. As for relatively small workloads, the proposed algorithm can reduce the finish time of workloads by at least 25% and 43% for relatively small workloads and large-scale workloads, respectively.

Keywords: multi-installment scheduling; divisible load; global optimization; evolutionary algorithm

---

收稿日期：2017-10-24。 作者简介：朱海(1988－)，男，博士，副教授。 基金项目：河南省科技发展计划资助项目(162102310589)；国家自然科学基金资助项目(U1404622,61103143,61402350)。
网络出版时间：2018-06-22 网络出版地址：http://kns.cnki.net/kcms/detail/61.1069.T.20180622.1721.014.html

随着物联网和互联网应用的蓬勃发展，源源不断的大数据涌入各行各业，对高性能计算的需求不断高涨。大规模计算平台如云计算、网格计算、雾计算等应运而生。然而，不管是何种分布式计算平台都离不开高效的任务调度策略[1]。

大数据任务[2]，如大规模信号处理、图像处理、矩阵运算等，虽然数据规模庞大，但是任务往往具有可分性，即可以切分为任意大小的子任务且各子任务之间没有优先关系[3]。可分任务调度模型分为单趟调度和多趟调度两大类，其中单趟调度将任务划分为与服务器具有相同数目的子任务，目标是寻找最优的任务分配方案使得任务的完成时间最短。近年来，针对各类分布式计算平台，如星型网络[4]、多层树网络[5]、高斯网格[6]、云平台[7]和非线性负载的线性网络[8]等，已有文献推导得到了最优或近似最优的任务分配方案的解析解。虽然单趟调度易于实现，但是由于服务器存在较长的等待时间，降低了服务器的利用率和完成任务的效率。

针对这一问题，Bharadwaj 等提出了多趟调度模型，将任务划分为数倍于服务器数的子任务并分多趟分配给各服务器[9]。通过减少服务器的等待时间提高完成任务的效率。多趟调度模型在各类计算平台得到了广泛应用，如星型网络[10]、K 维网[11]、完全 B-Ary 树网[12]和云平台[13]等。与单趟调度模型相比，多趟调度模型求解的复杂度急剧增大。一是因为模型引入了新的变量——调度趟数，二是多趟调度的任务分配方案不再是一个向量而是一个矩阵，其中矩阵的每个元素表示各趟调度中每个服务器的任务分配量。为了得到多趟调度的最优解，Suresh 等提出了一种混合实数编码进化算法[14]，然而当系统规模和调度趟数很大时，算法很难在可容忍的时间范围内收敛到全局最优解。为了得到问题的近似最优解，Yang 等提出了均匀多趟调度模型，将任务分配矩阵简化为向量，并推导得到了任务分配向量的解析解[15]。Shokripour 等对模型进一步优化，提出了周期性多趟调度模型，将调度分为内部调度和最后一趟调度两部分[16]。最后一趟调度用来确保所有服务器同时完成计算，从而进一步降低了任务的完成时间。但是，文献[10-16]并没有找到服务器的最优调度顺序，因此不能最小化任务的完成时间。鉴于此，本文以任务的完成时间最短为目标，建立了可分任务的周期性多趟调度优化模型，并设计了全局优化进化算法对模型进行求解，得到了最优的服务器数、服务器调度顺序、最优调度趟数和最优的任务分配方案。

1　周期性多趟调度优化模型

分布式计算平台由 $M+1$ 台服务器通过星型网络互连，其中 p_0 为主服务器，$\{p_i \mid i\in\{1,2,\cdots,M\}\}$ 为从服务器，l_i 是连接 p_0 和 p_i 的通信链路，总任务大小为 W。p_0 负责划分和调度任务，并不参与任务计算。多趟调度过程分为内部调度和最后一趟调度。设内部调度趟数为 n，则总趟数为 $n+1$。设参与计算的从服务器数为 m，且 $2\leqslant m\leqslant M$。每趟调度所有服务器完成的任务量相同，记为 $V=W/(n+1)$。

多趟调度过程如图 1 所示，纵坐标 $p_{\sigma_1},p_{\sigma_2},\cdots,p_{\sigma_m}$ 表示服务器的调度顺序，其中 $\sigma=(\sigma_1,\sigma_2,\cdots,\sigma_m)$ 是 $1\sim m$ 的全排列。设内部调度和最后一趟调度中分配给服务器 p_{σ_i} 的任务量分别为 $\alpha_{\sigma_i}V$ 和 $\beta_{\sigma_i}V$，其中 $\sum_{i=1}^{m}\alpha_{\sigma_i}=1$ 和 $\sum_{i=1}^{m}\beta_{\sigma_i}=1$。记 o_{σ_i} 为链路 l_{σ_i} 的通信启动开销，$z\sigma_i$ 为 l_{σ_i} 传输单位任务所需要的时间，则 p_0 传输 $\alpha_{\sigma_i}V$ 给 p_{σ_i} 所需时间为 $o_{\sigma_i}+z_{\sigma_i}\alpha_{\sigma_i}V$。设 s_{σ_i} 为 p_{σ_i} 的计算启动开销，w_i 为 p_{σ_i} 计算单位任务所需的时间，则 p_{σ_i} 计算 $\alpha_{\sigma_i}V$ 所需时间为 $s_{\sigma_i}+w_{\sigma_i}\alpha_{\sigma_i}V$。问题在于如何确定最优的 m、n、σ、α 和 β，使得任务的完成时间最短。

1.1　内部调度

为了获得最短的任务完成时间，内部调度中每个服务器在相邻两趟调度之间不应存在空闲时间。由图 1 可以看出，$t_1\sim t_2$ 时间段内，服务器 p_{σ_1} 先后完成了子任务 $\alpha_{\sigma_1}V$ 的计算（$s_{\sigma_1}+w_{\sigma_1}\alpha_{\sigma_1}V$）和传输（$o_{\sigma_1}+z_1\alpha_{\sigma_1}V$），同时 p_{σ_2} 先后完成了子任务 $\alpha_{\sigma_2}V$ 的传输（$o_{\sigma_2}+z_2\alpha_{\sigma_2}V$）和计算（$s_{\sigma_2}+w_{\sigma_2}\alpha_{\sigma_2}V$），因此 $t_1\sim t_2$ 时段内可以建立恒等式关系。以此类推可得

$$\alpha_{\sigma_1}V(z_{\sigma_1}+w_{\sigma_1})+o_{\sigma_1}+s_{\sigma_1}=\alpha_{\sigma_2}V(z_{\sigma_2}+w_{\sigma_2})+o_{\sigma_2}+s_{\sigma_2}=\cdots=\alpha_{\sigma_m}V(z_{\sigma_m}+w_{\sigma_m})+o_{\sigma_m}+s_{\sigma_m} \tag{1}$$

已知

$$\alpha_{\sigma_1}+\alpha_{\sigma_2}+\cdots+\alpha_{\sigma_m}=1 \tag{2}$$

利用式(1)将 α_{σ_i} 用 α_{σ_1} 表示出来并代入式(2)得

$$\alpha_{\sigma_1}+\frac{\alpha_{\sigma_1}V(z_{\sigma_1}+w_{\sigma_1})+o_{\sigma_1}+s_{\sigma_1}-o_{\sigma_2}-s_{\sigma_2}}{V(z_{\sigma_2}+w_{\sigma_2})}+\cdots+\frac{\alpha_{\sigma_1}V(z_{\sigma_1}+w_{\sigma_1})+o_{\sigma_1}+s_{\sigma_1}-o_{\sigma_m}-s_{\sigma_m}}{V(z_{\sigma_m}+w_{\sigma_m})}=1 \tag{3}$$

定义下面 2 个新的变量

$$\Delta_{\sigma_i}=\frac{z_{\sigma_1}+w_{\sigma_1}}{z_{\sigma_i}+w_{\sigma_i}},\quad \Phi_{\sigma_i}=\frac{o_{\sigma_1}+s_{\sigma_1}-o_{\sigma_i}-s_{\sigma_i}}{z_{\sigma_i}+w_{\sigma_i}} \tag{4}$$

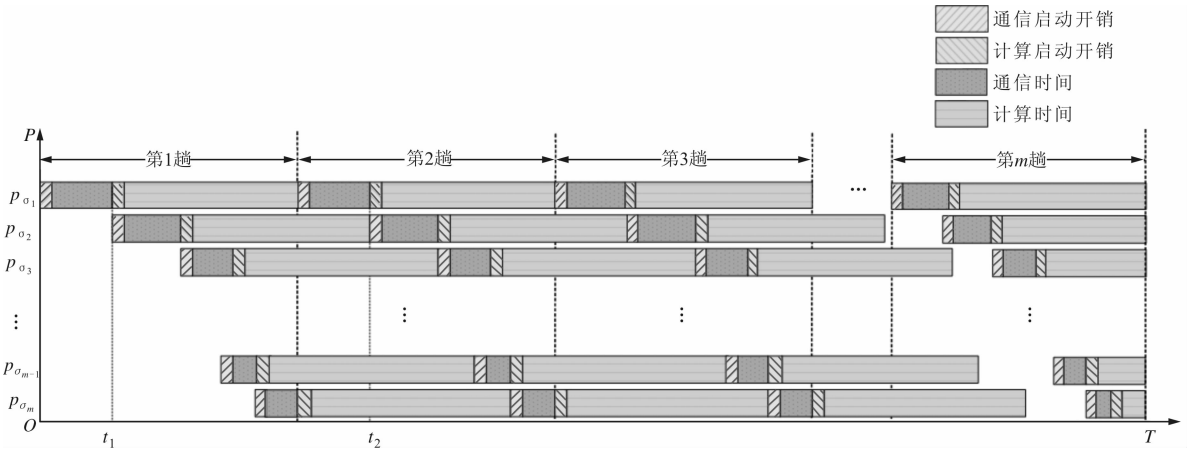


图 1 可分任务的周期性多趟调度

则式(3)可简化为

$$\alpha_{\sigma_1}\left(1+\sum_{i=2}^{m}\Delta_{\sigma_i}\right)+\frac{1}{V}\sum_{i=2}^{m}\Phi_{\sigma_i}=1 \tag{5}$$

由式(1)(5)可得内部调度的任务分配方案为

$$\left.\begin{aligned}\alpha_{\sigma_1}&=\left(1-\frac{1}{V}\sum_{i=2}^{m}\Phi_{\sigma_i}\right)\Big/\left(1+\sum_{i=2}^{m}\Delta_{\sigma_i}\right)\\ \alpha_{\sigma_i}&=\alpha_{\sigma_1}\Delta_{\sigma_i}+\frac{1}{V}\Phi_{\sigma_i},\quad i=2,\cdots,m\end{aligned}\right\} \tag{6}$$

1.2 最后一趟调度

为了得到任务的最短完成时间,最后一趟调度中所有服务器必须同时完成计算,即所有服务器在 T 时刻同时完成计算,因此

$$s_{\sigma_i}+w_{\sigma_i}\beta_{\sigma_i}V=o_{\sigma_{i+1}}+z_{\sigma_{i+1}}\beta_{\sigma_{i+1}}V+s_{\sigma_{i+1}}+w_{\sigma_{i+1}}\beta_{\sigma_{i+1}}V \tag{7}$$

$$\beta_{\sigma_{i+1}}=\frac{s_{\sigma_i}-(s_{\sigma_{i+1}}+o_{\sigma_{i+1}})}{V(w_{\sigma_{i+1}}+z_{\sigma_{i+1}})}+\frac{w_{\sigma_i}}{w_{\sigma_{i+1}}+z_{\sigma_{i+1}}}\beta_{\sigma_i} \tag{8}$$

为了简化公式,定义如下 2 个变量

$$\delta_{\sigma_{i+1}}=\frac{s_{\sigma_i}-(s_{\sigma_{i+1}}+o_{\sigma_{i+1}})}{w_{\sigma_{i+1}}+z_{\sigma_{i+1}}}\text{ 和 }\varepsilon_{\sigma_{i+1}}=\frac{w_{\sigma_i}}{w_{\sigma_{i+1}}+z_{\sigma_{i+1}}} \tag{9}$$

式(8)可简化为

$$\beta_{\sigma_{i+1}}=\frac{1}{V}\delta_{\sigma_{i+1}}+\varepsilon_{\sigma_{i+1}}\beta_{\sigma_i},\quad i=2,\cdots,m \tag{10}$$

定义如下 2 个变量

$$E_{\sigma_i}=\prod_{j=2}^{i}\varepsilon_{\sigma_j}\text{ 和 }\Gamma_{\sigma_i}=\sum_{j=2}^{i}\left(\delta_{\sigma_j}\prod_{k=j+1}^{i}\varepsilon_{\sigma_k}\right) \tag{11}$$

由 $\sum_{i=1}^{m}\beta_{\sigma_i}=1$,递归求解式(10)可得

$$\left.\begin{aligned}\beta_{\sigma_1}&=\left(1-\frac{1}{V}\sum_{i=2}^{m}\Gamma_{\sigma_i}\right)\Big/\left(1+\sum_{i=2}^{m}E_{\sigma_i}\right)\\ \beta_{\sigma_i}&=E_{\sigma_i}\beta_{\sigma_1}+\frac{1}{V}\Gamma_{\sigma_i},\quad i=2,\cdots,m\end{aligned}\right\} \tag{12}$$

1.3 n 和 m 的最优解

由图 1 可得任务的总完成时间为

$$T=n(\alpha_{\sigma_1}V(z_{\sigma_1}+w_{\sigma_1})+o_{\sigma_1}+s_{\sigma_1})+\beta_{\sigma_1}V(z_{\sigma_1}+w_{\sigma_1})+o_{\sigma_1}+s_{\sigma_1} \tag{13}$$

调度的趟数越多,每趟调度分配的任务量就越少,服务器的空闲等待时间就越短。但是,由于平台存在启动开销,调度的趟数越多,额外的开销越多,任务的总完成时间也就越大。任务的完成时间 T 随调度趟数 n 的增加先减小后增加,因此可以通过寻找函数 $T(n)$ 的转折点找到调度趟数的最优解。为此,需要首先确定 n 的上界和下界。由前文的分析可知,对于所有参与调度的服务器有 $\beta_{\sigma_i}>0$。展开式(12)可得

$$\beta_{\sigma_i}=E_{\sigma_i}\beta_{\sigma_1}+\frac{1}{V}\Gamma_{\sigma_i}=E_{\sigma_i}\left(1-\frac{1}{V}\sum_{i=2}^{m}\Gamma_{\sigma_i}\right)\Big/\left(1+\sum_{i=2}^{m}E_{\sigma_i}\right)+\frac{1}{V}\Gamma_{\sigma_i}>0 \tag{14}$$

由 $\left(1+\sum_{i=2}^{m}E_{\sigma_i}\right)>0$ 且 $V>0$,可得

$$E_{\sigma_i}\left(V-\sum_{i=2}^{m}\Gamma_{\sigma_i}\right)+\Gamma_{\sigma_i}\left(1+\sum_{i=2}^{m}E_{\sigma_i}\right)>0 \tag{15}$$

将 $V=W/(m+1)$ 代入式(15),化简得

$$(n+1)\left[E_{\sigma_i}\sum_{i=2}^{m}\Gamma_{\sigma_i}-\Gamma_{\sigma_i}\left(1+\sum_{i=2}^{m}E_{\sigma_i}\right)\right]<E_{\sigma_i}W \tag{16}$$

引入新的符号

$$u_i=\left[E_{\sigma_i}\sum_{i=2}^{m}\Gamma_{\sigma_i}-\Gamma_{\sigma_i}\left(1+\sum_{i=2}^{m}E_{\sigma_i}\right)\right] \tag{17}$$

当 $u_i<0$ 时,$E_{\sigma_i}W>0$ 恒成立。当 $u_i>0$ 时,$n<\frac{E_{\sigma_i}W}{u_i}-1$,其中 $i=2,\cdots,m$。因此,可以令 $i=2,\cdots,m$,找出 $u_i>0$ 时,满足 $n<E_{\sigma_i}W/u_i-1$ 的最小

的 n 值作为 n 的上限，同时将 2 作为 n 的下限。

1.4 σ 的最优解

给定服务器调度顺序 σ 就可以求得最优的 n 和 m，从而可以通过式(6)(12)和(13)推导得到任务分配方案和任务的完成时间 T。为了求得最优的服务器调度顺序，建立以下优化模型

$$\min_{\sigma} T = \min\left[n\left(\alpha_{\sigma_1}V\left(z_{\sigma_1}+w_{\sigma_1}\right)+o_{\sigma_1}+s_{\sigma_1}\right)+\beta_{\sigma_1}V\left(z_{\sigma_1}+w_{\sigma_1}\right)+o_{\sigma_1}+s_{\sigma_1}\right]$$

$$\text{s.t.}\begin{cases}(1)\,2\leqslant m\leqslant M \text{ 且 } n\geqslant 2\\ (2)\begin{cases}\alpha_{\sigma_1}=\left(1-\dfrac{1}{V}\sum\limits_{i=2}^{m}\Phi_{\sigma_i}\right)\Big/\left(1+\sum\limits_{i=2}^{m}\Delta_{\sigma_i}\right)\\ \alpha_{\sigma_i}=\alpha_{\sigma_1}\Delta_{\sigma_i}+\dfrac{1}{V}\Phi_{\sigma_i},\quad i=2,\cdots,m\end{cases}\\ (3)\begin{cases}\beta_{\sigma_1}=\left(1-\dfrac{1}{V}\sum\limits_{i=2}^{m}\Gamma_{\sigma_i}\right)\Big/\left(1+\sum\limits_{i=2}^{m}E_{\sigma_i}\right)\\ \beta_{\sigma_i}=E_{\sigma_i}\beta_{\sigma_1}+\dfrac{1}{V}\Gamma_{\sigma_i},\quad i=2,\cdots,m\end{cases}\\ (4)\alpha_{\sigma_i}>0 \text{ 且 } \beta_{\sigma_i}>0,\text{其中 } i=2,\cdots,m\\ (5)\sum\limits_{i=1}^{m}\alpha_{\sigma_i}=1 \text{ 和} \sum\limits_{i=1}^{m}\beta_{\sigma_i}=1\\ (6)\sigma=(\sigma_1,\sigma_2,\cdots,\sigma_m)\text{是 1 到 } m \text{ 的全排列}\end{cases}$$

2 全局优化进化算法

所提模型仅含有一组变量 $\sigma=(\sigma_1,\sigma_2,\cdots,\sigma_m)$，即服务器的调度顺序，它是 $1\sim m$ 的一个全排列。可见，该问题属于组合优化问题。当系统规模较大时，传统优化算法很难在可容忍的时间范围内找到问题的全局最优解或局部最优解，往往需要借助智能优化算法对问题进行求解，而进化算法作为智能优化算法类的杰出代表，非常擅长于求解组合优化问题。因此，本文设计了一个新的全局优化进化算法求解多趟调度优化模型。

2.1 编码与适应度函数

本文直接对模型的变量进行编码，即个体由 $I=(\sigma)$ 来表示，其中 σ 表示服务器的调度顺序。注意，为满足模型的约束条件，个体的编码必须包含所有参与计算的服务器编号且每个服务器编号仅能包含一次。任何服务器编号的缺失或者重复都将导致产生非法个体。

随机产生 N 个个体作为初始种群。每产生一个个体，也就给定了一种服务器的调度顺序，进而可以求得当前处理机调度顺序下最优的 n 和 m，然后通过式(6)和式(12)可以推导得到任务的分配方案，最后通过式(13)求得 T。将 T 的倒数作为个体的适应度值，T 越小，个体的适应度值越大，表明个体越优秀。

2.2 交叉与变异

本文采用部分匹配的交叉方法。对于任意两个父代个体 I_1 和 I_2

$I_1=(12,10,6,9,4,1,13,0,11,5,3,7,14,2,8)$

$I_2=(14,12,10,8,4,11,6,13,5,2,1,0,9,3,7)$

交叉的具体过程如下。

步骤 1　随机选取两个交叉点 k_1 和 k_2，要求 $k_1<k_2$，例如 $k_1=5$ 和 $k_2=11$。建立 k_1 和 k_2 交叉点之间个体 I_1 和 I_2 匹配区内基因的对应关系：$1\leftrightarrow 11$，$13\leftrightarrow 6$，$0\leftrightarrow 13$，$11\leftrightarrow 5$，$5\leftrightarrow 2$，$3\leftrightarrow 1$，$7\leftrightarrow 0$。

步骤 2　交换匹配区两父代个体 I_1 和 I_2 对应的基因位可得两个子代个体 I'_1 和 I'_2

$I_1=(12,10,6,9,4,\overset{k_1}{|}1,13,0,11,5,3,7,\overset{k_2}{|}14,2,8)$

$I_2=(14,12,10,8,4,|11,6,13,5,2,1,0,|9,3,7)$

$I'_1=(12,10,6,9,4,|11,6,13,5,2,1,0,|14,2,8)$

$I'_2=(14,12,10,8,4,|1,13,0,11,5,3,7,|9,3,7)$

观察 I'_1 和 I'_2 可以看出，两个个体均存在重复和缺失的处理机编号，因此是非法个体，需要修正。

步骤 3　按照匹配区内父代各基因位上的对应关系替换匹配区外的重复位。例如，个体 I'_1 在匹配区外的左侧有重复编码 6。按照步骤 1 给出的对应关系，6 换为 13。编码 13 依然与匹配区内编码重复，继续按照对应关系将 13 换为 0。编码 0 依然重复，因此继续按对应关系将 0 换为 7，没有重复，结束替换。个体 I'_1 和 I'_2 替换后可以得到如下两个合法的子代个体 I''_1 和 I''_2

$I''_1=(12,10,7,9,4,11,6,13,5,2,1,0,14,3,8)$

$I''_2=(14,12,10,8,4,1,13,0,11,5,3,7,9,2,6)$

交叉产生的后代个体以概率 p_s 进行变异，本文采用对换变异的方法。对某一后代个体，随机选取两个变异位 k_1 和 k_2，要求 $k_1<k_2$，交换这两个位置对应的基因。例如，对如下个体 p，取 $k_1=5$ 和 $k_2=11$

$p=(12,10,6,9,\overset{k_1}{4},1,13,0,11,5,\overset{k_2}{3},7,14,2,8)$

交换这两个位置的基因可以得到变异后的个体

$p'=(12,10,6,9,3,1,13,0,11,5,4,7,14,2,8)$

2.3 局部搜索算子

考虑到高性能分布式平台的系统规模较为庞大，为了加快算法的收敛速度，本文为进化算法引入局部搜索算子。其基本思想是对于任意一个个体 I，计算个体的适应度值 $f(I)$，搜索个体的邻域空间，即按序交换其编码相邻两位基因，更新个体 I'，

计算个体 I' 的适应度值 $f(I')$，若新个体 I' 的适应度优于原来的个体 I，即 $f(I')>f(I)$，则用新个体 I' 替换原来的个体 I，并继续在个体 I' 的邻域空间内执行搜索；否则，不替换。

例如，对于个体 $p=(12,10,6,9,4,1,13,0,11,5,3,7,14,2,8)$，局部搜索的步骤为：交换第 1 位基因 12 和第 2 位基因 10，得到新的个体 $p'=(10,12,6,9,4,1,13,0,11,5,3,7,14,2,8)$。计算个体 p 和 p' 的适应度值，如果 p' 的适应度值大于 p 的适应度值，即 p' 的任务完成时间小于 p 的任务完成时间，则交换这两个基因位，否则，不交换。然后，对个体 p 的第 2 位和第 3 位基因执行相同的操作，直到最后两位完成试探交换。

2.4 全局优化进化算法

本文提出多趟调度全局优化进化算法的步骤如下。

步骤 1 初始化。产生初始种群 $P(t)$，设置代数 $t=0$。

步骤 2 交叉。对于种群 $P(t)$ 中的每个个体 I_j，$j=1,2,\cdots,N$，计算适应度值，轮盘赌选择 N 个个体存入交叉池。对交叉池中的 $N/2$ 对个体按照交叉概率 p_c 进行部分匹配交叉，后代个体的集合记为 O_1，后代个体的数量记为 R_1。

步骤 3 变异。对于 $P(t)\cup O_1$ 中的每个个体 I_j，$j=1,2,\cdots,N+R_1$，随机产生实数 $r\in[0,1]$，若 $r<p_s$，对个体进行对换变异操作，变异所得个体的集合记为 O_2，后代个体的数量记为 R_2。

步骤 4 局部搜索。对 $P(t)\cup O_1\cup O_2$ 中的每个个体 I_j，$j=1,2,\cdots,N+R_1+R_2$，进行局部搜索操作，更新后的个体集合记为 O_3。

步骤 5 选择。首先根据精英保留策略从 $P(t)\cup O_3$ 中选出 E 个最佳个体直接保存到下一代 $P(t+1)$ 中，然后通过轮盘赌方法在 $P(t)\cup O_1\cup O_2$ 中选择剩下的 N-E 个个体放入 $P(t+1)$ 中。

步骤 6 终止条件。若满足终止条件，终止算法并输出当前种群的最优个体。否则，跳到步骤 2。

3 实验与结果分析

将本文算法与文献[10-16]采用的 IZ 和 IW 方法以及 Hsu 等提出的 ESCR 算法[17]进行了比较。平台含有 15 台服务器，每台服务器的相关参数见表 1。本文算法的相关参数设置如下：种群大小 $N=100$，交叉概率 $p_c=0.6$，变异概率 $p_s=0.02$，精英保留数 $E=5$，终止条件 $t=200$。若增加平台服务器的数量，可仅将终止条件的进化代数适量调大，其他参数保持不变。给定测试任务大小 $W\in\{100,500,1\,000,5\,000,10\,000\}$，表 2 给出了本文、IZ、IW 和 ESCR 等 4 种调度算法的对比实验结果。由表 2 可以看出，给定任务量，4 种调度算法求得的最优调度趟数 n、参与计算的处理机数 m 和处理机调度顺序 σ 各不相同，因此任务的分配方案和最终获得的任务完成时间也各不相同。对任一任务，与其他调度算法相比，本文算法都能获得最短的任务完成时间，可见模型和算法的有效性。

表 1 分布式系统的参数

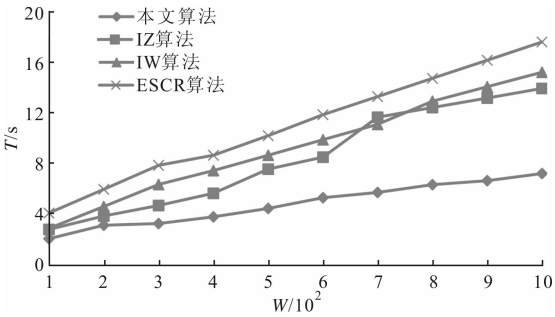
P	o	s	z	w
p_1	70.55	57.95	0.53	2.90
p_2	30.19	1.40	0.77	7.61
p_3	81.45	4.54	0.71	4.14
p_4	86.26	37.35	0.79	9.62
p_5	87.14	94.96	0.06	3.64
p_6	52.49	5.35	0.77	5.92
p_7	46.87	62.27	0.30	6.48
p_8	26.38	82.98	0.28	8.25
p_9	58.92	91.10	0.99	2.27
p_{10}	69.51	24.39	0.98	5.34
p_{11}	10.64	67.62	1.01	1.57
p_{12}	57.52	10.30	0.10	7.99
p_{13}	28.45	29.58	0.05	3.82
p_{14}	30.09	97.98	0.95	4.01
p_{15}	27.83	16.28	0.16	6.47

为了更直观地体现各算法的性能差异，图 2 给出了 4 种调度算法处理不同数量级任务的完成时间随任务量的变化曲线，其中图 2a 针对小规模任务 $W\in[100,1\,000]$，图 2b 针对中规模任务 $W\in[1\,000,10\,000]$，图 2c 针对大规模任务 $W\in[10\,000,100\,000]$。从图 2a 可以看出，当任务量相对较小时，本文算法能够获得最短的任务完成时间，IW 与 IZ 算法并没有明显的优劣关系，而 ESCR 算法的任务完成时间最大。这是因为 ESCR 算法并没有令所有服务器同时完成任务计算，而且即便任务量较小的情况下也令所有的服务器都参与任务的计算，导致服务器的空闲等待时间较长，既降低了系统的资源利用率又没有最大化任务的执行效率。与已有算法相比，本文算法降低了至少 25% 的任务完成时间。从图 2b 可以看出，当任务量逐渐增大时，ESCR 和 IW 算法的时间性能并没有明显的优劣之分，而 IZ 和本文算法明显优于其他 2 种算法，本文

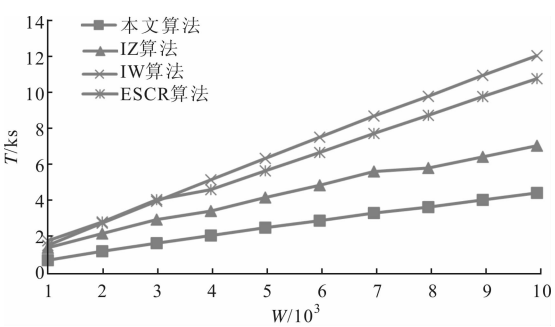
表 2　4 种算法的实验结果比较

算法	W	n	m	$\sigma=(\sigma_1,\sigma_2,\cdots,\sigma_m)$	T/s
本文	100	1	4	12,10,14,1,8,0,11,3,5,2,13,4,7,6,9	214
IZ		1	4	12,4,11,14,7,6,0,2,5,1,3,13,9,8,10	287
ESCR		1	15	3,7,1,11,6,14,5,9,2,13,4,12,0,8,10	416
IW		1	2	10,8,0,4,12,13,2,9,5,14,6,1,11,7,3	295
本文	500	1	10	14,12,10,0,13,8,4,11,7,6,2,1,5,3,9	453
IZ		2	6	12,4,11,14,7,6,0,2,5,1,3,13,9,8,10	763
ESCR		1	15	3,7,1,11,6,14,5,9,2,13,4,12,0,8,10	1 027
IW		2	3	10,8,0,4,12,13,2,9,5,14,6,1,11,7,3	873
本文	1 000	3	15	14,12,10,0,13,8,4,2,1,5,11,7,3,6,9	728
IZ		2	7	12,4,11,14,7,6,0,2,5,1,3,13,9,8,10	1 401
ESCR		1	15	3,7,1,11,6,14,5,9,2,13,4,12,0,8,10	1 768
IW		3	3	10,8,0,4,12,13,2,9,5,14,6,1,11,7,3	1 529
本文	5 000	3	15	14,12,4,10,6,0,1,8,11,7,2,5,3,13,9	2 474
IZ		4	8	12,4,11,14,7,6,0,2,5,1,3,13,9,8,10	4 103
ESCR		1	15	3,7,1,11,6,14,5,9,2,13,4,12,0,8,10	5 537
IW		7	3	10,8,0,4,12,13,2,9,5,14,6,1,11,7,3	6 197
本文	10 000	4	15	14,12,4,0,11,6,2,5,13,8,7,1,3,9,10	4 343
IZ		5	9	12,4,11,14,7,6,0,2,5,1,3,13,9,8,10	6 878
ESCR		1	15	3,7,1,11,6,14,5,9,2,13,4,12,0,8,10	10 494
IW		10	3	10,8,0,4,12,13,2,9,5,14,6,1,11,7,3	11 722

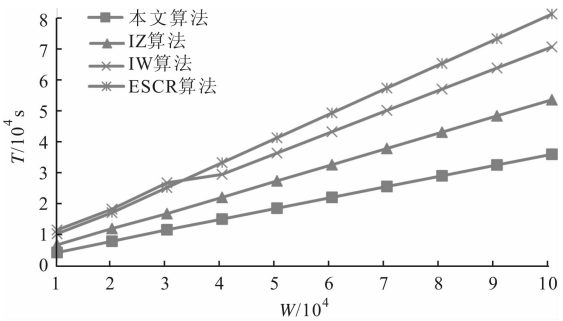
算法时间性能最佳。从图 2c 可以看出，当任务量较大时，各调度算法的时间性能差异随着任务量的增大差距越来越大。ESCR 算法获得的任务完成时间最大(最差)，其次是 IW 和 IZ 算法，本文算法在时间性能上有明显的优势，能够最小化任务的完成时间。与已有算法相比，本文算法降低了至少 43% 的任务完成时间。



(a)任务量从 100 到 1 000



(b)任务量从 1 000 到 10 000



(c)任务量从 10 000 到 100 000

图 2　4 种算法的时间性能对比

4　结　论

随着大数据时代的到来，对企业而言，数据即生产力，如何提高数据的处理效率受到越来越多的关注。本文以任务的完成时间最短为目标，建立了可分任务的周期性多趟调度优化模型，并设计了全局

优化进化算法对模型进行求解，获得了最优的调度趟数、最优的参与计算的服务器数、最优的任务分配方案和最优的处理机调度顺序。实验结果表明，与已有调度算法相比，本文算法在任务执行效率方面提升了至少25%的性能。

参考文献：

[1] 孙健，张兴军，董小社. 异构系统中带可用性约束的性能优化调度算法 [J]. 西安交通大学学报，2018，52(2)：18-23.
SUN Jian，ZHANG Xingjun，DONG Xiaoshe，A performance optimization scheduling strategy with availability constraints for heterogeneous systems [J]. Journal of Xi'an Jiaotong University，2018，52(2)：18-23.

[2] IYER G N，VEERAVALLI B，KRISHNAMOORTHY S G. On handling large-scale polynomial multiplications in compute cloud environments using divisible load paradigm [J]. IEEE Transactions on Aerospace & Electronic Systems，2012，48(1)：820-831.

[3] MILLOT D，PARROT C. Optimization of the processing of data streams on roughly characterized distributed resources [J]. IEEE Transactions on Parallel & Distributed Systems，2016，27(5)：1415-1429.

[4] 赖俊凡，王宇平，王晓丽. 考虑处理机时间窗口的可分任务调度优化模型 [J]. 西安交通大学学报，2017，51(9)：118-124.
LAI Junfan，WANG Yuping，WANG Xiaoli，An optimization model for divisible-load scheduling considering processor time-window [J]. Journal of Xi'an Jiaotong University，2017，51(9)：118-124.

[5] GHANBARI S，OTHMAN M，BAKAR M R A，et al. Multi-objective method for divisible load scheduling in multi-level tree network [J]. Future Generation Computer Systems，2016，54(C)：132-143.

[6] 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.

[7] SINGH S，CHANA I. Cloud resource provisioning：survey，status and future research directions [J]. Knowledge & Information Systems，2016，49(3)：1-65.

[8] CHEN C Y，CHU C P. A novel computational model for non-linear divisible loads on a linear network [J]. IEEE Transactions on Computers，2016，65(1)：53-65.

[9] BHARADWAJ V，GHOSE D，MANI V. Multi-installment load distribution in tree networks with delays [J]. IEEE Transactions on Aerospace and Electronic Systems，1995，31(2)：555-567.

[10] BRANDÃO J S，NORONHA T F，RESENDE M G C，et al. A biased random-key genetic algorithm for scheduling heterogeneous multi-round systems [J]. International Transactions in Operational Research，2017，24(5)：1061-1077.

[11] CHANG Y K，WU J H，CHEN C Y，et al. Improved methods for divisible load distribution on k-dimensional meshes using multi-installment [J]. IEEE Transactions on Parallel & Distributed Systems，2007，18(11)：1618-1629.

[12] 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.

[13] ZHAO T，JING M. Bandwidth-aware multi round task scheduling algorithm for cloud computing [J]. Journal of Intelligent & Fuzzy Systems，2016，31(2)：1053-1063.

[14] SURESH S，HUANG H，KIM H J. Hybrid real-coded genetic algorithm for data partitioning in multi-round load distribution and scheduling in heterogeneous systems [J]. Applied Soft Computing，2014，24(C)：500-510.

[15] YANG Y，RAADT K V D，CASANOVA H. Multi round algorithms for scheduling divisible loads [J]. IEEE Transactions on Parallel & Distributed Systems，2005，16(11)：1092-1102.

[16] SHOKRIPOUR A，OTHMAN M，IBRAHIM H，et al. A method for scheduling heterogeneous multi-installment systems [J]. Future Generation Computer Systems，2012，28(8)：1205-1216.

[17] HSU C H，CHEN T L，PARK J H. On improving resource utilization and system throughput of master slave job scheduling in heterogeneous systems [J]. Journal of Supercomputing，2008，45(1)：129-150.

（编辑 武红江）