分布式计算中基于资源分级的自适应 Min – Min 算法*

巩子杰[a]，张亚平[a]，张铭栋[b]
(天津大学 a. 软件学院; b. 环境科学与工程学院，天津 300072)

摘 要: Min-Min 任务调度算法的思路总是优先调度执行时间较短的小任务，无法得到理想的最优跨度及资源负载平衡。针对该问题，提出基于资源分级的自适应 Min-Min 算法。分配任务前，先参考现有资源的属性进行分级处理，再与任务在资源中的最小完成时间作乘积得到的最小任务资源组合进行调度; 在任务调度过程中，引入自适应阈值，调节长任务的调度等级，从而达到优化效果。通过模拟仿真实验，表明该算法在时间跨度和负载平衡上均有较好性能。

关键词: 分布式计算; 任务调度; Min-Min 算法; 资源分级; 负载平衡
中图分类号: TP301.6 **文献标志码:** A **文章编号:** 1001-3695(2016)03-0716-04
doi:10.3969/j.issn.1001-3695.2016.03.018

Adaptive Min-Min algorithm based on classification of resources in distributed computing

Gong Zijie[a], Zhang Yaping[a], Zhang Mingdong[b]
(*a. School of Computer Software, b. School of Environmental Science & Engineering, Tianjin University, Tianjin* 300072, *China*)

Abstract: The Min-Min task scheduling algorithm in distributed computing is simple and clear, which is always the priority scheduling perform small tasks for a short time, unable to make in respect of the optimal span and resource load balancing the ideal balance. In order to solve this problem, this paper proposed an adaptive Min-Min task scheduling algorithm based on resource classification. Firstly, it classified the existing resource according to its own properties, and then multiplied the minimum completion time by the task in the resources to get the optimal combination of tasks and resources. Secondly, it adjusted the long task priorities to achieve good performance with adaptive threshold valve. The simulation experiment results show that the algorithm has obvious improvement on the time span and load balancing.

Key words: distributed computing; task scheduling; Min-Min algorithm; resource classification; load balancing

0 引言

随着互联网技术的飞速发展，如今已经进入以网络为核心的发展时代。网络用户量的急剧增加以及网络服务的多元化都对服务器的性能带来了巨大的挑战。众多的研究人员，将研究方向转向分布式计算，来解决服务器的性能问题。通过分解任务，将子任务分配给若干个计算机进行计算，并将各个子任务的计算结果返回到主计算机进行综合计算分析得到最后的结果[1~3]。

从分布式计算的基本思想不难发现，庞大数量的计算任务使得任务调度及资源分配的问题显得尤为突出。目前，针对分布式计算中任务调度的问题，国内外也在相关领域进行了研究，并提出多种静态和动态的调度算法。目前，主要已有的算法是遗传算法、Min-Min 算法、Max-Min 算法、Sufferage 算法等[4~7]。传统调度算法研究的重点集中在如何在较短的时间内完成任务，可以快速地进行任务的调度。实际中，大量的资源是异构并且变化的，不同资源的计算能力也不相同，并且子任务之间存在竞争关系，这使得任务调度成为一项极其复杂的工作，传统的任务调度算法受到很大的限制。

本文通过分析 Min-Min 算法，从资源分类及任务调度过程入手，提出一种基于资源分级的自适应任务调度算法，能较好地保证任务调度的快速性和负载平衡性。

1 Min-Min 算法

1.1 Min-Min 算法的原理

传统的 Min-Min 算法非常简单，调度速度较快，基本思想是: 优先将小任务分配到执行时间最短的资源上，追求单个任务时间最短。算法运行过程是: 将子任务与各资源进行一一映射并计算出完成时间，从中找出每个子任务最早完成时间及对应资源; 再从中找出最小最早完成时间的任务进行分配，任务完成后将其从任务集合中删除。重复此过程，直到任务集为空[8~10]。

令任务集为 T，其中 T_i 表示第 i 个任务，基于执行时间最短的资源调度原则得到资源集为 H，其中 H_i 表示资源 i，执行时间 ETC(expected time to compute) 为任务执行时间与任务等待时间(资源准备时间) 之和。任务集 T 与资源集 H 形成一个 $m \times n$ 的矩阵 ETC，再从 ETC 搜索最小执行时间的任务—资源组合，建立 MCT(minimum completion time) 矩阵，MMCT 为

收稿日期: 2014-11-15; 修回日期: 2015-01-09 基金项目: 国家自然科学基金资助项目(60776807)

作者简介: 巩子杰(1988-)，男，山西太原人，硕士研究生，主要研究方向为信息安全(gzjsx_2009@163.com); 张亚平(1973-)，男，副教授，博士，主要研究方向为信息安全、云计算; 张铭栋(1990-)，男，硕士研究生，主要研究方向为计算机模拟随机水质.

遍历 *MCT* 矩阵中的最小值。

Min-Min 算法的伪代码如下：

```
while( 集合 T 不为空)
{
  for( 集合 T 中的每个任务)
  {
    通过计算得到预计执行时间 ETC 矩阵;
    遍历矩阵 ETC 找出最小完成时间 MCT 矩阵;
    遍历矩阵 MCT 确认最短执行时间的任务;
    选取该任务 T_s 和资源 H_j;
    分配任务 T_i 给资源 H_j 完成;
    删除任务 T_i，并更新 ETC;
  }
}
```

1.2　Min-Min 算法的不足

Min-Min 算法的核心思想是一种贪心算法，选取最早完成执行时间最短的任务进行分配，视为最优的选择。这种做法优先考虑短任务，只有当机器空闲的时候才调度长任务，显然这样的做法无法保证短任务和长任务并发进行，进而使得总任务的执行时间较长。所以，在复杂的分布式计算环境中，单纯地考虑某一任务完成的时间因素是远远不够的。

可见，传统的 Min-Min 算法主要有以下的两个问题：

a) 任务资源分配不合理，利用率不高，当面对复杂的任务情况时会出现负载不平衡的问题；

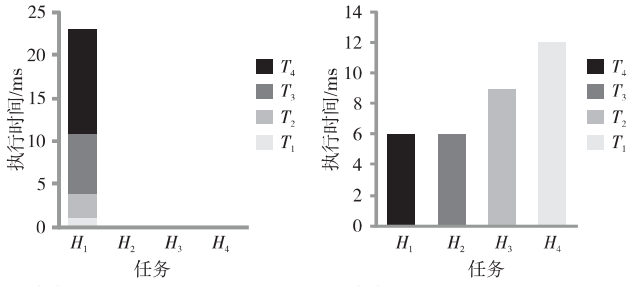
b) 由于任务总是被调度到执行时间较长的机器上，任务等待时间增加，总体执行时间也会随之增加，不是较好的选择。

为此，本文通过一个简单的例子可以比较清楚地理解 Min-Min 算法的两个主要问题。表 1 表示各任务在各资源上的执行情况，选取 Min-Min 算法完成任务调度。

表 1　作业任务在资源集上的执行时间

任务	在 H_1 上的执行时间/ms	在 H_2 上的执行时间/ms	在 H_3 上的执行时间/ms	在 H_3 上的执行时间/ms
T_1	1	3	5	6
T_2	3	4	6	7
T_3	7	9	10	11
T_4	12	13	15	14

基于 Min-Min 算法的任务调度方案，逐次调度任务 T_1、T_2、T_3、T_4，并且每个人都调度给资源 H_1 进行执行。显然，这个调度方案增加了调度跨度，并且造成了严重的资源浪费，使得资源负载不平衡。最优的调度方案是将任务 T_1 调度给 H_4，将任务 T_2 调度给 H_3，将任务 T_3 分配给 H_2，将任务 T_4 分配给 H_1。通过图 1 的显示可知，当前方案相对于 Min-Min 算法有明显的改进。



(a) Min-Min 算法的执行时间　　(b) 最优调度方案的执行时间

图 1　Min-Min 算法和最优调度方案的执行时间对比图

基于上面的分析，本文提出一种方案针对 Min-Min 算法中负载不平衡的问题和最优调度跨度的问题进行改进优化。从资源本身因素角度考虑，先对资源进行分类，然后再调度任务；从任务长短因素角度考虑，将其长度具体化，设定自适应阈值，调解长短任务的调度优先级，提出一种基于资源分类的自适应任务调度算法(reservation category self-Min-Min，RCSMM)。

2　基于资源分级的自适应 Min-Min 任务调度算法

2.1　建立基于资源分级的任务调度模型

从资源本身因素角度考虑，先对资源进行分级，然后再调度任务，基于这样的思想，首先建立资源分级调度模型。由于需要对计算资源预先分级，所以在执行调度任务之前，将资源载入分级模块，按其属性进行分级；然后将分级信息载入调度模块调用自适应 Min-Min 算法输出结果。调度模型如图 2 所示。该调度模型的工作模式如下：

a) 输入资源集属性，资源分级器对资源进行分级输出结果，提交至资源管理模块存储分级结果；

b) 调用资源分级结果，输入计算任务分解的 n 个子任务至任务调度模块；

c) 任务调度模块调用改进算法完成任务—资源的分配。

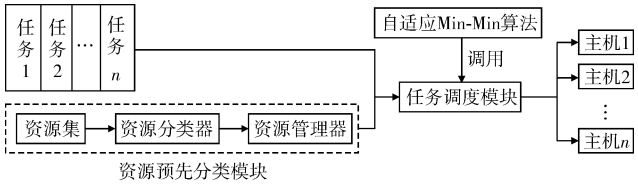


图 2　基于资源分级的任务调度模型

2.2　资源分级方法

分布式计算的实际应用中涉及不同的服务类型，主要的类型为数据密集型和交互密集型。不同的服务类型，在任务调度的过程中采取的策略和评价指标也不同。本文主要针对计算型的任务调度进行算法的改进优化。为了较好地利用分布式计算平台中的资源，特别是计算能力较强的资源，先对资源进行划分级别，然后在调度任务分配算法的时候将执行时间与资源级别相结合，同时保证完工时间较短及资源负载平衡。

对计算资源在分布式计算平台中自身属性进行评估分析，选取可以体现其计算能力和通信能力的属性进行标志，采用计算资源识别度 δ 这个指标对计算资源进行分级。计算资源的重要属性包括：

a) 衡量资源计算能力的属性：CPU 处理能力 P，CPU 数量 n，内存性能 R，磁盘容量 V(MB)。

b) 衡量资源通信能力的属性：通信带宽 B(Mbps)。

计算资源识别度 δ 的计算公式如下：

$$\delta = o \times P \times n + p \times V + q \times B + R \times y \tag{1}$$

式中：o、p、q、y 为各资源属性参数的比例系数，考虑到针对计算型的任务进行调度，取值 $o = 30\%$、$p = 10\%$、$q = 30\%$、$y = 30\%$。

依据得到的资源识别度 δ 进行分类，按照 δ 的值从大到小排列，分为 5 级，0 ~ 30% 为第 1 级，31% ~ 60% 为第 2 级、61% ~ 80% 为第 3 级、81% ~ 90% 为第 4 级、91% ~ 100% 为第 5 级。

2.3　自适应 Min-Min 任务调度算法

从任务长短因素角度考虑，将其长度具体化，设定自适应阈值，调解长短任务的调度优先级。本文将重点参考长任务在

所有子任务中的比例，设定自适应阈值的大小，然后通过自适应阈值重新划分长任务的优先级，从而得到改进的自适应 Min-Min 任务调度算法(SMM)。

2.3.1 相关定义

定义1 设定 $w[i]$ 表示任务 $s[i]$ 的平均预测执行时间，即 *ETC* 矩阵中的第 i 行的平均值。由 $w[i]$ 组成向量 $W=\{w[1],w[2],\cdots,w[n]\}$。令 $w_{\min}$ 为 W 中最小值、$w_{\max}$ 为 W 中最大值、$\overline{w}$ 为 W 中的平均值。其中，$\overline{w}=\frac{1}{n}\sum_{i=1}^{n}w[i]$。

定义2 设 $T_1=w_{\max}-\overline{w}>0$，$T_2=\overline{w}-w_{\min}>0$；当 $T_1>T_2$ 时，令 $A=T_1$，$B=T_2$；当 $T_1<T_2$ 时，令 $B=T_1$，$A=T_2$；记 $M=\frac{A}{B}$。定义任务 $s[i]$ 的平均预测执行时间为 $t[i]$，当 $t[i]>M\cdot\overline{t}$ 时，则称 $s[i]$ 为长任务；当 $t[i]\leqslant M\cdot\overline{t}$ 时，则称 $s[i]$ 为一般任务。

定义3 设集合 S 为待分配任务集，集合 H 表示被分配的所有子任务集，则 $S\subseteq H$；设集合 S 的元素个数为 p，则 p 为未分配的任务数。于是长任务所占比例 $I=\frac{q}{p}$，其中 q 为长任务的个数。定义自适应阈值为 $\zeta=kI$，其中 k 为常量，且 $k>0$。

接下来考察长任务定义的合理性，即按此定义能否找出数量符合实际情况的长任务。本文从下面两个特殊的情况来予以讨论：

a) 当 $\lim(w_{\min}-\overline{w})=0$，此时 I 值较小，且 $M=\frac{A}{B}>1$，依定义2可找到长任务，并且 M 值偏大，故长任务的数量 q 较少，即可按照定义2进行处理。

b) 当 $\lim(w_{\max}-\overline{w})=0$，此时 $M=\frac{A}{B}>1$，依定义2无法找到长任务。此时可直接用 Min-Min 算法进行调度，而没有必要区分长任务与一般任务。

2.3.2 RCSMM 算法描述

基于资源分级的方法以及上述自适应 Min-Min 任务调度方法的分析，RCSMM 算法的运行过程描述如下：

首先建立需要计算的子任务序列，将子任务在所有资源上的执行时间与资源等级作乘积运算，即 $t_{\text{update}}=t\times\delta$，从而得到基于资源分级后的 *RETC* 矩阵和向量 W；初始化 *MCT*，以及总体任务分解后的子任务集 T，任务集 T 映射到待分配任务集 S，MCT 矩阵中的每个元素为对应作业任务在各机器上的预测执行时间与资源就绪时间之和，即 $\text{MCT}[i,j]=\text{RETC}[i,j]+r[j]$，其中 $r[j]$ 为资源就绪时间。

a) 循环取值从第1行到第 n 行，计算 *RETC* 矩阵中每行的均值，将均值结果存储到向量 W 中，即 $w[i]$ 表示任务 $s[i]$ 的平均预测执行时间。

b) 判断：如果待分配任务集 S 不为空，完成步骤 b) ~ i)；否则任务调度结束。

c) 从向量 W 找出最大值 $w_{\max}$、最小值 $w_{\min}$，并计算出均值 w 和向量元素数量 p。

d) 参考定义，从子任务集找出长任务，并确定数量 q。判断：如果存在长任务，即 $q>0$，则跳转到步骤 e)；如果不存在长任务，即 $q=0$，则跳转到步骤 g)。

e) 计算得到长任务占整个任务集的比例 $k=q/p$；

f) 得到自适应阈值 $\zeta=kI$，在 *MCT* 矩阵中对长任务所占的行作乘 ζ 的运算。

g) 遍历矩阵 *MTC* 找出最小完成时间矩阵 *MMCT*；

h) 遍历矩阵 *MMCT* 确认当前最短执行时间的任务，记录该任务并将其调度到指定的机器上完成计算任务；

i) 将该任务从未分配的子任务集中删除，更新资源就绪时间 $r[j]$、W 和 *MCT*。跳转到 b)。

图3为 RCSMM 算法执行流程图。

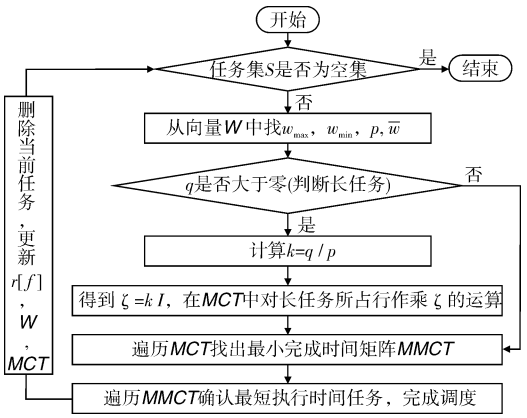


图3 RCSMM 算法执行流程图

3 实验验证及结果分析

3.1 模拟计算环境

为了验证本文所提出算法的有效性，选择仿真软件 GridSim 模拟实际的计算环境，并设计实验进行算法验证。仿真实验是通过量化评价指标，给出 RCSMM 和 Min-Min 算法的性能评价。

该模拟实验的硬件环境：CPU Intel Core i5-3210M 2.4 GHz、8 GB 内存、500 GB 硬盘。

该模拟实验的操作系统：Windows 7 Ultimate。

该模拟实验的软件环境：Visual Modeler，Eclipse；其中 Visual Modeler 用来配置分布式计算环境并产生相应的代码，Eclipse则为实现过程的开发工具。

3.2 评价指标

本实验将针对时间跨度 Makespan 和负载平衡 Balance 两个任务分配算法的重要指标来评价改进算法的性能。参照下面两个公式，进行测试实验：

$$\text{Makespan}=\max\{g[1],g[2],\cdots,g[n]\} \tag{2}$$

$$\text{Balance}=1-\frac{\sum_{i=1}^{u}\left|B[i]-\frac{n}{u}\right|}{\left(n-\frac{n}{u}\right)+\frac{n}{u(u-1)}} \tag{3}$$

式(2)(3)中，n 为任务总数，u 为资源总数。

在3.1节中的实验环境下，从每个模拟的计算节点中找到从开始到完成该资源上最后一个任务的时间跨度，然后从所有计算节点中找出时间跨度的最大值，如式(2)中 $g[i]$ 为对应任务 i 的完成时间跨度，从所有任务中找到最大值，即为整个任务的时间跨度值。式(3)中，$B[i]$ 为对应资源 i 上分配的任务数。

评价原则：

a) $\text{Makespan}\in(0,+\infty)$，Makespan 值越小性能越优越。

b) $\text{Balance}\in[0,1]$，Balance 值越大性能越优越。

3.3 实验设计

仿真软件 GridSim 是基于离散事件的模拟器[9,11]，通过映

射器可以很好地建立用户任务与资源之间的关系，将任务分配给对应的资源，并完成计算任务，反馈执行结果。根据上述的算法结构，结合工具中提供现有的函数，完成对模拟实验的架构搭建及代码实现，执行过程如图 4 所示。

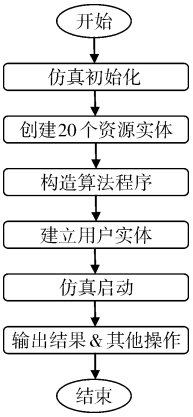


图 4　GridSim 仿真实验执行图

充分考虑到实验的可靠性，对于两种算法过程模拟相同的实验环境，保证每一次的比较测试都是在相同的任务数和资源数下进行的，经过反复实验取均值。根据定义中的描述，选取 $k=2$，自适应阈值 $\zeta=2I$。

实验 1　对比 Min-Min 与 RCSMM 算法的时间跨度 Makespan 值。

资源数为 25，任务数从 50 变化到 500，间隔为 50，每一组实验反复进行 20 次求平均获得时间跨度 Makespan 值，并记录数据。

实验 2　对比 Min-Min 与 RCSMM 算法的 Balance 值。

资源数为 25，任务数从 50 变化到 500，以 50 为变化单元，每一组实验反复进行 20 次求平均获得负载平衡 Balance 值，并记录数据。

3.4　实验结果分析

根据实验 1 和 2 记录的数据，如表 2、3 所示，绘制评价参数与任务数量变化曲线，其中图 5 为时间跨度 Makespan 随着任务数量变化的曲线图，图 6 为负载平衡 Balance 随着任务数量变化的曲线图。

表 2　MM 和 RCSMM 算法时间跨度的对比

任务数量	Min-Min 算法/s	RCSMM 算法 /s	改进程度/%
50	412.05	307.54	25.36
100	1108.33	899.90	26.03
150	1 932.31	1 482.12	43.61
200	2 876.58	2 212.85	47.60
250	3 421.09	2 673.11	61.91
300	3 907.76	3 129.34	63.05
350	4 733.81	3 515.46	75.93
400	5 756.32	4 097.23	73.97
450	6 877.01	4 482.84	86.13
500	7 801.17	5 113.15	81.01

从实验 1 可以看出，在资源数为 25，模拟环境相同的条件下，RCSMM 算法的时间跨度 Makespan 值均小于原始 Min-Min 算法的值，平均减少的比例大约在 53% 左右，根据评价原则 a)，RCSMM 算法的性能相对参照组有明显的改进，并且随着任务数量的增加，RCSMM 算法在时间跨度 Makespan 上的优势更加明显，从时间跨度 Makespan 曲线变化图上可以较为清晰地得出。

表 3　MM 和 RCSMM 算法负载均衡的对比

任务数量	Min-Min 算法	RCSMM 算法	改进程度/%
50	0.11	0.15	4
100	0.32	0.35	3
150	0.40	0.45	5
200	0.39	0.55	16
250	0.35	0.54	19
300	0.37	0.52	15
350	0.42	0.56	14
400	0.45	0.61	16
450	0.38	0.66	28
500	0.39	0.59	20

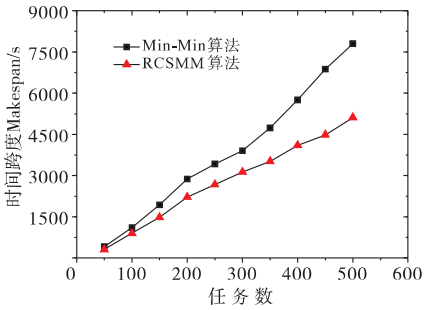


图 5　时间跨度随任务变化的曲线

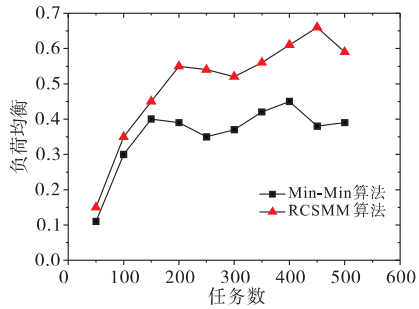


图 6　负载平衡随任务变化的曲线

从实验 2 可以看出，在资源数为 25，模拟环境相同的条件下，RCSMM 算法的负载平衡 Balance 值均大于原始 Min-Min 算法的值，根据评价原则 b)，RCSMM 算法的性能相对参照组有明显的改进。从负载平衡 Balance 曲线变化图可清楚地知道，原始 Min-Min 算法负载平衡利用率均小于 50%；任务总数小于 200 时，两种算法差别不是很大，当任务数量大于 300 时，RCSMM 算法的负载平衡利用率大于 50%，显然在任务量不断增加的情况下，RCSMM 算法能够较好地平衡资源负载和总体执行时间的问题。

通过两个实验结果的证明，随着任务数量的增加，RCSMM 算法在最优跨度上具有较好的性能，并且在负载平衡上也有一定的改进，RCSMM 算法在密集型的计算服务情况下体现出较好的适应性。由此可见，对资源进行预先分类的处理，使得资源得到更合理的分配与规划；引入自适应阈值的概念，帮助提高对长任务处理的优先级，既可以平衡资源的任务负载量，同时也减少了总体任务的执行时间。

4　结束语

本文通过对原始的 Min-Min 算法的分析讨论，发现了该算法中负载不平衡及总体执行时间较长的问题。因此，从计算资源的等级和不同任务的长度两个方面考虑，分别引入资源预先分级的思想和调节长任务运行级别的自适应阈值，对其进行改进与优化，提出了基于资源分级的自适应 Min-Min 算法。模拟仿真实验证明：资源得到了合理分配，总体执行 (下转第 725 页)

模型进行优化后，制动效能因数比优化前提高了 14%，制动鼓体积减小了 25%，制动器温升降低了 6%。由此可见，采用 OXNSGA-Ⅱ 算法对汽车鼓式制动器进行多目标优化设计效果比 NSGA-Ⅱ 算法更好，在保持较高制动效能因数的前提下，制动鼓体积更小，制动器温升更小。其表明文中建立的优化模型合理，提出的改进算法求解制动器多目标优化设计问题是有效可行的。

5　结束语

本文建立了制动器多目标优化数学模型，同时考虑制动器制动效能因数最大，制动鼓体积最小以及制动器温升最低这三个目标函数，改善了现有研究将多目标优化问题通过加权组合转换为单目标函数问题后求解的不足。

为了对多目标优化模型进行有效求解，本文提出了一种基于正交设计的 NSGA-Ⅱ 算法。该算法是在传统 NSGA-Ⅱ 算法的基础上引入正交设计的思想，利用正交表均衡分散的特点从父代个体构造出多个子代个体，然后选取其中最优的个体进入下一代种群。运用测试函数对算法进行测试，结果表明提出的改进算法相对于当前典型的几种多目标算法，在解决多目标优化问题时具有一定优势。利用该算法对制动器优化算例进行计算，并与传统的 NSGA-Ⅱ 算法对比，结果表明改进算法求解此类问题是有效的。

参考文献：

[1] 黄其柏，周明刚，王勇. 基于遗传算法的鼓式制动器结构参数优化设计 [J]. 机械制造，2006，44(11)：24-26.

[2] 米洁，吴欲龙. 汽车鼓式制动器多目标优化设计 [J]. 机械设计与制造，2007(1)：25-26.

[3] 杨仁华. 基于 MATLAB 遗传算法的汽车鼓式制动器多目标优化设计 [J]. 机床与液压，2011，39(23)：91-93.

[4] 李军. 基于混合粒子群算法的拖拉机制动器优化设计 [J]. 中国农机化，2011(4)：93-96.

[5] 秦广乐，王道明，陈小辉. 基于改进遗传算法的盘式制动器的优化设计 [J]. 组合机床与自动化加工技术，2011(7)：101-103.

[6] 王晗. 基于蚁群算法的制动器参数优化设计 [D]. 长春：吉林大学，2007.

[7] Ye Hongtao，Luo Fei，Xu Yuge. Differential evolution for solving multi-objective optimization problems：a survey of the state of the art [J]. Control Theory & Applications，2013，30(7)：922-927.

[8] Franciso G M，Raul B，Consolacion G，*et al*. Minimization of voltage deviation and power losses in power networks using Pareto optimization methods [J]. Engineering Applications of Artificial Intelligence，2010，23(5)：695-703.

[9] Andreas K，Yang Kun，Zhang Qingfu，*et al*. A multi-objective evolutionary algorithm for the deployment and power assignment problem in wireless sensor networks [J]. Computer Networks，2010，54(6)：960-976.

[10] Deb K，Pratab A，Agarwal S，*et al*. A fast and elitist multiobjective genetic algorithm：NSGA-Ⅱ [J]. IEEE Trans on Evolutionary Computation，2002，6(2)：182-197.

[11] Zhang Qingfu，Li Hui. MOEA/D：a multiobjective evolutionary algorithm based on decomposition [J]. IEEE Trans on Evolutionary Computation，2007，11(6)：712-731.

[12] Nebro A J，Durillo J J，Luna F，*et al*. MOCell：a cellular genetic algorithm for multiobjective optimization [J]. International Journal of Intelligent Systems，2009，24(7)：726-746.

[13] 王望予. 汽车设计 [M]. 北京：机械工业出版社，2004.

[14] 刘惟信. 汽车制动系的结构分析与设计计算 [M]. 北京：清华大学出版社，2004.

[15] Wang Yong，Cai Zixing，Zhang Qingfu. Enhancing the search ability of different evolution through orthogonal crossover [J]. Information Sciences，2012，185(1)：153-177.

[16] Deb K，Thiele L，Laumanns M，*et al*. Scalable test problems for evolutionary multi-objective optimization [C]//Proc of the Congress on Evolutionary Computation. Berlin：Springer，2002：852-890.

[17] Zitzler E，Thiele L. Multiobjctive evolutionary algorithms：a comparative case study and the strength Pareto approach [J]. IEEE Trans on Evolutionary Computation，1999，3(4)：257-271.

[18] 李密青，郑金华. 一种多目标进化算法解集分布广度评价方法 [J]. 计算机学报，2011，34(4)：647-663.

[19] Durillo J J，Nebro A J. jMetal：a Java framework for multi-objective optimization [J]. Advances in Engineering Software，2011，42(10)：760-771.

(上接第 719 页) 时间也有一定的减少，但仍需要进一步的工作来完善，如通过更多的实验确定最合适的常量参数，保证自适应阈值的准确，以及针对交互密集型任务调度的处理。

参考文献：

[1] Joseph J，Fellenstein C. 网格计算 [M]. 北京：清华大学出版社，2005.

[2] Bawa R K，Sharma G. Modified min-min heuristic for job scheduling based on QoS in grid environment [C]//Proc of Information Management in the Knowledge Economy International Conference. [S. l.]：IEEE Press，2013：166-171.

[3] Vouk M A. Cloud computing：issues，research and implementations [J]. Journal of Computing and Information Technology，2008，42(4)：235-246.

[4] 左利云，曹志波. 云计算中调度问题研究综述 [J]. 计算机应用研究，2012，29(11)：4023-4027.

[5] 李冬梅，施海虎. 负载平衡调度问题的一般模型研究 [J]. 计算机工程与应用，2007，43(8)：121-125.

[6] 夏家莉，陈辉，杨兵. 一种动态优先级实时任务调度算法 [J]. 计算机学报，2012，35(12)：2685-2695.

[7] Yu Xiaogao，Yu Xiaopeng. A new grid computation-based Min-Min algorithm [C]//Proc of the 6th International Conference on Fuzzy Systems and Knowledge Discovery. [S. l.]：IEEE Press，2009：43-45.

[8] Chauhan S，Joshi R C. A weighted mean time Min-Min Max-Min selective scheduling strategy for independent tasks on grid [C]//Advance Computing Conference. [S. l.]：IEEE Press，2010：4-9.

[9] Liu Juefu，Li Gang. An improved Min-Min grid tasks scheduling algorithm based on QoS constraints [C]//Proc of Optics Photonics and Energy Engineering International Conference. [S. l.]：IEEE Press，2010：281-283.

[10] Hao Fang，Lakshman T V，Mukherje E S. Enhancing dynamic cloud-based services using network virtualization [J]. Computer Communication Review，2010，40(1)：67-74.

[11] Chen Huankai，Wang F，Na Helian. User-priority guided Min-Min scheduling algorithmfor load balancing in cloud computing [C]//Proc of Parallel Computing Technologies Conference. [S. l.]：IEEE Press，2013：1-8.