大规模云计算服务器优化调度问题的最优二元交换算法研究

王万良[1]，臧泽林[1]，陈国棋[1]，屠杭垚[1]，王宇乐[1]，陆琳彦[2]

（1. 浙江工业大学计算机科学与技术学院，浙江 杭州 310027；2. 伦敦大学国王学院工程科学学院，英国 伦敦 WC2R 2LS）

摘 要：随着云计算产业的不断兴盛，云计算服务器的合理管理与科学调度成为了一个重要的课题。在模型方面，提出了一个新的携带亲和约束与反亲和约束的混合整数规划（MIP）模型，并将其用于描述大规模云计算服务器调度问题。考虑到求解大规模 MIP 问题的时间成本，在分枝定界法与局部搜索算法的基础上提出了最优二元交换算法。该算法通过不断地从完整的调度问题中提取 MIP 子问题，并使用分支定界法解决该子问题的思想，不断地对服务器调度方案进行优化，从而使调度方案接近最优解。实验结果表明，所提算法在测试数据集 ALISS 上与其他方法相比有较大优势，在完成相同任务的情况下，可以使云计算中心的资源消耗减少 4%以上。

关键词：服务器调度；混合整数规划模型；最优二元交换算法；云计算

中图分类号：TP302

文献标识码：A

doi: 10.11959/j.issn.1000−436x.2019105

Research on optimal two element exchange algorithm for large scale cloud computing server scheduling problem

WANG Wanliang[1], ZANG Zelin[1], CHEN Guoqi[1], TU Hangyao[1], WANG Yule[1], LU Linyan[2]

1. School of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310027, China
2. Faculty of Natural and Mathematical Sciences, King's College London, London WC2R 2LS, England

Abstract: With the flourishing of cloud computing industry, the rational management and scientific scheduling of cloud computing servers has become an important issue. In terms of model, a new mixed integer programming (MIP) model with affinity constraints and anti-affinity constraints was proposed to describe the scheduling problem of large scale cloud computing server. Considering the time cost of solving large-scale MIP problems, an optimal two element exchange algorithm was designed with the basics of branch and bound method and local search algorithm. By constantly extracting MIP sub-problems from completing scheduling problems and using branch and bound method to solve the sub-problems, the algorithm continuously optimized the server scheduling schemes, so that the scheduling schemes approached the optimal solution. The experimental results show that the algorithm has great advantages over the other methods in testing data set ALISS, and can reduce the resource consumption of cloud computing center by more than 4% when the same task is completed.

Key words: server scheduling, mixed integer programming model, optimal two element exchange algorithm, cloud computing

1 引言

云计算是分布式计算（distributed computing）、网络存储（network storage technology）、负载均衡（load balance）等传统计算机和通信技术发展融合的产物，由于云计算技术的通用性和高可靠性，该技术得到了广泛的发展[1]。随着云计算规模的逐步扩大，谷歌、阿里巴巴、百度等大型互联网企业纷纷搭建了自己的云服务平台。它们建立的平台可容纳超过万台的服务器，为超

收稿日期：2018−12−07；**修回日期**：2019−04−19

通信作者：臧泽林，zangzelin@gmail.com

基金项目：国家自然科学基金资助项目（No.61873240, No.51875524）

Foundation Item: The National Natural Science Foundation of China (No.61873240, No.51875524)

过百万的用户进行服务。大规模云服务平台的建立，使云服务的调度系统变得更加重要[2]。在庞大的云服务平台基数下，云服务平台的性能即使有 1%的提高也会带来巨大的收益[3]。

目前，云计算调度分为云服务请求实时调度和基于历史数据的云服务优化调度。

云服务请求实时调度以调度系统的快速性、稳定性为出发点，研究如何实时地将云计算任务指派给一台或几台宿主机服务器实现负载均衡。在这类研究中，传统的方法有 Max-Min 方法[4]、RR 算法[5]、FCFS 算法、FIFO 算法[6]等。另外，一系列的商业软件如 Google 的 Borg 调度系统[7-8]、阿里巴巴的 Sigma 调度系统[9]和 MapReduce 等开源软件都做出了相应的学术贡献。但是上述研究存在以下 2 个尚待解决的问题：1)上述方法只能应用于过程迭代和异构任务类型，无法执行递归或复杂的业务流；2)上述方法会出现占用特定的虚拟机 I/O，造成其他作业“饥饿”的情况。

在基于历史数据的云服务优化调度领域，同样出现了大量的研究成果。其中，Tsai 等[10]阐述了实时的云计算的生命周期问题，并在此基础上提出了一种面向云计算实时调度的框架，该框架能够有效地解决云计算实时调度的一般问题。在此基础上，Zhu 等[11]提出了一种用于虚拟化云中实时任务调度的新型滚动调度架构，然后提出并分析了面向任务的能耗模型。另外，Zhu 等[11]开发了一种新的能量感知调度算法（EARH, energy awar），并通过实验证明了该算法能够在可调度性和节能之间做出良好的权衡。王吉等[12]在考虑调度快速性和有效性的背景下，又考虑了调度的容错性，提出了一种在虚拟云平台中的容错调度算法（FSVC, fault-tolerant scheduling algorithm in virtualized cloud），通过主副版本方法实现对物理主机的容错控制，采用副版本重叠技术与虚拟机迁移技术提高算法的调度性能。郭平等[13]将云平台的调度问题简化为本地化的调度问题，并且结合主导资源公平调度策略 DRF 和 Delay 调度约束机制，提出了一种满足本地化计算的集群资源调度策略（DDRF, delay-dominant resource fairness），并讨论了本地化计算时延对作业执行效率的影响。另外，对于提供 GPU 服务的云平台，Peng 等[14]提出了一种高效的、动态的深度学习资源调度方法—— Optimus，该方法使用在线训练来拟合训练模型，并建立性能模型以准确地估计每个作业的训练速度。就工作完成时间而言，Optimus 的调度性能有 63%的提高。

基于历史数据的云服务优化调度将同一云计算请求源产生的计算请求看成是可预测的时间序列。调度算法以该时间序列为基础对宿主机服务器的分配进行调度[15]。很多经典的优化方法和优化理论可以引入调度框架[16]中，如遗传算法（GA, genetic algorithm）[17]、群智能算法、局部搜索（LS, local search）算法[18]等。同时，有部分学者提出了专门面向服务器的调度算法。例如，林伟伟等[19]通过对约束满足问题对异构的云数据中心的能耗优化资源调度问题建模，并且在此基础上提出了能耗优化的资源分配算法（DY, dynamic power）。Li 等[20]提出了一种协调调度算法来解决过度生成虚拟机（VM, virtual machine）实例的问题，并使用实际生产中的数据验证了该方法的有效性。Dong 等[21]将云服务器调度问题建模为混合整数规划（MIP, mixed integer programming）问题，将目标函数设置为使数据中心服务器消耗的能量最小，并提出了一个有效的服务器优先任务调度方案。

为了克服分支定界算法解决大规模问题时间消耗庞大的问题，本文将 LS 算法[22]的框架融入分枝定界算法，设计了最优二元交换算法（OTECA, optimal two element exchange algorithm）。OTECA 并不是使用分支定界法一次性解决全部问题，而是每次只解决其中的部分子问题。然后使用 LS 的思想不断地选择子问题进行解决。因此，OTECA 可以通过求解多个子问题获得原问题的解。上述求解策略可以快速有效地求解大规模的服务器调度问题。实验证明，OTECA 优于 LS、GA 等算法。

2　云计算服务器调度模型

本节建立一个 MIP 模型，对云计算服务器调度问题进行描述。模型框架如图 1 所示。

图 1 显示了云计算调度资源、调度方案、目标函数、约束条件与运算请求间的相互关系。本节将用数学方式对上述单元以建模的方式进行描述。

2.1　云计算服务器的资源表示与数据描述

在整个调度过程中，需要处理的资源被抽象为 5 个集合：服务器集合 M、应用集合 A、实例集合 S、亲和约束集合 Q 与反亲和约束集合 F。

每个宿主机服务器 m_j 都可以提供多种类型的计算资源，服务器集合 M 可描述为

$$M=\{m_1,m_2,m_3,\cdots,m_j,\cdots,m_n\} \tag{1}$$

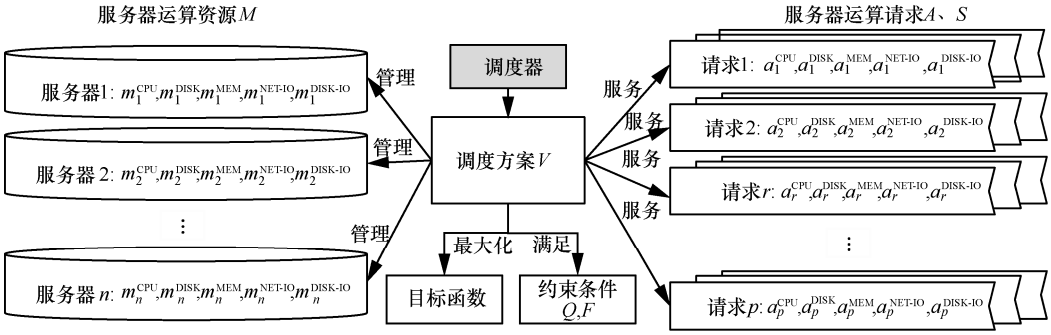

图 1　云计算服务调度模型框架

其中，n 为服务器的数量，m_j 为第 j 台服务器所能提供的资源，即

$$m_j=\{m_j^{\text{CPU}},m_j^{\text{DISK}},m_j^{\text{MEM}},m_j^{\text{NET-IO}},m_j^{\text{DISK-IO}}\} \tag{2}$$

其中，m_j^{CPU}、m_j^{DISK}、m_j^{MEM}、$m_j^{\text{NET-IO}}$、$m_j^{\text{DISK-IO}}$ 分别为宿主机服务器 m_j 可以提供的处理器时间分片资源（CPU）、磁盘空间资源（DISK）、内存资源（MEM）、网络 I/O 资源（NET-IO）与磁盘 I/O 资源（DISK-IO）。

应用集合 A 描述了请求服务器资源的应用与其对服务器提供资源的消耗，即

$$A=\{a_1,a_2,a_3,\cdots,a_r,\cdots,a_p\} \tag{3}$$

其中，p 为应用的总量，a_r 为第 r 个应用需求的资源，即

$$a_r=\{a_r^{\text{CPU}},a_r^{\text{DISK}},a_r^{\text{MEM}},a_r^{\text{NET-IO}},a_r^{\text{DISK-IO}}\} \tag{4}$$

其中，CPU、DISK 与 MEM 的每日资源消耗按照时间曲线给出，NET-IO、DISK-IO 的每日资源消耗以常数的形式给出。$\boldsymbol{a}_{r,k}^{\text{CPU}}$、$\boldsymbol{a}_{r,k}^{\text{DISK}}$、$\boldsymbol{a}_{r,k}^{\text{MEM}}$ 为元素个数为 N 的列向量，其中 k 为时间维度的检索句柄，满足 $k\in\{1,2,3,\cdots,N\}$。

相互独立的运算请求被称为实例。实例集合 S 包含调度过程中应用产生的所有实例，即

$$S=\{s_1,s_2,s_3,\cdots,s_i,\cdots,s_q\} \tag{5}$$

其中，q 为实例的总量，实例 s_i 为

$$s_i=\{s_i^{\,a\text{-ID}}\} \tag{6}$$

其中，$s_i^{\,a\text{-ID}}$ 为实例对应的应用的名称。

亲和约束与反亲和约束描述了调度场景中的约束关系，主要分为 2 种情况：应用与应用之间的亲和/反亲和关系，应用与服务器之间的亲和/反亲和关系。

亲和/反亲和关系可以描述为 2 个应用之间（或者应用与宿主机服务器之间）存在相互依赖或者相互排斥的关系。具体的亲和/反亲和关系描述如表 1 所示。

表 1　　**亲和/反亲和关系描述**

情况	亲和/反亲和关系
2 个应用间有功能上的依赖关系	2 个应用亲和
2 个应用的某一维度的资源存在强烈的竞争	2 个应用反亲和
2 个应用都是重要的应用，如果将 2 个应用布置到同一服务器会增大故障风险	2 个应用反亲和
应用需要用到部分宿主机服务器的计算资源（如 GPU、HDFS 模块）	宿主机服务器与应用亲和
应用由于资源问题不适合某一宿主机服务器（如 GPU、HDFS 模块）	宿主机服务器与应用反亲和

亲和/反亲和的关系使用 0-1 矩阵 $\boldsymbol{H}^{\text{aa}}$、$\boldsymbol{H}^{\text{am}}$、$\boldsymbol{F}^{\text{aa}}$、与 $\boldsymbol{F}^{\text{am}}$ 描述。应用与应用间的亲和关系 $\boldsymbol{H}^{\text{aa}}$ 的结构描述为

$$\boldsymbol{H}^{\text{aa}}=\left(h_{r_1,r_2}^{\text{aa}}\right)_{p\times p},\ r_1,r_2\in A \tag{7}$$

其中，h_{r_1,r_2}^{aa} 为 0-1 变量，当 $h_{r_1,r_2}^{\text{aa}}=0$ 时，r_1 应用与 r_2 应用不存在亲和关系，反之存在。

$\boldsymbol{F}^{\text{aa}}$ 的结构与 $\boldsymbol{H}^{\text{aa}}$ 的结构相同，其矩阵内元素为 f_{r_1,r_2}^{aa}，当 $f_{r_1,r_2}^{\text{aa}}=0$ 时，r_1 应用与 r_2 应用不存在反亲和关系，反之存在。$\boldsymbol{H}^{\text{am}}$ 的结构描述为

$$\boldsymbol{H}^{\text{am}}=\left(h_{r,j}^{\text{am}}\right)_{p\times n},\ r\in A,j\in M \tag{8}$$

其中，$\boldsymbol{H}^{\text{am}}$ 描述应用与机器间的亲和关系。

$\boldsymbol{F}^{\text{am}}$ 的结构与 $\boldsymbol{H}^{\text{am}}$ 的结构相同，其矩阵内元素为 $f_{r,j}^{\text{am}}$，当 $f_{r,j}^{\text{am}}=0$ 时，r 应用与 j 机器不存在反亲和关系，反之存在。

2.2　云计算服务器调度的决策变量与目标函数

云计算服务器的调度问题乃至大部分的资源调度问题都可以转化为一个多维度的背包问题。调度的目的是为每一个计算请求（实例 s_i）寻找一个

合适的宿主机服务器 m_j，减少系统的运行成本。因此以 0-1 整数的形式，定义二维决策变量 V 描述实例到机器的分配关系，即

$$V=\left(v_{i,j}\right)_{q\times n},\ i\in S, j\in M \tag{9}$$

其中，$v_{i,j}$ 为 0-1 决策变量，$v_{i,j}$=1 代表实例 s_1 被调度到机器 m_j 中执行，反之没有。

为了方便对亲和/反亲和约束描述，另外，设定辅助决策变量 W 来描述应用到机器的分配关系，即

$$W=\left(w_{i,j}\right)_{p\times n},\ i\in A, j\in M \tag{10}$$

其中，$w_{i,j}$ 为 0-1 决策变量，$w_{i,j}$=1 代表应用 a_i 被调度在机器 m_j 中执行，反之没有。V 与 W 之间存在的约束关系为

$$v_{i,j}-w_{R(i),j}=1\ ,j\in M \tag{11}$$

其中，$R_{(i)}$为一个从实例 S_i 到其对应的应用的映射。优化调度的目标描述为：在一定调度时间段内，寻找一种使用服务器数量最少、负载均衡的服务器调度方案。为了避免由于模型不精确带来的资源使用溢出风险，本文通过惩罚的形式定义目标函数为

$$\min:\frac{\sum_{k=1}^{N}\sum_{j\in M}F_{j,k}}{C} \tag{12}$$

其中，k 为时间索引，满足 $k\in\{1,2,3,\cdots,N\}$；C 为时间采样频率；F_j 为宿主机服务器 m_j 获得的分数值，其可以描述为

$$F_{j,k}=\begin{cases}0, x_{j,k}=0\\ 1, x_{j,k}<\beta\\ 1+\alpha(\max(x_{j,k}^2-\beta^2,0)), x_{j,k}>\beta\end{cases} \tag{13}$$

其中，β 为惩罚阈值，当宿主机服务器的 CPU 使用率大于惩罚阈值时，开始引入额外平方项对调度方案进行惩罚；α 为惩罚增量系数，描述进行惩罚时的惩罚力度；$x_{j,k}$ 为机器 m_j 在 k 时刻的 CPU 使用率，计算式为

$$x_{j,k}=\frac{\sum_{i\in S}v_{i,j}a_{R(i),k}^{\mathrm{CPU}}}{m_j^{\mathrm{CPU}}} \tag{14}$$

单一机器 CPU 使用率与目标分数关系如图 2 所示。由图 2 可知，当机器在所有的时间内保持 CPU 使用率为 0 时，目标分数为 0；当机器的 CPU 使用率大于 0 且小于惩罚阈值 β 时，目标分数为 1；但当 CPU 使用率大于惩罚阈值时，目标分数从 1 开始以二次幂形式上升。

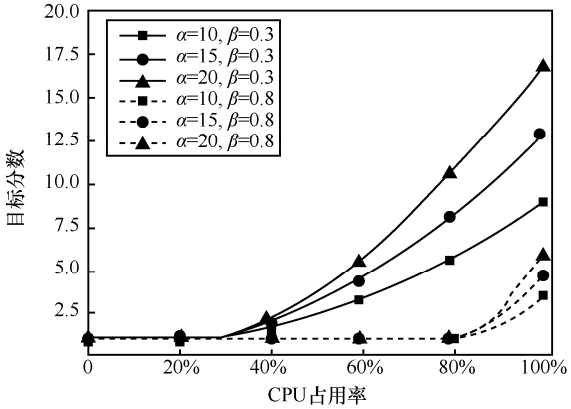

图 2　单一机器 CPU 使用率与目标分数关系

2.3　云计算服务器调度的约束条件

实际的云计算服务器调度环境中存在大量由硬件条件和应用功能导致的限制。模型将这些限制转化为模型约束、资源约束、亲和/反亲和约束和优先级约束进行处理。

模型约束描述了调度模型的基本约束，包括每个实例都应被安排到一个宿主机服务器上，且只能被安排到一台宿主机服务器上，即

$$\sum_{j\in M}v_{i,j}=1, i\in A \tag{15}$$

资源约束是一种由于资源限制造成的约束，每个宿主机服务器上配置的不同类型的资源是有容量上限的。

CPU 资源约束为

$$\sum_{i\in S}v_{i,j}a_{R(i),k}^{\mathrm{CPU}}\leqslant m_j^{\mathrm{CPU}}, j\in M \tag{16}$$

DISK 资源约束为

$$\sum_{i\in S}v_{i,j}a_{R(i),k}^{\mathrm{DISK}}\leqslant m_j^{\mathrm{DISK}}, j\in M \tag{17}$$

MEM 资源约束为

$$\sum_{i\in S}v_{i,j}a_{R(i),k}^{\mathrm{MEM}}\leqslant m_j^{\mathrm{MEM}}, j\in M \tag{18}$$

NET-IO 资源约束为

$$\sum_{i\in S}v_{i,j}a_{R(i)}^{\text{NET-IO}}\leqslant m_j^{\text{NET-IO}}, j\in M \tag{19}$$

DISK-IO 资源约束为

$$\sum_{i\in S}v_{i,j}a_{R(i)}^{\text{DISK-IO}}\leqslant m_j^{\text{DISK-IO}}, j\in M \tag{20}$$

亲和/反亲和约束可以分为以下 4 种情况。

1) 应用与应用的亲和约束

$$h_{a_u,a_v}^{aa}(w_{a_u,j}-w_{a_v,j})=0,\ a_u,a_v\in A,\ j\in M \tag{21}$$

当 2 个应用存在亲和约束时，$h^{aa}_{r_1,r_2}=1$，此时约束退化成 $w_{a_u,j}-w_{a_v,j}=0$。又因为 $w_{a_u,j}$、$w_{a_v,j}$ 都是 0-1 决策变量，所以该约束能够约束 2 个决策变量同时为 1 或者同时为 0。

2) 应用与应用的反亲和约束

$$f^{aa}_{i,j}(w_{a_u,j}+w_{a_v,j})\leqslant 1,\ a_u,a_v\in A,\ j\in M \tag{22}$$

当 2 个应用存在亲和约束时，$f^{aa}_{r_1,r_2}=1$，因为 $w_{a_u,j}$、$w_{a_v,j}$ 都是 0-1 决策变量，所以该约束能够约束 2 个决策变量不能同时为 1。

3) 应用与机器的亲和约束

$$h^{am}_{a_u,j}-w_{a_u,j}\leqslant 0,\ a_u\in A,\ j\in M \tag{23}$$

约束 $h^{am}_{a_u,j}=1$ 时，$w_{a_u,j}$ 必须为 1。

4) 应用与机器的反亲和约束

$$f^{am}_{a_u,j}+w_{a_u,j}\leqslant 1,\ a_u\in A,\ j\in M \tag{24}$$

约束决策变量不能同时为 1。

3 大规模云计算二元交换分析论证

将式(13)写成 max 函数形式，定义评分函数为

$$f(x)=1+\alpha(\max(x^2-\beta^2,0)),x\in[0,1] \tag{25}$$

其中，x 为宿主机服务器某一时刻的 CPU 使用率，机器 j 的 CPU 使用率为 x_j。

引理 1 在所有对 2 台机器进行的 CPU 使用率 (x_1,x_2) 的重新分配 (x_1^*,x_2^*) 中，至少存在一个使评价分 $O_t(x_1^*,x_2^*)=f(x_1^*)+f(x_2^*)$ 最小的最优分配，且该最优分配 $V_0=(x_1^*,x_2^*)$ 可由计算式 $x_1^*=\dfrac{c(cx_1+x_2)}{c^2+1}$ 和 $x_2^*=\dfrac{c(cx_1+x_2)}{c^2+1}$ 求得，其中，c 为 2 台机器的 CPU 容量的比例，即 $c=\dfrac{m_1^{\mathrm{CPU}}}{m_2^{\mathrm{CPU}}}$。

证明 将式(14)代入 $O_t(x_1,x_2)=f(x_1)+f(x_2)$ 可得

$$O_t(x_1^*,x_2^*)=2+\alpha(\max(\frac{x_1^{*2}}{\beta^2},0)-1)+\alpha(\max(\frac{x_2^{*2}}{\beta^2},0)-1) \tag{26}$$

由于实例的重新分配不改变其 CPU 使用率绝对值之和，因此有

$$x_1^*m_1^{\mathrm{CPU}}+x_2^*m_2^{\mathrm{CPU}}=x_1m_1^{\mathrm{CPU}}+x_2m_2^{\mathrm{CPU}} \tag{27}$$

因此，p_2^* 可以表示成 $l(\cdot)$，即 $x_2^*=l(x_1^*,c)=x_2+c(x_1-x_1^*)$，将其代入式(26)，可得变量为 x_1^* 的方程为

$$O_t(x_1^*,l(x_1^*,c))=2+\alpha(\max(\frac{x_1^{*2}}{\beta^2},0)-1)+\alpha(\max(\frac{l(x_1^*,c)^2}{\beta^2},0)-1) \tag{28}$$

对式(28)进行分段讨论后再对 x_1^* 求导可得

$$O_t'(x_1^*,l(x_1^*,c))=\begin{cases}\alpha(\dfrac{x_1^{*2}}{\beta^2}-\dfrac{cl(x_1^*,c)^2}{\beta^2}),\ \beta\leqslant x_1^*\leqslant b\\ \dfrac{-\alpha cl(x_1^*,c)^2}{\beta^2},\ x_1^*\leqslant\beta,x_1^*\leqslant b\\ \dfrac{\alpha x_1^{*2}}{\beta^2},\ x_1^*\geqslant\beta,x_1^*\geqslant b\\ 0,\ b\leqslant x_1^*\leqslant\beta\end{cases} \tag{29}$$

其中，$b=x_2+cx_1-\dfrac{\beta}{c}$。

当 $b=x_2+cx_1-\dfrac{\beta}{c}<\beta$ 成立时，说明 2 台机器中的实例对应的 CPU 负载之和非常低，以至于将所有的实例全部移入任何一台机器都不会造成 2 台机器的得分变化（2 台机器皆得 1 分）。因此其得分的导数为 0。

当上述不等式不能被满足时，求解等式 $O'_t(x_1^*,l(x_1^*,c))=0$，解得

$$x_1^*=\frac{c(cx_1+x_2)}{c^2+1} \tag{30}$$

当 $x_1^*\geqslant\dfrac{c(cx_1+x_2)}{c^2+1}$ 时，有

$$O_t'(x_1^*,l(x_1^*,c))\geqslant\min(\alpha(\frac{x_1^{*2}}{\beta^2}-\frac{cl(x_1^*,c)^2}{\beta^2}),\frac{\alpha x_1^{*2}}{\beta^2})\geqslant\alpha(\frac{x_1^{*2}}{\beta^2}-\frac{cl(x_1^*,c)^2}{\beta^2})\Big|_{x_1^*=\frac{c(cx_1+x_2)}{c^2+1}}\geqslant 0 \tag{31}$$

当 $x_1^*\leqslant\dfrac{c(cx_1+x_2)}{c^2+1}$ 时，有

$$O_t'(x_1^*,l(x_1^*,c))\leqslant\max(\alpha(\frac{x_1^{*2}}{\beta^2}-\frac{cl(x_1^*,c)^2}{\beta^2}),-\frac{\alpha cl(x_1^*,c)^2}{\beta^2})\leqslant\alpha(\frac{x_1^{*2}}{\beta^2}-\frac{cl(x_1^*,c)^2}{\beta^2})\Big|_{x_1^*=\frac{c(cx_1+x_2)}{c^2+1}}\leqslant 0 \tag{32}$$

因此，$x_1^*=\dfrac{c(cx_1+x_2)}{c^2+1}$ 是函数的一个极小值点，

此时，$x_2^*=\frac{cx_1+x_2}{c^2+1}$。证毕。

引理 2　定义机器 j 的特征计算式为 $g_j=\frac{x_j}{m_j^{\mathrm{CPU}}}$，$x_j$ 为机器 j 的平均 CPU 使用率。在 2 台机器的平均 CPU 占比为(x_1,x_2)的情况下，最优分配(x_1^*,x_2^*)所带来的评分之和的下降量 $O_d(x_1,x_2,c)=O_t(x_1,x_2)-O_t(x_1^*,x_2^*,c)$与 2 台机器的特征计算式的差的平方$(g_1-g_2)^2$呈正相关。

证明　对特征公式的差的平方$(g_1-g_2)^2$进行展开，有

$$(g_1-g_2)^2=\frac{c^2x_2^2-2cx_1x_2+x_1^2}{(m_1^{\mathrm{CPU}})^2} \tag{33}$$

构造函数

$$D(x_1,x_2,c)=\frac{\mathrm{d}O_d(x_1,x_2,c)}{(x_1-x_2)^2} \tag{34}$$

对于所有的自由变量，使用链式法则进行展开，可得

$$\begin{aligned}D_{l1}&=\frac{\mathrm{d}O_d(x_1,x_2,c)}{\mathrm{d}x_1}\times\frac{\mathrm{d}x_1}{\mathrm{d}(x_1-x_2)^2}\\D_{l2}&=\frac{\mathrm{d}O_d(x_1,x_2,c)}{\mathrm{d}x_2}\times\frac{\mathrm{d}x_2}{\mathrm{d}(x_1-x_2)^2}\\D_{l3}&=\frac{\mathrm{d}O_d(x_1,x_2,c)}{\mathrm{d}c}\times\frac{\mathrm{d}c}{\mathrm{d}(x_1-x_2)^2}\end{aligned} \tag{35}$$

其中

$$D_{l1}=\frac{2(m_1^{\mathrm{CPU}})^2\alpha\left(cx_2-x_1\right)}{\beta^2\left(c^2+1\right)(2cx_2+2x_1)}=\frac{(\alpha m_1^{\mathrm{CPU}})^2}{\beta^2\left(c^2+1\right)} \tag{36}$$

$$D_{l2}=\frac{2(m_1^{\mathrm{CPU}})^2\alpha c\left(cx_2-x_1\right)}{\beta^2\left(c^2+1\right)(2c\left(cx_2-x_1\right))}=\frac{\alpha(m_1^{\mathrm{CPU}})^2}{\beta^2\left(c^2+1\right)} \tag{37}$$

$$D_{l3}=\frac{2(m_1^{\mathrm{CPU}})^2\alpha\left(c^2x_1x_2-cx_1^2+cx_2^2-x_1x_2\right)}{\beta^2\left(c^4+2c^2+1\right)(2x_2\left(cx_2-x_1\right))}=\frac{\alpha(m_1^{\mathrm{CPU}})^2\left(cp_1+p_2\right)}{\beta^2p_2\left(c^4+2c^2+1\right)} \tag{38}$$

由于 α、β 与 c 皆大于或等于 0，O_d 对于特征计算式 p 的导数恒大于 0，即 $D(x_1,x_2,c)>0$，因此 2 个方程呈正相关。证毕。

4　算法设计

混合整数规划问题是一个 NP 完全问题，即求解该问题的最优解的时间复杂度与问题的规模呈指数关系。随着数据量的增长，分支定界法不能在有效时间内得到理想的解，基于此，本文设计了 OTECA，寻找一个时间花费与求解精度的平衡点，快速有效地解决云计算服务器调度问题。

4.1　可行解生成方法

可行解生成方法负责生成占用尽量少的服务器资源且满足约束要求的初始解。求解的流程如算法 1 所示。

算法 1　可行解生成方法（GSSM, good solution generation method）

输入　服务器集合 M，应用集合 A，实例集合 S

输出　调度初始可行解 V，状态矩阵 $\boldsymbol{E}$

1) 初始化可行解 V，使用服务器数下限 $N_{\mathrm{ml}}=0$，使用服务器数上限 $N_{\mathrm{mh}}=p$

2) while N_{ml} != N_{mh} do

3)　　机器指示 $m=0$

4)　　for 实例索引 $s=0, s\leqslant q, s$++

5)　　if 实例 s 可装入机器 m then

6)　　V,E=put(s,m)　%将 s 放入 m 中，并更新状态

7)　　else

8)　　$m=m+1, s=s-1$

9)　　end if

10)　　if $m==N_{\mathrm{mh}}$ then %机器数选取过少

11)　　$N_{\mathrm{mh}}=[\frac{3}{2}N_{\mathrm{mh}}+\frac{1}{2}N_{\mathrm{ml}}]$

12)　　$N_{\mathrm{ml}}=[\frac{2}{3}N_{\mathrm{mh}}+\frac{1}{3}N_{\mathrm{ml}}]$

13)　　break

14)　　end if

15)　　if $s==q$ then % 机器数选取过多

16)　　$N_{\mathrm{mh}}=[\frac{1}{2}N_{\mathrm{mh}}+\frac{1}{2}N_{\mathrm{ml}}]$

17)　　end if

18)　　end for

19) end while

20) return V,E

算法通过二分法的形式对使用服务器数量的上限和下限不断更新，从而不断地拉近使用服务器的上下限的差距，最终在上下限相等时停止程序，完成对最合适的可行解的生成。步骤 5)验证机器 m 装入实例 s 后，是否仍然满足模型约束。步骤 10)与步骤 15)判断当前的机器数量上限 N_{mh} 能否得到可行解，如果可以得到可行解，则进一步减少机器数量，如果无法得到可行解，则增加机器的数量，

直至上下限相等则退出方法。

在时间复杂度方面，由于每次进入步骤 11)、步骤 12)与步骤 16)判断都会造成 N_{mh} 与 N_{ml} 的差减少一半，因此外部循环的复杂度为 $\log(p)$。步骤 4)~步骤 18)的内部循环的次数受到实例数 p 与机器数 n 的影响，在最坏的情况下将进行 $n+p$ 次循环，因此整个算法的时间复杂度为 $(n+p)\log(p)$。

4.2 最优二元交换算法

分支定界法求解大规模 MIP 问题消耗的时间常常是不能接受的，但是求解较小规模的 MIP 问题却有很高的效率。该算法将通过求解一系列小规模的 MIP 子问题，去逼近整个大规模的 MIP 问题。

可以证明，在所有的实例都满足约束被分配进入宿主机服务器的情况下，将其中 2 台机器中的实例以 BBM 方法重新分配到原来的宿主机服务器中，不会造成调度目标值的上升（如引理 1 所示），且这 2 台机器的特征计算式 g_j 差距越大，其获得更高分数下降的可能性也就越大（如引理 2 所示）。

OTECA 流程如图 3 所示。

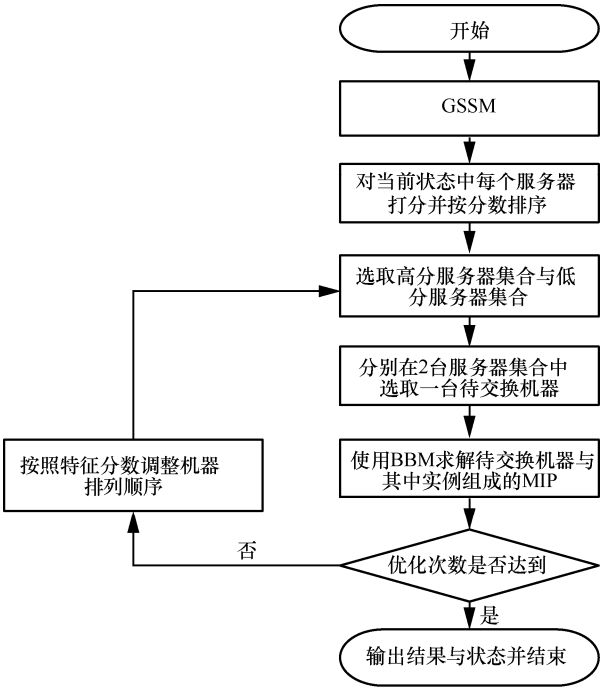

图 3 OTECA 流程

最优二元交换算法伪代码如算法 2 所示。

算法 2 OTECA

输入 服务器集合 M，应用集合 A，实例集合 S，惩罚系数 α 和 β，集合选择系数 γ，小规模 MIP 子问题求解次数 N_s

输出 服务器调度结果 V^{best}

1) V=GSSM(M,A,S) %生成可行解

2) M,ScoreList=Score(E,M) %按照式(14)对当前可行状态中每个服务器进行打分，并按照分数进行排序

3) for loop=0, loop<N_{aim}, loop++

4) M=SMWS(M, ScoreList) %按照分数调整顺序

5) S^{mh},S^{ml}=CMSet(M_c,A,S,α,β,γ) %选取高分服务器集合与低分服务器集合

6) M^{h},M^{l}=ChoMach(S^{mh},S^{ml}) %分别在 2 台服务器集合中选取一台待交换机器

7) V_{MIP},E=BBM(M^{h},M^{l},A,α,β) %分支定界法求解该小规模 MIP 问题，得到新的分配方案与机器状态

8) end for

9) return V^{best}

最优二元优化部分是算法的核心部分，该部分为寻找一个优秀的大规模服务器调度方案，其主要循环中包括高分机器集合与低分机器集合的维护、MIP 子问题选择、MIP 子问题求解等步骤。

步骤 4)和步骤 5)生成一个高分机器集合与低分机器集合，其中特征分数概念由目标函数中目标分数的概念变化而来。该特征分数综合考虑了机器当前负载与机器的容量这 2 个特性，可以表现机器未来的负载能力，如引理 2 所示。步骤 5)中，使用最高分 s_{h} 与最低分 s_{l} 生成高分集合与低分集合，生成的标准为

$$S^{\mathrm{mh}}=\{j\mid w_j>\frac{(N_s-\gamma L)s_{\mathrm{h}}+\gamma L s_{\mathrm{l}}}{N_s},j\in M\} \quad (39)$$

$$S^{\mathrm{ml}}=\{j\mid w_j<\frac{(N_s-\gamma L)s_{\mathrm{l}}+\gamma L s_{\mathrm{h}}}{N_s},j\in M\} \quad (40)$$

其中，γ 为集合选择系数，一个较大的值可以使高分机器集合和低分机器集合随循环次数的选择增长得更快；S^{mh} 为高分机器集合，S^{ml} 为低分机器集合。

步骤 7)中设定当机器得分不变的前提下，实例负载尽量装入容量较大的服务器，从而尽量减少服务器的使用。步骤 4)~步骤 7)不断解决子问题优化调度结果。

每个循环子问题由高分机器、低分机器与机器中的实例组成。子问题的最优解一定是上述实例在

上述机器上的重新组合。可以证明，新解的目标分数之和一定优于原解（如引理 1 所示）。另外，高分机器与低分机器的分差越高，重新组合后分数的优化越显著（如引理 2 所示）。上述引理是算法挑选高分集合和低分集合进行混合的原因。因此，最优二元优化算法比传统的基于随机或启发的算法有更强的目的性，在前期拥有更高的收敛速度，在后期拥有更强的调度精确度。

算法初始阶段将快速地消除高负载机器，造成低负载机器分数上升、高负载机器分数下降，这时高分机器集合和低分机器集合将出现一定重叠。随着这种重叠现象的逐步加深，算法的子问题选择将逐渐退化为对机器的随机优化，从而对机器中实例进行整理，并减小宿主机服务器的数量。最后该算法将返回从实例到宿主机服务器的优化分配。

在时间复杂度方面，4.1 节已经论证了算法 2 步骤 1)的复杂度为$(n+p)\log(p)$，步骤 2)作为一个排序的工作，复杂度为 $n\log(n)$。因为步骤 3)~步骤 7)在最坏情况下复杂度为 $N_{\text{aim}}2^x$，其中 x 为子问题中实例的数量，这并不意味着该步骤是十分耗时的，因为在子问题中，x 受到机器容量的约束。在实际的应用环境中，步骤 7)均可以在可接受的时间（秒级）范围内得到响应，则算法 2 总的时间复杂度为$(n+p)\log(p)+n\log(n)$。

5　仿真与验证

5.1　实验硬件架构描述与数据集描述

为了保证改进算法得到充分稳定的测试，本文使用不同规模的调度测试数据与阿里云计算中心公开的实际数据集（ALISS）进行仿真实验。数据中心拥有 10 000 台不同配置的宿主机服务器。实验仿真硬件平台为：曙光天阔 W580-G20 服务器，CPU E5-2620 v4 2.1 GHz；软件环境为：ubuntu 18 LTS，Python3.6。

实验的硬件架构如图 4 所示。硬件架构按照功能由 4 种节点组成，分别为资源节点、管理节点、调度节点和服务节点。其中，服务节点负责收集云计算请求，并以集合 A、S 的形式提交给调度节点。管理节点负责监控资源节点的资源使用和工作情况，并以集合 M 的形式提交给调度节点。调度节点综合整个云计算服务平台的情况进行调度工作，将运算请求调度到资源节点中。

ALISS 数据集中每一组数据由 4 张表格组成，分别是机器信息列表、应用信息列表、实例信息列表、亲和与反亲和信息列表。其中，机器信息列表中含有 M-id、M-c、M-m、M-d，M-id 为机器的 ID，M-c、M-m、M-d 分别对应机器集合的 m_i^{CPU}、m_i^{DISK}、m_i^{MEM}。应用信息列表中含有 A-id、A-c、A-m、A-d，A-id 为应用的 ID，A-c、A-m、A-d 分别对应集合中的 a_i^{CPU}、a_i^{DISK}、a_i^{MEM}。时变数据 CPU-use、MEM-use 的采样周期为 15 min，即 C=96。

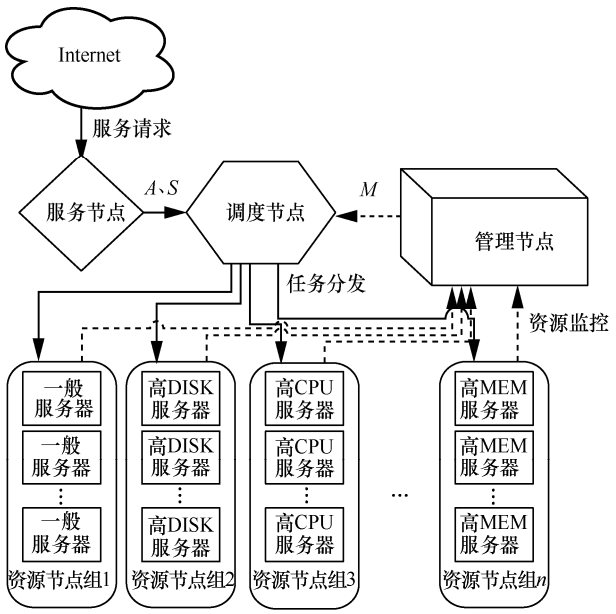

图 4　实验的硬件架构

5.2　OTECA 在 ALISS 上的求解结果

本节使用 OTECA 求解 ALISS-2 数据集中的调度问题。为了保证服务器运行的稳定性与资源分配的合理性，设置模型参数为 α=10，β=0.5，选取 OTECA 参数为择优选取集合选择系数 γ=1.5，子问题求解次数 N_s=300，并对问题进行求解。求解过程如图 5~图 7 所示。

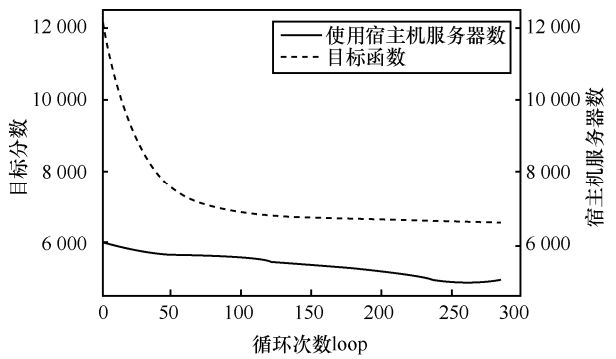

图 5　ALISS-2 问题宿主机服务器数、目标函数与循环次数关系

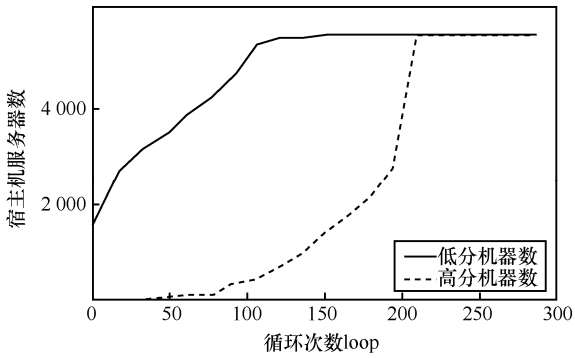

图 6　ALISS-2 问题高低分机器数与循环次数关系变化

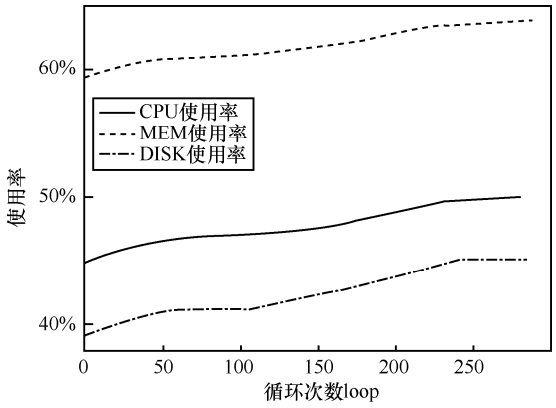

图 7　ALISS-2 宿主机服务器使用率与循环次数关系变化

优化开始时，宿主机服务器平均 CPU 使用率仅在 45%左右，这证明优化开始时服务器组的负载并不均衡。随着算法迭代循环次数的增加，算法将得分较高的机器中的实例与得分较低的机器中的实例使用整数规划的方法进行最优的重新组合，重新分配到原来的机器中，并将负载进行有效的重组。由图 5 可知，随着优化的进行，得分在快速地下降。在最理想的情况下，算法将所有的机器的负载都调整到从下方逼近 β 的情况。由图 6 可知，由于优化开始时机器分数差异较大，以最高分和最低分划定的高分机器集合和低分机器集合中元素较少。随着优化的进行，机器的负载逐步平均，分数的分布逐步集中，高分机器集合和低分机器集合逐渐包含同一部分机器。

当 loop>100 时，由于机器的负载已经达到了平均水平，机器数与得分的差距基本不再变化，此时分数的下降主要来自机器数的削减。当某台机器不执行任何程序时，其得分为 0。因此在交换无法使负载更加均衡的情况下，OTECA 的同样的交换操作将尽量清除多余的宿主机服务器。由图 7 可知，随着循环次数的上升，CPU 的平均使用率逐渐接近 β，达到接近最优解的状态。

5.3　不同规模服务器调度问题上算法性能对比

本文在不同规模的调度问题中进行算法性能的比较。实验算法包括 OTECA、分离化差分进化算法（C-MSDE, cost modified separation differential evolution）[23]、成本时间进化算法（CT-DE, cost time differential evolution）[24]、GA[17]和 LS[18]。

择优选取 OTECA 中参数 γ=1.5，N_s=300；C-MSDE 中选择变异因子为 1，交叉因子为 0.5；CT-DE 中 P_{time}=P_{cost}=0.5；GA 中选择概率为 0.2，交叉概率为 0.2，变异概率为 0.2；LS 中搜索范围为 3；C-MSDE、CT-DE、GA 的种群数为 300。测试使用的服务器规格数据与云计算要求数据皆从阿里巴巴公司公布的 Alibaba Cluster Data V2018 中抽取，测试结果如表 2 所示。

表 2 中显示了不同规模（SSP_1~SSP_6）的调度问题，其中 SSP_1~SSP_3 是规模与实例数较小的服务器调度问题，SSP_4~SSP_6 是规模与实例数较大的服务器调度问题。其中，N_m 代表算法调度结果使用的宿主机服务器数量，Score 代表算法所得目标分数，CPU 代表调度结果平均 CPU 使用率，目标为 50%。处理小规模问题时，进行比较的算法拥有相似的性能表现，但是 OTECA、CT-DE 与 LS 拥有较好的表

表 2　　**不同规模服务器调度问题上 5 种算法性能对比**

不同规模	实例数	OTECA			C-MSDE			CT-DE			GA			LS		
		N_m	Score	CPU	N_m	Score	CPU	N_m	Score	CPU	N_m	Score	CPU	N_m	Score	CPU
SSP_1	100	9	9.01	49%	10	10.00	46%	9	9.01	49%	11	11.01	44%	9	9.01	49%
SSP_2	500	41	48.70	48%	41	48.57	47%	41	48.84	48%	41	48.48	48%	41	48.46	48%
SSP_3	1 000	95	102.04	45%	97	109.62	44%	102	100.52	44%	105	102.88	45%	105	107.56	45%
SSP_4	7 000	480	607.76	45%	482	613.26	47%	481	626.15	47%	480	608.95	47%	480	619.58	47%
SSP_5	14 000	1 413	1 709.48	45%	1 433	1 756.14	43%	1 426	1 716.54	45%	1 452	1 786.51	42%	1422	1 712.15	45%
SSP_6	70 000	4 731	5 544.98	47%	5 098	6 034.26	44%	4 969	6 167.03	45%	5 260	5 989.14	44%	4816	5 861.35	45%
平均	—	1 128.17	1 337.00	47%	1 193.50	1 428.64	45%	1 171.33	1 444.68	46%	1 224.83	1 424.50	45%	1 145.50	1 393.02	47%

现，目标分数的平均值相比其他算法优秀 1%。处理较大规模问题时，OTECA 展现出比较突出的性能优势，目标分数的平均值比其他算法中最优秀的 LS 优秀 3%。综合上述 2 种情况，OTECA 表现更好。

5.4　5 种算法在 ALISS 数据集上性能对比

设置算法参数与 5.3 节相同，以实例在宿主机服务器中的转移次数为横坐标，以分数为纵坐标，得到 5 种算法求解 ALISS-1 问题如图 8 所示。

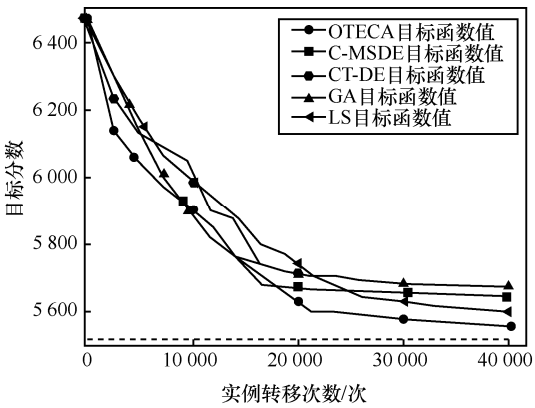

图 8　5 种算法求解 ALISS-1 问题

图 8 中最下方的虚线为该问题的最优解。在算法的初始阶段，OTECA 与 C-MSDE 的效率较高。随着优化的进行，目标值快速下降，GA、LS 与 CT-DE 因优化策略原因效率较低。在优化的中后阶段，OTECA 表现出了较强的调度能力与精确性，通过解决子问题，有目的地将组合不合理的机器挑选出来并进行重新组合，该操作在效果上优于基于随机迁移的 GA、LS、C-MSDE 等其他算法，OTECA 得到的调度结果能够跳出局部最优解。

算法对服务器组负载的均衡能力是衡量算法优劣程度的重要指标，本文对于 ALISS-1 的求解过程绘制机器负载量直方图，如图 9 所示。

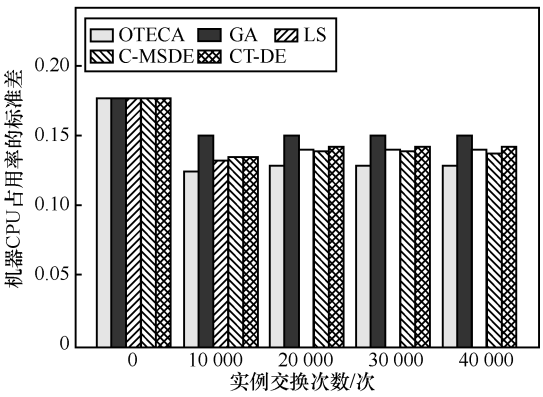

图 9　机器 CPU 使用率标准差直方图

由图 9 可知，当实例交换次数为 0 时，5 种算法以相同的基础进行优化。随着实例交换次数的增加，5 种优化方法都能起到有效的负载均衡效果。当实例的交换次数为 20 000 次时，OTECA 的效果最为显著，其对应的机器 CPU 使用率的标准差最小，相比初始值变化的幅度是 GA 的 2 倍。随着优化的继续进行，其他算法的标准差几乎没有发生改变，而 OTECA 的标准差有少量提升。这是因为随着交换的进行，OTECA 能够有效地减少使用宿主机服务器的数量，从而提高了仍在工作的宿主机服务器使用率的标准差。5 种算法求解 ALISS 数据集的结果对比如表 3 所示。

表 3 中评价了 5 种算法在数据集上的 5 个指标，分别是目标分数（Score）、平均机器 CPU 使用率（CPU）、平均机器内存使用率（DISK）、平均机器硬盘使用率（MEM）、机器 CPU 使用率标准差（STD）。在 7 个测试数据中，OTECA 表现最佳，目标分数的平均分为 6 665.70，比第二优秀的 C-MSDE 优秀 4%。

数据问题的差异性会引起算法表现的差异，但是 OTECA 综合性能占有一定优势。对于 ALISS-3 与 ALISS-4 这类实例粒度较小（单个实例占用的资源较小）的问题，5 种算法的表现相似。但是对于实例粒度较大的问题，GA、LS、C-MSDE 与 CT-DE 等算法表现稍逊于 OTECA，原因是不满足约束的情况会降低上述算法的搜索效率。综合考虑上述情况，OTECA 具有更加优秀的性能。

表 3 显示，C-MSDE、CT-DE、GA 与 LS 这 4 种算法在解决有相对严格约束的服务器调度问题时没有明显优势，这是因为 4 种算法的解集更新具有随机性，新生成的解很容易不满足问题的约束。因此，检查新解是否满足约束与丢弃不满足约束的解花费大量的运算时间，从而大大降低了算法性能。OTECA 不需要维护多个解的集合，也从不产生可能不满足约束的解。OTECA 只通过求解 MIP 子问题不断地对解集进行更新，因此拥有较高的效率与性能。

6　结束语

本文针对大规模云服务器调度问题进行研究。在对大规模云服务器调度问题进行 MIP 建模的基础上，为了解决传统方法很难及时地求解出最优调度方案的问题，本文提出了 OTECA。

表 3　　5 种算法求解 ALISS 数据集的结果对比

算法	指标	ALISS-1	ALISS-2	ALISS-3	ALISS-4	ALISS-5	ALISS-6	ALISS-7	平均
OTECA	Score	5 534.3	6 214.5	5 234.9	5 240.2	6 894.6	6 825.1	9 584.9	6 665.70
	CPU	0.34	0.49	0.51	0.5	0.49	0.51	0.51	0.50
	MEM	0.42	0.63	0.62	0.61	0.65	0.63	0.72	0.64
	DISK	0.99	0.44	0.5	0.49	0.46	0.45	0.39	0.46
	STD	0.128	0.083	0.042	0.052	0.153	0.146	0.126	0.10
C-MSDE	Score	5 668.8	6 715.6	5 214.6	5 244.9	7 015.9	7 163.4	10 125.6	6 913.33
	CPU	0.32	0.44	0.49	0.49	0.42	0.44	0.38	0.44
	MEM	0.40	0.57	0.60	0.60	0.56	0.54	0.54	0.57
	DISK	0.93	0.40	0.48	0.48	0.39	0.39	0.29	0.40
	STD	0.15	0.106	0.042	0.052	0.154	0.142	0.121	0.10
CT-DE	Score	5 679.6	7 832.5	5 235.7	5 247.3	7 948.1	7 216.4	11 244.8	7 454.02
	CPU	0.32	0.44	0.50	0.50	0.40	0.40	0.39	0.44
	MEM	0.40	0.57	0.61	0.61	0.53	0.49	0.55	0.56
	DISK	0.93	0.40	0.49	0.49	0.38	0.35	0.30	0.40
	STD	0.14	0.098	0.043	0.053	0.133	0.142	0.134	0.10
GA	Score	5 683.9	7 816.5	5 247.6	5 256.3	7 365.2	7 201.6	10 165.6	7 175.37
	CPU	0.33	0.42	0.50	0.50	0.40	0.43	0.36	0.44
	MEM	0.41	0.54	0.61	0.61	0.53	0.53	0.51	0.55
	DISK	0.96	0.38	0.49	0.49	0.38	0.38	0.28	0.40
	STD	0.15	0.106	0.043	0.052	0.157	0.142	0.146	0.11
LS	Score	5 630.6	7 834.4	5 235	5 247.3	7 952.5	7 265.2	12 033.9	7 594.72
	CPU	0.33	0.44	0.5	0.5	0.42	0.41	0.35	0.44
	MEM	0.41	0.57	0.61	0.61	0.56	0.51	0.49	0.56
	DISK	0.96	0.40	0.49	0.49	0.39	0.36	0.27	0.40
	STD	0.14	0.098	0.043	0.053	0.134	0.142	0.136	0.10

OTECA 首先通过可行解生成算法求出可行解，然后通过在循环中不断选取与解决 MIP 子问题的方式优化可行解，从而得到全局调度方案。结果表明，OTECA 可以快速优化大规模服务器调度方案，在测试数据集 ALISS 上较传统的 GA、LS、C-MSDE 与 CT-DE 算法有较大的优势。在完成相同任务的情况下，OTECA 使云计算中心的资源消耗减少 4%以上。

参考文献：

[1] MARINESCU D C. Cloud computing: theory and practice[M]. Morgan Kaufmann, 2017.

[2] HASSAN M M, SONG B, HUH E N. A market-oriented dynamic collaborative cloud services platform[J]. Annals of Telecommunications, 2010, 65(11-12): 669-688.

[3] AL-DHURAIBI Y, PARAISO F, DJARALLAH N, et al. Elasticity in cloud computing: state of the art and research challenges[J]. IEEE Transactions on Services Computing, 2017, 12(5): e0176321.

[4] ADAN I, KLEINER I, RIGHTER R, et al. FCFS parallel service systems and matching models[J]. arXiv Preprint, arXiv:1805.04266, 2018.

[5] LI J, MA T, TANG M, et al. Improved FIFO scheduling algorithm based on fuzzy clustering in cloud computing[J]. Information, 2017, 8(1): 25.

[6] BURNS B, GRANT B, OPPENHEIMER D, et al. Borg, omega, and kubernetes[J]. Queue, 2016, 14(1): 70-93.

[7] MIJUMBI R, SERRAT J, GORRICHO J, et al. Network function virtualization: state-of-the-art and research challenges[J]. IEEE Communications Surveys & Tutorials, 2015, 18(c): 236-262.

[8] VERMA A, PEDROSA L, KORUPOLU M, et al. Large-scale cluster management at Google with Borg[C]//The Tenth European Conference on Computer Systems. ACM, 2015: 18.

[9] CHENG Y, CHAI Z, ANWAR A. Characterizing co-located datacenter workloads: an alibaba case study[J]. arXiv Preprint. arXiv: 1808. 02919, 2018.

[10] TSAI W, SHAO Q, SUN X, et al. Real-time service-oriented cloud computing[C]//2010 6th World Congress on Services. 2010: 473-478.

[11] ZHU X, YANG L T, CHEN H, et al. Real-time tasks oriented energy-aware scheduling in virtualized clouds[J]. IEEE Transactions on Cloud Computing, 2014, 2(2): 168-180.

[12] 王吉, 包卫东, 朱晓敏. 虚拟化云平台中实时任务容错调度算法研究[J]. 通信学报, 2014, 35(10): 171-180.
WANG J, BAO W D, ZHU X M. Fault tolerant scheduling algorithm for real time tasks in virtualized cloud[J]. Journal on Communications, 2014, 35(10): 171-180.

[13] 郭平, 宁立江, 陈海珠, 等. 满足本地化计算的集群资源调度策略[J]. 通信学报, 2014, 35(Z2): 1-8.
GUO P, NING L J, CHEN H Z, et al. Scheduling strategy for achieving locality in cluster[J]. Journal on Communications, 2014, 35(Z2): 1-8.

[14] PENG Y, BAO Y, CHEN Y, et al. Optimus: an efficient dynamic resource scheduler for deep learning clusters[C]//The Thirteenth EuroSys Conference on EuroSys'18. 2018: 1-14.

[15] SINGH S, CHANA I. A survey on resource scheduling in cloud computing: issues and challenges[J]. Journal of Grid Computing, 2016, 14(2): 217-264.

[16] RIMAL B P, MAIER M. Workflow scheduling in multi-tenant cloud computing environments[J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(1): 290-304.

[17] PETEGHEM V V, VANHOUCKE M. 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.

[18] PAGNOZZI F, STUTZLE T. Speeding up local search for the insert neighborhood in the weighted tardiness permutation flowshop problem[J]. Optimization Letters, 2017, 11(7): 1283-1292.

[19] 林伟伟, 刘波, 朱良昌, 等. 基于 CSP 的能耗高效云计算资源调度模型与算法[J]. 通信学报, 2013, 34(12): 33-41.
LIN W W, LIU B, ZHU L C, et al. CSP-based resource allocation model and algorithms for energy-efficient cloud computing[J]. Journal on Communications, 2013, 34(12): 33-41.

[20] LI J, SU S, CHENG X, et al. Cost-efficient coordinated scheduling for leasing cloud resources on hybrid workloads[J]. Parallel Computing, 2015, 44: 1-17.

[21] DONG Z, LIU N, ROJAS-CESSA R. Greedy scheduling of tasks with time constraints for energy-efficient cloud-computing data centers[J]. Journal of Cloud Computing, 2015, 4(1): 5.

[22] KRAMER O. Briefs in applied sciences and technology[M]. London: Springer, 2014.

[23] 糜培培. 基于云计算的改进差分进化算法的研究与实现[D]. 成都: 电子科技大学, 2018.
MI P P. Research and implementation of improved differential evolution evolution algorithm based on cloud computing[D]. Chengdu: University of Electronic Science and Technology of China, 2018.

[24] DAS S, MULLICK S S, SUGANTHAN P N. Recent advances in differential evolution – an updated survey[J]. Swarm and Evolutionary Computation, 2016, 11(9): 30-45.

[作者简介]

王万良（1957– ），男，江苏高邮人，博士，浙江工业大学教授，主要研究方向为人工智能、运筹优化、大数据。

臧泽林（1994– ），男，河北秦皇岛人，浙江工业大学硕士生，主要研究方向为人工智能、运筹优化、深度学习。

陈国棋（1996– ），男，浙江温州人，浙江工业大学硕士生，主要研究方向为深度学习、增强学习。

屠杭垚（1996– ），男，浙江杭州人，浙江工业大学硕士生，主要研究方向为智能算法、人工智能、多目标优化。

王宇乐（1990– ），男，江苏常州人，浙江工业大学博士生，主要研究方向为演化计算、多目标优化。

陆琳彦（1996– ），女，浙江杭州人，伦敦大学国王学院硕士生，主要研究方向为通信网络技术。