云计算资源分配的分层多代理优化

Xiangqiang Gao , Rongke Liu , *Senior Member, IEEE*, and Aryan Kaushik , *Member, IEEE*

摘要 在云计算中，一个重要的问题是将服务节点的可用资源按需分配给所请求的任务，并使目标函数最优，即资源利用率最大化、报酬最大化和可用带宽最大化。本文提出了一种分层多代理优化（HMAO）算法，以实现云计算的资源利用率最大化和带宽成本最小化。所提出的 HMAO 算法是遗传算法（GA）和多代理优化（MAO）算法的结合。为了最大限度地提高资源利用率，采用了改进的遗传算法来寻找一组服务节点，用于部署所请求的任务。为了使带宽成本最小化，提出了一种基于去中心化的 MAO 算法。我们通过田口方法研究了 HMAO 算法关键参数的影响，并评估了性能结果。结果表明，在解决资源分配的大规模优化问题时，HMAO 算法比遗传算法（GA）和快速精英非支配排序遗传算法（NSGA-II）这两种基线算法更有效。
此外，我们还比较了 HMAO 算法与两种启发式 Greedy 算法和 Viterbi 算法在在线资源分配方面的性能。

索引词条--云计算、资源分配、资源利用、带宽成本、遗传算法、多代理优化

Ç

1 导言

F 一些电子设备是由专用硬件设备（如现场可编程门阵列（FPGA）、数字信号处理器（DSP）和集成电路（IC））组成的，因此很难保证不同请求任务的兼容性，而且随着请求任务数量的增加，系统也会变得更加复杂。软件定义网络（SDN）和虚拟化技术是云计算的基础，为促进资源分配提供了一种前景广阔的灵活方法[1], [2], [3]。云服务提供商可根据供需情况，将服务节点相关的可用资源分配给请求的任务。当一个任务由多个子任务组成时，这些子任务可部署在多个服务节点上，并形成一个服务链，服务链是依次通过服务节点的数据流，可表示为有向无环图（DAG）[4]。每个子任务都需要中央处理器（CPU）、内存或图形处理器（GPU）等物理资源。此外，在不同的服务节点上传输数据还需要带宽成本。例如，就数据传输而言，它包括五个子任务，这些子任务的服务链可以表示为：网络接收！

捕获！

● *高祥强和刘荣科，北京航空航天大学电子与信息工程学院，中国北京 100191。*
E-mail：{xggao, rongke_liu}@buaa.edu.cn.

● *Aryan Kaushik现供职于英国伦敦大学学院（UCL）电子与电气工程系，地址：WC1E 7JE London。*
电子邮件：a.kaushik@ucl.ac.uk.

2019 年 11 月 26 日收到手稿；2020 年 9 月 2 日修订；2020 年 10 月 9 日接受。发表日期：2020年10月14日；当前版本日期：2020年10月22日。（通讯作者：刘荣科）由O. Rana推荐接受。
数字对象标识符编号10.1109/TPDS.2020.3030920

跟踪！ 同步！ 解码，其中每个功能模块都是通过软件编程实现的，可以在常用的计算机系统上运行。云计算可以有效降低系统的复杂性和开发成本，提高灵活性和可扩展性。然而，云计算面临的一个新挑战是如何有效地将服务节点相关的可用资源分配给所请求的任务，这就导致了一个组合优化问题[5], [6]。

1.1 文献综述

云计算中的资源分配优化问题已被广泛研究[7]、[8]、[9]、[10]、[11]，其中[12]、[13]、[14]对这些问题的复杂性进行了分析，并证明这些问题是 NP 难的。元启发式算法是解决这些资源分配问题的有效优化方法。文献[7]、[8]、[9]开发了几种遗传算法（GA）的变体来提高资源分配解决方案的性能，文献[15]详细介绍了快速精英非支配排序遗传算法（NSGA-II），该算法也被用来解决这一问题[10]、[11]。文献[16]和[17]分别提出了两种改进的粒子群优化算法（MPSO），用于重新分配迁移的虚拟机，实现基于灵活成本的资源管理。此外，[18] 还提出了处理非线性资源分配问题的蚁群优化（ACO）。为了提高寻求最优解的效率，作者在文献[19]中引入了模拟退火和人工蜂群的混合优化算法（ABC-SA）。

然而，随着优化问题规模的扩大，需要搜索的可行解空间也越来越大，寻求最优解的计算复杂度也随之增加。因此，求解

使用元启发式算法可以减少问题的数量 [20]，[21]。为了进一步解决这个问题，人们提出了一些基于分解和合作协同进化方法的优化算法，即将一个大问题分成若干个小问题，通过合作协同进化方法解决这些子问题，从而获得全局最优解[22], [23]。参考文献[24]提出了一种新的合作协同进化框架（CCFR），它可以根据不同子群的贡献有效地分配计算资源，以解决大规模优化问题。

与集中式优化方法相比，基于多代理系统（MAS）[25] 的分布式优化算法在解决云计算中的任务部署和资源分配问题上具有明显的潜在优势[26], [27], [28]。在[26]中，供应链市场的产品分配问题被认为是一个分散的资源分配问题，并通过基于多代理的分布式优化算法来解决。此外，还有人提出了一种多代理的高效贪婪算法来解决社交网络中的任务分配问题[27]，其中代理只需要各自本地的任务和资源信息，并通过拍卖机制为任务提供资源。文献[28]提出了另一种基于拍卖的多代理系统虚拟机资源分配方法，以节约能源成本，并通过基于本地协商的方法交换分配给不同代理的虚拟机。

在我们的工作中，提出了一种分层多代理优化（HMAO）算法，以解决资源利用率最大化和带宽成本最小化的问题。所提出的 HMAO 算法由改进的 GA 算法和多代理优化（MAO）算法组成。为了降低目标问题求解的复杂度，我们将其分解为两个子目标问题：资源利用率最大化和带宽成本最小化。改进后的 GA 用于获取请求任务使用的最优服务节点集。为了最小化带宽成本，提出了 MAO 算法。在 MAO 算法中，我们设计了一个共享代理和多个服务代理。共享代理可以掌握所有服务节点的资源分配和任务部署信息。服务代理可以通过访问共享代理来感知云计算环境。此外，我们还通过概率方法实现了选择和交换操作符，以迁移和交换服务节点上的子任务。根据所提出的 HMAO 算法，随着请求任务数量的增加，我们可以提高解决资源分配问题的性能。

1.2 捐款

在本文中，我们假设预先给出了所请求任务和服务节点的相关信息，如任务类型、任务数量、每个子任务的资源需求和服务节点的资源容量值。此外，我们还提出了资源分配问题，即在资源约束条件下，资源利用率最大化，带宽成本最小化。资源利用率定义为资源数量与服务节点数量之比

所有请求任务使用的资源总数与用于部署请求任务的服务节点的资源总数之比。为了解决优化问题，我们提出了一种分层多代理优化（HMAO）算法，它是改进的 GA 算法和多代理优化（MAO）算法的结合。

首先，我们将资源利用率最大化和带宽成本最小化的主目标分解为两个子目标：资源利用率最大化和带宽成本最小化[24]。这两个子目标的优化问题相互冲突，我们假设前者优先考虑。改进后的 GA 用于寻求最优解，以最大限度地提高资源利用率，最优解可以表示为一组服务节点，用于部署所请求的任务。在 MAO 算法中，有两种代理：服务代理和共享代理。服务代理分配给每个服务节点，以协助资源管理。共享代理掌握所有服务节点的资源分配信息，并在*访问*和更新过程中为服务代理提供支持。代理具有环境感知、自治、社会行为和负载平衡等特性[29]。服务代理访问共享代理以获取所有服务代理的资源分配信息，它们可以通过基于概率方法设计的选择和交换操作符相互迁移和交换子任务。此外，考虑到服务节点的负载平衡和不同子任务之间的关系，为选择算子实施了基于优先级的源子任务选择机制。因此，通过合作共同进化法寻求服务代理的最优解，将获得全局最优解[23]。拟议的 HMAO 算法具有以下贡献。

1) 将资源利用率和带宽成本的联合优化问题作为一个整体来解决，会增加优化问题的计算复杂度，削弱寻求最优解的性能，尤其是对于高维问题。本文将整个优化问题分解为两个子问题。相应地，我们提出了一种分层多代理优化算法，它结合了改进的 GA 算法和 MAO 算法，用于求解这两个优化子问题。这样，我们就能有效降低整体优化问题的计算复杂度。
2) 改进后的 GA 可以获得可用服务节点集，该集合代表了资源利用最大化的最优解。然后提出 MAO 算法来解决带宽成本最小化的子问题。我们考虑了代理的四个主要特征：环境感知、自主性、社会行为和负载平衡。此外，我们还设计了服务代理和共享代理的行动集和行为标准，以及不同代理之间的关系。

我们将根据有组织的架构对代理进行说明。为了迁移和交换服务节点上的子任务，我们实现了选择和交换两个操作符，其中选择操作符包括源子任务选择和目标子任务选择。考虑到服务节点上的负载平衡和不同子任务之间的关系，源子任务由基于优先级的选择机制提供。在 MAO 算法中，一个可行的解决方案被划分为多个小规模解决方案，每个服务代理表示一个部分。我们可以通过合作协同进化法优化服务代理的目标，找到全局最优解。

3) 保持可行解决方案的多样性，避免预先成熟的衔接、选择和交流

实现了 MAO 算法的运算符基于概率方法。对于前者，我们可以通过选择概率从服务节点中随机选择一个子任务作为目标子任务。同样，如果服务代理的目标结果在交换后没有改善，也可以利用交换概率继续执行。通过在选择和交换算子中引入概率方法，我们可以进一步提高原型 HMAO 算法的性能。

最后，针对不同的任务进行了实验，以验证所提出的 HMAO 算法的性能。结果表明，所提出的 HMAO 算法是解决资源分配优化问题的有效方法。与GA和NSGA-II相比，所提出的HMAO算法在高维问题的求解质量、收敛时间和稳定性方面表现更好。

本文的其余部分安排如下。第 2 节介绍资源分配系统模型。第 3 节提供了系统模型的问题表述。第 4 节提出了一种分层多代理优化算法。第 5 节研究了关键参数对所提出的 HMAO 算法的影响，并评估了该算法与 HMAO 算法的性能比较。

现有的优化算法。结论
本文将在第 6 节进行讨论。

2 系统模型

云计算系统可以建模为一个图 $G = < V; E >$，其中 $V = \{v_1; v_{01}; \ldots; v_{K-1}\}$ 表示有 K 个服务

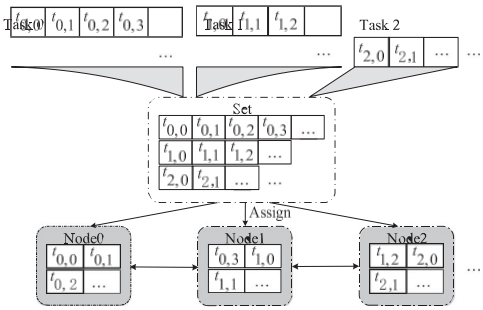


图 1.资源分配程序。

不同子任务之间的关系。也就是说，在所有前置任务完成之前，$_{l;n}$。对 CPU、内存、GPU 和带宽的资源要求 $_{l;n}; m^i$

在 v_i 上运行的 $t_{l;n}$ 的宽度描述为 c^i $_{l;n}$；

$g^i_{l;n}; b^i_{l;n}$ 分别为在云计算中，有效的资源分配方法用于分配可用的将服务节点的资源分配给请求的任务。需要注意的是，任何服务节点上的资源总数都 不 能 超过其容量，当两个相邻的子任务被放置在不同的服务节点上时，将考虑带宽成本。此外，我们可以在一个服务节点上部署尽可能多的子任务，以提高资源利用率 [7]、[11]、[12]。

在我们的系统模型中，资源分配的流程如图 1 所示，事先获取请求任务和服务节点的信息，将请求任务中的所有子任务同时部署到多个服务节点上，相邻的子任务可以位于同一个服务节点上，也可以位于不同的服务节点上。因此，不同的资源分配方案会对系统性能产生影响[12]。

图 2 展示了采用不同方案分配资源的示例。网络中有五个带通信链路的服务节点，两个任务分别为 $t_0 = \{t_{0;0}; t; t_{0;10;2}; t; t_{0;30;4}$ 和 $t_1 = t_{1;0}; t; t_{1;11;2}; t_{1;3}$。图 2a、对于任务 t_0，子任务 $t_{0;0}$；$t_{0;1}$ 部署在服务节点 v_0 上，子任务 $t_{0;2}$；$t_{0;3}$ 和 $t_{0;4}$ 分别部署在服务节点 v_1；v_3 和 v_4 上。t_0 的服务链依次建立在服务节点 $v_0; v; v_{13}$ 和 v_4 上，子任务 $t_{0;0}$ 和 $t_{0;1}$ 之间不存在带宽成本，因为它们位于同一个服务节点上。

服务节点。因此，t_0 的带宽成本为 b^0 $_{0;2}$ + $b^3_{0;3} + b^4_{0;4}$.对于任务 t_1，子任务 $t_{1;0}$ 和 $t_{1;1}$ 被分配在服务节点 v_2，子任务 $t_{1;2}$ 和 $t_{1;3}$ 位于服务节点 v_3 和 v_4。t_1 的带宽成本为 表示为 $b^2_{1;0} + b^3_{1;2} +$ $^4_{1;3}$.不同的资源分配

b

节点的服务节点集，v_i 是第 i 个服务节点。E 表示这些

服务节点之间的一组链接，$< v_i ; v_j > \in E$ 表示服务节点 v_i 和 v_j 之间的链接。服务节点 v_i 包含四种资源：CPU、内存、GPU 和带宽的容量分别表示为 C^i ; M^i ; G^i 和 B^i 。我们还假设任意两个服务节点可以通过相互连接的网络进行通信，即 $6 < v_i ; v_j > \in$。

*E.*设 $T = \{t_0 ; t_1 ; \ldots ; t_{L-1}\}$ 是一组任务，任务总数为 L ，t_l 表示*第 l 个*任务。t_l *由* N *个* $_l$ 子任务组成，表示为 $t_l = t_{l;0} ; t_{l;1} ; \ldots ; t_{l;N\text{l-}1}$ 、是一个有序列表，并且有一个优先级

任务 t_0 和 t_1 的方案如图 2b 所示。

任务 $t_{0;0}$; $t_{0;1}$ 和 $t_{0;2}$ 分配给服务节点 v_0 ，子任务 $t_{0;3}$ 和 $t_{0;4}$ 分配给服务节点 v_1 和 v_4 、

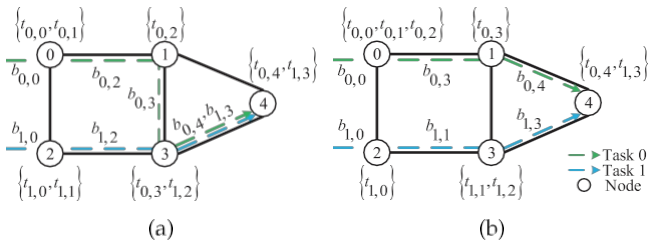


图 2.不同方案的资源分配示例。

t_0 的带宽成本分别为 b^0 b^4 。同样，t_1 的带宽成本为 b^2 + b^3 如图 2 所示，不同的带宽成本会有所不同。资源分配方案。

3 问题的提出

本节将对我们的系统模型进行资源分配的数学描述。我们的目的是在云计算中最大限度地提高资源利用率，并最大限度地降低带宽成本。此外，本文不考虑路由器和交换机等网络资源限制。

需要注意的是，在实际的云环境中，使用所提出的方法需要放置多个冗余服务节点，以维持服务交付。一般来说，这些冗余服务节点处于休眠或关闭状态，以减少服务交付。运营成本 [30]、[31]。当少数服务节点出现故障时当服务节点发生故障时，冗余服务节点会被迅速唤醒，以处理其动态工作负载。故障服务节点上的动态工作负载可以迁移到冗余服务节点上，从而确保我们提出的方法在服务节点故障时也能有效地维持服务交付。

表 1 汇总了用于提出问题的主要符号。

为了进一步讨论这个问题，我们定义了二进制解码器。

分变量 x^i $_{l;n}$ 表示子任务 $t_{l;n}$ 是否是部署在服务节点 v_i 上，即 x^i $_{l;n}$ = 1 表示 $t_{l;n}$ 是异构的。

v_i ，否则没有。此外，另一个二元判定变量 $y^{i;j}$ 用于描述是否有一个 $_{l;n;n^\wedge}$

子任务 $t_{l;n}$ 和 $t_{l;n^\wedge}$ 之间的带宽成本。让我们表示 $t_{l;n}$ 作为 U^p 的一组前辈，以及 t 的一组后辈 $_{l;n}$

$t_{l;n^\wedge}$ 作为 U^s ^。我们假设 $t_{l;n}$ 和 $t_{l;n^\wedge}$ 位于 v_i 和 $_{i;j}$ v_j 分别为当 $U_{l;n}$ /= Ø ; $t_{l;n^\wedge}$ ∈ $U_{l;n}$ 时，则 $yl;n_{;n^\wedge}$ = 1，或否则 $y^{i;j}_{l;n;n^\wedge}$ = 0 之间的带宽成本。任务，源节点也不会被忽视。带宽 $t_{l;n}$ 的成本可写成

$$\tilde{b}^{i}_{l;n} = \begin{cases} b^{i}_{l;n} & \text{如果 } U^{p}_{l;n} = \emptyset \quad \text{或 } y^{i;j}_{l;n;n^\wedge} = 1; \\ 0; & \text{否则。} \end{cases} \tag{1}$$

根据物理资源约束[9]，服务节点上请求任务使用的资源数量应小于资源容量。因此，CPU、内存和 GPU 的资源限制如下：

表 1
符号列表

符号	定义
V	云计算中的服务节点集。E 云计算中的网络链接集。K 服务节点的最大数量。
V_a	已部署任务的服务节点集。
K_a	已部署任务的服务节点数。
T	一组任务。
L	任务数量
N_l	第 l 项任务的子任务数。
t_l	第 l 项任务的子任务集。
$t_{l;n}$	第1 个任务的第 n 个子任务。
v_i	第 i 个服务节点。
C^i	v 上 CPU 的资源容量$_i$.
M^i	v_i 上的内存资源容量。
G^i	v_i 上 GPU 的资源容量。
B^i	v 上的带宽资源容量$_i$.
$c^i_{l;n}$	v_i 上 CPU 的 $t_{l;n}$ 资源需求。
$m^i_{l;n}$	v_i 上内存的 $t_{l;n}$ 资源需求。
$g^i_{l;n}$	v_i 上 GPU 的 t 资源需求。$_{l;n}$
$b^i_{l;n}$	$t_{l;n}$ 对 v_i 上带宽的资源需求。说明 $t_{l;n}$ 是否分配给了 v 。 $_i$
$x^{n\,i}_{l;}$	
$y^{i;j}_{l;n;n^\wedge}$	指明带宽成本是否可用于 $tl;_n;\ tl;_{n^\wedge}$.
$\tilde{l}^{i}_{l;n}$	
$U^{l;n}$	运行 t 的带宽成本$_{l;n}$ 。
$U^{p}_{l;n}$	t 的后继集$_{l;n}$ 。
	t 的前代集合 。$_{l;n}$
Z1	资源利用的目标函数。
Z_2	带宽成本的目标函数。
Z	总目标函数。
a; b	重量值。

$$\sum_{l=0}^{L-1}\sum_{n=0}^{Nl-1} b\sim i.xi_{l;n}\ _{l;n} \le B\ 。\ ^{i} \tag{3}$$

此外，任务集 T 中的所有子任务都是任何子任务都只能分配一次。因此，我们可以得到以下结果：

$$\sum_{i=0}^{K-1}\sum_{l=0}^{L-1}\sum_{n=0}^{Nl-1} {}^{i}\ xl_{;n} = \sum_{l=0}^{XL-1} N.l \tag{4}$$

8> XL-1 NXl-1

本文的目标之一是通过减少使用的服务节点数量来提高云计算的资源利用率。对于服务节点 v_i，资源利用率包括三个

$$\begin{cases} c^i_{l;n} x^i_{l;n} \le C^i; \\ \sum_{l=0}^{NXl-1} \sum_{n=0}^{XL-1} m^i_{l;n} x^i_{l;n} \le M^i; \\ \sum_{l=0}^{NXl-1} \sum_{n=0}^{XL-1} g^i_{l;n} x^i_{l;n} \le G^i. \end{cases} \tag{2}$$

同样，由于服务节点上使用的带宽量不能超过容量，因此带宽资源约束条件表示为

部分：CPU 利用率、内存利用率和 GPU 利用率分别用 u^i_c、u^i_m、u^i_g 表示。它们的计算方法如下：

$$\begin{cases} u^i_c = \sum_{l=0}^{Nl-1} \sum_{n=0}^{Nl-1} c^i_{l;n} x^i_{l;n} / C^i; \\ u^i_m = \sum_{l=0}^{Nl-1} \sum_{n=0}^{Nl-1} m^i_{l;n} x^i_{l;n} / M^i; \\ u^i_g = \sum_{l=0}^{Nl-1} \sum_{n=0}^{Nl-1} g^i_{l;n} x^i_{l;n} / G^i. \end{cases} \tag{5}$$

根据不同资源类型的偏好，我们将 CPU、内存和 GPU 的权重值分别设为 ω_c、ω_m 和 ω_g。资源利用率 Z_1 为

我们的系统模型是通过线性加权和法得到的，具体如下

$$Z_1(X) = \frac{1}{K_a} \sum_{i=0}^{KXa} \left(a_{c\,c}^{\,i} + a_{m\,m} + a_{g\,g}^{\,i} \right)^{-1}; \quad (6)$$

其中，$X = nx_{l;n}^{i};\ 6v_i \in V;\ 6t_{l;n} \in T$ 是一个可行解，为将服务节点的可用资源分配给云计算中的请求任务。参数 K_a ($K_a \le K$) 表示用于部署请求任务的服务节点数量。此外，一个$_c$ + 一个$_m$ + 一个$_g$ = 1。

另一个目标是通过以下方式优化带宽成本
将相邻的子任务部署到同一服务节点。我们将带宽成本记为 Z_2 ，其表达式如下：

$$Z_2(X) = \frac{1}{\text{卡}} \sum_{i=0}^{Ka-1} \sum_{l=0}^{L-1} \sum_{n=0}^{Nl-1} b_{l;n}\, x_{l;n}^{i} / B^{i}. \quad (7)$$

在云环境中，我们认为当一个服务节点
运行会增加运行成本 [30]，[31]。为了降低运行成本，请求任务使用的服务节点数量应尽可能少，处于空闲模式的服务节点可以处于休眠或关闭状态 [31]。此外，将任务中相邻的子任务分配到同一个服务节点上，以节省带宽和延迟成本，提高云计算的性能。我们需要同时优化这两个目标，即 $Z_1(X)$ 和 $Z_2(X)$。具体来说，我们的目的是寻求一个最优解，最大化资源利用率 $Z_1(X)$，最小化 $Z_2(X)$。因此，综合目标 $Z(X)$ 包括最大化 $Z_1(X)$和最小化 $Z_2(X)$，可表示为

$$\text{最大化 } Z(X) = b1Z_1(X) + {}_{b2}(1 - Z_2(X)); \quad (8)$$

其中，$_{b1}$ 和 $_{b2}$ 分别是 $Z_1(X)$ 和 $1 - Z_2(X)$ 的权重值。不同目标的偏好可以通过改变 $_{b1}$ 和 $_{b2}$ 来调整，我们认为 $_{b1}$ + $_{b2}$ = 1。然而，这个优化问题被认为是 NP-困难问题，这意味着找到最优解的计算顺序非常复杂。元启发式优化算法是解决组合优化问题的有效方法，但随着问题规模的增大，求解质量和收敛时间都会下降。基于分解和多代理系统的分布式优化算法可以有效改善优化解。

在下一节中，我们提出了 HMAO 算法，这是一种使用分层多代理框架的优化方法，用于解决云计算中的资源分配问题。

4 分层多代理优化

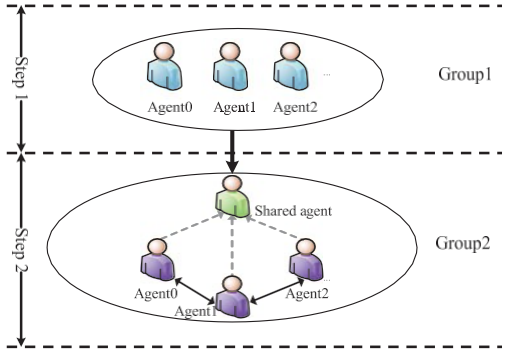


图 3.拟议的分层多代理框架。

为了降低优化问题的计算复杂度，我们将总目标分解为两个优化子问题：最大化 $Z_1(X)$和最小化 $Z_2(X)$，并求解这两个优化子问题、
因此。

首先，引入改进的 GA，以找到使 $Z_1(X)$ 最大化的最优解。T

本文探讨了云计算系统中资源利用率最大化和降低带宽成本的联合优化问题 [32], [33]。为了降低

组成一个有序列表，作为代表可行解决方案的个体。我们采用轮盘选择法 [34] 来获取适应度值较高的个体。此外，我们还使用了两点交叉和信号点突变 [35]。改进后的 GA 可以提供一组服务节点 V_a 作为最优解，用于部署所要求的任务。

然后，提出一种多代理优化算法来解决最小化 $Z_2(X)$ 的子问题。我们使用共享代理来保存服务节点的资源分配信息，并为每个服务节点分配服务代理来协助资源管理[36]。不同的服务代理可以相互合作、协调和竞争，以优化其与行为准则相关的目标[28]。为了保持可行解的多样性，避免过早收敛，选择和交换算子[36]都是通过概率列表法实现的。此外，可行解由可用服务代理的所有子任务组成，服务代理上的这些子任务被视为可行解的一部分。我们可以通过合作共同进化法优化服务代理的目标，从而获得全局最优解[23]。

图 3 显示了拟议的分层多代理框架。解决优化问题的程序分为两步。第一步，使用改进后的 GA 算法寻找使 $Z_1(X)$ 最大化的最优解，每个代理代表一个个体。第二步，采用 MAO 算法寻求 $Z_2(X)$ 最小的最优解。共享代理可保存所有服务节点的任务部署和资源分配信息，并支持服务代理实时访问和更新任务部署和资源分配信息。服务代理被分配到每个服务节点，为资源管理提供帮助。服务代理可通过访问共享代理获取任务部署和资源分配信息，并相互交换子任务，以改进资源分配方案。

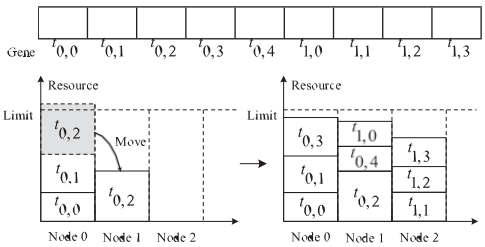


图 4.解码个体的程序

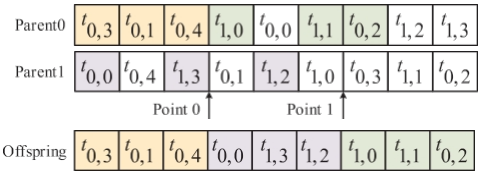


图 5.两点交叉算子示例。

4.1 改进的 GA

对于优化资源利用率的子问题，目标是最大化 $Z_1(X)$，并且需要满足约束条件。为解决这一优化问题，引入了改进的 GA。改进包括两部分：解码个体的过程和初始群体的生成。我们通过索引优先和第一匹配规则实现了两种解码个体的程序
[37]、[38]。一种基于索引优先级用于离线资源分配和而在线资源分配则采用另一种方法，即通过第一拟合规则进行解码。此外，对于在线资源分配中提出的 HMAO 算法，为了提高初始解的质量，一个个体被依次编码为这些子任务的相互依赖关系，其余个体被随机编码，以保持可行解的多样性。在改进的遗传算法中，一个种群包括 P 个个体，个体 $p \in P$ 由 $\sum_{l=0}^{L-1} N_l$ 个排列组合编码表示法[35]，[39]，由 L 基因组成。
顺序，其中一个基因代表一个子任务。在解码过程中，V 中的所有服务节点按索引从高到低排序，索引低的服务节点具有高优先级，将与"..... "一起部署到"..... "中。
要求的任务。因此，子任务的部署取决于
服务节点的优先级，直到资源
满足要求。通过解码方法，可以将个体的子任务依次分配给 V 中的这些服务节点，资源分配的结果表明解决方案是可行的。

Fig. 4 illustrates the procedure for decoding an individual. There are two tasks: t_0 with 5 sub-tasks and t_1 with 4 sub-tasks, an individual is expressed as $p = \{t_{0;0}; t_{0;1}; t_{0;2}; t_{0;3}; t_{0;4}; t_{1;0}; t_{1;1}; t_{1;2}; t_{1;3}\}$. Three service nodes are denoted as v_0; v_1 and v_2, respectively, and their resource capacities are limited. First, it is seen that sub-tasks $t_{0;0}$ and $t_{0;1}$ are allocated to service node v_0, but the required resources for v_0 are more than the capacities after deploying $t_{0;2}$ to v_0. As a result, sub-task $t_{0;2}$ is moved to service node v_1. Due to the high priority of service nodes with low index, sub-task $t_{0;3}$ will be placed to service node v_0. Similarly, we deploy sub-

计算出相应的概率分布函数（CDF）。然后，我们通过 "适者生存 "的概念来选择候选者，这意味着高适应度的个体更有可能被选中。

- *交叉算子：*我们使用两点交叉[35]作为交叉算子，交叉概率为 p_c。我们为两个个体随机选择两个基因点（分别来自 n o
母亲和父亲）的实地 1；$^{PL-1}N-2$ 至 $\vdash 0$
相互交配，每个后代都会继承一些
the genes from their parents, respectively. An example of two-point crossover is described in Fig. 5. Two individuals $p_0 = \{t_{0;3}; t_{0;1}; t_{0;4}; t_{1;0}; t_{0;0}; t_{1;1}; t_{0;2}; t_{1;2}; t_{1;3}\}$; $p_1 = \{t_{0;0}; t_{0;4}; t_{1;3}; t_{0;1}; t_{1;2}; t_{1;0}; t_{0;3}; t_{1;1}; t_{0;2}\}$ 作为亲本，并随机产生两个基因点，如 3 和 6。首先，离体弹簧从*父本* 0 继承 $t_{0;3}$；$t_{0;1}$ 和 $t_{0;4}$ 作为自己的基因。然后，需要从低到高搜索亲代 *1 的* 3 个基因（point1 - *point0* = 3），而且这些基因不能与子代的基因相同。因此，*父代* 1 中的基因 $t_{0;0}$；$t_{1;3}$ 和 $t_{1;2}$ 被继承为子代的基因。同样，子代也可以遗传 ````` 1 的基因 $t_{1;0}$；$t_{1;1}$ 和 $t_{0;2}$。来自*父代 0*。因此，子代表示为
$\{t_{0;3}; t_{0;1}; t_{0;4}; t_{0;0}; t_{1;3}; t_{1;2}; t_{1;0}; t_{1;1}; t_{0;2}\}$。类似 $t_{0;0}$; 地、
tasks $t_{0;4}$ and $t_{1;0}$ to service node v_1, sub-tasks $t_{1;1}$; $t_{1;2}$ and $t_{1;3}$ to service node v_2.
选择、交叉和变异使用了以下三种算子：

- *选择算子：*采用轮盘选择[34]来获取高适应度值的个体。所有个体都按其适应度值升序排序，累计的

$t_{0;1}$; t_{3} ; t ; $t_{0;30;1}$; t ; t ; $t_{1;01;21;1}$; $t_{0;2}$ } 可以代表另一种情况。

- *突变算子*突变算子[35]的突变概率为 p_m ，我们从一个个体中随机选择两个基因点，交换它们的基因，生成一个新个体。

改进后的 GA 算法如算法 1 所述。我们将最大迭代次数记为 $Iter_{max}$ 。开始时，随机产生初始种群 P，我们计算种群 P 中个体的目标函数拟合度并得到 CDF。然后，分别运行选择、交叉和变异算子，即可得到子代种群。在迭代进化过程中，将给出最大化 $Z_1(X)$ 的最优解。当迭代次数大于 $Iter_{max}$ 时，即满足终止标准。

根据改进后的遗传算法，我们得到了资源利用率最大化的最优解，最优解表示用于部署请求任务的服务节点集 V_a ，包含 K_a 服务节点。不过，这个子问题没有考虑带宽成本和负载平衡。下文第 4.2 节将讨论基于服务节点集 V_a 的带宽成本和负载平衡最小化子问题。

算法 1.改进的 GA

1: 初始化：种群 P , 个体 p∈ P , 交叉概率 p_c , 突变概率 p_m , 最大迭代次数 $Iter$ ； $_{max}$
2: for $Iter$ = 0 to $Iter_{max}$ do
3: 计算所有个体 P 的适应度；
4: 根据适配值获取 CDF；
5: 运行选择运算符，得到 P 个候选者； 6
： for i = 0 to $\frac{length(P)}{2}$ do
7: 随机选择两个人；
8: 生成一个随机数 p_r ; 9: if p_r ≤ p_c then
10: 运行两点交叉运算；
11: end if
12: 结束
13: 产生后代种群； 14: for i = 0 to $length(P)$ do
15: 随机选择一个人；
16: 生成一个随机数 p_r ; 17: if p_r ≤ p_m then
18: 运行突变运算符
19: end if
20: 结束
21: 更新人口 P ；
22: 结束 for

4.2 多代理优化

对于服务节点集 V_a ，我们提出了一种多代理优化方法，以解决在约束条件下最小化 $Z_2(X)$ 的优化子问题。就 MAO 而言，代理具有四个特征：环境感知、自动运行、社会行为和负载平衡[29]，具体描述如下：

1) *环境感知：* MAO 算法中的环境由所有代理和它们之间的关系组成，其中代理被预先设计为一个共享代理和 K_a 个服务代理，我们用一个有组织的架构来说明它们之间的关系。服务代理可以通过访问共享代理获得资源分配信息，并与其他服务代理进行交互，以更好地适应当前环境条件，即获得更高的适应度值。共享代理为服务代理提供访问服务，以帮助进行资源管理。

2) *自主性：* 当环境条件发生变化时，代理可以自主地做出决定，以调整其适应度值，具体做法是

3) *社会行为：* 根据社交行为，代理共享有关资源分配的信息，部署在不同服务代理上的子任务可以相互交换。社交行为有两种，一种是共享代理与服务代理之间的社交行为，另一种是服务代理之间的社交行为。对于前者，代理可以与所有服务代理共享有关资源分配的信息。对于后者，目的是交换部署在不同服务代理上的子任务，以提高其目标。

4) *负载平衡：* 负载平衡是确保充分使用服务节点的一个重要问题。因此，考虑到负载平衡，服务代理通过与其他服务代理合作、协调和竞争来运行交换运营商。

为了更好地描述 MAO 算法，下面给出了一些重要说明：

- *共享代理：* 共享代理（用 $_s$ 表示）持有所有服务代理的任务部署和资源分配信息，它可以支持服务代理实时访问和更新信息。
- *服务代理：* 服务代理被分配到 V_a 中的服务节点，以协助资源管理，它们可以访问和更新共享代理上的资源分配信息。此外，部署在不同服务代理上的子任务可以通过选择和交换操作员相 互迁移和交换。分配给 v_i 的服务代理表示为 $_i$ ，所有服务代理组成服务代理集 A 。 $_a$
- *相邻代理：* 我们假设 $6t_{l;n} \in a_i$, $Et_{l;n} \in a_j$;i / = j，其中 $t_{l;n^{\wedge}} \in$ ℧ ∪ ℧ ，则 a_i ; a_j 被视为相邻代理。
- *活动子任务：* 对于子任务 $6t_{l;n} \in a_i$ ，其相邻的子任务会迁移和交换到 $_i$ ，以通过选择和交换操作员来改善带宽成本。子任务 $t_{l;n}$ 被定义为活动子任务。
- *主机服务代理：* 如果活动子任务 $t_{l;n} \in a_i$ ，则服务代理 a_i 被视为主机服务代理。
- *可行解决方案：* 置于服务代理 a_i 上的所有子任务均表示为 L_i ，其中 L_i 被视为可行解的一部分。因此，一个可行的解决方案可表示为 X = S L_i 。 $^{Ka\text{-}1}_{i=0}$
- *目标函数：* 可行解的分解P分为 K 个 $_a$ 部分，因此目标可改写为

$z_2\ (x) = {}^{1}Ka{-}^{1}\ z_2\ \ (\mathrm{l}_i\ ).$

行动集和行为标准。对于服务代理来说，其行动集包括四个部分：访问、选择、交换和更新。首先，它通过访问共享代理获取有关资源分配的信息。然后分别执行选择和交换操作。服务代理的交互结果将在共享代理上更新。此外，共享代理的操作集可提供三种服务，即存储、访问和更新信息。

$\neq \quad i{=}0$

在 MAO 算法中，有一个共享代理 a_s 和 K 个 $_a$ 服务代理，代理之间可以通过网络链接进行通信，如 $_s$ 和 $_i$ ，以及 $_i$ 和 $_j$ $;i\ /= j$。服务代理可以通过访问共享代理 a_s 共享资源分配信息，相互迁移和交换子任务，以降低带宽成本。为了保持可行解决方案的多样性，选择和交换操作符是基于一种概率方法实现的，本节稍后将讨论这种方法。

一般来说，MAO 的程序包括四个部分：获取相邻代理、选择源和目标

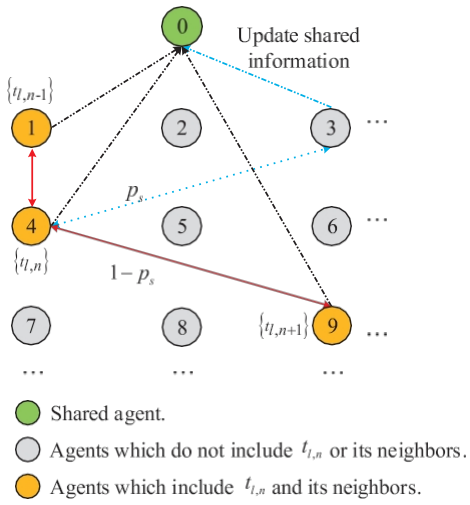


图 6.多代理优化架构。

子任务，执行交换操作符，并更新共享信息。首先，选择子任务 $6t_{l;n} \in a_i$ 作为活动子任务，其所有相邻子任务 $U_{l;n}$ =

$U^{p}_{l;n}$ ∪ $U^{s}_{l;n}$ 及其服务代理可通过访问

然后，可以从相应的 U 中对每个子任务执行选择和交换操作。然后可以从对应的 $U_{l;n}$ 中对每个子任务执行选择和交换操作。对于选择操作，我们需要选择源子任务和目标子任务，其中目标子任务通过概率方法选择，而来自$_i$ 的源子任务则通过基于优先级的选择机制获得。基于优先级的选择机制由两部分组成，一部分是相邻两个服务代理的子任务的依赖关系，另一部分是服务器节点使用的资源，详细说明见下表

第 4.2.2 节对于交换操作员，我们可以迁移和

在不同服务器之间交换源任务和目标子任务

第 4.2.3 节清楚地描述了具有多重约束条件的副代理，其中包括目标优化、资源约束、负载平衡和可行解决方案的多样性。此外，如果目标契合度在此时没有得到改善，则可以交换概率 p_e 进行交换操作。交换完成后，共享信息将被更新。

图 6 举例说明了 MAO 算法。我们假设主机服务代理是$_4$ ，活动子任务是 $t_{l;n}$ 。服务代理 a_4 可以获取信息

$_{l;n}$通过访问共享的

代理，并且有两个相邻的子任务 $t_{l;n-1} \in a_1$; $t_{l;n+1} \in a_9$ 。对于子任务 $t_{l;n+1}$ ，以 1 - p_s 的选择概率选择它作为目标子任务，否则选择一个随机子任务 $_{tl^{\wedge},}$n^∈ a_3 作为目标子任务。$_4$ 中的源子任务由基于优先级的选择机制选出。然后，我们就可以运行交换操作并更新共享信息。同样，我们

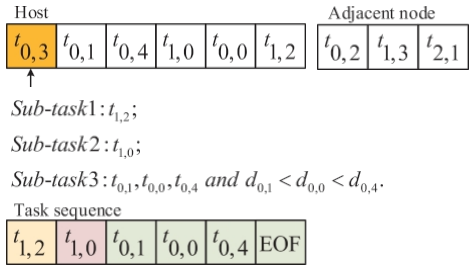


图 7.源子任务候选清单。

使用概率方法选择目标子任务，以避免过早收敛。

对于活动子任务 $t_{l;n} \in a_i$ ，其相邻子任务集合为 $U_{l;n}$ ，我们假设候选相邻子任务为 $t_{l;n^{\wedge}} \in U_{l;n}$ ； $t_{l;n^{\wedge}} \in a_j$;i /= j，随机子任务表示为 $_{tl^{\wedge},}$n^∈ a_k ;i /= k。因此，目标子任务为

$_{tl^{\wedge};n^{\wedge}}$ 的概率为 p_s ，否则 $t_{l;n^{\wedge}}$ 。

算法 2 中对获取选择进行了描述。

4.2.2 *来源选择*

考虑到负载平衡和目标优化，一个

基于优先级的源子任务选择机制是由

假定。在主机服务代理中，活动子任务 $t_{l;n} \in a_i$ 的所有可用子任务按优先级降序排序，其中包括两类：主机和相邻服务代理的子任务依赖关系和已用资源。根据依赖关系可分为以下三种子任务类型：

- *子任务1：* 让我们用 $t_{l;n} \in a_i$ 表示活动子任务，用$_j$;i /= j 表示相邻服务代理，用 $W_{l;n}$ $_{l;n}$表示可用子任务集。如果 $_{tl';n'} \in W_{l;n}$; t '' ∈ $_{Ul';n'}$ ，则使

  $Et'_{l;n'} \in W_{l;n}$ 和 ∈ a_j ，设 $_{tl';n'}$ 为子任务 1。

- *子任务 2：* 如果 $_{tl';n'} \in W_{l;n;\, tl';n} \in U_{Ul';n'}$ ，则使 $_{6tl';n'} \in W_{l;n}$ 和 t '' $_{l;n} \in a_j$ ，设 $_{tl';n'}$ 为子任务 2。

- ****务 3：如果 t | ' ; n ' ∈ $W_{l;n}$; t $_{l;n}$ ∈ $_{Ul';n'}$ ，则使 $Et'' \in W$; t ∈ Ul' ;n' ，则使 Et ∈W ; t ∈ $_{Ul';n'}$ 。

  $W_{l;n}$ ，设 $_{tl';n'}$ 为子任务 3。

三种子任务类型的优先级关系排序为：子任务 1 > *子任务* 2 > *子任务* 3。此外，对于同一子任务类型，我们会计算$_i$ 与每个候选子任务的系统平均值之间的资源利用率差异。所有候选子任务按差值由大到小排序。

也可以

$_{l;n-1}$ 。

接下来，我们将讨论 MAO 算法的三个主要部分。

4.2.1 目标选择

在 MAO 算法中，为了优化带宽成本，我们的目的是尽可 能 将任务中相邻的子任务分配给同一个服务代理。此外，一个

算法 2.目标选择

1: 初始化：概率 p；$_s$

2：对于 $6t_{l;n} \in a_i$; $Et_{l;n^\wedge} \in U_{l;n}$; $t_{l;n^\wedge} \in a$; a_{ji} ; $a_j \in A_a$; i /= j，$_{tl^\wedge;n^\wedge} \in a_k$; i /= k；

3: 生成一个随机数 p_r ; 4: 如果 p_r ≥ p_s 那么

5: 让 $t_{l;n^\wedge}$ 成为目标子任务；

6: 否则

7: 让 $_{tl^\wedge;n^\wedge}$ 成为目标子任务；

8: 如果结束

图 7 描述了源子任务日期列表的过程。 主机服务代理表示为 $t_{0;3}$; $t_{0;1}$; t ; $t_{0;41;0}$; $t_{0;0}$; $t_{1;2}$ ，活动子任务为 $t_{0;3}$ ，广告服务代理由 $t_{0;2}$; $t_{1;3}$ 和 $t_{2;1}$ 组成。我们可以发现

即 $t_{1;2}$ 为子任务 1，$t_{1;0}$ 为子任务 2，$t_{0;1}$；$t_{0;0}$ 和 $t_{0;4}$ 为子任务 3。对于子任务 3，子任务的差异排序为 $d_{0;1} < d_{0;0} < d_{0;4}$。因此，任务列表可表示为 $t_{1;2}$；t；$t_{1;00;1}$；$t_{0;0}$；$t_{0;4}$。

对于子任务候选列表，可根据优先级选择源子任务，不同的源子任务会对性能产生影响。源选择算法如算法 3 所示。

算法 3.源选择

1: 对于活动子任务 t ∈a，相邻的服务代理 a；t 并
2: 制作一组可用的子任务 $W_{l;n}$；3: for $6tl'$;n' ∈ $W_{l;n}$ do
4: 4: 获取相邻子任务 Ul' ;n' 的集合；
5: For t '' ∈ Ul' ;n'；
6: 如果 Et' 和 t' ∈a，则
∈/ W l' ;n' $l;n$ l' ;n' j
7: 设 tl' ;n' 为子任务 1；
8: else if $6t$ '' ∈/ $W_{l;n}$ and ' ∈/ a_j 那么
t' l ;n l ;n
9: 设 tl' ;n' 为子任务 2
；
10: 其他
11: 设 tl' ;n' 为子任务 3
；
12: end if
13: 假定 tl' ;n' 是源子任务，并计算其与 tl' ;n' 的差值。
i 的资源利用率与系统平均值之间的差异；
14: 结束 for
15: 按优先级对子任务候选列表进行排序。

4.2.3 交换程序

对于活动子任务 $t_{l;n}$，通过目标和来源选择操作给出目标子任务和来源子任务候选列表，然后服务代理可以通过交换算子相互合作、协调和竞争，迁移和交换各自的子任务，以改善目标。在这种情况下，需要考虑以下四种情况：

- *目标优化：*对于交换程序，我们的目标是将这些子任务迁移和交换到不同的服务代理上，以降低其带宽成本。也就是说，在运行交换操作员后，主机服务代理的目标适配性应得到提高。
- *资源限制：*每个服务代理使用的资源数量不能超过其资源容量。
- *负载平衡：*负载平衡通过基于优先级的源子任务选择机制以及子任务迁移和交换程序来实现。
- *可行解决方案的多样性：*交换算子是

如果等式（9）中的负载平衡约束条件得到满足，我们就会将目标子任务从$_j$ 迁移到$_i$，而不会对源子任务做任何处理。此外，当候选列表中没有可用的源子任务时，也会考虑迁移。负载平衡约束可表示为

$$_{hi} < _{hj} + 2 * h^{-}; \tag{9}$$

其中，hi 和 $_{hj}$ 表示一个$_i$ 和一个$_j$

分别表示迁移后的资源利用率。参数 h¯ 表示所有子任务类型的平均资源利用率。在大多数情况下，我们需要交换目标子任务和源子任务。
之间的相互关系。 i j
和交换满足资源限制。
交换程序如算法 4 所示。

算法 4.交换程序

1：对于目标子任务 tl^;n^，源子任务候选列表 $Q_{l;n}$，交换概率 p；$_e$
2: for $6t_{l'}$;n' ∈ $Q_{l;n}$ do

3: 如果目标优化无效，那么

4: 生成一个随机数 p_r ; 5： if p_r $> p_e$ then

6: 继续；

7: end if

8: end if

9: 如果（$t_{l';n'} = EOF$ 或（$t_{l';n'} /= EOF$ 并符合公式 (9)））并满足资源限制，则

10: 运行迁移并中断；

11: else if $t_{l';n'} /= EOF$ and hold resource constraints then 12： 运行交换并中断；

13: end if

14: 结束 for

15: 更新共享代理上的共享信息。

概率方法实现的。
如果不能满足最优化要求，交换程序

这样就能以较低的概率继续执行该程序。

接下来，我们将详细介绍交换程序。

我们用 $Q_{l;n}$ 表示源子任务列表。对于 $6t_{l';n}' \in Q_{l;n}$ ，我们首先要确保交换操作能 改善目标结果。如果没有，交换过程可以继续运行，交换概率为 p_e 。

算法 5.MAO 算法

1: 输入：最大迭代次数 M，选择概率 p_s ，交换概率 p ； $_e$

2: for m = 0 to M do

3: for $6a_i \in A_a$ do

4: for $6t_{l;n} \in a_i$ do

5: 通过访问$_s$ 获取 $U_{l;n}$ ；

6: for $6t_{l;n^\wedge} \in U_{l;n}$; $t_{l;n^\wedge} \in a_j$;$i /= j$ do

7: 运行目标选择操作；

8: 运行信号源选择操作；

9: 运行交换程序；

10: 结束

11: 结束

12: 更新 L$_i$ 为$_i$ ；

13: 结束

14: 获取可行解 X = $\bigcup_{i=0}^{Ka-1}$ L$_i$ 并计算目标健合值； 15: 结束 for

16: 获得近似最优解。

MAO 算法是一种基于多代理系统的迭代优化算法，算法 5 描述了整个 MAO 算法。我们假设

在迭代进化过程中，每个服务代理的所有子任务都会执行选择和交换操作，该服务代理的目标值会有很大概率得到改善。所有服务代理都可以通过合作共同进化法协同工作。因此，通过增加迭代次数，可以找到全局最优解。

4.3 在线资源分配中的 HMAO 算法

本文提出的 HMAO 算法只需稍加修改，即可轻松应用于在线资源分配。在本文中，我们考虑的场景是以批处理模式将服务节点相关的可用资源分配给请求的任务。也就是说，所请求的任务被收集起来，并在一个固定的时间段内处理。

与服务节点相关的所有可用资源构成一个可用资源池。此外，我们假设每个时隙都会出现一些新的请求任务和几个旧的请求任务。在每次运行假定的 HMAO 算法之前，我们需要检查上一个时隙是否有几个已完成的旧请求任务。这些已完成的旧请求任务所使用的资源可被释放并放入资源池，以便在当前时隙部署新的请求任务。然后，我们就可以运行建议的 HMAO 算法，将资源池中的可用资源分配给新的请求任务。

算法 6.在线资源分配中的 HMAO 算法

1: t 时的输入：$T_{new;t} \in T$ 的任务，设 $T_{old;t-1} \in T$ 的任务在 t - 1 结束；
2: 释放 $T_{old;t-1}$ 中旧任务所需的资源，并更新服务节点的资源信息；
3: 生成 T 的初始种群 $P_{new;t}$；
4: 运行 GA，获得用于部署新任务的服务节点集 $V_{a;t}$；
5: 对 $V_{a;t}$ 执行 MAO 算法；
6: 获得将可用资源分配给 $T_{n;t}$ 的近似最优解。
7: 时间：$t \leftarrow t + 1$。

为了提高遗传算法初始解的质量，一个个体按这些子任务的相互依赖关系依次编码，而群体中的其他个体则随机编码，以保持可行解的多样性。在遗传算法的解码过程中，我们将用于部署请求任务的服务节点按资源利用率升序排序。个体的所有子任务都会按照 "最先适合 "规则分配给可用的服务节点[37]、[38]，当任何一个可用的服务节点无法满足要分配的子任务的资源要求时，就会激活一个新的服务节点。算法 6 描述了在线资源分配中的 HMAO 算法。对于时隙 t，我们将新请求任务的集合表示为 $T_{new;t} \in T$，将即将结束的旧请求任务的集合表示为 $T_{old;t-1} \in T$。首先，我们结束 $T_{old;t-1}$ 中的旧任务，并释放已使用的所需资源。然后

表 2
任务和服务节点的资源需求

名称	中央处理器（兆赫）	内存（GB）	GPU	带宽（兆比特/秒）
网络接收	290	9.6	0	100
捕获	319	11.52	1	97
跟踪	435	12.48	1	95
同步	638	12.48	1	92
解码	145	4.8	1	90
服务器节点	2900	96	8	1000

根据已使用的现有资源和物理资源限制，我们使用提议的 HMAO 算法将服务节点的可用资源分配给新请求的任务。

5 绩效评估

在本节中，我们针对不同数量的通信任务进行了实验，通过计算机仿真结果来验证所提出的 HMAO 算法的性能。同时，通过与现有的两种基线算法（GA 和 NSGA-II）进行比较，分析了所提出的 HMAO 算法的性能。实验平台为高性能服务器，CPU 为 i7-4790k，内存为 16 GB，操作系统为 Windows 10。每个通信任务包括 5 个部分：网络接收、容量、跟踪、同步和解码。所有服务节点都是同质的，具有相同的资源能力。任务和服务节点的资源需求见表 2。需要注意的是，本文没有考虑交换机和路由器等网络链接的限制。公式（8）用于比较拟议的 HMAO 算法和其他现有基线算法的性能指标。

5.1 模拟参数设置

首先，我们假设等式（8）中的权重值相同，可以描述为 $a_c = a_m = a = {}_g^1$ 和 $_{b1=b2}={}^1$。改进型 GA 的参数为 $P = 16$；$Iter_{max} = 5$；$p_c = 1.0$；$p_m = 0.1$。在 MAO 算法中，有三个主要参数：选择概率 p_s、交换概率 p_e 和迭代次数，它们对性能结果有影响。为了更好地估算不同参数组合对性能的影响，我们采用田口试验设计法（DOE）来生成试验案例，并对试验结果进行分析。

对实验结果进行分析[29]。有 3 个因素，每个因素包含 4 个水平，田口方法的不同参数组合如表 3 所示。此外，我们还制作了正交表 $L_{16}(4^3)$，其中有 16 个案例，每个案例分别对 8、16、24、32 个任务执行 10 次，得出平均结果。表 4 提供了正交表 $L_{16}(4^3)$ 和不同任务近似解的相关值。

图 8 显示了不同任务平均值的主效应图。很明显，随着 p_s ; p_e 和迭代次数的增加，目标值得到了改善。

$\overline{3}$

$\overline{2}$

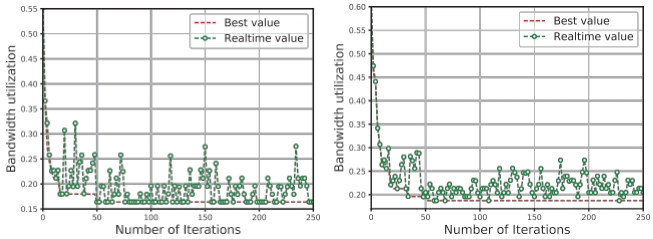
表 3
田口方法参数

系数	水平			
	1	2	3	4
p_s	0.0s1	0.05	0.10	0.15
p_e	0.01	0.05	0.10	0.15
M	250	500	750	1000

表 4
正交表 $L_{16}(4^3)$ 和 $Z(X)$ 的最优解

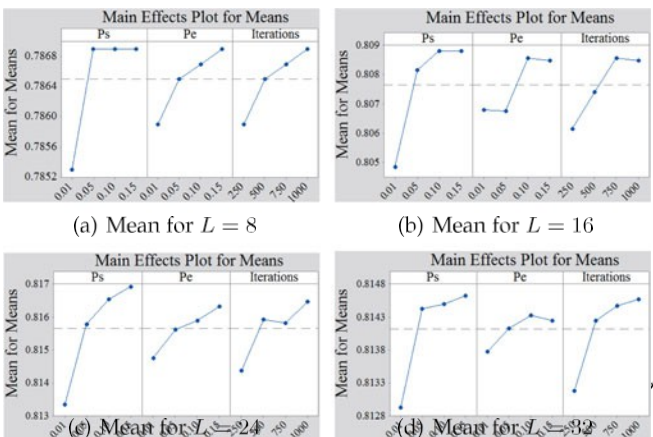
不	系数			$Z(X)$			
	p	P	M_s $_e$	$L=8$	$L=16$	$L=24$	$L=32$
0	0.01	0.01	250	0.7829	0.8008	0.8103	0.8109
1	0.01	0.05	500	0.7853	0.8032	0.8141	0.8133
2	0.01	0.10	750	0.7861	0.8079	0.8138	0.8136
3	0.01	0.15	1000	0.7869	0.8075	0.8152	0.8139
4	0.05	0.01	500	0.7869	0.8088	0.8156	0.8146
5	0.05	0.05	250	0.7869	0.8062	0.8144	0.8136
6	0.05	0.10	1000	0.7869	0.8088	0.8167	0.8148
7	0.05	0.15	750	0.7869	0.8088	0.8164	0.8147
8	0.10	0.01	750	0.7869	0.8088	0.8161	0.8148
9	0.10	0.05	1000	0.7869	0.8088	0.8170	0.8148
10	0.10	0.10	250	0.7869	0.8088	0.8161	0.8141
11	0.10	0.15	500	0.7869	0.8088	0.8170	0.8143
12	0.15	0.01	1000	0.7869	0.8088	0.8170	0.8148
13	0.15	0.05	750	0.7869	0.8088	0.8170	0.8148
14	0.15	0.10	500	0.7869	0.8088	0.8170	0.8148
15	0.15	0.15	250	0.7869	0.8088	0.8167	0.8141

对于所提出的 HMAO 算法，我们可以根据可行解的多样性调整 p_s ; p_e 的值，但是，这将花费更多的时间为 $Z(X)$。对于 8、16、24、32 项任务，p_s 的理想值为 0.15，对于 8、24 项任务，p_e 的理想值为 0.1，对于 16、32 项任务，p_e 的理想值为 0.15。



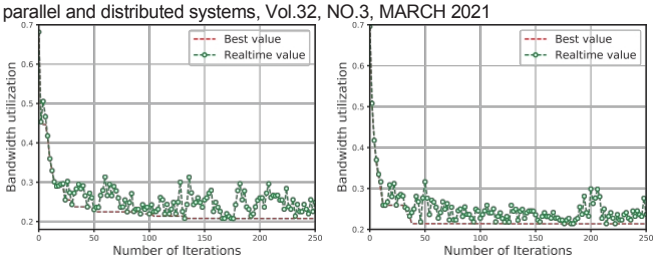
(a) $L=8, p_s=0.15, p_e=0.15$ (b) $L=16, p_s=0.15, p_e=0.10$

图 8.$L=8$; 16; 24; 32 的平均值。



(a) Mean for $L=8$ (b) Mean for $L=16$

(c) Mean for $L=24$ (d) Mean for $L=32$



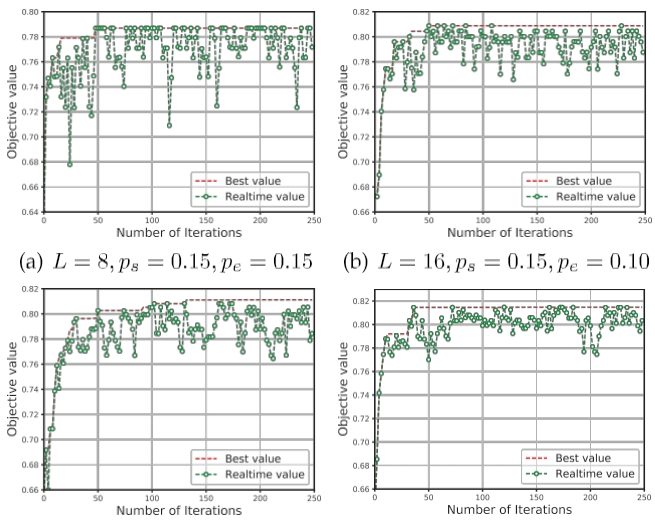
(c) $L = 24, p_s = 0.15, p_e = 0.15$ (d) $L = 32, p_s = 0.15, p_e = 0.10$

图 9.$L = 8$；16；24；32 时的带宽利用率。

5.2 数值模拟

根据上述参数，设定最大迭代次数为 250 次，我们进行了四次不同的实验，请求的任务数分别为 8、16、24 和 32，以评估所提出的 HMAO 算法的收敛性。如图 9 所示，我们展示了不同任务的迭代次数与带宽利用率之间的关系。图 9a 和 9c 显示了在 $p_s = 0.15$、$p_e = 0.15$ 时寻求带宽利用率最优解的过程，分别在迭代 49 次和 176 次时观察到最佳值 0.1640 和 0.0.2076。对于 $L = 16; 32$，$p_s = 0.15$，$p_e = 0.10$，其优化带宽利用率的结果如图 9b 和图 9d 所示，分别在 73 次和 121 次迭代时得到最优解结果 0.1872 和 0.2137。

同样，图 10 描述了不同任务迭代最优解的演化图，其中包括最佳和实时目标值。图 10a 和 10c



(a) $L = 8, p_s = 0.15, p_e = 0.15$ (b) $L = 16, p_s = 0.15, p_e = 0.10$

(c) $L = 24, p_s = 0.15, p_e = 0.15$ (d) $L = 32, p_s = 0.15, p_e = 0.10$

图 10.$L = 8$；16；24；32 时的目标值。

表 5
HMAO 算法的模拟结果

L	M	Min	Max	Mean	Std	Time(min)
8	250	0.7869	0.7869	0.7869	0.0	0.2375
16		0.8046	0.8088	0.8084	0.0013	0.4777
24		0.8141	0.8170	0.8164	0.0012	0.7205
32		0.8104	0.8148	0.8137	0.0015	0.9703
8	500	0.7869	0.7869	0.7869	0.0	0.4995
16		0.8088	0.8088	0.8088	0.0	1.0125
24		0.8170	0.8170	0.8170	0.0	1.5177
32		0.8146	0.8148	0.8147	0.0001	2.0164

图 10b 和图 10d 显示了 8、24 项任务的模拟结果，其中 p_s = 0.15，p_e = 0.*15*；图 10b 和图 10d 显示了 16、32 项任务的模拟结果，其中 p_s = 0.15，p_e = 0.10。可以看出，8、16、24 和 32 项任务的目标值收敛到近似解结果，分别为 0.7869、0 . 8088、0.8112 和 0.8112。

迭代次数分别为 49 次、73 次、176 次和 121 次时的收敛性为 0.8148。由此可见，所提出的 HMAO 算法是解决云计算资源分配问题的有效优化算法，具有较好的收敛性能。

由于迭代次数对 HMAO 算法的性能有影响，因此分别对 L = 8、16、24 和 32 进行了 250 次和 500 次迭代实验。每种情况运行 10 次，我们可以得到最优解的最小值、最大值、平均值和标准偏差以及平均计算时间，如表 5 所示。从表 5 中可以看出，当迭代次数增加时，所提出的 HMAO 算法的性能较高，即迭代 250 次的 8、16、24 和 32 任务的平均值分别为 0.5、0.5、0.5、0.5、0.5、0.5、0.5、0.5、0.5、0.5、0.5、0.5 和 0.5。

分别为 0.7869、0 . 8084、0.8164 和 0.8137。

迭代 500 次后的平均值分别为 0.7869、0.8088、0.8170 和 0.8147。也就是说，通过迭代探索和利用解空间，更有可能找到 Z（X）的潜在最优解。此外，结果表明，随着迭代次数的增加，拟议 HMAO 算法的计算时间几乎呈线性增长。

5.3 与基准算法的性能比较

为了进一步讨论所提出的 HMAO 算法的有效性，我们将针对不同任务提出的 HMAO 算法与现有的两种算法（即 GA 和 NSGA-II）进行了比较。

*遗传算法*本文采用了 [9] 中描述的遗传算法。我们从群体 P 中随机选取两个个体作为亲代，它们可以通过两点交叉算子（交叉概率为 p_c）相互交配，从而产生

表 6
HMAO、GA 和 NSGA-II 的参数设置

参数	数值
任务数 L	{4; 5; 6; 7; 8; 9; 10; 11; 12; 13; 14; 15; 16}
服务节点数 K	{4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16}

子代。如果子代的适应度值优于父代，我们将把它作为下一代的候选者。否则，我们将使用突变算子，突变概率为 p_m 。如果子代与其父代的适配值更高，则该个体将被视为下一代的新个体。

*NSGA-II.*我们在 [15] 中引入了 NSGA-II 来解决我们的问题，两个优化子问题分别是 Z_1 *(X)* 和 1 - Z_2 *(X)*。二元锦标赛选择方法

[34]来决定选择父母

最大迭代次数 M	HMAO: 1000; GA,NSGA-II:
5000 种群大小 P	HMAO: 16; GA,NSGA-II: 100
选择概率 p_e	0.15
交换概率 p_s	0.15
交叉概率 p_c	1.0
突变概率 p_m	0.1
运行时间	10

在交叉算子中，我们随机选择一个个体的两个基因点与另一个个体等价交换基因，交叉概率为 p 。在交叉算子中，我们随机选取一个个体的两个基因点，在交叉概率 p_c 的条件下，与另一个个体等价交换基因。在突变概率 p_m 的条件下，执行位向突变。

根据上述对拟议 HMAO 算法的分析，我们将拟议 HMAO 算法的最大迭代次数设为 1 000 次，将 GA 和 NSGA-II 设为 5 000 次，将拟议 HMAO 算法的种群 P 设为 16 个，将 GA 和 NSGA-II 设为 100 个。此外，$p_e = p_s = 0.15$，$p_c = 1.0$，$p_m = 0$ 1。任务数为 4 至 16，与初始服务节点数相对应分别为 4、5、16。每个测试案例运行 10 次，计算平均结果。表 6 列出了拟议的 HMAO 算法、GA 算法和 NSGA-II 算法所用参数的总和。

首先，我们利用表 6 中指定的参数，通过拟议的 HMAO 算法、GA 算法和 NSGA-II 算法模拟所有测试案例，得出这三种算 法 的平均结果，包括资源利用率、带宽利用率、目标值和时间成本。需要注意的是，初始服务节点数 K 会影响 GA 和 NSGA-II 的结果。为了研究拟议的 HMAO 算法的最优解随时间的变化情况，我们比较了拟议的 HMAO 算法、GA 和 NSGA-II 在 $L = 4$、6、8、10 时的 演化性能，三种算法的结果如图 11 所示。可以看出，在 $L = 4$、6、8 时，拟议 HMAO 算法的性能与 GA 算法相似；在 $L = 4$、8 时，拟议 HMAO 算法的性能与 NSGA-II 算法相似。Moreover, the convergence of the proposed HMAO algorithm is better than GA and NSGA-II as the 任务数量增加。

其原因在于，拟议 HMAO 算法的选择和交换算子是在更细粒度的子任务级别上执行的，优化目标在每次迭代演化过程中都以高概率趋向良好结果。此外，基于概率的选择和交换算子可实现可行解的多样性，避免过早进入局部最优解。然而，GA 的可行解空间

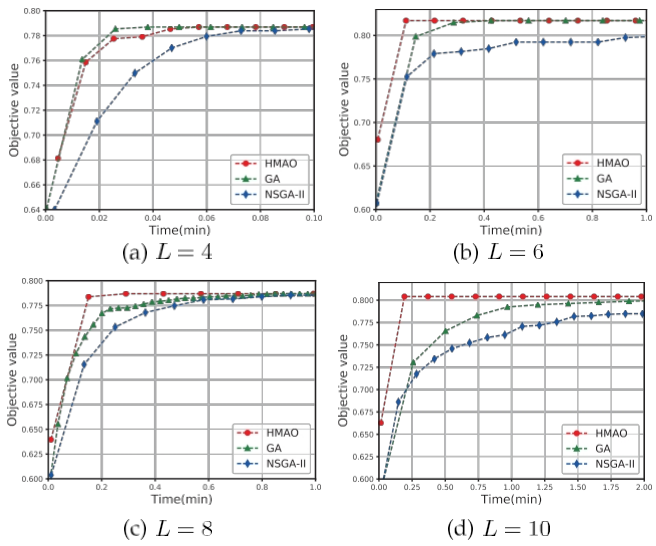


图 11.使用 HMAO、GA 和 NSGA- II 时 $L = 4$；6；8；10 的目标值。

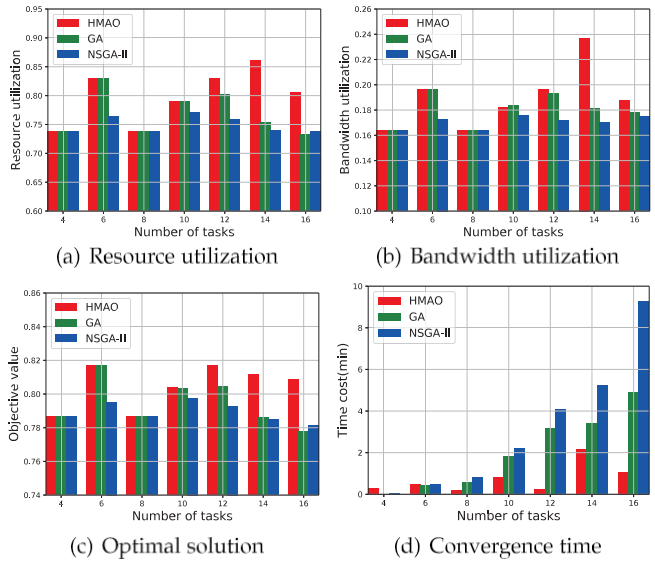


图 12.HMAO、GA 和 NSGA-II 的性能比较。

随着任务数的增加，GA 和 NSGA-II 的适配值会变大，即搜索最优解需要更多时间。此外，从图 11 可以看出，在 $L = 4$、6、8、10 时，GA 的性能优于 NSGA-II，这是因为 GA 产生的每个子代都会被选为下一代的新个体，其适应度值优于父代，这就保证了每次迭代进化都能提高优化问题的求解性能。

为了进一步分析所提出的 HMAO 算法的性能，我们提供了所提出的 HMAO 算法、GA 和 NSGA-II 的仿真结果的最大值、最小值、平均值、标准偏差和平均收敛时间，详细信息可参见表 7。在除 $L = 11$ 以外的所有测试条件下，HMAO 算法得到的最优解的最大值、最小值、平均值和标准偏差均优于或等于 GA 和 NSGA-II 、

此外，在 L = 4；6 时，所提出的 HMAO 算法的收敛时间比 GA、NSGA-II 慢，但在其他情况下收敛时间较快。此外，在 $L = 4$ 和 6 的情况下，所提出的 HMAO 算法的收敛时间比 GA 和 NSGA-II 慢，但在其他情况下收敛时间较快，随着任务数的增加，差异更加明显。例如，在 $L = 14$ 的情况下，我们可以观察到所提出的 HMAO 算法的性能分别提高了 3.27 和 3.27。3.44%，收敛时间分别缩短了 35.67% 和 GA 和 NSGA-II 分别为 58.38%。当 $L = 16$ 时，拟议的 HMAO 算法的平均结果分别提高了 4.26% 和 3.49%，收敛时间缩短了 GA和NSGA-II分别为78.23%和88.57%。结果表明，所提出的 HMAO 算法是一种有效的资源分配优化方法，在解的质量、鲁棒性和收敛性方面都有较好的表现。

此外，我们还在图 12 中比较了所提出的 HMAO 算法与 GA 和 NSGA-II 的结果、

表 7
HMAO、GA 和 NSGA-II 的模拟结果

L	HMAO					GA					NSGA-II				
	最小	最大	平均值	标准	时间（分钟）	最小	最大	平均值	标准	时间（分钟）	最小	最大	平均值	标准	时间（分钟）
4	0.7869	0.7869	0.7869	0.0000	0.2737	0.7869	0.7869	0.7869	0.0000	0.0231	0.7869	0.7869	0.7869	0.0000	0.0569
5	0.7718	0.7718	0.7718	0.0000	0.0520	0.7718	0.7718	0.7718	0.0000	0.0481	0.7603	0.7718	0.7706	0.0034	0.1408
6	0.8170	0.8170	0.8170	0.0000	0.4752	0.817	0.817	0.817	0.0000	0.4573	0.7628	0.8170	0.7953	0.0265	0.4666
7	0.7989	0.7989	0.7989	0.0000	0.0645	0.7989	0.7989	0.7989	0.0000	0.2841	0.7567	0.7989	0.7937	0.0126	1.0380
8	0.7869	0.7869	0.7869	0.0000	0.1903	0.7869	0.7869	0.7869	0.0000	0.5710	0.7869	0.7869	0.7869	0.0000	0.8126

9	0.8170	0.8170	0.8170	0.0000	0.5219	0.8170	0.8170	0.8170	0.0000	2.3043	0.7782	0.8170	0.7899	0.0177	2.1777
10	0.8041	0.8041	0.8041	0.0000	0.8174	0.7975	0.8041	0.8034	0.0020	1.8199	0.7718	0.8041	0.7976	0.0129	2.1903
11	0.7368	0.8102	0.7874	0.0283	0.8174	0.7887	0.7944	0.7938	0.0017	1.7007	0.7771	0.7944	0.7909	0.0052	2.3302
12	0.8170	0.8170	0.8170	0.0000	0.2446	0.7818	0.8170	0.8045	0.0135	3.1743	0.7628	0.8170	0.7929	0.0161	4.0658
13	0.8070	0.8070	0.8070	0.0000	1.0464	0.7715	0.8069	0.7973	0.0104	3.8402	0.7762	0.8070	0.7977	0.0131	4.2793
14	0.8116	0.8119	0.8118	0.0001	2.1852	0.7677	0.7989	0.7861	0.0127	3.3968	0.7675	0.7989	0.7848	0.0119	5.2505
15	0.8170	0.8170	0.8170	0.0000	0.9704	0.7590	0.7923	0.7811	0.0099	4.3592	0.7680	0.8075	0.7875	0.0115	7.7129
16	0.8088	0.8088	0.8088	0.0000	1.0624	0.7576	0.7869	0.7757	0.0092	4.8812	0.7714	0.7868	0.7815	0.0049	9.3008

包括资源和带宽利用率、最优解和平均收敛时间。图 12a 显示了不同任务的资源利用率，在 $L = 4$、6、8、10 的情况下，拟议 HMAO 算法的结果与 GA 算法相当，在 $L = 4$、8 的情况下与 NSGA-II 算法相当，而在其他情况下则优于 GA 算法和 NSGA-II。资源利用率较低的解决方案意味着有更多的服务节点可用于部署请求的任务。带宽利用率的结果如图 12b 所示。我们可以观察到，在 $L = 4$、6、8、10 时，拟议的 HMAO 算法的性能优于或等于 GA 和 NSGA-II。在 $L = 12$、14、16 时，这是由于拟议的 HMAO 算法的服务节点数量少于其他两种基准算法。由于请求任务使用的服务节点数量较少，将任务的相邻子任务放置到不同服务节点的概率较大，这可能导致带宽成本增加。图 12c 展示了最优解的结果，在任务数较少时，拟议的 HMAO 算法、GA 和 NSGA-II 的性能大致相当，随着任务数的增加，拟议的 HMAO 算法的性能优于 GA 和 NSGA-II。从图 12d 可以看出三种算法的平均收敛时间，当任务数较少时，拟议的 HMAO 算法的收敛时间比 GA 和 NSGA-II 算法的收敛时间短，但随着任务数的增加，拟议的 HMAO 算法的收敛时间优于 GA 和 NSGA-II。

5.4 评估在线资源分配中的 HMAO 为进一步研究建议的 HMAO 算法在在线资源分配中的性能，我们在动态环境中实施了资源分配模型，并设计了以下实验，将可用资源动态分配给请求的任务。我们为改进的 GA 算法设置了 $P = 16$；$p_c = 1.0$ 和 $p_m = 0.1$；为 MAO 算法设置了 $p_e = 0.15$；$p_s = 0.15$ 和 $M = 1000$。服务器节点总数为较大的 $K = 64$，以保证每个时隙内最大可用资源足以部署请求的任务，其中请求的任务不使用的服务器节点处于休眠或关闭状态，以节省运行成本，请求的任务只能部署到活动的服务器节点上。当请求任务的资源需求无法满足时，我们会将服务器节点从空闲或关闭状态唤醒至激活状态，并按需为请求任务提供可用资源。我们假设每个时隙都有新的请求任务出现，旧的请求任务结束，其编号从 $L_i = \{4 * i + 1; \ldots ; 4 * i + 9\}; i = 0; 1; \ldots ; 10$ 中随机产生。每个案例也运行 10 次，时隙从 0 到 30，并通过所提出的 HMAO 算法获得最优解的平均结果。此外，我们还从资源利用率、带宽利用率和最优解的角度，将提出的 HMAO 算法与 Viterbi [30] 和 Greedy [40] 两种启发式算法进行了比较。

*维特比算法*对于 [30] 中的 Viterbi 算法，放置服务链的问题由状态及其关系建模为多阶段有向图。我们可以通过 Viterbi 算法计算每个状态的累积成本。

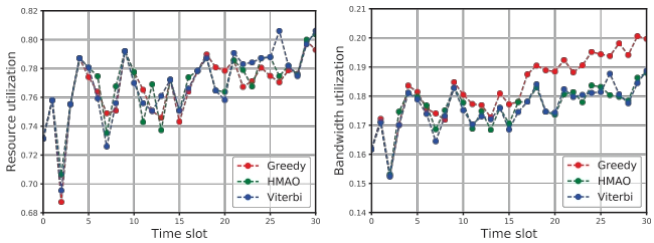


图 13.HMAO、Viterbi 和 Greedy 的性能比较。

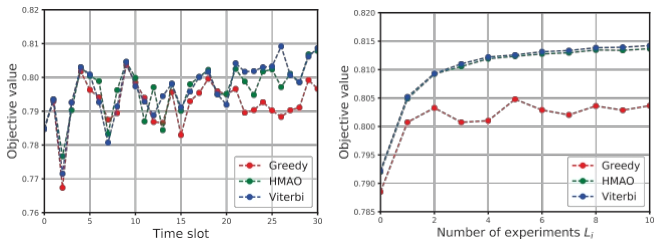
当最后阶段的成本计算结束后，成本最小的状态序列被认为是最优解。

贪心算法[40] 中描述的 Greedy 算法分为两个步骤。首先，作者用一个重要因子对所有中间盒进行降序排序，该因子代表包含相同中间盒的策略数量。其次，迭代地将中间盒分配给这些开关。为了放置中间件箱，作者可以搜索所有可用的交换机，计算相应的成本分数，找到部署中间件箱的最低成本。为了减少这些未放置的中间件的影响，作者引入了未分配中间件的加权平均成本得分来计算总成本得分。

从图 13 中可以观察到 L_0 情况下的模拟结果，图 13d 中给出了拟议的 HMAO 算法、Greedy 算法和 Viterbi 算法在所有 $L_0 \sim L_{10}$ 情况下的最优解结果。

图 13a 显示了 L_0 的计算资源利用率。从图 13a 中可以看出，在每个时隙中，建议的 HMAO 算法与 Greedy 和 Viterbi 两种比较算法的计算资源数量比较接近。L_0 的带宽利用率结果如图 13b 所示，从图中可以看出，所提出的 HMAO 算法获得的带宽利用率值优于 Greedy 算法，平均性能提高了 3.63%。这是因为拟议的 HMAO 算法可将任务的相邻子任务尽可能迁移和交换到同一服务节点，以减少请求任务所占用的带宽资源。拟议的 HMAO 算法与 Viterbi 算法性能相当。图 13c 描述了 L_0 的最优解的客观结果，从图中我们可以看出，建议的 HMAO 算法比 Greedy 方法表现更好，建议的 HMAO 算法的性能平均提高了 0.54%。此外，拟议的 HMAO 算法和 Viterbi 算法

(a) Resource utilization with L_0 (b) Bandwidth utilization with L_0



(c) Optimal solution with L_0 (d) Optimal solutions with $L_{0\sim10}$

显示了类似的结果。为了更好地分析拟议的 HMAO 算法在一个时隙内处理不同数量的请求任务时的性能，我们在图 13d 中提供了 $L_0 \sim L_{10}$ 的最优解的平均结果。很明显，在 $L_0 \sim L_{10}$ 的情况下，建议的 HMAO 算法优于 Greedy 方法，建议的 HMAO 算法的平均性能提高了近 1.06%。我们提出的 HMAO 算法与 Viterbi 算法性能接近，平均性能差异为 0.04%。从图 13 可以看出，在动态资源分配方面，拟议的 HMAO 算法优于 Greedy 算法，接近 Viterbi 算法。

6 结论

本文研究云计算系统中的资源分配问题。我们的目标是最大化基于 CPU、内存和 GPU 的资源利用率，并最小化带宽成本。为了解决这个问题，我们提出了 HMAO 算法，该算法结合了改进的 GA 算法和 MAO 算法，其中改进的 GA 算法旨在找到最优资源利用方案，而 MAO 算法旨在使带宽成本最小化。在 MAO 算法中，我们使用基于优先级的选择机制来获取候选源子任务，并通过概率方法设计选择和交换算子，以便在多个服务代理上迁移和交换子任务。所提出的 HMAO 算法可以通过合作共同进化的方法获得客观最优结果。

最后，我们通过仿真实验来验证和评估所提出的 HMAO 算法的性能。与 GA 和 NSGA-II 相比，我们可以发现，随着任务数量的增加，所提出的 HMAO 算法在求解质量、收敛时间和鲁棒性方面都优于它们。对于 $L = 14$，所提出的 HMAO 算法的性能比 GA 提高了 3.38%，比 NSGA-II 提高了 3.44%，平均收敛时间比 GA 缩短了 19.34%，比 NSGA-II 缩短了 58.38%。此外，我们还比较了拟议的 HMAO 算法与 Greedy 和 Viterbi 算法在在线资源分配方面的性能。我们可以发现，拟议的 HMAO 算法与 Viterbi 算法的性能几乎相同，比 Greedy 算法提高了约 1.06%。

参考资料

[1] H.Kim 和 N. Feamster，"利用软件定义网络改进网络管理"，《*电气和电子工程师学会通讯》杂志，第 51 卷第 2 期。杂志》*，第 51 卷，第 2 期、pp.114-119, Feb. 2013.

[2] S.Sezer *等人*，"我们为 SDN 做好准备了吗？软件定义网络的实施挑战》，*IEEE Commun.杂志》*，第 51 卷，第 7 期，第 36-43 页，2013 年 7 月。

[3] H.Erdogmus, "云计算：涅槃是否隐藏在星云之后？*IEEE Softw.》*，第 26 卷，第 2 期，第 4-6 页，2009 年 3 月/4 月。

[4] H.Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *Proc.Conf.Netw.服务管理》*，2014 年，第 418-423 页。

[5] S.S. M. A. Kazmi, N. H. Tran, T. M. Ho, and C. S. Hong, "Hierarchical matching game for service selection and resource purchasing in wireless network virtualization," *IEEE Commun. Lett.Lett.*, vol. 22, no. 1, pp.

[6] H.Zheng, Y. Feng, and J. Tan, "A hybrid energy-aware resource allocation approach in cloud manufacturing environment," *IEEE Access, vol.* 5, pp.

[7] W.Rankothge, F. Le, A. Russo, and J. Lobo, "Optimizing resource allocation for virtualized network functions in a cloud center using genetic algorithms," *IEEE Trans.Netw.服务管理》*，第 14 卷，第 2 期，第 343-356 页，2017 年 6 月。

[8] L.Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," *IEEE Trans.通信》*，第 64 卷，第 9 期，第 3746-3758 页，2016 年 9 月。

[9] F.-H. Tseng, X. Wang, L.-D.Tseng, X. Wang, L.-D. Chou, H.-C.Chou, H.-C.Chao, and V. C. M. Leung, "Dynamic resource prediction and allocation for cloud data center using the multiobjective genetic algorithm," *IEEE Syst.J.》*，第 12 卷，第 2 期，第 1688-1699 页，2018 年 6 月。

[10] B.Tan, H. Ma, and Y. Mei, "A NSGA-II-based approach for service resource allocation in cloud," in *Proc.Evol. Com- put.*, 2017, pp.

[11] S.Khebbache, M. Hadji, and D. Zeghlache, "A multi-objective non-dominated sorting genetic algorithm for VNF chains placement," in *Proc.Consum.Commun.Netw.Conf.*, 2018, pp.

[12] Q.Sun, P. Lu, W. Lu, and Z. Zhu, "*Forecast-assisted* NFV service chain deployment based on *affiliation-aware* vNF placement," in *Proc.Conf.*, 2016, pp.

[13] E.Amaldi *等人*，"论虚拟网络嵌入问题的计算复杂性"，*Electron.Notes Discrete Math.》*，第 52 卷，第 213-220 页，2016 年。

[14] B.Addis, M. Gao, and G. Carello, "On the complexity of a virtual network function placement and routing problem," *Electron.Notes Discrete Math.》*，第 69 卷，第 197-204 页，2018 年。

[15] K.Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm：NSGA-II," *IEEE Trans.*6, no. 2, pp.

[16] S.E. Dashti and A. M. Rahmani, "Dynamic VMs placement for energy efficiency by PSO in cloud computing," *J. Exp. Theor.Artif.Intell.*, vol. 28, no. 1/2, pp.

[17] K.Mani and R. M. Krishnan, "Flexible cost based cloud resource provisioning using enhanced PSO," *Int.J. Comput.Intell.Res.》*，第 13 卷，第 6 期，第 1441-1453 页，2017 年。

[18] P.-Y. Yin 和 J.-Y.Yin and J.-Y. Wang, "Ant colony optimization for the nonlin-ear resource allocation problem," Appl.Wang, "Ant colony optimization for the nonlin- ear resource allocation problem," *Appl.计算》*，第 174 卷，第 2 期，第 1438-1453 页，2006 年。

[19] B.Muthulakshmi and K. Somasundaram, "A hybrid ABC-SA based optimized scheduling and resource allocation for cloud environment," *Cluster Comput., vol.* 22, pp.

[20] M.N. Omidvar, X. Li, and K. Tang, "Designing benchmark problems for large-scale continuous optimization," *Inf.科学》*，第 316 卷、
pp.419-436, 2015.

[21] Y.Liu, X. Yao, Q. Zhao, and T. Higuchi, "Scaling up fast evolutionary programming with cooperative coevolution," in *Proc.进化计算》*，2001 年，第 1101-1108 页。

[22] X.Wu, Y. Wang, J. Liu, and N. Fan, "A new hybrid algorithm for solving large scale global optimization problems," *IEEE Access*, vol. 7, pp.

[23] Z.Ren, Y. Liang, A. Zhang, Y. Yang, Z. Feng, and L. Wang, "Boosting cooperative coevolution for large scale optimization with a fine-grained computation resource allocation strategy," *IEEE Trans.Cybern.》*，第 49 卷第 12 期，第 4180-4193 页，2019 年 12 月。

[24] M.Yang *et al.*, "Efficient resource allocation in cooperative co-evolution for large-scale global optimization," *IEEE Trans.Evol.Com-put.*，vol.21，no.4, pp.

[25] P.Stone and M. Veloso, "Multiagent systems：A survey from a machine learning perspective," *Auton.Robots*, vol. 8, no.3, pp.

[26] T.Kaihara, "Multi-agent based supply chain modelling with dynamic environment," *Int.J. Prod.*85, no. 2, pp.

[27] M.M. de Weerdt, Y. Zhang, and T. Klos, "Multiagent task allocation in social networks," *Auton.*25, no. 1, pp.

[28] W.Wang, Y. Jiang, and W. Wu, "Multiagent-based resource allocation for energy minimization in cloud computing systems," *IEEE Trans.Syst.Syst.》*，第 47 卷，第 2 期，第 205-220 页，2017 年 2 月。

[29] X.-L. Zheng and L. Wang, "A multi-agent optimization algorithm for resource constrained project scheduling problem," *Expert Syst.应用》*，第 42 卷，第 15/16 期，第 6039-6049 页，2015 年。

[30] F. 巴里、S . R . 乔杜里、R . 艾哈迈德、R . 布塔巴和 O.C. M. B. Duarte，"协调虚拟化网络功能"。*IEEE Trans.Netw.服务管理》*，第 13 卷，第 4 期，第 725-739 页，2016 年 12 月。4, pp.

[31] B.Kar, E. H.-K.Wu, and Y.-D. Lin, "Energy cost optimization in dynamic placement in virtualized network function chains," IEEE Trans.Lin, "Energy cost optimization in dynamic placement of virtualized network function chains," *IEEE Trans.Netw.服务管理》*，第 15 卷，第 1 期，第 372-386 页，2018 年 3 月。

[32] N.Srinivas and K. Deb, "Muiltiobjective optimization using non-dominated sorting in genetic a l g o r i t h m s ," *Evol. Comput*.3, pp.

[33] L.L. F. Nawaf, S. M. Allen, and O. Rana, "Optimizing infrastructure placement in wireless mesh networks using NSGA-II," in *Proc.Conf.High Perform.Comput.Commun.IEEE 16th Int.Conf.智能城市 IEEE 4th Int.Conf.数据科学与系统》*，2018 年，第 1669-1676 页。

[34] N.N. M. Razali *等人*，"在求解 TSP 时采用不同选择策略的遗传算法性能"，*Proc.World Congr.Eng.*，2011 年，第 1-6 页。

[35] S.Hartmann, "A competitive genetic algorithm for resource-constrained project scheduling," *Nav.Res.物流》*，第 45 卷，第 7 期、
pp.733-750, 1998.

[36] L.Nawaf, S. M. Allen, and O. Rana, "*Internet* transit access point placement and bandidth allocation in wireless mesh networks," in *Proc.Comput.Commun.Workshop Conf*、
pp.1-8.

[37] D.Bhamare *et al.*, "Optimal virtual network function placement in multi-cloud service function chaining architecture," *Comput.Commun.*，第 102 卷，第 1-16 页，2017 年。

[38] D.Espling, L. Larsson, W. Li, J. Tordsson, and E. Elmroth, "Modeling and placement of cloud services with internal structure," *IEEE Trans.云计算》*，第 4 卷，第 4 期，第 429-44 页。4, pp.

[39] S.Kaur 和 A. Verma，"云计算环境中任务调度遗传算法的有效方法"，*Int.J. Inf.Technol.Comput.Sci.*，第 4 卷，第 10 期，第 74-79 页，2012 年。

[40] J.Liu, Y. Li, Y. Zhang, L. Su, and D. Jin, "Improve service chaining performance with optimized middlebox placement," *IEEE Trans.服务计算》，第*10 卷，第 4 期，第 560-573 页。4, pp.

高祥强 2012 年获西安电子科技大学电子工程学院学士学位，2015 年获西安微电子技术研究所硕士学位。目前，他正在北京航空航天大学电子与信息工程学院攻读博士学位。他的研究兴趣包括无鼠码、软件定义网络和网络功能虚拟化。

刘荣科（IEEE 高级会员）分别于 1996 年和 2002 年获得中国北京航空航天大学学士和博士学位。他曾分别于 2006 年、2015 年和 2018 年在美国佛罗里达理工学院、日本东京大学和英国爱丁堡大学担任客座教授。现任北京航空航天大学电子信息工程学院正教授。曾获教育部新世纪优秀人才支持计划、
中国。他参加过许多特别项目，如中国地面数字广播标准。他在国际会议和期刊上撰写或合著了 100 多篇论文。他已获得 20 多项专利。他的研究兴趣包括无线通信和空间信息网络。

Aryan Kaushik（电气和电子工程师学会会员）于 2015 年获得香港科技大学电信硕士学位，并于 2020 年获得英国爱丁堡大学工程学院数字通信研究所通信工程博士学位。他目前是英国伦敦大学学院通信与互联系统研究所的通信与雷达传输研究员。
2019-20年在英国伦敦帝国学院无线通信与信号处理实验室任研究员，2018年在卢森堡大学跨学科安全、可靠性与信任中心任研究员，2017-19年在中国北京航空航天大学电子与信息工程学院任研究员。他的研究兴趣大致包括信号处理、雷达、无线通信、毫米波和多天线通信。