匿名稿件编号。（由编辑填写）

具有安全约束的跨多云工作流成本与可靠性感知调度

Atherve Tekawade · Suman Banerjee

arXiv:2304.00313v1 [cs.DC] 1 Apr 2023

收稿日期：日期 / 录用日期：日期

摘要 许多现实世界中的科学工作流都可以表示为有向无环图（DAG），其中每个节点代表一个任务，而有向边表示两个任务之间的依赖关系。由于这些工作流的计算资源需求不断增加，它们被部署在多云系统上执行。本文提出一种调度算法，采用高效的列表调度方法，根据成本、处理时间和可靠性等参数为工作流中的任务分配资源。随后，对于给定的任务-资源映射，我们提出一种密码分配算法，在满足给定安全约束的前提下，以时间最优的方式为负责传输数据的边分配安全服务。我们分析了所提算法的时间和空间需求。我们实现了所提调度算法和密码分配算法，并在两个现实世界的科学工作流——表观基因组学和 Cybershake——上进行了实验。我们将所提调度算法的性能与先进的进化方法进行了比较。结果表明，我们的方法在成本和可靠性方面始终优于先进方法，但在某些情况下，完工时间方面的表现较差。

关键词 多云系统 · 工作流 · 虚拟机 · 数据-安全性 · 定价方案 · 调度器

Atherve Tekawade，印度理工学院查谟分校计算机科学与工程系。电子邮箱：2018uee0137@iitjammu.ac.in Suman Banerjee，印度理工学院卡哈拉格普尔分校计算机科学与工程系。电子邮箱：suman.banerjee@iitjammu.ac.in

1 引言

近年来，云计算 已成为一种替代性的计算框架，并因许多特性而广受欢迎，包括“按使用付费”类的计费策略、虚拟化、快速弹性、按需使用等 [20]。科学工作流被定义为一组任务，其中任务之间存在多种依赖关系 [32]。近年来，云基础设施已被广泛用于工作流的执行，来自不同领域的许多科学工作流，例如生物信息学中的表观基因组学、地震工程中的 *Cybershake* 等，均已成功部署在商业云上 [11, 32]。最近，由于大规模计算资源和多样化需求，多个云服务提供商联合起来形成更大的基础设施，这种框架被称为多云系统[26]。在许多情况下，它们由第三方管理。每个云服务提供商都提供自己的一组虚拟机（VM）和计费机制。

研究动机 正如文献所述，近年来，执行这些工作流所需的资源大幅增加[13]。在这种情况下，一种解决方案是使用多云系统来执行此类超大规模工作流。多云系统由各自的硬件、软件和网络基础设施构成，具有高度异构性 [21]。因此，它容易发生故障。工作流中的任务彼此依赖，一个任务发生故障可能导致整个工作流失效。为了成功执行工作流，云基础设施必须提供高度可靠的计算服务。此外，在多云系统中，来自两种不同云服务的虚拟机可能位于不同地点；连接这两台服务器（各自承载一台虚拟机）的网络链路可能会被攻击者访问。分配给这些虚拟机的两个任务可能存在依赖关系。因此，在将数据从一台虚拟机传输到另一台之前，需要采取某种安全措施。总之，要在多云系统中成功执行工作流，安全性和可靠性都是重要问题。

相关工作 过去几年，人们对多云系统进行了广泛研究 [10, 19, 27]。针对多云系统，已经提出了多种调度策略 [17, 21]。近期，也出现了一些用于多云系统中工作流调度的方法 [1, 18, 23]。通常，在科学工作流中，任务与其前序任务之间具有非常高程度的数据和控制流依赖关系。近年来，有多项研究聚焦于工作流执行的可靠性问题 [9, 14, 28, 31]。Guo 等人 [13] 研究了在满足截止期限约束的条件下，通过一种结合遗传算法和粒子群优化的混合启发式方法进行工作流调度以最小化成本的问题。Tang 等人 [26] 考虑了工作流调度问题，目标是最小化完工时间和成本，并最大化可靠性。一些研究同时考虑了工作流的安全性和可靠性。Wen 等人 [28] 提出了一种算法，用于在联邦云中为工作流寻找可靠、成本更低且满足安全要求的部署方案

算法，用于在联邦云中为工作流寻找可靠、成本更低且满足安全要求的部署方案。Zhu 等人 [33] 研究了受可靠性和安全约束限制的任务调度问题。除此之外，众所周知，工作流也会在多处理器平台上进行调度，同时考虑完工时间、能耗和可靠性要求。 [31] -[29]。据我们所知，关于工作流执行的安全性和可靠性的文献仍然有限。本文重点研究工作流调度中的这两个方面。

本文的贡献 本文的主要贡献如下：

– 我们提出了一种系统模型，用于研究多云系统中的工作流调度，并考虑不同云服务提供商的通信成本和计费机制。– 我们假设数据使用不同强度的密码算法进行加密，并在模型中融入数据机密性的概念。我们设计了一种数据与密码算法的最优分配方案，以最大限度地降低加解密总开销。– 我们还对任务执行进行可靠性分析，并使用泊松分布对任务故障进行建模。– 我们提出了一种基于列表的启发式算法，并结合迭代局部搜索算法，根据完工时间、成本和可靠性为任务分配最佳资源。– 我们将所提出的求解方法应用于真实世界的科学工作流，并将其与最先进的方法进行了比较。

论文结构 本文其余部分安排如下。第2节介绍系统模型和问题形式化。第3节介绍所提出的基于列表的启发式求解方法。第4 节对所提出的求解方法进行实验评估。最后，第5 节总结本文并展望未来研究方向。

2 系统模型与问题形式化

本节介绍系统模型，并对问题进行形式化描述。对于任意正整数 n， $[n]$ 表示集合 $\{1, 2, \ldots, n\}$。

2.1 任务与工作流

任务是需要在多云系统上执行的作业。一个科学工作流 由多个相互关联的任务组成，并建模为一个有向无环图 (DAG) $G(V, E)$，其中 $V(G) = \{v_i : i \in [n]\}$ 是任务集合， $E(G) = \{e_h = (v_i, v_j) : v_i, v_j \in V(G)\}$ 是边集。这里，边 (v_i, v_j) 表示任务 v_i 和 v_j之间的优先关系，即只有在任务 v_i 执行完毕后，才能开始执行任务 v_j

并将其输出传输给 v_j。边 (v_iv_j) 的权重记为 w_{v_i,v_j}，表示需要传输的数据量。任务 v_i 的计算资源需求记为 w_{v_i}。接下来，我们将介绍一些与工作流相关的定义。

定义 1（任务前驱）对于任意任务 v_i，$pred(v_i)$ 表示该任务的直接前驱任务集合，定义为 $pred(v_i) = \{v_j : (v_j, v_i) \in E\}$。

定义 2（任务后继）对于任意任务 v_i，$succ(v_i)$ 表示该任务的直接后继任务集合，定义为 $succ(v_i) = \{v_j : (v_i, v_j) \in E\}$。

定义 3（入口任务）它是一个冗余任务，记为 v_{entry}，它通过权重为零的出边连接到每个 v，使得 $pred(v) = \emptyset$。

定义 4（出口任务） v_{exit} 表示出口任务。它是一个冗余节点，通过权重为零的入边从每个 v 连接到该节点，使得 $succ(v) = \emptyset$。

为简单起见，我们假设 $v_{entry} = v_1$ 和 $v_{exit} = v_n$。

定义 5（拓扑层级）对于任务 v，$top\ level(v)$ 表示其拓扑层级，其定义见公式 1。

$$top_level(v) = \begin{cases} 0 & \text{if } v = v_1 \\ \max\limits_{u \in pred(v)} \{top_level(u) + 1\} & \text{otherwise} \end{cases} \tag{1}$$

可以使用广度优先搜索算法（广度优先搜索）计算拓扑层级 [3]。

2.2 多云系统

我们的模型由 m 个不同的云服务提供商组成，每个提供商都提供自己的资源。这些资源以虚拟机的形式提供（以下简称虚拟机）[24]。假设第 k^{th} 个云服务提供商共提供 m_k 种不同的虚拟机类型。令 $VM(k,p)$ 表示第 k^{th} 个云服务提供商提供的第 p^{th} 种虚拟机。虚拟机由其 CPU、磁盘和内存决定。我们假设虚拟机有足够的内存来执行工作流任务 [22]。令 $w_{k,p}$ 表示 $VM(k,p)$ 的处理能力，该能力与每秒百万条指令（每秒完成的计算量）成正比。虚拟机的处理能力越高，执行任务的速度就越快。在 $VM(k,p)$ 上执行 v_i 所需的时间记为 $T_{exec}[v_i, VM(k,p)]$，其计算方式见公式 2。此外，我们假设新启动的虚拟机需要一定的初始启动时间，记为 $T_{boot}[VM(k,p)]$。

$$T_{exec}[v_i, VM(k,p)] = \frac{w_{v_i}}{w_{k,p}} \tag{2}$$

2.3 网络

两个虚拟机之间的带宽取决于多个因素，例如它们的物理位置、云服务提供商等。为简单起见，我们假设属于同一数据中心的虚拟机通过高速内部网络相连。相比之下，属于不同数据中心的虚拟机则通过速度较慢的外部网络相连[26]。令 B_k 和 $B_{k,k'}$ 分别表示第 k 个云中的虚拟机通信链路带宽，以及连接第 k 个云与第 k' 个云的通信链路带宽。两个任务 v_i 和 v_j 之间的通信时间记为 $T_{comm}[v_i, v_j]$。这取决于待传输的数据量以及任务所在的云。可使用公式 3 计算。

$$T_{comm}[v_i, v_j] = \begin{cases} 0 & \text{if } v_i, v_j \text{ are scheduled on the same VM instance} \\ \frac{w_{v_i,v_j}}{B_k} & \text{else if } k' = k \\ \frac{w_{v_i,v_j}}{B_{k,k'}} & \text{otherwise} \end{cases} \tag{3}$$

2.4 资源模型与时间指标

由于云服务提供商提供了灵活的资源获取方式，客户可以运行给定虚拟机 [22] 的任意数量的实例。因此，实例的最大数量至多为 $\mathcal{N} \cdot n$，其中 $\mathcal{N} = \sum_{k=1}^{m} \sum_{p=1}^{m_k} 1$ 假设每个任务都在每种可能的虚拟机类型上运行。令 $R = \{vm_r : r \in [\mathcal{N} \cdot n]\}$ 表示待租用的虚拟机实例集合（资源池），LST_{vm_r} 和LFT_{vm_r} 分别表示租用 vm_r 的开始和结束时间。对于任务 v_i，当 v_j 和 v_j 之间的所有数据 $\forall v_j \in pred(v_i)$接收完毕后，便开始解密。随后，任务 v_i 在其分配的虚拟机上开始执行，之后对 v_i 和 v_j之间的所有数据进行加密 $\forall v_j \in succ(v_i)$。最后，需要传输数据。虚拟机必须保持开启，直到任务将数据传输给其所有后继节点[22]。因此，如公式4所示，以 PT_{v_i} 表示的总处理时间包括解密、执行、加密和通信时间。

$$PT_{v_i} = \begin{cases} T_{exec}[v_i, VM(k,p)] \\ \quad + \sum_{v_j \in succ(v_i)} (T_{enc}[v_i, v_j] + T_{comm}[v_i, v_j]) & \text{if } v_i = v_1 \\ \sum_{v_j \in pred(v_i)} T_{dec}[v_j, v_i] + T_{exec}[v_i, VM(k,p)] & \text{else if } v_i = v_n \\ \sum_{v_j \in pred(v_i)} T_{dec}[v_j, v_i] + T_{exec}[v_i, VM(k,p)] \\ \quad + \sum_{v_j \in succ(v_i)} (T_{enc}[v_i, v_j] + T_{comm}[v_i, v_j]) & \text{otherwise} \end{cases} \tag{4}$$

其中 v_i 假设在类型为 $VM(k,p)$ 的虚拟机上执行。任务 v_i 的开始时间和完成时间分别记为 ST_{v_i} 和 FT_{v_i}。如公式 5 所示，任务只有在其所有前驱任务完成后才能开始。

$$ST_{v_i} = \begin{cases} 0 & \text{if } v_i = v_1 \\ \max_{v_j \in pred(v_i)} FT_{v_j} & \text{otherwise} \end{cases} \tag{5}$$

完成时间等于开始时间与处理时间之和，如公式 6 所示。

$$FT_{v_i} = ST_{v_i} + PT_{v_i} \tag{6}$$

完工时间是执行工作流所需的总时间，即最后一个任务完成时的时间。

$$makespan = FT_{v_n} \tag{7}$$

2.5 定价机制

在本研究中，我们考虑三种常见的云服务：Microsoft Azure（MA）、Amazon Web Services（AWS）和 Google Cloud Platform（GCP）。
每个云服务提供商在指定的计费周期结束后向客户收费 τ。令$c_{k,\ p}$表示租用 $VM(k, p)$ 一个计费周期的价格。令租用虚拟机 vm_r 的成本表示为 C_{vm_r}。下面讨论不同云服务提供商采用的定价机制 [26]。

– MA 采用细粒度方案，按使用量逐分钟向客户收费，即 $\tau = 1$ 分钟。成本如方程编号 8所示。

$$C_{vm_r} = \lceil \frac{LFT_{vm_r} - LST_{vm_r}}{\tau} \rceil \cdot c_{k,p} \tag{8}$$

– AWS 采用粗粒度定价机制，客户按每小时使用量计费。即 $\tau = 1$ 小时。可通过公式编号获取成本。 8。
– GCP 采用混合定价机制，客户至少按十分钟计费，此后按分钟计费。成本的计算公式见公式编号 9 ，其中 $C_{k,\ p}$ 表示前十分钟的价格， $\tau = 1$ 表示每分钟价格。

$$C_{vm_r} = C_{k,p} + \max(0, \lceil \frac{LFT_{vm_r} - LST_{vm_r} - 10 \cdot \tau}{\tau} \rceil) \cdot c_{k,p} \tag{9}$$

表1 展示了各种虚拟机的价格 [26]。除此之外，每个云服务提供商都有针对数据发送出站的特定定价方案。通常，入站数据不收费 [8], [6], [7]。令$c_{k,k'}$ 表示在第 k个与第 k'个云服务提供商之间发送单位数据的价格。该价格取决于许多因素：虚拟机所在的位置、数据大小等。表2 列出了从官方网站获取的数据传输费率 [8],[6], [7]。在表中，跨中心指由同一云服务提供商管理但位于不同地点的数据中心。价格因地点（美国、欧洲、亚洲等）而异。为简单起见，我们采用所有地点的中位数值。跨云指通过外部互联网在不同云服务提供商之间进行的数据传输。同样，该价格取决于

MA		*AWS*		*GCP*		
VM	每分钟***($)***	***VM***	每小时***($)***	***VM***	十分钟***($)***	每分钟***($)***
B2MS	0.0015	**m1.small**	0.06	**n1-highcpu-2**	0.014	0.0012
B4MS	0.003	**m1.medium**	0.12	**n1-highcpu-4**	0.025	0.0023
B8MS	0.006	**m1.large**	0.24	**n1-highcpu-8**	0.05	0.0047
B16MS	0.012	**m1.xlarge**	0.45	**n1-highcpu-16**	0.1	0.0093

表 1 不同云服务提供商的虚拟机成本

MA		*AWS*		*GCP*	
同一数据中心 - 免费		**同一数据中心- 免费**		**同一数据中心 - 免费**	
跨中心 - $0.08/GB		**跨中心 - $0.02/GB**		**跨中心 - $0.05/GB**	
跨云					
数据大小	每 ***GB*** （***$***）	数据大小	每 ***GB***（美元）	数据大小	每 ***GB***（美元）
最高 100GB	Free	最高 100GB	Free	0-1太字节	0.19
前10 太字节	0.11	前10 太字节	0.09	1-10太字节	0.18
接下来的40太字节	0.075	接下来的40太字节	0.085	10太字节+	0.15
接下来的100太字节	0.07	接下来的100太字节	0.07	-	
接下来的350太字节	0.06	超过150太字节	0.05	-	

表2 不同云服务提供商的数据传输成本

级别	轮数	明文	Vul	时间（$\mu s/block$）
1	4	2^{29}	98	3.08
2	8	2^{61}	67	3.58
3	12	2^{94}	34	4.15
4	16	2^{118}	10	4.63
5	20	2^{128}	0	5.21

表 3 RC6 变体的安全性比较。

地点，因此我们采用中位数值。将数据从 v_i 传输到 v_j 所需的成本记为C_{v_i,v_j} 并由公式编号10给出。

$$C_{v_i,v_j} = w_{v_i,v_j} \cdot c_{k,k'} \tag{10}$$

其中， v_i, v_j 假定分别调度在第 k 个和第 k' 个云上。

如公式11所示，总执行成本是虚拟机上任务执行成本与任务之间数据传输成本之和。

$$cost = \sum_{vm_r \in R} C_{vm_r} + \sum_{i=1}^{n} \sum_{v_j \in succ(v_i)} C_{v_i,v_j} \tag{11}$$

2.6 安全机制

由于任务之间存在数据依赖，因此需要在任务之间传输数据。为确保安全通信，可以采用不同的安全机制来实现机密性目标。类似于 [16], ，我们重点研究使用分组密码实现数据机密性。 安全级别用于衡量密码的强度。安全级别与加密轮数成正比。设计者必须在安全性和时间之间进行权衡：为了确保更高的安全性，必须选择具有更多加密轮数的密码。表3 展示了这种权衡，其中每一行对应一个级别为 L_i的密码。Plaintexts 列表示成功实施密码分析攻击所需的明文数量，Time 列表示加密一个数据块（128 位）所需的时间。Vul 列表示漏洞 V_i ，其定义为暴力搜索所需的最大明文数量 PT_{bf}与使用所选密码分析算法所需的明文数量 $PT_{cc}(L_i)$之比的对数，如公式 12 [4]所示。该表比较了 RC6 的不同变体的参数；RC6 是一种广泛使用的分组密码 [15]。

$$V_i = \log_2 \lfloor \frac{PT_{bf}}{PT_{cc}(L_i)} \rfloor \tag{12}$$

系统脆弱性定义为各数据项脆弱性的加权和，如公式 13 [16] 所示。

$$V_{system} = \sum_{i=1}^{n} \sum_{v_j \in succ(v_i)} W_{v_i,v_j} \cdot V_{v_i,v_j} \tag{13}$$

其中$W_{v_i,\ v_j}$ 和$V_{v_i,\ v_j}$ 分别表示数据在 v_i 和 v_j 之间的权重和脆弱性。最大脆弱性 $V_{\max}$ 通过在公式中将$V_{v_i,\ v_j}$ 设为具有最大脆弱性的密码可得13。

加密时间可以用数据大小和所选安全级别 [30]的线性函数来表示。如果数据在 v_i 和 v_j 之间使用级别为 L_i 的安全性进行加密，则所需的加密时间记为 $T_{enc}[v_i, v_j]$，其表达式见公式14。

$$T_{enc}[v_i, v_j] = \begin{cases} 0 & \text{if } v_i, v_j \text{ are scheduled on the same VM instance} \\ t_i \cdot \frac{w_{v_i,v_j}}{B} \cdot \frac{1}{w_{k,p}} & \text{otherwise} \end{cases} \tag{14}$$

其中 t_i 表示使用级别为 L_i的密码加密一个数据块所需的时间， B 表示数据块大小（128 位），并且 v_i 假设在 $VM(k,\ p)$ 上执行。解密开销与加密开销类似 [16] ，记为 $T_{dec}[v_i, v_j]$，唯一的区别是它在执行 v_j的虚拟机上执行。

2.7 可靠性分析

可靠性定义为工作流无故障执行的概率。故障的发生采用泊松分布 [11]建模。设$\lambda_{k,\ p}$表示 $VM(k,\ p)$ 的分布参数。类似地，设λ_k 和$\lambda_{k',\ k}$ 分别表示第 k个云的通信链路以及第 k'个云与第 k个云之间通信链路的参数。在 v_i 与 v_j 之间成功传输数据的概率记为$\mathcal{R}_{v_i,\ v_j}$ 由方程15给出。

$$\mathcal{R}_{v_i,v_j} = \begin{cases} 1 & \text{if } v_i, v_j \text{ are scheduled on the same VM instance} \\ e^{-\lambda_k \cdot T_{comm}[v_i,v_j]} & \text{if } k' = k \\ e^{-\lambda_{k',k} \cdot T_{comm}[v_i,v_j]} & \text{otherwise} \end{cases} \tag{15}$$

其中，v_i，v_j 分别假设调度在第 k' 个和第 k 个云上。

虚拟机实例 vm_r 成功执行的概率记为 $\mathcal{R}_{vm_r}$，如公式 16 所示。

$$\mathcal{R}_{vm_r} = e^{-\lambda_{k,p} \cdot (LFT_{vm_r} - LST_{vm_r})} \tag{16}$$

当所有数据均已传输且所有任务均成功执行时，工作流才能成功执行。假设各故障相互独立，公式 17给出了工作流的可靠性。

$$reliability = \prod_{i=1}^{n} \prod_{v_j \in pred(v_i)} \mathcal{R}_{v_j,v_i} \cdot \prod_{vm_r \in R} \mathcal{R}_{vm_r} \tag{17}$$

2.8 问题定义

调度 $\mathcal{S} = (R, \mathcal{M}, \mathcal{C}, makespan, cost, reliability)$ 定义为由以下内容组成的元组：资源池（R）、任务到资源的映射（$\mathcal{M}$）以及边到密码的映射（$\mathcal{C}$）。 R 是从每个资源到其租用时间的映射，由形式为（$vm_r, LST_{vm_r}, LFT_{vm_r}$）的元组组成。$\mathcal{M}$ 是由形式为（$v_i, vm_{u_i}, ST_{v_i}, FT_{v_i}$）的元组组成的映射，其中任务 v_i 分配给资源 vm_{u_i}。 $\mathcal{C}$ 是由形式为（v_i, v_j, C_{v_i,v_j}）的元组组成的映射，其中 $v_j \in succ$（v_i）和 $C_{v_i,\ v_j}$ 是用于加密 v_i 与 v_j 之间数据的密码。本文所研究的问题是在满足下述安全约束的条件下，最小化总完工时间和成本，并最大化可靠性。

$$\textbf{Minimize} \begin{cases} makespan \\ cost \end{cases} \quad \textbf{Maximize} \begin{cases} reliability \end{cases}$$

$$\textbf{Subject To} \begin{cases} V_{system} \leq UV_{req} \\ V_{v_i,v_j} \leq UV_{v_i,v_j} \end{cases}$$

其中 UV_{req}和$UV_{v_i,\ v_j}$分别表示系统以及 v_i 和 v_j 之间数据项漏洞的上限。其权衡关系在于，更高的安全性和时间开销会导致更长的完工时间、更高的成本以及更低的可靠性。在介绍我们的算法之前，我们首先给出算法1中将任务到资源的映射转换为调度的伪代码，该算法类似于 [13] 并描述如下。我们将截至目前租用的资源R_{curr} 和任务到资源的映射 $\mathcal{M}$ 初始化为空。如果任务存在前驱任务，则只有在这些任务完成后才能开始，如第 9、10 行所示。对于每个任务，我们使用算法 2 中的 $process\ task()$ 例程，计算与数据传输相关的解密、加密、传输时间、传输成本和可靠性。该例程使用公式14计算加密和解密时间，使用公式3计算传输时间，使用公式 10 计算传输成本，并使用公式 15计算可靠性。处理时间在第 13 行使用公式 4计算。如果资源已经托管，则可知任务只有在当前租约结束时间之后才能开始$LFT_{vm_{u_i}}$ ，如第 15、16 行所示。否则，我们首先启动相应的虚拟机实例，并将租约开始时间 $LST_{vm_{u_i}}$ 设置为第 19 行所示的时间；任务开始时间则为第 18 行所示的启动完成后时间。最后，如第 21 行所示，通过将处理时间加到开始时间上，计算任务的完成时间。在第 22 行，租约结束时间更新为任务的完成时间。处理完所有任务后，完工时间即为最后一个任务的完成时间，如第 24 行所示。第 25 至 30 行计算执行任务所产生的成本和可靠性。

3 所提算法

我们的方法分为两部分：将每项任务分配给一台虚拟机，然后为其分配适当的密码算法。我们首先介绍将密码算法分配给数据的过程，因为该过程适用于任何任务到资源的分配。

3.1 密码算法分配

给定特定的任务到资源的映射，我们希望为数据分配密码算法，在满足下文所述安全约束的前提下，使加解密开销总和最小。

$$\textbf{Minimize} \left\{ \sum_{i=1}^{n} \sum_{v_j \in succ(v_i)} T_{enc}[v_i, v_j] + T_{dec}[v_i, v_j] \right.$$

$$\textbf{Subject To} \begin{cases} V_{system} \leq UV_{req} \\ V_{v_i,v_j} \leq UV_{v_i,v_j} \end{cases}$$

我们采用动态规划（DP）策略来解决此问题，具体方法见算法3，描述如下。DP表项 $dp[h][Vul]$

算法1： 粒子到调度的映射

输入： $G(V,\ E)$、多云系统参数、资源池（R）、安全性密码表，类似于表3（$cipher\ tab$）、数据到密码的映射（$\mathcal{C}$），任务到资源的映射（X）

输出： $\mathcal{S}$

1 $\mathcal{M}, R_{curr} \longleftarrow \emptyset$;
2 $cost \longleftarrow 0$;
3 $reliability \longleftarrow 1$;
4 **对** $v_i \in V$ **执行**
5 $vm_{u_i} \longleftarrow R[X[v_i]]$;
6 $VM(k,p) \longleftarrow type(vm_{u_i})$;
7 $exec_time \longleftarrow \frac{w(v_i)}{w(VM(k,\ p))}$;
8 $ST_{v_i} \longleftarrow 0$;
9 **对于** $v_j \in pred(v_i)$ **执行**
10 $ST_{v_i} \longleftarrow \max(ST_{v_i},\ FT_{v_j})$;
11 $dec_time,\ transfer_time,\ transfer_cost,\ enc_time,\ rel \longleftarrow process_task(v_i)$;
12 $reliability \longleftarrow reliability \cdot rel$;
13 $PT_{v_i} \longleftarrow dec_time + exec_time + transfer_time + enc_time$;
14 $cost \longleftarrow cost + transfer_cost$;
15 **如果** $vm_{u_i} \in R_{curr}$ **then**
16 $ST_{v_i} \longleftarrow$ 最大值$(ST_{v_i},\ LFT_{vm_{u_i}})$;
17 **else**
18 $ST_{v_i} \longleftarrow$ 最大值$(ST_{v_i}, T_{boot}[VM(k,p)])$;
19 $LST_{vm_{u_i}} \longleftarrow ST_{v_i} - T_{boot}[VM(k,p)]$;
20 $R_{curr} \longleftarrow R_{curr} \cup \{vm_{u_i}\}$;
21 $FT_{v_i} \longleftarrow ST_{v_i} + PT_{v_i}$;
22 $LFT_{vm_{u_i}} \longleftarrow FT_{v_i}$;
23 $\mathcal{M} \longleftarrow \mathcal{M} \cup \{(v_i,\ vm_{u_i},\ ST_{v_i},\ FT_{v_i})\}$;
24 $makespan \longleftarrow FT_{v_n}$;
25 </style>适用于 $vm_r \in R_{curr}$ **do**
26 $VM(k,p) \longleftarrow type(vm_r)$;
27 计算 $cost_{vm_r}$ 对应于租赁周期 $LFT_{vm_r} - LST_{vm_r}$ 使用公式编号 8，9;
28 $cost \longleftarrow cost + cost_{vm_r}$;
29 计算 $\mathcal{R}_{vm_r}$ 对应租赁周期 $LFT_{vm_r} - LST_{vm_r}$ 使用公式编号16;
30 $reliability \longleftarrow reliability \cdot \mathcal{R}_{vm_r}$;
31 $\mathcal{S} \longleftarrow (R, \mathcal{M}, \mathcal{C}, makespan, cost, reliability)$;
32 **返回** $\mathcal{S}$;

表示对前 h 条边进行加密和解密所需的最短时间，以及第 $h-$条边对应的密码算法，如安全约束 $V_{system} \leq Vul$在第21行所示。首先，在第4行找出任务 v_i，v_j 所运行的虚拟机。然后，在第8行将所有可能的密码算法分配给第 $h-$条边。如果满足第9行中的约束条件，则在第10行计算加密和解密时间。如果这是第一条边（$h=1$），则按照第13行求出最短加密时间。否则，在第16行根据方程 $V_{remaining}$ 求出前 $h-1$ 条边需要满足的剩余脆弱性，见第13式。总时间将

算法 2: *process task*

输入： $G(V,\ E)$，多云系统参数，资源池（R），安全性与表 3 （*cipher tab*）类似的密码表，数据到密码的映射（$\mathcal{C}$），任务到资源的映射（X），v_i

输出： *dec time*，*transfer time*，*transfer-cost*，*enc_time*, *rel*

1 $vm_{u_i} \longleftarrow R[X[v_i]]$;
2 $VM(k,p) \longleftarrow type(vm_{u_i})$;
3 $dec\text{-}time, transfer\text{-}time, transfer\text{-}cost, enc\text{-}time \longleftarrow 0$;
4 $rel \longleftarrow 1$;
5 **对于** $v_j \in pred(v_i)$ **执行**
6 $vm_{u_j} \longleftarrow R[X[v_j]]$;
7 **如果** $u_i \neq u_j$ **则**
8 从 $\mathcal{C}$中查找用于在 v_j 和 v_i 之间加密数据的 *cipher*；
9 从中查找使用 *cipher* 加密并在 $VM(k,\ p)$上执行的 $T_{dec}[v_j, v_i]$ 方程编号 14;
10 $dec\text{-}time \longleftarrow dec\text{-}time + T_{dec}[v_j, v_i]$;
11 **对于** $v_j \in succ\ (v_i)$ **执行**
12 $vm_{u_j} \longleftarrow R[X[v_j]]$;
13 $VM\ (k',\ p') \longleftarrow type\ (vm_{u_i})$；
14 **如果** $u_i \neq u_j$，**则**
15 找出用于加密 v_i 与v_j 之间数据的 *cipher*，其中 $\mathcal{C}$;
16 找出使用 *cipher* 加密并在 VM（$k,\ p$）上执行的 $T_{enc}[v_i, v_j]$，其中公式编号 14;
17 **如果** $k' \neq k$ **那么**
18 $transfer_time \longleftarrow transfer_time + \frac{w(v_i,\ v_j)}{B_{k,k'}}$;
19 查找定价 $c_{k\ \ ,k'}$ 根据表 2;
20 $transfer_cost \longleftarrow transfer_cost + c_{k,k'} \cdot w(v_i,v_j)$;
21 $rel \longleftarrow rel \cdot e^{\lambda_{k,k'} \cdot \frac{w(v_i,\ v_j)}{B_{k,k'}}}$;
22 **else**
23 $transfer_time \longleftarrow transfer_time + \frac{w(v_i,\ v_j)}{B_k}$;
$rel \longleftarrow rel \cdot e^{\lambda_k \cdot \frac{(v_i, v_j) B_k}{}}$
24 ;
25 $enc_time \longleftarrow enc_time + T_{enc}[v_i, v_j]$;
26 **返回** *dec time*，*transfer time*，*transfer cost*，*enc time*，*rel*;

由加密 $h-1$ 条边的最优时间与加密 $h-$条边的时间相加得到，该时间在第 17 行计算。第 19 行更新总最短时间。总体递推关系如公式 18所示。最后，我们在第 23–27 行开始为每条边分配密码算法。我们按相反的顺序添加密码算法，即先为最后一条边分配密码算法，再为倒数第二条边分配，依此类推。如前所述，动态规划表项 $dp[|E|][UV_{req}]$ 存储总最短时间，以及满足约束 UV_{req}时对应于最后一条边的密码算法。因此，我们从 $Vul = UV_{req}$ 开始，如第 22 行所示。对应的密码算法从第 25 行的动态规划表表项 $dp[h][Vul]$ 中获取，并更新 Vul 。

根据公式 13 ，将其更新为缩减边集 $\{e_1, e_2, \ldots, e_{h-1}\}$ 对应的新约束，见第 27 行。

$$dp[h][Vul][0] = \begin{cases} \min_{cipher \in cipher_tab} T_{enc}[v_i, v_j] + T_{dec}[v_i, v_j] & \text{if h = 0} \\ \min_{cipher \in cipher_tab} T_{enc}[v_i, v_j] + T_{dec}[v_i, v_j] & \\ \quad + dp[h-1][Vul - W_{v_i,v_j} \cdot V_{v_i,v_j}] & \text{otherwise} \end{cases} \tag{18}$$

注意，权重 $W_{v_i,\ v_j}$ 和约束 UV_{req} 可能是小数，因此我们通过乘以 10 的幂将它们转换为整数。

算法3： 密码分配算法

输入： $G(V,\ E)$、多云系统参数、任务到资源的映射（X），安全密码表，与表3 （$cipher\ tab$）相似，W_{v_i,v_j}，UV_{req}, UV_{v_i,v_j}

输出： $\mathcal{C}$

1 初始化动态规划表 $dp[|E|+1][UV_{re\ \ q}+1]$;
2 **对于** $h \longleftarrow 1\ to\ |E|$，**执行**
3 $(v_i,\ v_j) \longleftarrow e_h$;
4 $VM(k,p) \longleftarrow type(R[X[v_i]])\ VM(k',p') \longleftarrow type(R[X[v_j]])$;
5 **对于** $Vul \longleftarrow 0\ to\ UV_{req}$ **do**
6 $best_time \longleftarrow INT_MAX$;
7 $best_cipher \longleftarrow None$;
8 **对于** $cipher \in cipher\ tab$ **执行**
9 **如果** $W_{v_i,\ v_j} \cdot V_{cipher} \leq Vul,\ V_{cipher} \leq UV_{v_i,\ v_j}$ **则**
10 计算与 $cipher$ 对应的 $T_{enc}[v_i, v_j],\ T_{dec}[v_i, v_j]$ 以及假设 $v_i,\ v_j$ 分别在 $VM(k,\ p)\ VM(k',\ p')$ 上执行分别根据方程编号 14;
11 **如果** $h = 1$，**则**
12 **如果** $T_{enc}[v_i, v_j] + T_{dec}[v_i, v_j] < best_time$，**则**
13 $best\text{-}time \longleftarrow T_{enc}[v_i, v_j] + T_{dec}[v_i, v_j]$;
14 $best_cipher \longleftarrow cipher$;
15 **else**
16 $V_{remaining} = Vul - W_{v_i v_j} \cdot V_{cipher}$;
17 $total_time \longleftarrow dp[h-1][V_{remaining}][0] + T_{enc}[v_i, v_j] + T_{dec}[v_i, v_j]$;
18 **如果** $total_time < best_time$ **则**
19 $best_time \longleftarrow total_time$;
20 $best_cipher \longleftarrow cipher$;
21 $dp[h][Vul] \longleftarrow [best_time, best_cipher]$;
22 $Vul \longleftarrow UV_{req}$;
23 **对于** $h \longleftarrow |E|\ to\ 1$ **执行**
24 $(v_i,\ v_j) \longleftarrow e_h$;
25 $C_{v_i v_j} \longleftarrow dp[h][Vul][1]$;
26 将元组 (v_i, v_j, C_{v_i,v_j}) 添加到 $\mathcal{C}$;
27 $Vul \longleftarrow Vul - W_{v_i v_j} \cdot V_{v_i v_j}$;
28 **返回** $\mathcal{C}$;

3.2 虚拟机分配

我们采用高效的基于列表的方法为任务分配资源。基于列表的调度通常分为两个阶段：根据某种级别对任务进行排序，然后按照顺序逐一为任务分配资源。首先，我们按照类似于[22] 如下方式缩小所考虑的资源池：令 $\mathcal{P}$ 表示可以并行执行的最大任务集。然后，通过考虑 R 中每个任务对应的每种类型的一个虚拟机实例，可以将 $R = \{vm_r$ 重新定义为： $r \in [|\mathcal{P}| \cdot n]\}$。寻找集合 $\mathcal{P}$ 的一种方法是找出一个最大的任务集，其中所有任务具有相同的拓扑层级。由于所有任务具有相同的拓扑层级，它们之间不存在依赖关系。以这种方式计算出的集合 $\mathcal{P}$ 不一定是最大的集合。其次，我们在定义6中定义任务级别的概念。

定义 6 $Rank$（v_i）表示任务 v_i 的秩，该值反映了从 v_i 到 v_n 的所有路径上的最坏情况处理时间，其计算公式见式19。

$$Rank_{v_i} = \begin{cases} \max_{v_j \in succ(v_i)}\{Rank(v_j)\} + \overline{T_{exec}[v_i, VM(k,p)]} \\ \quad + \sum_{v_j \in succ(v_i)} \overline{T_{comm}[v_i, v_j]} & \text{if } v_i = v_1 \\ \sum_{v_j \in pred(v_i)} \overline{T_{exec}[v_i, VM(k,p)]} & \text{else if } v_i = v_n \\ \max_{v_j \in succ(v_i)}\{Rank(v_j)\} + \overline{T_{exec}[v_i, VM(k,p)]} \\ \quad + \sum_{v_j \in succ(v_i)} + \overline{T_{comm}[v_i, v_j]} & \text{otherwise} \end{cases} \tag{19}$$

其中，$\overline{T_{exec}[v_i, VM(k,p)]}$ 表示在虚拟机类型上执行 v_i的平均执行时间，$T_{comm}[v_i, v_j] = \frac{w_{(v_i,v_j)}}{B}$，$B$表示平均带宽。称为基于列表的调度器（LBS）的分配算法见算法4，具体描述如下：我们从底向上开始分配任务，因为任务的处理时间主要取决于后继节点，如公式4所示。为此，我们按照拓扑层级递减的顺序对任务进行排序，在第 1、2 行优先处理级别较高的任务。由于具有相同拓扑层级的任务可以同时执行，因此我们为每个任务分配不同的虚拟机实例，并使用集合$vms\ allocated$ 记录截至目前已分配的虚拟机，该集合在第 8 行初始化为空集。我们使用 $process\ task$ 例程生成处理时间所需的组成部分，当 v_i 在第 12、13 行被假定在 vm_r 上执行时，这些组成部分表示为$PT_{v_i、\ vm_r}$。由于我们知道每个后继任务被分配到何处，因此可以计算 $transfer\ time$、$transfer\ cost$、rel。由于尚未确定密码算法分配，我们假设 $enc\ time$、$dec\ time$ 为零。在第 14-17 行计算与处理时间和数据传输相关的成本与可靠性。之后，在第 20 行基于成本、处理时间和可靠性的线性组合计算指标$metric_{v_i、\ vm_r}$，其中每个参数均使用最小-最大归一化进行归一化。权重 α、β和γ 表示各参数之间的相对重要性。在本研究中，我们更加重视成本，并选择权重：$\alpha = 0.7$、$\beta = 0.2$、$\gamma = 0.1$。A

值较小的虚拟机实例$metric_{v_i,\ vm_r}$更为理想。因此，我们按照$metric_{v_i,\ vm_r}$递增的顺序对实例进行排序，并在第 22-27 行为其分配与同一拓扑层级中的先前任务尚未分配的第一个虚拟机实例，以提高可并行性。我们的算法不同于其他算法中使用的基于列表的方法 [26] [29], ，因为它以相反的顺序分配任务，即先分配后继任务，再分配前驱任务。这是因为如前文所述，处理时间主要取决于后继节点。

最后，我们提出一种迭代局部搜索算法（LS），该算法以虚拟机分配为输入，并将虚拟机重新分配给任务，以优化完工时间、成本和可靠性，如算法5所示。首先，按排名对任务排序（第1行）。在固定次数的迭代中，算法首先为每个任务 v_i 重新分配虚拟机实例 vm_r ，并计算相应的 $makespan$($s_{v_i,\ vm_r}$) $cost$($cost_{v_i,\ vm_r}$) $reliability$($\mathcal{R}_{v_i,\ vm_r}$)。与算法4类似，我们使用最小-最大归一化，在第9行计算得到一个包含 $\alpha = 0.6,\ \beta = 0.2,\ \gamma = 0.2$ 的指标。第10行将值最小的虚拟机实例 vm_{u_i} 分配给任务 v_i 。如果分配没有变化，则如第11、12行所示，终止该过程。

整体算法按以下步骤执行：

– 使用广度优先搜索 [3]计算缩减资源池 R 。– 使用算法4获取任务到虚拟机的分配。– （可选）使用算法5获取改进后的任务到虚拟机分配。– 使用算法3计算与所得任务到虚拟机分配相对应的边到密码的映射 $\mathcal{C}$ 。– 使用算法1计算最终调度 $\mathcal{S}$。

3.3 复杂度分析

$process\ task$ 例程的时间复杂度为 $\mathcal{O}(n)$，因为每个任务最多有 n 个后继任务或前驱任务，因此，算法1中处理一个任务所需的时间为 $\mathcal{O}(n)$。处理所有任务所需的总时间为 $\mathcal{O}(n^2)$。最后，在算法1的第 25 行遍历 R_{curr} 最多需要$\mathcal{O}(\mathcal{N} \cdot n)$时间，因为 $|R_{curr}| \leq \mathcal{N} \cdot n$。算法3采用基于动态规划的方法，计算一个条目 $dp[i][j]$ 所需的时间取决于可用于加密的密码算法数量$|cipher\ tab|$ ，如第 7 行所示。因此，计算所有条目所需的总时间为 $\mathcal{O}(|cipher\ tab| \cdot UV_{req} \cdot n^2)$，因为 $|E| \leq n^2$。在算法4中，计算拓扑层级和排名需要 $\mathcal{O}(n^2)$时间。对于每个任务和每个虚拟机实例对，计算处理时间、成本和可靠性需要 $\mathcal{O}(n)$时间，如 $process\ task$ 例程所示。对所有虚拟机实例对的指标进行排序需要 $\mathcal{O}(|R| \cdot log|R|)$时间。因此，处理一个任务所需的时间为 $\mathcal{O}(|R| \cdot (n + \log|R|))$。处理所有任务的时间复杂度为 $\mathcal{O}(n \cdot |R| \cdot (n + \log|R|))$。由于 $|R| \leq \mathcal{N} \cdot n$，时间复杂度至多为 $\mathcal{O}(\mathcal{N} \cdot n^2 \cdot (n + \log\ n + \log|\mathcal{N}|))$。大多数方法

算法4： LBS分配算法

输入： $G(V,\ E)$、多云系统参数、资源池（R）
输出： 任务到资源的映射（X）

按照列表 Q 中任务的排名降序排列；
按照任务的拓扑层级降序排列，若相同则依据 Q中的顺序。将得到的顺序存储在列表 L 中；
$prev_lvl \longleftarrow -1$;
$vms_allocated \longleftarrow \emptyset$;
对于 $v_i \in L$，**执行**
  $curr_lvl \longleftarrow top_level(v_i)$;
  如果 $prev\ lvl \neq curr\ lvl$，**则**
    $vms_allocated \longleftarrow \emptyset$;
  对 $vm_r \in R$ **执行**
    $VM(k,p) \longleftarrow type(vm_r)$;
    $exec_time \longleftarrow \frac{w(v_i)}{w(VM(k,\ p))}$;
    $dec_time,\ transfer_time,\ transfer\ cost,$ $enc_time,\ rel \longleftarrow process\ task(v_i)$;
    $PT_{v_i,\ vm_r} \longleftarrow dec_time + exec_time + transfer_time + enc_time$;
    计算租赁周期 $PT_{v_i,\ vm_r}$ 对应的 $lease\ cost$，使用公式编号8，9；
    $cost_{v_i,\ vm_r} \longleftarrow lease_cost + transfer_cost$;
    使用方程计算对应租赁周期 $PT_{v_i,\ vm_r}$ 的 $lease\ rel$ 编号。16；
    $\mathcal{R}_{v_i,\ vm_r} \longleftarrow rel \cdot lease\ rel$;
  计算 $PT_{v_i,\ 最小值}$ $PT_{v_i,\ 最大值}$ $cost_{v_i,\ 最小值}$ $cost_{v_i,\ 最大值}$，
  $\mathcal{R}_{v_i,\ 最小值}$ $\mathcal{R}_{v_i,\ 最大值}$，遍历 R 中的所有虚拟机实例；
  对于 $vm_r \in R$，**执行**
    $metric_{v_i,\ vm_r} \longleftarrow$
    $\alpha \cdot \frac{cost_{v_i,\ vm_r} - cost_{v_i,\ 最小值}}{cost_{v_i,\ 最大值} - cost_{v_i,\ 最小值}} + \beta \cdot \frac{PT_{v_i,\ vm_r} - PT_{v_i,\ 最小值}}{PT_{v_i,\ 最大值} - PT_{v_i,\ 最小值}} + \gamma \cdot \frac{\mathcal{R}_{v_i,\ 最大值} - \mathcal{R}_{v_i,\ vm_r}}{\mathcal{R}_{v_i,\ 最大值} - \mathcal{R}_{v_i,\ 最小值}}$
  按$metric_{v_i,\ vm_r}$ 的升序对列表 U中的虚拟机实例排序；
  对于 $vm_r \in U$，**执行**
    如果 $vm_r \notin vms\ allocated$，**则**
      $X[v_i] \longleftarrow r$;
      $vms\text{-}allocated \longleftarrow vms\text{-}allocated \cup \{vm_r\}$;
      中断;
  $prev_lvl \longleftarrow curr_lvl$;
返回 X;

采用进化方法进行虚拟机分配 [5], [2], [13], 与算法4相比，这些方法非常耗时。最后，我们来看算法5所需的时间。对于第 5 行中将虚拟机实例重新分配给任务的每次操作，我们都会在第 6 行计算调度，这需要 $\mathcal{O}(n^2)$ 时间。因此，所有迭代、任务和虚拟机实例所需的总时间为$\mathcal{O}(num\ iter \cdot \mathcal{N} \cdot n^4)$。

算法5： LS算法

输入： $G(V,\ E)$，多云系统参数，资源池（R），任务到资源的映射（X）

输出： 新的任务到资源的映射（X'）

按列表 L 中任务的排名降序排列；
对于 $itr \longleftarrow 1\ to\ num\ iter$，**执行**
  对于 $v_i \in L$，**执行**
    对于 $vm_r \in R$，**执行**
      将 vm_r 赋给 v_i，以获得新的调度 X'；
      计算 $s_{v_i,\ vm_r}$，$cost_{v_i,\ vm_r}$，$\mathcal{R}_{v_i,\ vm_r}$，这些值对应于使用 X' 算法1；
    计算 $s_{v_i,\ \text{最小值}}$ $s_{v_i,\ \text{最大值}}$$cost_{v_i,\ \text{最小值}}$ $cost_{v_i,\ \text{最大值}}$
    $\mathcal{R}_{v_i,\ \text{最小值}}$ $\mathcal{R}_{v_i,\ \text{最大值}}$ 所有位于 R 中的虚拟机实例。
    $vm_{u_i} \longleftarrow$ 最小值$_{vm_r \in R}\{\alpha \cdot \frac{cost_{v_i,vm_r} - cost_{v_i,\text{最小值}}}{cost_{v_i,\ \text{最大值}} - cost_{v_i,\ \text{最小值}}} + \beta \cdot \frac{s_{v_i,\ vm_r} - s_{v_i,\ \text{最小值}}}{s_{v_i,\ \text{最大值}} - s_{v_i,\ \text{最小值}}} + \gamma \cdot \frac{\mathcal{R}_{v_i,\ \text{最大值}} - \mathcal{R}_{v_i,\ vm_r}}{\mathcal{R}_{v_i,\ \text{最大值}} - \mathcal{R}_{v_i,\ \text{最小值}}}\}$；
    $X'[v_i] \longleftarrow u_i$；
  如果 $X' = X$ **则**
    中断；
返回 X'；

4 实验评估

本节介绍所提出解决方案的实验评估。首先，我们描述实验设置和示例任务图。

4.1 实验设置、任务图和算法 thms

我们在配备 i712$^{\text{th}}$ 代处理器和 16GB 内存的工作站系统上，使用 Python 3.8.10 实现了所提出的方法。本研究考虑两种在文献中广泛用于比较的真实世界任务图：表观基因组学和 LIGO [26], [25] 工作流。我们考虑的表观基因组学工作流规模为 $n = 24$，100，以及规模为 $n = 30$，100 [25] 的 Cybershake 工作流。每个工作流的结构可从网站以 XML 格式获取 [12]。每种工作流中规模较小的称为 Small（S），规模较大的称为 Large（L）。受篇幅限制，我们仅考虑两种工作流，每种各有两种规模。有关这些工作流的更多信息，请参阅 [**?**]。

4.2 竞争算法

将 LBS 与以下方法进行比较：

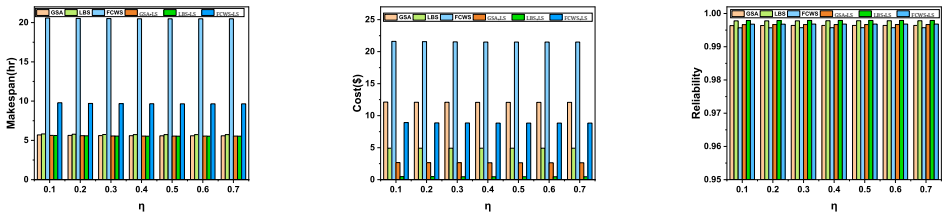
– 引力搜索算法 (GSA) [5]：旨在最小化单云中工作流调度的完工时间和成本。

– 容错成本效益工作流调度 (FCWS) [26]：这是一种基于列表的方法，用于在多云系统上调度工作流，同时考虑完工时间、成本和可靠性。虚拟机根据成本与可靠性的线性组合进行分配。容错依据用于可靠性分析的分布的风险率。在我们的泊松分布案例中，FCWS算法所采用的容错条件不适用，因此我们不予考虑。此外，FCWS在分配虚拟机时未考虑通信成本，因此我们将通信成本计入虚拟机分配成本。

由于 LBS、GSA 和 FCWS 是任务到虚拟机分配算法，我们在其基础上使用算法3来决定用于加密的密码算法。我们不与其他考虑安全约束的工作流调度算法进行比较，例如 [28]，因为它不考虑完工时间。

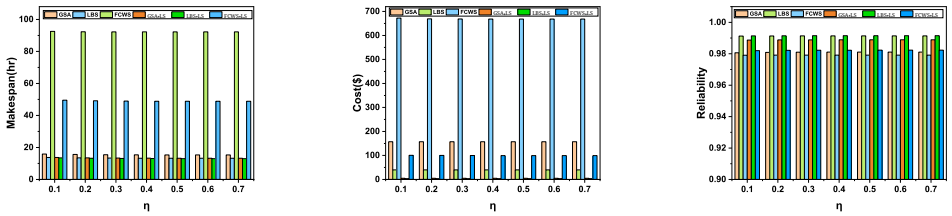
4.3 实验描述

我们在一台配备 $i5$、 10^{th}代处理器和32GB内存的工作站上，使用Python 3.8.10实现所提出的方法。多云系统的参数设置如 [26]所示。虚拟机的计算能力设置为从1到32不等。云内部的平均带宽设置为20Mbps，外部带宽设置为100Mbps。定价机制按照虚拟机计算能力的比例进行设置，如表1 所示；数据传输成本根据表2设置。每台虚拟机的启动时间设置为97秒[22]。所考虑的云服务提供商数量为6个，每种类型各有两个，分别来自MA、AWS和GCP。假设同一类型的两个云服务提供商位于不同的数据中心。加密级别和开销按照表3设置。我们假设$w_{k,\ p} = 1$。安全约束设置如下 [16]： $UV_{req} = \eta \cdot V_{\max}$，其中 $\eta \in [0.1, 0.7]$ 以0.1为步长。$UV_{v_i,\ v_j}$是从表3中随机选择的密码，权重为$W_{v_i,\ v_j} \in [0.1, 1]$。泊松分布的参数在给定范围内均匀随机设置$\lambda \in [10^{-8}, 10^{-7}]$。我们在每个算法之上使用算法5，以观察所获得的改进。每项实验执行15次，并报告平均结果。



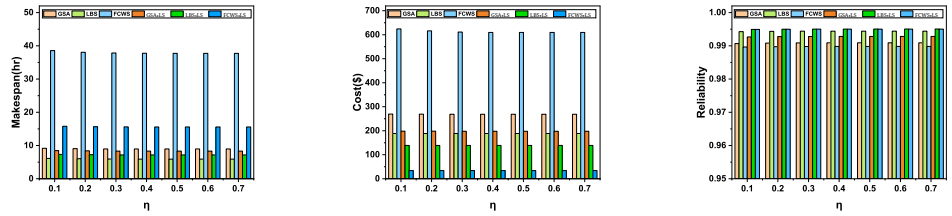
(a) (b) (c)

图 1 带有 $n = 24$ 的表观基因组学图



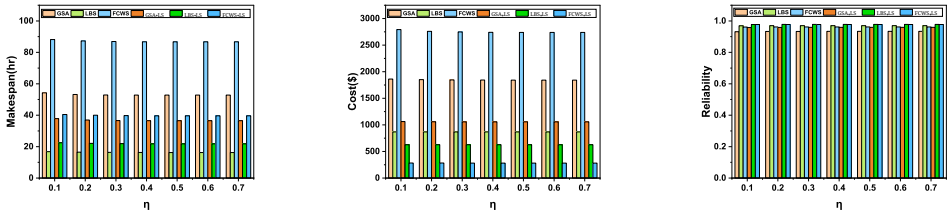
(a) (b) (c)

图2 带有 $n = 100$的表观基因组学图



(a) (b) (c)

图3 带有 $n = 30$的Cybershake图



(a) (b) (c)

图 4 含有 $n = 100$ 的 Cybershake 图

1. 表观基因组学：对于小规模实例，GSA的完工时间最短，LBS的完工时间略长；如图1 (a)所示，两者的表现均明显优于FCWS，优势达71.93%。从图1 (b)可以看出，LBS的成本比GSA低59.37%，随后是FCWS，低77.18%。从图1 (c)可以看出，就可靠性而言，所有算法的表现相近，其中LBS分别比GSA和FCWS略好0.15%和0.2%。应用LS后，GSA和LBS的完工时间略有改善，而FCWS则显著改善了52.74%。成本方面的改善是

在所有算法中都有显著改善：GSA 为 78.09%，LBS 为 90.52%，FCWS 为 58.86%。可靠性的提升幅度不大。
对于大规模问题，LBS 的完工时间最短，比 GSA 快 13.6%，比 FCWS 快 85.5%，如图 2 (a) 所示。从图2 (b) 可见，在成本方面，LBS 的表现显著优于 GSA 和 FCWS，分别降低了 74.66% 和 94.04%。在可靠性方面，GSA 和 FCWS 表现相近，而 LBS 优于二者 1.06%，如图 2 (c) 所示。采用 LS 后，LBS 的完工时间略有改善，而 GSA 和 FCWS 则分别显著改善了 13.47% 和 52.74%。所有算法的成本均显著降低：GSA 降低 96.84%，LBS 降低 92.02%，FCWS 降低 85.09%。

2. *Cybershake*：对于小规模，LBS 的完工时间最短，优于 GSA 33.81%，其次是 FCWS，优于其 84.27%，如图 3(a) 所示。从图 3 (b) 可见，LBS 的成本最低，优于 GSA 29.92%，优于 FCWS 69.24%。从图 3 (c) 可见，就可靠性而言，所有算法的表现相近。应用 LS 后，LBS 的完工时间下降了 20.71%，而 GSA 和 FCWS 则分别显著提升了 7.34% 和 58.78%。所有算法的成本均显著改善：GSA 提升 26.39%，LBS 提升 26.24%，FCWS 提升 94.44%。可靠性仅在 GSA 中有显著改善，提升了 0.53%。

对于大规模，LBS 的完工时间最短，优于 GSA 69.12%，其次是 FCWS，优于其 81.16%，如图 2 (a) 所示。从图 2 (b) 可见，就成本而言，LBS 的表现显著优于 GSA 53.02%，优于 FCWS 68.47%。就可靠性而言，LBS 的可靠性最高，优于 FCWS 0.84%，优于 GSA 3.92%，如图2 (c) 所示。应用 LS 后，LBS 的完工时间下降了 33.83%，而 GSA 和 FCWS 则分别显著提升了 30.72% 和 54.26%。所有算法的成本和可靠性均显著改善：GSA 分别提升 42.69% 和 2.77%，LBS 分别提升 27.73% 和 0.84%，FCWS 分别提升 89.75% 和 1.63%。

总之，随着任务图规模的增大，性能会下降，因为任务节点增多，因此需要更多处理时间。随着 η 的增大，完工时间和成本会降低，而可靠性会提高。这是因为安全约束放宽后，我们可以使用开销较低的密码算法来加密数据，从而减少处理时间。在大多数情况下，就成本、可靠性和完工时间而言，LBS 的表现始终优于其他算法。应用 LS 可提升 GSA 和 FCWS 的完工时间、成本和可靠性。对于 LBS，应用 LS 始终能改善成本和可靠性；在某些情况下，完工时间也会有所改善。

5 结论与未来研究方向

本文研究了多云系统中科学工作流任务的调度问题。调度的目标是

在满足安全约束的前提下，最小化完工时间和成本，并最大化可靠性。针对这一问题，我们开发了一种高效的求解方法，并使用真实科学工作流对其进行了验证。未来，我们希望将容错纳入模型，并探索更高效的求解方法。

参考文献

1. Al-Maytami BA, Fan P, Hussain A, Baker T, Liatsis P（2019）一种基于任务计算时间预测算法、可缩短完工时间的云计算任务调度算法。I EEE Access 7：160916–160926 2. Beegom A, Rajasree M（2015）云计算系统中双目标任务调度的遗传算法框架。载于：分布式计算与互联网技术国际会议，Springer，第356–359页 3. Bundy A, Wallen L（1984）广度优先搜索。载于：《人工智能工具目录》，Springer，第13–13页 4. Chandramouli R, Bapatla S, Subbalakshmi K, Uma R（2006）电池功耗感知加密。ACM信息与系统安全汇刊（TISSEC）9(2)：162–180 5. Choudhary A, Gupta I, Singh V, Jana PK（2018）一种基于GSA的混合算法，用于云计算中的双目标工作流调度。Future Generation Computer Systems 83：14–26 6. AWS数据传输成本（访问日期：2022年9月11日） [在线]。https://aws.amazon.com/ec2/pricing/on-demand/7. GCP数据传输成本（访问日期：2022年9月11日） [在线]。https://cloud.google.com/vpc/network-pricing8. MA数据传输成本（访问日期：2022年9月11日） [在线]。https://azure.microsoft.com/en-in/pricing/details/bandwidth/9. Di S, Robert Y, Vivien F, Cappello F（2016）在两级HPC检查点模型下寻求最优在线检查点方案。IEEE并行与分布式系统汇刊 28(1)：244–259 10. Diaz-Montes J, Diaz-Granados M, Zou M, Tao S, Parashar M（2015）支持软件定义联邦多云中的数据密集型工作流。IEEE云计算汇刊 6(1)：250–263 11. Farid M, Latip R, Hussin M, Hamid NAWA（2020）在多云环境中使用具有模糊资源利用率的多目标算法调度科学工作流。IEEE Access 8：24309–24322 12. Generator W（访问日期：2022年9月14日）[在线]。https://confluence.pegasus.isi.edu/display/pegasus/Deprecated+工作流+生成器13. Guo W, Lin B, Chen G, Chen Y, Liang F（2018）面向多个云中基于截止期限的工作流的成本驱动调度。IEEE网络与服务管理汇刊 15(4)：1571–1585

14. Han L, Canon LC, Casanova H, Robert Y, Vivien F（2018）针对故障停止错误的工作流检查点处理。IEEE计算机汇刊 67(8)：1105–1120 15. Jiang K, Eles P, Peng Z（2011）具有实时约束的分布式嵌入式系统消息加密优化。载于：第14届IEEE电子电路与系统设计及诊断国际研讨会，IEEE，第243–248页 16. Jiang W, Pop P, Jiang K（2017）面向安全与保障关键型分布式实时应用的设计优化。Microprocessors and Microsystems 52：401–415 17. Kang S, Veeravalli B, Aung KMM（2018）一种动态调度策略，通过高效预测节点可用性来处理多云系统中的可分割负载。Journal of Parallel and Distributed Computing 113：1–16 18. Li R, Wu CQ, Hou A, Wang Y, Gao T, Xu M（2018）预算约束下多云环境中高吞吐量科学工作流的调度。载于：2018年IEEE并行与分布式处理及应用、普适计算与通信、大数据与云计算、社会计算与网络、可持续计算与通信国际会议（ISPA/IUCC/BDCloud/SocialCom/SustainCom），IEEE，第1087–1094页 19. Li X, Ma H, Yao W, Gui X（2015）面向大规模多云协作服务的数据驱动与反馈增强型信任计算模式。IEEE服务计算汇刊 11(4)：671–684 20. Mastelic T, Oleksiak A, Claussen H, Brandic I, Pierson JM, Vasilakos AV（2014）云计算：能源效率综述。ACM计算调查（CSUR）47(2)：1–36 21. Miraftabzadeh SA, Rad P, Jamshidi M（2016）一种用于在多云上调度负载感知作业的高效分布式算法。载于：2016年第11届系统之系统工程会议（SoSE），IEEE，第1–8页 22. Rodriguez MA, Buyya R（2014）面向云上科学工作流的基于截止期限的资源配置与调度算法。IEEE云计算汇刊 2(2)：222–235 23. Sooezi N, Abrishami S, Lotfian M（2015）多云环境中的数据驱动型工作流调度。载于：2015年IEEE第7届云计算技术与科学国际会议（CloudCom），IEEE，第163–167页 24. Stillwell ML, Vivien F, Casanova H（2012）异构分布式平台上用于服务托管的虚拟机资源分配。载于：IPDPS 2012，IEEE，x-pays= 美国 25. Szabo C, Kroeger T（2012）在公有云上分配科学工作流任务的多目标策略演化。载于：2012年IEEE进化计算大会，IEEE，第1–8页 26. Tang X（2021）多云系统中具有可靠性感知的成本高效科学工作流调度策略。IEEE云计算汇刊

27. Tekawade A, Banerjee S (2023) 面向多云系统任务图的成本效益型可靠性感知调度器。载于：第15届通信系统与网络国际会议（COMSNETS 2023），印度班加罗尔，2023年1月3日至8日，IEEE，第295–303页，DOI 10.1109/COMSNETS56262.2023.10041331，网址：https://doi.org/10.1109/COMSNETS56262.2023.1004133128. WenZ, Cala J, Watson P, Romanovsky A (2016) 联邦云上的经济、可靠且安全的工作流部署。IEEE服务计算汇刊 10(6):929–941 29. Xie G, Chen Y, Xiao X, Xu C, Li R, Li K (2017) 异构分布式嵌入式系统中可靠并行应用的节能容错调度。IEEE Transactions on Sustainable Computing 3(3):167–181 30. Xie T, Qin X (2007) 通过调度提高嵌入式系统中周期性任务的安全性。ACM嵌入式计算系统汇刊（TECS）6(3):20–es 31. Zhou A, Wang S, Cheng B, Zheng Z, Yang F, Chang RN, Lyu MR, Buyya R (2016) 通过优化虚拟机放置提升云服务可靠性。IEEE服务计算汇刊 10(6):902–913 32. Zhou X, Zhang G, Sun J, Zhou J, Wei T, Hu S (2019) 使用基于异构最早完成时间优先的模糊支配排序，最小化云中工作流调度的成本和完工时间。Future Generation Computer Systems 93:278–289 33. Zhu QH, Tang H, Huang JJ, Hou Y (2021) 满足安全性和可靠性约束的多云计算任务调度。IEEE/CAA自动化学报 8(4):848–865