支持 SaaS 应用多维异构性能需求的云资源放置方法

郭 伟[1)] 张凯强[1)] 崔立真[1)] 徐 猛[1),2)]

[1)](山东大学软件学院 济南 250101)

[2)](山东省软件工程重点实验室 济南 250101)

摘 要 SaaS(Software as a Service)应用是以云计算资源为基础,以按需定制及按需付费的服务模式向用户提供云计算软件服务的应用系统.云中的 SaaS 应用一般为多层多节点部署的大型软件应用系统,对于云计算 SaaS 服务提供商来说,往往需要在云数据中心中同时快速交付和部署多个不同的 SaaS 应用,需要满足不同租户对于不同的 SaaS 应用多样化性能、网络、存储和操作系统需求,即多维异构的性能环境需求.因此,如何快速选择合适的云资源来部署大规模 SaaS 应用系统,满足大规模不同租户的多维异构性能需求,同时节省云服务提供商的成本,是实现 SaaS 应用敏捷交付部署的关键.传统的按照等级和供需的云资源匹配方法已经很难满足云数据中心大规模 SaaS 应用敏捷化交付部署要求.为此,提出一种基于图匹配的 SaaS 应用云资源放置方法,将大规模 SaaS 应用的个性化云服务放置问题映射为云资源节点拓扑图的子图查询匹配问题,即 SaaS 应用的多节点多维性能需求和云资源节点拓扑均表示为带多维属性标签的异构图,基于偏序异构图查询匹配方法得到一组满足用户需求的云资源节点集合,用于放置 SaaS 应用及其数据,从而实现大规模 SaaS 应用的敏捷化交付部署.实验结果表明该方法能有效提高大规模复杂 SaaS 应用多维异构云资源放置的执行效率.

关键词 SaaS 应用;云资源放置;多维属性;图匹配;偏序关系;云计算

中图法分类号 TP391 **DOI 号** 10.11897/SP.J.1016.2018.01225

A Cloud Resources Placement Method Supporting SaaS Applications with Multi-Dimensional and Heterogeneous Requirements

GUO Wei[1)] ZHANG Kai-Qiang[1)] CUI Li-Zhen[1)] XU Meng[1),2)]

[1)](*Software College, Shandong University, Jinan* 250101)

[2)](*Shandong Provincial Key Laboratory of Software Engineering, Jinan* 250101)

Abstract Software-as-a-Service (SaaS) is a new software delivery model that provides on-demand customization and payment for tenants based cloud platform. SaaS has drawn considerable research attention for its capability of transferring software goods for services in light of the Internet based platforms. Individual software vendors release resources and services to tenants by deploying the SaaS applications accordingly. SaaS applications in cloud are generally large scale multi-layer software application systems, which deployed in multiple nodes and are very important to tenants. On the other hand, SaaS service providers wish to reduce the total cost and gain more benefit. To this end, SaaS service providers require to quickly deliver and place multiple diversified SaaS applications in the cloud data center to meet the multi-dimensional and heterogeneous requirements, such as performance, network utilization, storage capability and operating system requirements. As a result,

收稿日期:2017-06-12;在线出版日期:2017-11-13.本课题得到国家自然科学基金(61572295)、国家重点研发计划项目(2016YFB1000602,2017YFB1400102)资助.**郭 伟**,男,1978 年生,博士,工程师,中国计算机学会(CCF)会员,主要研究方向为云计算与大数据管理、智能数据分析.E-mail:guowei@sdu.edu.cn.**张凯强**,男,1995 年生,学士,主要研究方向为云资源放置.**崔立真**,男,1976 年生,博士,教授,博士生导师,中国计算机学会(CCF)会员,主要研究领域为大数据管理与大数据分析、大数据人工智能、服务计算与协同计算、云计算架构技术.**徐 猛**,男,1978 年生,博士,讲师,中国计算机学会(CCF)会员,主要研究方向为云计算与大数据管理、业务流程管理.

it is essential to achieve the SaaS application agile placement to quickly select appropriate cloud resources to place large scale SaaS application system to meet the needs of large scale heterogeneous performance requirements of different tenants and hence reduce the cost of cloud service providers at the same time. Therefore, the key point of placing SaaS applications in an efficient way lies on choosing the appropriate cloud resources to deploy the SaaS applications. Traditional cloud resources matching methods based on Service Level Agreement SLA and supply demand have been difficult to adapt to the requirements of agile placement of large scale SaaS applications in cloud for the complicated requirements of large-scale heterogeneous performance. In this paper, the strategy of placing SaaS applications based on graph matching theory is proposed which models the SaaS applications placement problem as the subgraph matching problem of the cloud resource graph. The computing resources, data resources and their inherent connections in the cloud are mapped into an isomerism graph with multi-dimensional attribute labels. In this graph, vertices represent the cloud resources required by SaaS applications, while edges represent topological relationships between cloud resources. The attribute values on vertices represent the performance requirements of SaaS applications for cloud resources. After constructing the above graph, an algorithm which contains four steps is presented. The first step is to mine the frequent subgraphs which satisfy a certain threshold in the cloud services resource graph. In the next step, the request graph is cut using frequent subgraphs. Then the subgraph set of the request graph is obtained. The third step is to filter the subgraph set of the request graph. In this process, the algorithm dexterously uses the coordinate division method to improve the efficiency of the filtering operation. In the final step, the candidate set is merged to reduce the complexity of the problem and solve the SaaS applications placement problem quickly and accurately. A set of cloud resources that meet the provider's requirements can be eventually obtained by using the partial order relation isomerism matching method, which is employed to agilely place SaaS applications and data. The extensive experimental results suggest that the proposed method illustrates significant improvement in the efficiency of multi-dimensional heterogeneous cloud resource placement strategy in complex SaaS applications.

Keywords SaaS; cloud resources placement; multi-dimensional attributes; graph matching; partial order; cloud computing

1 引 言

SaaS(Software as a Service,软件即服务)[1-2]是一种全新的软件应用模式,是以云计算资源为基础设施,以按需定制及按需付费的服务模式向用户和租户提供云软件服务的应用系统.该模式是通过将软件变为服务来降低客户的使用及维护成本,其根本方法是将软件从部署在客户服务器变为部署在云数据中心中.因为按需定制、按需付费的优势,越来越多的传统应用开始向云数据中心迁移.SaaS 应用一般为多层多节点部署的大型软件应用系统,对于提供 SaaS 应用的独立软件供应商(Independent Software Vendors,ISV)来说,往往需要在云中同时快速交付和部署多个不同的 SaaS 应用[3-4],不同的 SaaS 应用以及 SaaS 应用的不同部署节点对于云资源的性能、网络、存储和操作系统等环境需求都不尽相同,呈现多维性能需求的特性.如何快速选择合适的云资源,满足大规模 SaaS 应用系统的部署请求,满足其多维异构的性能需求.即 SaaS 应用的云资源放置问题[5],节省云服务提供商的成本,是实现 SaaS 应用敏捷交付部署的关键.

云资源节点的海量异构性和大规模 SaaS 应用的多节点多维性能需求的复杂性增加了实现 SaaS 应用敏捷交付部署的难度,主要表现在:(1)异构性.ISV 指定云资源拓扑关系上的节点属性是不一样的,可能分别作为计算节点、存储节点或分发节点等提供相应的云资源;(2)多维性.ISV 对于部署

SaaS 应用的云资源集合中的不同云资源都有计算能力、存储能力、维护成本等不同需求；(3) 偏序性. ISV 要求云资源的计算能力、存储能力、维护成本等指标满足不同的偏序关系(即要求云资源集合中的云资源能够满足 ISV 的性能需求)；(4) 动态性. 云平台上的云资源是动态变化的，这就要求 SaaS 应用放置算法应准确高效，时间复杂度低；(5) 并发性. 云中的 SaaS 应用一般为多层多节点部署的大型软件应用系统，对于云计算 SaaS 服务提供商来说，往往需要在云中同时快速交付和部署多个不同的 SaaS 应用. 同时，海量的部署应用的节点之间有依赖关系. 而云资源拓扑关系匹配时使用的子图匹配是一个 NP 完全问题[6-7]，最坏情况下的时间复杂度随图的规模呈指数级增长.

传统的按照云资源供给与应用服务资源需求等级匹配的云服务放置方法[8-9]已经很难满足云计算时代大规模 SaaS 应用敏捷化交付部署的要求. 因此，如何在云平台大量的云资源中选择合适的资源集合来部署 SaaS 应用，达到既满足 ISV 对云资源性能的需求，同时节省云资源、降低云资源提供商的成本，成为当前 SaaS 应用放置问题中的一大挑战.

分析 SaaS 应用服务的云资源放置问题不难发现，云数据中心的资源节点拓扑表现为一个复杂的异构网络图，同时对于不同租户提出的 SaaS 服务部署需求也可以表示为一个带有多维性能需求属性的异构网络图. 为此，本文提出一种基于图匹配的 SaaS 应用云资源放置方法，将大规模 SaaS 应用的云资源放置问题映射为云资源节点拓扑图的子图查询匹配问题[10]，即 SaaS 应用的多节点多维性能需求和云资源节点均表示为带多维属性标签的异构图，基于偏序关系异构图查询匹配方法得到一组满足用户需求的云资源节点集合，包括计算资源节点和存储节点，用以放置 SaaS 应用及其数据. 利用频繁子图挖掘方法[11-12]得到资源图的频繁子图集合，将请求图利用频繁子图集合拆分，经过图匹配算法和合并算法，得到满足用户 SaaS 应用性能需求的云资源子图，即实现大规模 SaaS 应用的敏捷化交付部署. 算法降低了图匹配问题时间复杂度，在处理 SaaS 应用的云资源放置问题时具有较高的效率.

本文第 2 节介绍 SaaS 应用云资源放置问题的相关研究工作，并与本文工作进行比较；第 3 节给出 SaaS 应用云资源放置的问题描述及相关定义；第 4 节详细讨论支持多维异构属性标签的图匹配方法；第 5 节给出实验结果和分析；最后一节对本文进行总结并给出未来的研究计划.

2　相关研究

针对 SaaS 应用的云资源放置问题的研究，主要方法可以分为两类：一类是将软件服务在云中的部署放置问题建模为资源优化问题；另一类方法是考虑到云中资源节点的拓扑特征和软件服务部署的拓扑需求，将 SaaS 应用服务的部署放置问题映射为图匹配问题进行求解.

在将 SaaS 应用服务在云中的部署放置问题建模成资源优化问题方面，文献[13-18]等结合遗传算法、模拟退火算法等来解决部署放置中的多目标优化问题，实现降低 SaaS 应用的运营成本，并通过对 SaaS 中服务和数据的综合考虑，防止算法陷入局部最优. Gullhav 等人[19]提出启发式算法 ALNS (Adaptive Large Neighborhood Search)，使用自适应的大规模邻域搜索算法求解. Su 等人[20]提出一个多租户排队网络模型，并基于该模型提出了一种平衡的 SLA 感知的租户部署算法. Bhardwaj 等人[21]提出一个数学模型，使用粒子群优化来尽量减少云中 SaaS 服务放置的总成本.

在应用图匹配算法求解 SaaS 应用服务放置问题方面，Zong 等人[22]提出一种子图匹配算法 Gradin (Graph index for dynamic graphs with numerical labels)，该方法利用 FracFilter 的网格策略，实现了对具有单个标签的节点的筛选，但对于多个数值标签的拓展支持存在时间复杂度过高的问题.

这些研究均没有考虑 SaaS 应用的云资源放置问题呈现出的资源需求异构性、性能需求属性多维化的特点. 本文将 SaaS 应用的云资源放置问题映射为云资源图和 SaaS 应用资源需求请求图的图匹配问题，利用频繁子图挖掘算法、图匹配算法、图合并算法等，得到 SaaS 应用的部署方案. 因此我们对于该类技术研究工作进行了大量的调研.

在图匹配算法方面，Fan 等人[6]提出了一种增量的图模式匹配算法，解决了云平台上节点的拓扑结构和数值频繁更新的问题. 但该算法没有很好地解决应用的初始放置的问题，且应用的放置策略无法同时满足多种偏序关系.

对于本文算法的核心部分，频繁子图挖掘算法，

国内外学者也有很多研究成果. Yan 等人[23]研究了图形数据集中频繁图形模式挖掘的方法,并提出了一种称为 gSpan(基于图形的子结构模式挖掘)的新算法来挖掘频繁子结构,并且不需要生成候选集. 方法同时在图形之间建立新的词典顺序,并将每个图形映射到唯一的最小深度优先搜索代码上. 基于这个词典顺序,采用深度优先搜索的方法来快速挖掘频繁子图.

上述频繁子图挖掘算法[23],未考虑顶点类型异构及顶点上的标签属性的偏序问题. 然而,数据中心网络通常具有相当稳定的结构,但是节点性能属性却是复杂多变的. 虽然有一些增量图索引算法可用[24],但并不能适应频繁的标签属性更新. 其次,现有支持近似的图匹配技术几乎都不能在 SaaS 服务放置时处理部分有序的数值标签.

3 问题定义与建模

SaaS 应用一般由多个服务构成,且每一服务对于硬件资源的需求不尽相同. 另一方面,构成 SaaS 应用的服务之间具有密切联系,在 SaaS 应用运行过程中需要进行数据交换,因此对各个服务所部署的节点之间的带宽也就提出了不同的需求.

云平台由若干节点组成,且节点之间均能建立网络连接,但是节点之间的带宽则有较大差别. 若两节点之间的带宽过低,则无法满足云服务对于带宽的需求,因此,对于带宽无法满足需求的,可以认为节点之间是没有连接的.

部署 SaaS 应用需要在云平台中找到符合 SaaS 应用性能需求的一组云资源,并且这些云资源之间满足特定的拓扑关系. 因此,可以将 SaaS 应用对于云资源的需求建模为图,图中的顶点为 SaaS 应用需要的云资源,图中的边为云资源之间的拓扑关系,SaaS 应用对于云资源的性能需求建模为图中顶点的属性. 同时,云平台中所有云资源及其之间的拓扑关系也可以建模为图,其中点为云资源,边为云资源之间的拓扑关系,点的属性为云资源的性能. 由此,SaaS 应用的云服务资源放置问题就转化为在云平台建模后的图中找到子图与 SaaS 应用对云资源的需求建模后的图相匹配的问题,即给定资源图 Res 和请求图 Req,求 Res 的子图集合,其中每个子图在结构上与 Req 图同构,并且其点的属性值与 Req 满足偏序关系,即满足 SaaS 应用的性能需求.

由于云平台中各资源都能够互通,且网络具有双向通信的特点,因此,本文主要考虑连通的无向标签图,为简单起见,顶点上保留两个经过标准化[4,25]的标签,分别表示该云资源的计算能力、存储能力. 通过简单修改,本文的算法也适用于点上有更多标签的图,即考虑云平台中云资源的更多性能指标,如网络拥塞[26]、传输成本[27]等.

定义 1(云服务资源图). 将云平台上所有云资源及它们之间的拓扑关系,定义为一个三元组 $Res=(V,U,E)$. 其中,V 是云平台中所有计算资源的集合,U 是云平台中所有存储资源的集合. $V=\{v_0,v_1,v_2,\cdots,v_m\}$,其中 $v_i=(M_i,C_i)$,M_i 代表该计算资源的内存大小,C_i 代表该计算资源的计算能力. $U=\{u_0,u_1,u_2,\cdots,u_n\}$,其中 $u_j=(S_j,Q_j)$,S_j 代表该存储资源的存储大小,Q_j 代表该存储资源的存取速度. E 是边集合,$E=\{e_0,e_1,e_2,e_3,\cdots,e_l\}$,其中,$e_q=(W_X,W_Y)$,$W_X$ 和 W_Y 代表边 e_q 连接的两个云资源,W_X 和 W_Y 可以为 $V\bigcup U$ 中的元素.

以下将云服务资源图简称为资源图.

一个资源图的实例如图 1 所示.

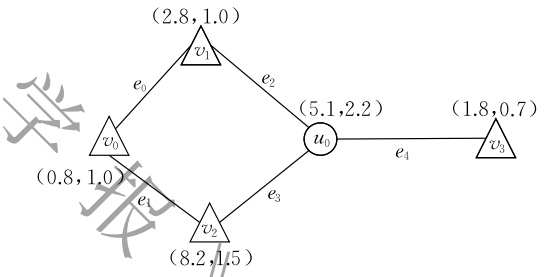

图 1 资源图实例

图 1 中展示的资源图,按照定义 1,该图的表示方式为

$$Res=(V,U,E),\ V=\{v_0,v_1,v_2,v_3\},$$
$$U=\{u_0\},\ E=\{e_0,e_1,e_2,e_3,e_4\},$$
$$v_0=(0.8,1.0),\ v_1=(2.8,1.0),$$
$$v_2=(8.2,1.5),\ v_3=(1.8,0.7),$$
$$u_0=(5.1,2.2),\ e_0=(v_0,v_1),$$
$$e_1=(v_0,v_2),\ e_2=(v_1,u_0),$$
$$e_3=(v_2,u_0),\ e_4=(u_0,v_3).$$

定义 2(SaaS 应用服务资源需求请求图). 将 SaaS 应用对云资源的需求及它们之间的拓扑关系,定义为一个三元组 $Req=(V',U',E')$. 其中,V' 是 SaaS 应用需要的计算资源的集合,U' 是 SaaS 应用需要的存储资源的集合. $V'=\{v'_0,v'_1,v'_2,\cdots,v'_m\}$,其中 $v'_i=(M'_i,C'_i)$,M'_i 代表所需要的计算资源的内存

大小，C'_i代表所需要的计算资源的计算能力. $U'=\{u'_0,u'_1,u'_2,\cdots,u'_n\}$，其中 $u'_j=(S'_j,Q'_j)$，S'_j 代表所需要的存储资源的存储大小，Q'_j代表所需要的存储资源的存取速度. E'是边集合，$E'=\{e'_0,e'_1,e'_2,e'_3,\cdots,e'_l\}$.

以下将 SaaS 应用服务资源需求请求图简称为请求图.

一个请求图的实例如图 2 所示.

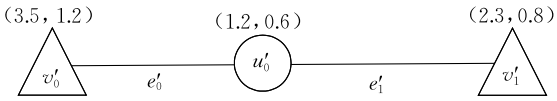

图 2　请求图实例

图 2 中展示的请求图，按照定义 2，该图的表示方式为

$Req=(V',U',E')$，$V'=\{v'_0,v'_1\}$，$U'=\{u'_0\}$，$E'=\{e'_0,e'_1\}$，$v'_0=(3.5,1.2)$，$v'_1=(2.3,0.8)$，$u'_0=(1.2,0.6)$，$e'_0=(v'_0,u'_0)$，$e'_1=(u'_0,v'_1)$.

定义 3(图同构).　图同构是一个双射函数 f：$VU(G)\leftrightarrow V'U'(G')$，$VU(G)$和 $V'U'(G')$代表 G 和 G' 上的点的集合. 对于图 $G=(V,U,E)$与图 $G'=(V',U',E')$，若 G 与 G' 是图同构的，则满足如下条件：

(1) $\forall v\in V$，$\forall u\in U$，存在 $f(v)\in V'$，$f(u)\in U'$；

(2) $\forall v\in V$，$\forall u\in U$，存在下述等价关系 $(v,u)\in E\Leftrightarrow(f(u),f(v))\in E'$.

如图 1、图 2 所示，图 2 与图 1 中的 3 个子图同构，分别为

$$G_1=(\{v_1,v_2\},\{u_0\},\{e_2,e_3\}),$$
$$G_2=(\{v_1,v_3\},\{u_0\},\{e_2,e_4\}),$$
$$G_3=(\{v_2,v_3\},\{u_0\},\{e_3,e_4\}).$$

定义 4(偏序关系).　给定资源图 $Res=(V,U,E)$和请求图 $Req=(V',U',E')$，若满足以下条件：

(1) Req 与 Res 中的子图是同构的；

(2) $\forall v'\in V'$，$\exists v\in V$，$M_v\geq M'_v$且 $C_v\geq C'_v$；

(3) $\forall u'\in U'$，$\exists u\in U$，$S_u\geq S'_u$且 $Q_u\geq Q'_u$.

则说明 Res 与 Req 存在偏序关系，记作 $Res\geq Req$.

如图 1、图 2 所示，与图 2 满足偏序关系的图 1 的子图为 $G=(\{v_1,v_2\},\{u_0\},\{e_2,e_3\})$.

定义 5(频繁子图).　对于给定的资源图 Res，其所有子图的集合为 GD，且给定最小频度阈值 $N_{\min}$，当且仅当某一图 G 与集合 GD 中至少 $N_{\min}$ 个元素同构，则称图 G 为资源图 Res 的频繁子图.

根据定义 1～5，云服务资源放置问题即要在资源图 Res 中找到频繁子图 G，使该子图 G 与请求图 Req 图同构，且 $G\geq Req$.

对于图 1 和图 2 所示的资源图和请求图实例，需要在图 1 中找到与图 2 匹配的子图，如图 3 中所选中的部分. 即子图 $G=(\{v_1,v_2\},\{u_0\},\{e_2,e_3\})$与图 2 的请求图实例匹配.

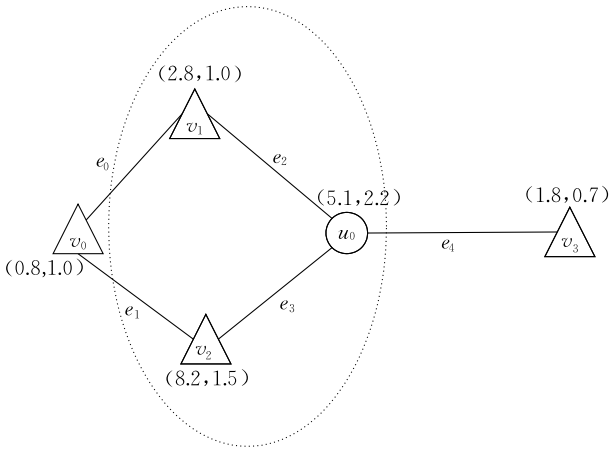

图 3　与请求图匹配的子图

4　云资源放置算法

本节详细介绍云平台上的服务器资源匹配算法. 算法的核心思想是将云平台中的所有云资源建模为资源图，并将 SaaS 应用对于云资源的需求建模为请求图，然后利用子图匹配算法搜索与请求图同构的资源图的子图. 具体步骤如下：首先，将云平台的服务器资源映射成顶点上带有数值标签的资源图 Res，然后挖掘资源图 Res 中的频繁子图集合 S_{frequent}，遍历资源图 Res，找到与 S_{frequent}中的任一频繁子图同构的 Res 的子图. 将这些 Res 的子图与同构的频繁子图对应起来并建立索引. 完成以上对资源图的预操作之后，再对客户的请求图 Req 进行操作. 利用频繁子图集合 S_{frequent} 对请求图 Req 进行切割操作，使得 Req 被拆分为一个个的频繁子图. 然后对照之前为资源图建立的索引，取出与这些子图同构的集合. 在得到图同构的集合以后，分层次对点上的数值标签进行比较，留下满足需求的那些子图，作为候选集. 最后，将这些候选集按照有无公共边缘点的关系进行合并，最终得到的结果就是满足客户需求的一组服务器的集合.

本节把算法部分的详细实现分为四步，分别是资源图的预处理、请求图的切割、候选集的筛选以及候选集的合并，具体的处理过程如算法 1 所示.

算法 1. 图匹配过程(*Graph_Matching*).

输入:资源图 *Res*,请求图 *Req*

输出:候选集 *Candidate*

IF *Res* 有变化

$S_{\text{frequent}}=Graph\_Mining(Res,N_{\min})$;

END IF

$S_{\text{request}}=Graph\_Cut(Req,S_{\text{frequent}})$;

$U'_s=Graph\_Filter(U_s,\alpha,S_{\text{request}})$;

$Candidate=Subgraph\_Join(Req,S_{\text{request}},U'_s)$;

RETURN *Candidate*;

4.1 资源图的预处理

设 *Res* 是一个资源图,为得到 *Res* 上的频繁子图集合 S_{frequent},对其进行频繁子图挖掘.其中需要设置一个最小出现次数阈值 $N_{\min}$,表示频繁子图出现次数的下限.首先进行数据导入处理,然后创建与存储相关的数据结构,最后采用递归调用,进行深度优先挖掘.该算法主要实现如下.

算法 2. 图挖掘过程(*Graph_Mining*).

输入:资源图 *Res*,最小出现次数阈值 $N_{\min}$

输出:频繁子图集合 S_{frequent},*Res* 中与频繁子图同构的集合 U_s

从 *Res* 中读图数据;

FOR EACH 边 $edge\in Res$

用边 *edge* 初始化搜索结构 *structure*;

$S_{\text{frequent}}=Subgraph\_Enumerate(Res,structure,N_{\min})$;

$Res\leftarrow Res-edge$;

END FOR

FOR EACH $structure\in S_{\text{frequent}}$

判断该 *structure* 的对称度 *symmetry*;

FOR(int $n=0$; $n<symmetry$; $n++$)

按照唯一顺序遍历 *Res*;

IF *Res* 中存在与 *structure* 图同构子图

将该子图加入 *structure* 的集合 U_s;

END IF

END FOR

END FOR

算法 2 的时间复杂度是 $O(|S_{\text{request}}|\times symmetry+e^3)$,此处,$e$ 是资源图 *Res* 中边的个数.

在算法 2 中,存在一些子图,从不同点开始遍历都可以得到图同构的子图,而点的位置有所不同,将这种可能事件出现的个数记为对称度.例如结构 $v_0\leftrightarrow v_1$ 中,$v_0\rightarrow v_1$ 与 $v_1\rightarrow v_0$ 是图同构的,因此该结构的对称度为 2.

在对资源图 *Res* 的挖掘过程中,一个重要的步骤是对于 *Res* 中个每条边枚举满足阈值 $N_{\min}$ 的频繁子图.这个过程如算法 3 所示.

算法 3. 子图枚举过程(*Subgraph_Enumerate*).

输入:资源图 *Res*,搜索结构 *structure*,最小出现次数阈值 $N_{\min}$

输出:更新后的频繁子图集合 S_{frequent}

IF S_{frequent} 中没有与 *structure* 图同构的子图

在 S_{frequent} 中加入 *structure*;

END IF

添加一条边,生成集合 *structure* 的生成图;

FOR EACH *child*,*child* 是 *structure* 的生成图

IF *child* 的在 *Res* 中出现的次数 $\geq N_{\min}$

$structure\leftarrow child$;

$S_{\text{frequent}}=Subgraph\_Enumerate(Res,structure,N_{\min})$;

END IF

END FOR

算法 3 中,使用了递归过程.通过本算法的递归,完成了频繁子图的枚举挑选过程,输出了更新后的频繁子图集合 S_{frequent}.对于每个频繁子图 $structure\in S_{\text{frequent}}$,搜索 *Res* 中所有结构图同构于 *structure* 的图的集合 U_s.用索引组织这些同类型的图,索引存储了子图与子图自身在一个或多个图中所在位置之间的映射.对于图 1 所示实例,以下图 4 为其频繁子图挖掘结果.

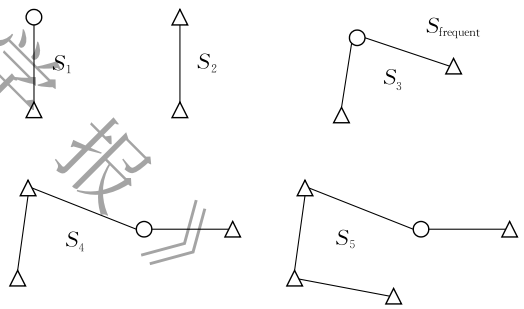

图 4 频繁子图集合

对应的每个 S_{frequent} 中的结构建立的索引见表 1,其中集合元素 $U_{S_{1,1}}$ 中仅存储了顶点的标号,虽然增加了少量的存储空间,但是会大幅提高图筛选过程的效率.

表 1 索引示例

频繁子图	同构图集	集合元素
S_1	U_{s_1}	$U_{s_{1,1}}U_{s_{1,2}}U_{s_{1,3}}\cdots$
S_2	U_{s_2}	$U_{s_{2,1}}U_{s_{2,2}}U_{s_{2,3}}\cdots$
S_3	U_{s_3}	$U_{s_{3,1}}U_{s_{3,2}}U_{s_{3,3}}\cdots$
S_4	U_{s_4}	$U_{s_{4,1}}U_{s_{4,2}}U_{s_{4,3}}\cdots$
S_5	U_{s_5}	$U_{s_{5,1}}U_{s_{5,2}}U_{s_{5,3}}\cdots$

4.2 请求图的切割

设 *Req* 是一个请求图,为了达到加快匹配速度

的目的，利用上一节从资源图 Res 中挖掘出来的频繁子图集 S_{frequent} 对 Req 进行切割. 将 S_{frequent} 中所有子图按复杂度降序排列以后，遍历排序后集合. 如果对于任意 $s\in S_{\text{frequent}}$，Req 中存在与 s 图同构的图，则将该图在保留边缘点的前提下从 Req 上切割下来，并存入结果集 S_{request} 中.

复杂度排序，是指对频繁子图集 S_{frequent} 中的子图按照边点的数量排序. 排序的原则如下：

(1) 边数多的子图的复杂度更高；

(2) 相同边数的子图中点数多的子图复杂度更高；

(3) 点数边数都相同的子图中有环等特殊结构的子图的复杂度更高.

该算法主要程序如下.

算法 4.　图切割过程($Graph\_Cut$).

输入：请求图 Req，频繁子图集 S_{frequent}

输出：请求图的子图集 S_{request}

从集合 S_{frequent} 中读频繁子图集合；

把频繁子图按复杂度降序排序，存于集合 S'_{frequent} 中；

FOR EACH 频繁子图 $sfre\in S'_{\text{frequent}}$

　查找请求图 Req 中与 $sfre$ 图同构的子图；

　IF Req 中存在与 $sfre$ 图同构的子图

　　将该子图存入结果集 S_{request} 中；

　　在 Req 中删除与该子图有关信息(边缘点除外)；

　END IF

END FOR

如算法 4 所示，在对请求图 Req 的切割过程中，首先从集合 S_{frequent} 中将所有的频繁子图的集合取出，然后按照复杂度对其进行降序排序，接下来对排序后的频繁子图集合中的每一个频繁子图，依次在请求图中查找与其满足图同构条件的子图，并对于同构的频繁子图加入到请求图的子图集合 S_{request} 中，最后将处理完成的 S_{request} 返回.

该算法的时间复杂度是 $O(|S_{\text{frequent}}|\times|Req|)$，此处，$|S_{\text{frequent}}|$ 是频繁子图集内元素个数，$|Req|$ 是请求图 Req 内边的个数.

将图 2 作为一个请求图，则对它的切割结果如图 5 所示.

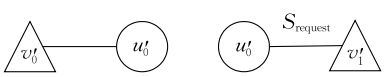

图 5　请求图切割

S'_{frequent} 为经过复杂度排序的频繁子图集，在遍历 S'_{frequent} 时，能够发现在 Req 中存在 s_1 的图同构子图，如图 5 中的结构.

4.3　候选集的筛选

首先，对于任意 $rs\in S_{\text{frequent}}$，利用在资源图的预处理时建立的索引，得到与 rs 图同构的子图集. 然后将子图集里的每个子图上的数值标签与 rs 比较，筛选出符合要求的子图，称之为候选集. 在此，着重介绍一下数值标签的比较操作.

在进行数值标签比较操作时，设计一种分层次的坐标筛选方法. 所谓分层，就是将需要考虑的不同要素分别放在不同层上，如在第一层考虑服务器的计算能力，在第二层考虑服务器的存储能力. 在每一层上，获取到该子图上的所有点的标签，然后按照子图中点的遍历顺序将每个点对应的标签映射为该子图每维上的坐标. 最后将这些坐标转化为点放入坐标系中，与 rs 上标签转化的点比较，满足要求的即为候选集.

在比较顶点上标签时，首先选取一个常量 α，将坐标系上的每维都划分为 α 部分. 这种划分使得整个坐标系被划分为 α^n 块，其中 n 为坐标系维度. 划分需要使得点均匀分布在块中. 如图 6 示例所示.

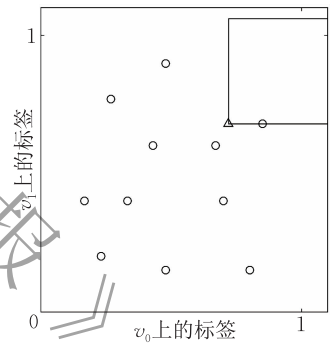

图 6　坐标划分

在以上例子中，将图 6 中的右边图划分成了 16 块($\alpha=4$)，并且使得点均匀落在每个块中. 基于已经划分完的坐标，根据请求图的子图所在块 t 的位置，将坐标图中的块归为 3 类：(1) 无需考虑即舍弃的块；(2) 有待进一步比较的块；(3) 无需考虑即通过的块. 其中，(1) 无需考虑即舍弃的块是指在某一维上的坐标比 t 小的块；(2) 有待进一步比较的块是指在某一维上的坐标与 t 相等的块(包括 t)；(3) 无需考虑即通过的块是指每维坐标都大于 t 的块. 按照以上方法，图 6 的分类如下：

(1) 块中的点不需要与请求子图进行比较；(2) 块中的点需要比较 1 次(t 中的点需要比较两次)；(3) 块中的点也不需要比较. 块划分之后的结果如图 7 所示.

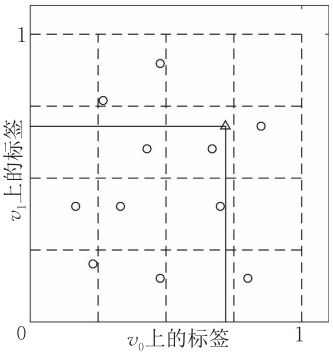

图 7 块划分

通过这种对坐标系的划分可以减少比较的次数,加快程序响应时间.算法主要程序如下.

算法 5. 图筛选过程(*Graph_Filter*).

输入:索引 U_s,常数 α,请求图的子图集 S_{request}

输出:经过筛选的索引 U_s'

FOR EACH 频繁子图 $reqt \in S_{\text{request}}$

按照 *reqt* 中点的遍历顺序将数值标签转化为坐标 *coordinate*,点类型设置为 *type*,放入对应的块中;

将坐标系划分为无需考虑即舍弃的块、有待进一步比较的块和无需考虑即通过的块;

FOR EACH U_{s_n},U_{s_n} 图同构于 *reqt*

IF U_{s_n} 落在无需考虑即通过的块中的点,或落在有待进一步比较的块且每维坐标都≥*coordinate* 且 *type* 相等的点

将这些点存入经过筛选的索引 U_s';

END IF

END FOR

END FOR

算法 5 利用 4.3 节介绍的思想对索引 U_s 进行过滤,从而提高整个方法的效率.算法首先将频繁子图集合中元素的数值标签转化为坐标,点类型设置为 *type*,然后将坐标系划分为无需考虑即舍弃的块、有待进一步比较的块和无需考虑即通过的块,对于无需考虑即通过的块中的点,或落在有待进一步比较的块且每维坐标都大于等于 *coordinate* 且 *type* 相等的点加入到筛选后的集合 U_s',其他点则舍弃.

该算法的时间复杂度是 $O(|S_{\text{request}}| \times |U_{s_n}|)$,此处,$|S_{\text{request}}|$ 是请求图子图集内元素个数,$|U_{s_n}|$ 是索引内子图的个数.

4.4 候选集的合并

得到候选集后,剩下的工作就是将这些候选集合并起来,得到一个完整的与请求图 *Req* 图同构的结果集合.因为在对请求图切割的时候是按照复杂度进行的,在这里也按照该顺序继续进行.首先,选取请求图的子图集 S_{request} 的第一个子图 rs_1,查找剩余子图中与其有公共边缘点(切割时保留的点)的子图,将它们进行合并.同样,按照这种邻接关系,合并候选集中与这些子图同构的子图集,直到合并的结果与请求图同构为止,此时的结果即是满足客户需求的一组服务器的集合.具体算法如算法 6 所示.

算法 6. 子图合并过程(*Subgraph_Join*).

输入:请求图 *Req*,请求图的子图集 S_{request},经过筛选的索引 U_s'

输出:合并后的候选集 *Candidate*

IF S_{request} 中第一个元素 sap_1 与 *Req* 图同构

RETURN 候选集 *Candidate*;

END IF

FOR EACH $sap_n \in S_{\text{request}}$,$sap_n$ 与 sap_1 有公共边缘点

将 sap_1 与 sap_n 合并,并存入 sap_1 中;

按 sap_1 与 sap_n 的邻接关系查找 U_s' 中的元素 U_{s_1} 与 U_{s_n} 中有同样邻接关系的子图;

将 U_{s_1} 与 U_{s_n} 中有此邻接关系的子图合并,存入 *Candidate* 中;

在 S_{request} 中删除 sap_n,在 U_s' 中删除 U_{s_n};

Subgraph_Join(*Req*,S_{request},U_s');

END FOR

经过算法 6,算法合并完成了与请求图同构的子图合并,最终输出合并后的候选集 *Candidate*,即满足客户需求的一组服务器的集合.

该算法的时间复杂度 T 为

$$T=O(\prod_{j=1}^{|S_{\text{request}}|} j \times \prod_{k=0}^{|U_s'|} |U_{s_k}'|).$$

此处,$|S_{\text{request}}|$ 是请求图子图集内元素个数,$|U_s'|$ 是候选图内子图种类数,$|U_{s_k}'|$ 是候选图内各子图中元素个数.

图 8 为对图 5 中图的合并结果,两个子图的公共边缘点为 u_0',将两图合并,并在经过筛选后的索引 U_{s_1} 里,找寻具有同样的邻接关系的子图合并.按照上述合并顺序将候选集合并以后,得到的一组与请求图 *Req* 图同构的资源图 *Res* 的子图,即满足客户应用所需的不同种类服务器的性能要求以及它们之间的拓扑关系的服务器集合.

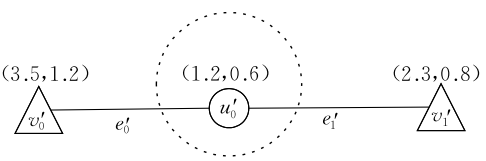

图 8 候选集合并

5　仿真实验与结果分析

本节将通过仿真实验评估本文提出的算法，文中所有仿真的实验平台环境为：INTEL Core i5 CPU，2.80 GHz，4 GB 内存.

资源图取自 CAIDA① 中包含最多节点的 2007 年 11 月 5 日的图，其中包含 26 475 个节点、106 762 条边. 其中计算资源与存储资源的比例在 2∶1 至 3∶1 之间随机生成. 计算资源上的数值标签来源于 ClusterData②，它包含了 1.1 万台计算机的 CPU 和内存使用情况，并表示成相应的数值[14]，并用随机映射函数将 ClusterData 中的标签与资源图中计算资源关联起来. 资源图中存储资源的存储大小由随机函数在 1 T 至 5 T 之间随机生成.

请求图一般规模较小，因此利用 5～25 条边的所有连通图，所需计算资源与存储资源的比例在 2∶1 至 3∶1 之间随机生成，其上节点的标签由随机函数生成.

5.1　算法性能分析

本文将频繁子图出现次数阈值 N_{min} 多次取值，对 CAIDA 进行挖掘. 其中，每个取值重复实验 10 次，并取均值作为实验结果. 实验结果如图 9 所示.

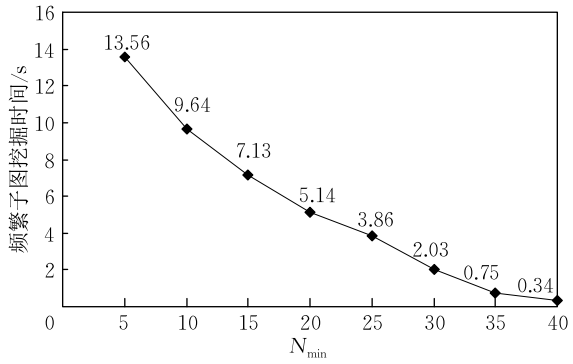

图 9　不同的 N_{min} 所需挖掘时间

本文选取的几个 N_{min} 取值依次为 5，10，15，20，25，30，35，40. 观察实验结果的平均值可以发现，随着 N_{min} 取值的增大，所需的挖掘时间在缩短. N_{min} 为 5 到 N_{min} 为 40 的挖掘时间从 13.56 s 减少到 0.34 s. 这是因为 N_{min} 越高的子图在资源图中出现的次数越少，所需的挖掘时间也成倍减少.

图 10 是对资源图进行频繁子图挖掘所得到频繁子图数量的实验.

从图 10 实验结果可以发现，随着 N_{min} 取值的增大，频繁子图的数目在减少，所需的挖掘时间也在缩短. N_{min} 为 5 的频繁子图的个数为 2657 个，而 N_{min} 为 40 的频繁子图个数则减少到了 24 个.

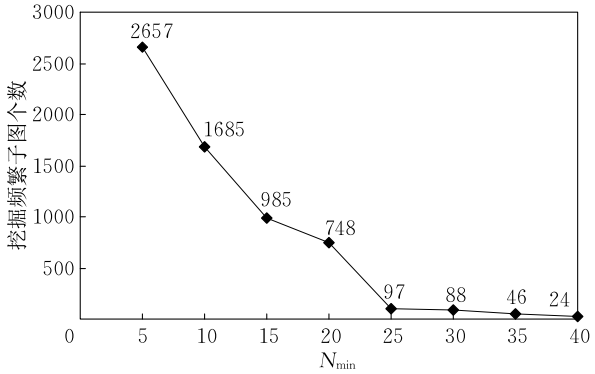

图 10　不同的 N_{min} 所能挖掘到的频繁子图数量

由图 9 和图 10 可以明显看出，随着 N_{min} 的增加，所需频繁子图挖掘时间减少，但是同时可以看出，在 N_{min} 大于 25 后，所能挖掘到的频繁子图数量急剧减少. 而频繁子图数量的减少意味着请求图匹配的成功率也随之降低. 通过大量实验数据表明，在 N_{min} 大于 25 时，匹配成功率不足 50%，因此，N_{min} 取值不能过大，本文后续的实验均是在 N_{min} 小于等于 25 的环境下进行的.

图 11 为算法对于不同规模的请求图和不同属性维度处理所需时间（N_{min} 均取值为 3），属性维度即为第 3 节中所述图中顶点上的标签属性的个数. 图 11 中横轴表示请求图的不同规模，纵轴表示算法的处理时间，5 条线分别表示属性维度取 1～5. 由图 11 可以看出，随着属性维度的增加，对于同一规模请求图算法的处理时间基本没有变化. 而对于同一属性维度，随着请求图规模的增长，算法的处理时间增长较快，然而考虑到请求图一般规模较小，因此，算法的处理时间不会过长.

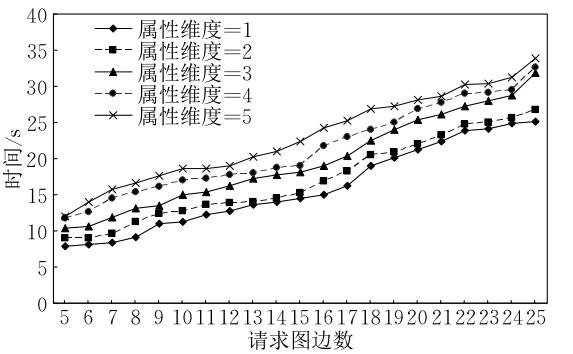

图 11　不同规模的请求图、不同属性维度的处理时间

① http://snap.stanford.edu/data/as-caida.html
② http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1

图 12 为算法对于不同规模的请求图和 N_{min} 取不同值时算法处理所需时间(属性维度均取值为 2). 图 12 中横轴表示 N_{min} 的不同取值,纵轴表示算法的处理时间,5 条线分别请求图的不同规模(边数). 由图 12 可以看出,随着 N_{min} 的增加,对于同一规模请求图算法的处理时间减少,直至无法找到匹配的子图. 而对于同一 N_{min},随着请求图规模的增长,算法的处理时间增大.

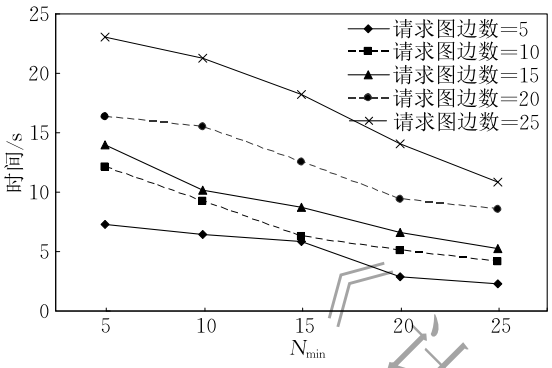

图 12 不同规模的请求图、不同 N_{min} 的处理时间

表 2 为算法对于不同规模的请求图和 N_{min} 取不同值时算法能够找到匹配子图的比例. 由表 2 可以看出,对于同一规模请求图,N_{min} 越小,算法找到匹配子图的概率越大. 而对于同一 N_{min},请求图规模越小,算法找到匹配子图的概率越大.

表 2 成功找到匹配子图的情况

请求图大小 *	N_{min}	实验次数	找到匹配子图次数	成功率/%
5	5	1000	872	87.2
5	10	1000	823	82.3
5	15	1000	716	71.6
5	20	1000	689	68.9
5	25	1000	651	65.1
10	5	1000	829	82.9
10	10	1000	811	81.1
10	15	1000	616	61.6
10	20	1000	539	53.9
10	25	1000	455	45.5
15	5	1000	773	77.3
15	10	1000	689	68.9
15	15	1000	613	61.3
15	20	1000	552	55.2
15	25	1000	403	40.3
20	5	1000	734	73.4
20	10	1000	642	64.2
20	15	1000	607	60.7
20	20	1000	523	52.3
20	25	1000	401	40.1
25	5	1000	721	72.1
25	10	1000	617	61.7
25	15	1000	582	58.2
25	20	1000	510	51.0
25	25	1000	386	38.6

5.2 与类似算法的比较

本节将本文所提出的算法与 Gradin 算法[22]、UpdNo 算法[12]、NaiveGrid 算法[12]进行实验比较,其中着重对算法的查询操作所需时间长短进行对比. 相对本文算法能够支持对多维属性、节点异构子图的挖掘,Gradin 算法可以挖掘一维属性且节点图同构的子图. UpdNo 算法是一种反向索引算法,其中每个条目指向相同结构的子图的列表,该算法从来不更新索引,因此需要大量的比较来搜索候选片段. NaiveGrid 算法是一种具有朴素验证算法的网格索引算法.

实验所采用的资源图结构为经过切割的 CAIDA 的子图,因此包含 500,5000,10 000,20 000,40 000,60 000,80 000,100 000 条边,请求图为 5,10,15,20,25 条边的所有连通图,N_{min} 在此分别取值 15,20,25. 实验结果如图 13 所示.

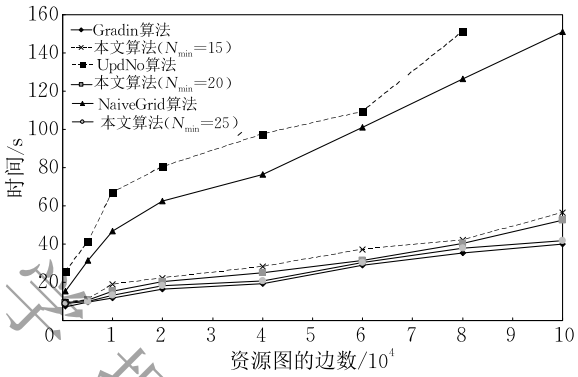

图 13 本文算法与类似算法比较

从图 13 所示的实验结果可以看出,随着资源图边数取值的增加,三种算法响应时间都会增长. 本文算法所需挖掘时间略多于 Gradin,其主要原因为增加了对于多维属性和异构节点的支持,但时间增加很少. UpdNo 算法因为需要大量的比较来搜索候选片段,因此它在所有算法中耗费时间最多. 并且当资源图边数达到一定数值时,UpdNo 无法在 160 s 内完成查询边数是 100 000 的资源图. NaiveGrid 算法采用网格索引,其查询所花费时间比 UpdNo 算法少. 随着请求图边数增加,其时间增长缓慢.

实验结果显示,不论资源图规模或者请求图边数如何变化,本文算法的花费时间均远少于 NaiveGrid 算法和 UpdNo 算法. 对于 N_{min} 取值为 15,20,25 的三种方案,都能在 100 s 之内找到需要查询的子图.

对比 N_{min} 取值为 15,20,25 的三种方案,当请求图边数为 5 和 10 时,三种方案的花费时间相差很小. 但是当请求图边数增加至 15,20 和 25 时,N_{min}

取值为 20 的方案花费的时间明显少于其他两种.

综合以上实验结果，可以看出本算法对于资源图上请求图的匹配问题具有良好的程序响应时间.

6 总　结

SaaS 是一种全新的软件应用模式，在 SaaS 应用部署过程中，如何找到一组具有足够计算能力、存储能力、合适的维护成本及网络带宽的云资源节点成为当前的一大挑战. 本文提出一种支持多维异构属性节点的图匹配算法，通过将云平台中的云资源及 SaaS 应用对云资源的需求建模为节点具有多维属性的异构图，并对资源图进行切割、筛选和合并，得到资源图的频繁子图集合，然后利用图匹配算法找到 SaaS 应用所需要的云资源，降低了匹配问题复杂度，快速准确地解决了 SaaS 服务放置问题. 实验结果表明，本算法具有理想的查询处理时间.

下一步，我们将研究面向资源图更新的图匹配算法. 另外，在资源图和请求图同时发生变化的情况下[28]，研究如何快速有效地进行图匹配也是一个非常有意义且具有挑战性的课题，是我们未来的研究方向.

参　考　文　献

[1] Huang K, Shen B. Service deployment strategies for efficient execution of composite SaaS applications on cloud platform. Journal of Systems and Software, 2015, 107(C): 127-141

[2] Weitek W, Bai X, Huang Y. Software-as-a-service(SaaS): Perspectives and challenges. Science China Information Sciences, 2014, 57(5): 1-15

[3] Benson T, Akella A, Shaikh A, Sahu S. Cloudnaas: A cloud networking platform for enterprise applications//Proceedings of the ACM Symposium on Cloud Computing. Cascais, Portugal, 2011: 1-13

[4] Ioana G, Claris C, Asser T, Malgorzata S. Enabling placement of virtual infrastructures in the cloud//Proceedings of the 13th ACM International Middleware Conference. Montreal, Canada, 2012: 332-353

[5] Liu Z, Hu Z, Jonepun L. Research on composite SaaS placement problem based on ant colony optimization algorithm with performance matching degree strategy. Journal of Digital Information Management, 2014, 12(4): 225-234

[6] Fan W, Li J, Luo J, Wu Y. Incremental graph pattern matching//Proceedings of the ACM SIGMOD International Conference on Management of Data. Bangalore, India, 2011: 925-936

[7] Jennings B, Stadler R. Resource management in clouds: Survey and research challenges. Journal of Network and Systems Management, 2014, 23(3): 567-619

[8] Gullhav A N, Nygreen B. A branch and price approach for deployment of multi-tier software services in clouds. Computers and Operations Research, 2016, 75: 12-27

[9] Jiao L, Lit J, Du W, Fu X. Multi-objective data placement for multi-cloud socially aware services//Proceedings of the IEEE Conference on Computer Communications. Toronto, Canada, 2014: 28-36

[10] Cheng J, Yu J, Ding B, Wang H. Fast graph pattern matching //Proceedings of the 24th International Conference on Data Engineering. Cancun, Mexico, 2008: 913-922

[11] Yan J, Han J. CloseGraph: Mining closed frequent graph patterns//Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Washington, USA, 2003: 286-295

[12] Huan J, Wang W, Prins J. Efficient mining of frequent subgraphs in the presence of isomorphism//Proceedings of the 3rd IEEE International Conference on Data Mining. Los Alamitos, USA, 2003: 549-552

[13] Shi Xue-Lin, Xu Ke. Utility maximization model of virtual machine scheduling in cloud environment. Chinese Journal of Computers, 2013, 36(2): 252-262(in Chinese)
(师雪霖，徐恪. 云虚拟机资源分配的效用最大化模型. 计算机学报，2013，36(2)：252-262)

[14] Qian B, Meng F, Chu D. A cost-driven multi-objective optimization algorithm for SaaS applications placement// Proceedings of the IEEE International Conference on DataCom. Chengdu, China, 2015: 1086-1091

[15] Meng Fan-Chao, Zhou Xue-Quan, Cao Zu-Feng, et al. Multi-tenant SaaS application placement algorithm based on cost optimization. Computer Integrated Manufacturing Systems, 2014, 20(6): 1508-1518(in Chinese)
(孟凡超，周学权，曹祖风等. 基于成本优化的多租户 SaaS 应用优化放置算法. 计算机集成制造系统，2014，20(6)：1508-1518)

[16] Amiri A. Application placement and backup service in computer clustering in Software as a Service(SaaS) networks. Computers and Operations Research, 2016, 69(C): 48-55

[17] Meng Fan-Chao, Chu Dian-Hui, Li Ke-Qiu, Zhou Xue-Quan. Solving SaaS components optimization placement problem with hybird genetic and simulated annealing algorithm. Journal of Software, 2016, 27(4): 916-932(in Chinese)
(孟凡超，初佃辉，李克秋，周学权. 基于混合遗传模拟退火算法的 SaaS 构件优化放置. 软件学报，2016，27(4)：916-932)

[18] Hajji M A, Mezni H. A composite particle swarm optimization approach for the composite SaaS placement in cloud environment. Soft Computing, 2017: 1-21

[19] Gullhav A N, Cordeau J, Hvattum L M, Nygreen B. Adaptive large neighborhood search heuristics for multi-tier service deployment problems in clouds. European Journal of Operational Research, 2017, 259(3): 829-846

[20] Su W, Hu J, Lin C, Shen S. SLA-aware tenant placement and dynamic resource provision in SaaS//Proceedings of the 2015 IEEE International Conference on Web Services. New York, NY, USA, 2015: 615-622

[21] Bhardwaj S, Sahoo B. A particle swarm optimization approach for cost effective SaaS placement on cloud//Proceedings of the International Conference on Computing, Communication and Automation. Uttar Pradesh, India, 2015: 686-690

[22] Zong B, Raghavendra R, Srivatsa M, et al. Cloud service placement via subgraph matching//Proceedings of the IEEE 30th International Conference on Data Engineering. Chicago, USA, 2014: 832-843

[23] Yan X, Han J. gSpan: Graph-based substructure pattern mining//Proceedings of the IEEE International Conference on Data Mining. Maebashi City, Japan, 2002: 721-724

[24] Yan X, Yu P, Han J. Graph indexing: A frequent structure-based approach//Proceedings of the International Conference on Management of Data. Mysore, India, 2009: 335-346

[25] Reiss C, Tumanov A, Ganger G, et al. Heterogeneity and dynamicity of clouds at scale: Google trace analysis//Proceedings of the 3rd ACM Symposium on Cloud Computing. San Jose, USA, 2012: 1-13

[26] Bansal N, Lee K, Nagarajan V, Zafer M. Minimum congestion mapping in a cloud//Proceedings of the ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing. San Jose, USA, 2011: 267-276

[27] Meng X, Pappas V, Zhang L. Improving the scalability of data center networks with traffic-aware virtual machine placement//Proceedings of the IEEE INFOCOM. San Diego, USA, 2010: 1-9

[28] Raghavendra R, Lobo J, Lee K. Dynamic graph query primitives for SDN-based cloud network management//Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks. Helsinki, Finland, 2012: 97-102

GUO Wei, born in 1978, Ph. D., engineer. His research interests include cloud computing and big data management, intelligent data analytics.

ZHANG Kai-Qiang, born in 1995, bachelor. His research interest is cloud resources placement.

CUI Li-Zhen, born in 1976, Ph. D., professor, Ph. D. supervisor. His research interests include big data management and big data analytics, big data artificial intelligence, service computing and collaborative computing, software architecture and technology in cloud.

XU Meng, born in 1978, Ph. D., lecturer. His research interests include cloud computing and big data management, business process management.

Background

Software-as-a-Service (SaaS) is a new software delivery model that provides on-demand customization and payment for tenants based cloud platform. SaaS has drawn considerable research attention for its capability of transferring software goods for services in light of the Internet based platforms. Individual software vendors release resources and services to tenants by deploying the SaaS applications accordingly. SaaS applications in cloud are generally large scale multi-layer software application systems, which deployed in multiple nodes and are very important to tenants. On the other hand, SaaS service providers wish to reduce the total cost and gain more benefit. To this end, SaaS service providers require to quickly deliver and place multiple diversified SaaS applications in the cloud data center to meet the multi-dimensional and heterogeneous requirements, such as performance, network utilization, storage capability and operating system requirements. As a result, it is essential to achieve the SaaS application agile placement to quickly select appropriate cloud resources to place large scale SaaS application system to meet the needs of large scale heterogeneous performance requirements of different tenants and hence reduce the cost of cloud service providers at the same time. Therefore, the key point of placing SaaS applications in an efficient way lies on choosing the appropriate cloud resources to deploy the SaaS applications. Traditional cloud resources matching methods based on Service Level Agreement SLA and supply demand have been difficult to adapt to the requirements of agile placement of large scale SaaS applications in cloud for the complicated requirements of large-scale heterogeneous performance. In this paper, the strategy of placing SaaS applications based on

graph matching theory is proposed which models the SaaS applications placement problem as the subgraph matching problem of the cloud resource graph. The computing resources, data resources and their inherent connections in the cloud are mapped into an isomerism graph with multi-dimensional attribute labels. In this graph, vertices represent the cloud resources required by SaaS applications, while edges represent topological relationships between cloud resources. The attribute values on vertices represent the performance requirements of SaaS applications for cloud resources. After constructing the above graph, an algorithm which contains four steps is presented. The first step is to mine the frequent subgraphs which satisfy a certain threshold in the cloud services resource graph. In the next step, the request graph is cut using frequent subgraphs. Then the subgraph set of the request graph is obtained. The third step is to filter the subgraph set of the request graph. In this process, the algorithm dexterously uses the coordinate division method to improve the efficiency of the filtering operation. In the final step, the candidate set is merged to reduce the complexity of the problem and solve the SaaS applications placement problem quickly and accurately. A set of cloud resources that meet the provider's requirements can be eventually obtained by using the partial order relation isomerism matching method, which is employed to agilely place SaaS applications and data. The extensive experimental results suggest that the proposed method illustrates significant improvement in the efficiency of multi-dimensional heterogeneous cloud resource placement strategy in complex SaaS applications.