doi:10.11959/j.issn.1000-436x.2016219

SDN 网络中受时延和容量限制的多控制器均衡部署

覃匡宇[1,2]，黄传河[1,2]，王才华[1]，史姣丽[1,2,3]，吴笛[1,2]，陈希[1,2]
（1. 武汉大学计算机学院软件工程国家重点实验室，湖北 武汉 430072；2. 武汉大学地理空间信息技术协同创新中心，湖北 武汉 430072；
3. 九江学院信息科学与技术学院，江西 九江 332005）

摘 要：软件定义网络（SDN）采用一种控制平面和数据平面分离的网络架构，其控制功能由控制器来实现。由于受到控制器处理能力的限制，在大型的 SDN 网络中，单一的控制器无法满足全体交换机的控制需要，必须使用多个控制器来处理所有的数据流。由于控制器和交换机之间的时延将显著地影响新流的转发，控制器的合理部署将有效地提高整个网络的性能。通过对网络进行子域划分，在谱聚类的基础上，通过为 *k*-means 增加均衡部署的目标函数，提出了在时延和容量限制下负载均衡的 SDN 网络多控制器部署算法。该算法中引入了一个惩罚函数来防止出现孤立节点。仿真结果表明该算法能均衡地对网络进行划分，使控制器和交换机之间保持较小的网络时延以及使各控制器的负载保持均衡。

关键词：软件定义网络；控制器部署；最小时延；负载均衡；*k*-means；谱聚类
中图分类号：TP393 **文献标识码**：A

Balanced multiple controllers placement with latency and capacity bound in software-defined network

QIN Kuang-yu[1,2], HUANG Chuan-he[1,2], WANG Cai-hua[1], SHI Jiao-li[1,2,3], WU Di[1,2], CHEN Xi[1,2]
(1. State Key Lab of Software Engineering, Computer School, Wuhan University, Wuhan 430072, China;
2. Collaborative Innovation Center of Geospatial Technology, Wuhan University, Wuhan 430072, China;
3. School of Information Science and Technology, Jiujiang University, Jiujiang 332005, China)

Abstract: Software-defined network (SDN) used a network architecture which separates the control plane and data plane. The control logic of SDN was implemented by the controller. Because controller's capacity was limited, in large scale SDN networks, single controller can not satisfy the requirement of all switches. Multiple controllers were needed to handle all data flows. By the reason that the latency between controller and switch would significantly affect the forwarding of new data flow, the rational placement of controllers would effectively improve the performance of entire network. By partition the network into multiple sub domains, on the base of spectral clustering, a method that added a balanced deployment object function into *k*-means was given and a balanced multiple controllers placement algorithm in SDN networks which has the latency and capacity limitations was proposed. In this approach, a penalty function was introduced in the algorithm to avoid isolation nodes appearing. The simulations show that this algorithm can balance partition the network, keep the latency between controller and switch small and keep loads balancing between controllers.

Key words: software-defined network, controller placement, minimal latency, load balancing, *k*-means, spectral clustering

1 引言

软件定义网络是一种新型的可以简化管理、提升扩展性的网络架构，其思想是将交换机网络的控制平面和数据平面分离[1]。在 SDN 网络中，交换机只负责数据转发，控制逻辑由称为控制器的服务器来提供。在目前广泛使用的 SDN 方案 OpenFlow 中，数据转发以流为单位，一个流根据网络地址、网络

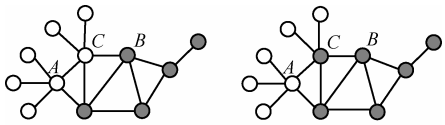
收稿日期：2016-02-25；**修回日期**：2016-09-06
通信作者：黄传河，huangch@whu.edu.cn
基金项目：国家自然科学基金资助项目(No.61373040, No.61572370)
Foundation Item: The National Natural Science Foundation of China (No.61373040, No.61572370)

端口和协议类型等信息来进行定义。OpenFlow 交换机中维持了多个流表来指导流的转发。当数据流进入 OpenFlow 交换机时，交换机会按照与该流匹配的流表项来转发数据。如果交换机的流表中没有与之匹配的转发项，则会向控制器发出查询。控制器做出决策后再将新的流表项下发到交换机。这套机制使网络管理者可以对数据的转发进行精细的控制，同时交换机可以被做得简单和便宜，交换机只需要实现数据转发功能，路由、安全策略等控制任务统一由控制器负责。SDN 网络带来的另外一个好处就是易于扩展。由于所有的控制逻辑都转移到控制器上，只要升级控制器的软件，整个网络就能够支持相应的新特性。

尽管 SDN 网络具有很多优点，但其缺点也同样明显。由于控制器担负了整个网络的控制工作，控制器的处理能力及控制器与交换机之间通信的时延对整个网络的性能有着重要的影响。对于大型的网络，靠单台控制器进行流表的分发无法胜任全体交换机的需求，这时需要使用分布式的多个控制器来分担整个系统的压力。如何在大型网络中有效地实现分布式的控制管理，是 SDN 网络的一个重要研究课题。本文研究如何在 SDN 网络中部署多个控制器的问题。多控制器的部署包括 3 方面：1) 确定需要多少个控制器；2) 这些控制器部署在哪里；3) 哪些交换机归由哪个控制器来进行管理。

要实现控制器的合理部署，实际上面临着较大的挑战。为了保证网络的性能，交换机到其管理控制器之间存在一个最大可容忍时延，同时单个控制器受限于处理能力，能管理的交换机个数有限。因而部署中受到最大时延和控制器容量的限制。在实际部署中，为了降低部署成本，应尽量减少控制器的个数。在保证低成本的同时还要提高性能，交换机到控制器的平均时延应该尽可能小，各控制器间负载应尽可能均衡，因而部署要兼顾控制器个数少、时延小、负载均衡 3 个指标。单独考虑时延最小这一个指标时，SDN 网络受时延和容量限制的多控制器部署问题可归约为带权的最小覆盖集问题，这是一个 NP 完全问题，理论上无法找到多项式时间内的解法。当同时考虑多个指标时，情况更加复杂。此外，当网络拓扑存在复杂的结构时，还会出现连通性的问题，如图 1 所示。

图 1(a)为找到的一个部署方案 1，此时网络分为 2 个域，其节点分别用浅色和深色表示，*A* 和 *B* 分别为 2 个域中控制器的位置，但浅色区域有 7 台交换机，深色区域只有 5 台交换机。出于负载均衡的需要，可以调整交换机的分配，将交换机 *C* 分配给控制器 *B*，因为 *C* 到 *A* 和 *B* 有相等时延，此时可以在不改变整体平均时延的情况下，获得更好的均衡性，如图 1(b)所示。但这样的调整却导致了浅色区域被割裂，使原来的域内通信变成跨域通信，产生性能、可靠性和安全性等一系列问题。这就是控制器部署中的域内连通性问题。如果不进行连通性分析，仅从优化公式上难以发现其问题，当网络拓扑为非凸结构时，这种情况很容易出现。因而，健壮的控制器部署算法应该能够识别类似图 1(b)的情况，绕过被割裂的方案，实现最合理的部署。



(a) 方案1　　(b) 调整节点*C*到深色域

图 1　优化中出现不连通情况

为了实现控制器的均衡部署，相比于先确定控制器位置再进行交换机的分配，若从一开始就对网络进行均衡的分区并求取各区域中心作为控制器的部署位置，后者能更好地实现降低时延和负载均衡的要求。本文通过对网络时延及拓扑结构进行分析，以改进的谱聚类方法来研究控制器的部署问题，较好地解决了这些问题。本文的贡献在于：1) 在谱聚类的基础上改进了 *k*-means 算法，通过在每一轮迭代的过程中加入了一次节点类别的调整，实现了 SDN 网络中受时延和容量限制的多控制器均衡部署；2) 提出了部署中的连通性问题，在交换机调整的目标函数中加入了孤立点惩罚因子，使调整过程有效地避免了域内出现孤立节点。

2　相关的研究工作

近年来，SDN 的相关研究一直保持着持续的热度。为了解决单控制器性能不足的问题，相关研究人员提出了许多解决方案。在 DevoFlow 中，研究人员提出在交换机中预先安装通配符规则，通过区分处理“老鼠流”和“大象流”，在交换机上平均减少了 10~53 倍的流表量和 10~42 倍的控制信息[2]。DIFANE 在交换机中设置权威交换机，控制器规则先下发到权威交换机中，普通交换机直接向权威交换机查询，权威交换机再将规则下发到普通交换机[3]。

在分布式控制器框架中，Onix 使用分布式散列表来管理控制器的状态分布[4]。HyperFlow 使用发布/订阅系统来处理控制器的状态分布[5]。Kandoo 框架包括 2 层控制器，下层是多个不互连的局部控制器，上层是一个处于逻辑中心的全局控制器，负责全局网络视图[6]。在控制器性能分析方面，文献[7]对 OpenFlow 控制器在主动式和被动式模式下进行了性能分析。文献[8]测量了 OpenFlow 网络在主动式、被动式和混合式模式下的控制器负载情况。文献[9]显示 NOX 控制器可以每秒安装 3 万条流并保证流处理时延在 10 ms 以下。

在 SDN 控制器的部署问题上，学者们也做了很多研究。其中，大量的文献以交换机到控制器的时延作为部署依据。Heller 等[10]最早提出了控制器的部署问题，使用了平均时延和最大时延这 2 个指标来分析控制器的部署，并利用贪心算法进行求解，但该算法未考虑控制器的容量问题。Sallahi 等[11]提出了控制器部署问题中基于部署代价的完整模型，但文献没有给出算法。Ishigaki[12]提出了一种压力中心的节点计算指标，并给出了基于该中心的控制器部署算法，但未涉及控制器容量及均衡问题。Jimenez 等[13]提出了 *K*-critical 算法，通过构建 Robust 树，根据最大允许时延来计算所需控制器的数量及部署位置，但该文献同样未考虑控制器的容量。文献[14]中提出了使用一种改进的 *k*-means 算法来对 SDN 控制器进行部署，其算法开始时只分一个区，随后逐次增加分区数量的方法来进行迭代。但其目标关注于控制器到各节点的距离最小，并未考虑负载均衡。Tracy 等[15]分别提出了基于贪心、基于原始对偶和基于分区的 3 个算法来进行控制器部署，但未能综合考虑最少控制器、最小时延和负载均衡问题。一些文献研究了控制器的容量问题，毕军等[16]提出了基于容量的控制器部署算法。文献[17]使用了粒子群算法来求解 SDN 的控制器部署问题，其优化目标为控制器到交换机及控制器之间延迟最小，并考虑了控制器的容量限制。文献[18]将粒子群算法和节点分区结合起来，提出了 NCPSO 控制器部署算法，兼顾了最小时延和负载均衡。Xiao 等[19]使用了谱聚类来解决广域网上的控制器部署问题，其算法具备一定的负载均衡效果。但以上几篇文献在给控制器分配交换机时均未考虑可能出现的域内节点不连通问题。

在基于 SDN 可靠度的部署方面，Neda 等[20]分析了 SDN 中控制平面和转发平面之间的连接恢复问题，并根据节点度提出了贪心的控制器部署算法和基于贪心路由树的算法。Lucas 等[21]提出了 Survivor 优化部署算法来提高网络的可靠性。Survivor 算法考虑 3 个方面：连通性、容量和恢复性。为了保证连通性，算法选择节点不相交路径最多的位置进行部署，同时还提出了备份控制器的选择方法。胡延楠等[22~24]研究了 SDN 中最大化 SDN 控制网络可靠性的控制器部署问题并提出了 SDN 网络可靠性的度量和部署算法。Guo 等[25]提出了 SDN 可靠度的度量并给出了基于根据 closeness 中心进行控制器部署。这些文献中的算法均是基于可靠度优化而不是以时延作为优化目标。Guo 等[26]给出了以网络状态延迟作为优化目标的 SDN 失效分析模型，并提出了 2 个最小化网络状态延迟的部署算法，但没有涉及负载均衡。

一些文献考虑了以多个优化目标作为部署依据，David 等[27, 28]在考虑多种网络失效情况的条件下，提出了基于 Pareto 最优的控制器部署框架 POCO，以及基于 Pareto 最优的控制器的动态部署方法[29]，随后又提出了基于 Pareto 模拟退火的启发式算法[30]。这些文献给出了控制器失效、链路失效、交换机到控制器延迟、控制器到控制器延迟和控制器负载差异等各种不同的评价指标，并根据这些指标给出了帕累托最优的算法。Vahid 等[31, 32]将多目标遗传算法 NSGA-II 引入到控制器部署问题中，并给出了相应的算法。但这些文献均未考虑控制器分配交换机时域内可能出现不连通的问题。

在动态控制器部署方面，Yao 等[33]提出了单控制域和多控制域的控制器部署方法，根据节点权重和路由代价作为部署依据，同时给出了交换机控制权的动态迁移算法。Bari 等[34]提出了根据网络状态实时动态分配控制器的算法，根据 4 种代价之和最小来提出了贪心背包算法和模拟退火算法。Rath 等[35]提出了基于非零和博弈的控制器优化部署方法，该方法认为每个控制器是一个游戏参与者，根据规则参与游戏并计算自身的收益。根据收益情况关闭控制器或是向邻居控制器卸载部分负载，实现网络均衡。以上这些动态算法均未考虑域内可能出现不连通的问题。此外，这些算法是先确定控制器位置后再进行交换机的分配调整，若能从网络开始划分时就考虑均衡性问题，则可使选出的控制器位置更好地位于最终域的中心。

综上，目前尚未有相关文献在同时考虑最少控制器数、最小时延和负载均衡的情况下又考虑了域内连通性。本文将谱聚类算法引入到 SDN 多控制器部署问题的求解，通过改造谱聚类中最后一步的 *k*-means 算法，在 *k*-means 算法的每一轮迭代后加入了一次根据目标函数的分类结果调整，同时在调整目标中加入了孤立节点惩罚函数，在保证了连通性的同时，实现了均衡的 SDN 控制域划分及控制器部署。

3　控制器部署模型

在 SDN 控制器的部署中，在满足需求的情况下，出于成本的考虑，应该使所需的控制器尽可能少。此外，由于受到控制器处理能力及带宽的限制，单台控制器能够管理的交换机台数是有限的，对交换机的分配要考虑控制器最大容量的问题。同时，为了保证基本的性能，应该使交换机到控制器的通信时延不超过一个可容忍的阈值。因而，SDN 网络中受时延和容量限制的多控制器部署可描述为：给定一个 SDN 网络，已知其交换机和链路的拓扑结构，以及链路间的时延情况，求解部署最少的控制器使交换机到控制器的查询时延不超过可容忍的最大时延 T，并且单台控制器管理的交换机台数不超过给定的上限值 B。同时使交换机到控制器的平均查询时延尽可能小，并使各控制器分配的负载尽可能均衡。

考虑到控制器是一种服务器，需要连接在某台交换机上，本文求取的部署位置指的是控制器所连接的交换机节点的位置。因控制器到其部署的交换机之间的时延未做约定，其值不改变部署选址结果，为简化起见，设控制器到其部署的交换机之间的时延为 0。交换机到控制器的可容忍时延也为受控交换机到控制器所连接的交换机的可容忍时延。同时控制器与交换机之间的控制信息与线路上传输的数据量相比并不大，控制器与各交换机之间通信可使用已有的交换机链路而无需为控制器到每个交换机布设专用链路。

将 SDN 网络建模为一个无向图 $G=(V, E, \boldsymbol{D})$，其中，V 为交换机集合，设共有 n 个节点，E 为连接各交换机的链路集合，$\boldsymbol{D}$ 为相邻节点的时延矩阵。设要选出的控制器集合为 C，共有 k 个控制器，其对应的交换机节点位置的序号集合为 CV。每个控制器管理一个域，控制器 j 管理的交换机集合表示为 SA_j。此外，用一个 n 维向量 ***Labels*** 来表示各交换机所在域的序号，即第 i 交换机的所属域为 $Labels_i$。以上关系可表达为

$$V=\{1,2,\cdots,n\} \tag{1}$$

$$C=\{1,2,\cdots,k\} \tag{2}$$

$$CV=\{c_j \mid c_j\in V, j\in C\} \tag{3}$$

$$SA_j=\{v \mid v\in V, v \text{ 由控制器}j\text{管理}\} \tag{4}$$

$$SA_i \cap SA_j=\phi,\ \ i,j\in C \tag{5}$$

$$\bigcup_{j=1}^{k} SA_j = V \tag{6}$$

$$Labels_i = j,\ \ i\in SA_j \tag{7}$$

$$SA=\{SA_1, SA_2, \cdots, SA_j, \cdots, SA_k\} \tag{8}$$

根据 OpenFlow 的工作机制，本文对控制器部署问题定义以下指标。

定义 1　总体部署代价（TPC, total placement cost）。基于使用带内通信的约定，无需为每个交换机额外安装控制链路，同时控制器部署在交换机位置处，则总体部署代价为各控制器的部署代价之和。为了简化起见，假设部署中使用相同类型的控制器，部署每台控制器的花费为 *price*，总共需部署 $|CV|$台，则总体的部署代价为

$$TPC = price\left|CV\right| \tag{9}$$

定义 2　平均控制策略查询时延（ACPRD, average control policy request delay）。当未知的新流进入交换机时，交换机需要查询控制器以获得该流的转发策略，从而产生控制策略查询时延。控制策略查询时延（CPRD, control policy request delay）是交换机发送 Packet-In 消息到本区域控制器，在控制器处理完毕后将控制信息返回交换机的各部分时延的总和。平均控制策略查询时延 ACPRD 是全体交换机控制策略查询时延的平均值。任意选择 2 个节点 i 和 j，若其中节点 i 作为交换机，节点 j 作为 i 的主控制器。设节点 i 到 j 的最短路径为 $p_{(i,\ j)}$，路径 $p_{(i,\ j)}$包括若干段子链路，用<u,v>表示其中的任一段子链路，则该子链路的传播时延为 $d_{<u,v>}$。设交换机设备数据分组排队加转发的平均时延为 t_f，控制器对每个请求的平均处理时延为 t_c，则控制策略查询时延为路径上各段子链路时延总和的 2 倍加上控制器对请求的处理时间。则交换机 i 的控制策略查询时延表示为

$$CPRD_{i,j}=2\sum_{<u,v>\in p_{(i,j)}}(d_{<u,v>}+t_f)+t_c \quad (10)$$

在 *CPRD* 的求取中，$d_{<u,v>}$、t_f和 t_c均为已知量，路径 $p_{(i,j)}$为变量，与控制器集合 *CV* 与交换机所属的域集合 *SA* 有关。网络全体交换机的平均控制策略查询时延为

$$ACPRD=\frac{1}{n}\sum_{i=1,j=CV_{Labels_i}}^{n}CPRD_{i,j} \quad (11)$$

定义 3 控制器负载差异度（CLDI, controller load difference index）。控制器负载差异度表示各控制管理的交换机个数的差异程度，通过各控制器管理的实际交换机个数与整个网络各域平均的交换机个数的差值来计算

$$CLDI=\frac{1}{k}\sum_{j=1}^{k}\left|\left|SA_j\right|-\frac{n}{k}\right| \quad (12)$$

基于以上 3 个指标的定义，受时延和容量限制的多控制器均衡部署问题是找到一个合适的部署方法，使这 3 个指标达到最小，可表示为

$$\begin{aligned}&\min F(CV,SA)=(TPC,ACPRD,CLDI)^{\mathrm{T}}\\&\text{s.t. }\left|SA_j\right|<B,\quad j=1,2,\cdots,k\\&\qquad CPRD_{i,j}<T,\ \ i=1,2,\cdots,n,\ \ j=CV_{Labels_i}\end{aligned} \quad (13)$$

该问题是一个带约束的多目标组合优化问题，难以在多项式时间内求得最优解。本文考虑使用谱聚类算法来求解该问题。为了获得可用于聚类的节点数据，首先需要将网络拓扑映射为 *n* 个节点的坐标向量。

在该部署问题中，由于给定了时延和容量的限制，在保障了网络基本性能的情况下，控制器的个数与部署成本成正比，成为最优先的指标。在部署控制器时，先假设需要 *k* 个控制器，由每个控制器负责一个特定的区域。用谱聚类算法将该图划分成 *k* 个子图，并求出子图中心作为部署位置。若聚类出来的结果能满足域内最大容忍时延和控制器容量的约束条件，再尝试更小的 *k* 值，否则尝试更大的 *k* 值，最终确定合理的 *k* 值。

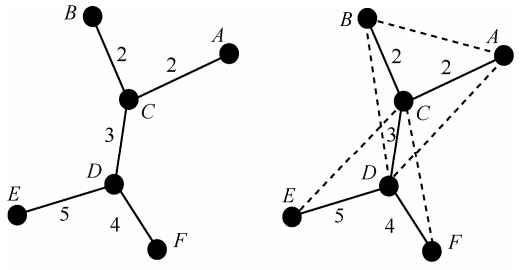
为了实现对网络合理地划分，在对图上的节点进行聚类时，原则上将相互靠近以及通信时延小的节点聚为一类。根据时延 *CPRD* 对图上的节点关系做一个变换，再对变换后的图做聚类使聚类结果在效果上达到同类间 *CPRD* 尽可能小，不同类间的 *CPRD* 尽可能大。在这里，本文引入一个称为相似度的度量值来计算节点间的相似程度，相似程度高的节点将被聚为同一类。

定义 4 节点相似度。节点相似度是对于任意 2 个交换机节点 *i* 和 *j*，判断其相似程度，衡量是否将这两点划分为同一区域的度量指标。该值越大，越倾向于将这两点划分为同一区域。

在对图做划分之前，先对无向图 *G* 做一个变换形成相似度图 $G'(V,E',\boldsymbol{W})$，新图中保持节点不变，但引入新的边集 *E'*和新边的权值矩阵 $\boldsymbol{W}$。图的转换规则为：对图上任意 2 个节点 *i* 和 *j*，考虑若将其中一个节点作为另一个节点的主控制器，计算这两点间的最短路径，并根据式(10)计算两点间的时延 $CPRD_{i,j}$，若该两点为非相邻节点，且时延小于可容忍的最大时延 *T*，则为这两点增加一条虚拟边。图 2 给出了一个示例。当虚拟边构建完毕后，对图上的所有的虚拟边和原有边(i, j)根据时延 $CPRD_{i,j}$ 来计算相似度 $w_{i,j}$。

$$w_{i,j}=\begin{cases}\dfrac{1}{CPRD_{i,j}}, & (i,j)\in E'\\ w_{i,j}=0, & (i,j)\notin E'\end{cases} \quad (14)$$

可以看出，当两点间查询时延越小，则相似度越大。相似度越大，则该两点越倾向于归为一类。



(a) 图G (b) 图G'

图 2 图 *G* 变换到图 *G'*，设最大延迟为 8，对距离小于 8 的节点对增加虚拟边

在图的划分中，根据相似度指标将相似度图划分成 *k* 个区域 $R_1,\cdots,R_k$，某区域与其他区域之间连接边的权值之和用割来表示，则整个图的割为

$$Cut_k(R_1,\cdots,R_k)=\frac{1}{2}\sum_{i=1}^{k}assoc(R_i,\overline{R_i}) \quad (15)$$

其中，*assoc* 表示任一区域连接其他区域边的权值之和。这里的$\frac{1}{2}$是为了防止每条边被计算 2 次。该图划分的原则是使相似度大的节点归为一类，不同

类之间的节点相似度尽可能小。由于式(14)中相似度与查询时延成反比关系，因而相互间时延小的节点将归为同一类，时延大的节点分为不同类。为了能调整区域的大小，本文采用归一化的割，通过在原有割上除以同一域中各边的权值之和来实现。

$$NCut(R_1,\cdots,R_k)=\frac{1}{2}\sum_{i=1}^{k}\frac{assoc(R_i,\overline{R_i})}{WS(R_i)}=\sum_{i=1}^{k}\frac{Cut_2(R_i,\overline{R_i})}{WS(R_i)} \tag{16}$$

这里，$WS(R_i)$为 R_i 内部所有边的权值的和。由于 $WS(R_i)$相当于同一类节点间的相似度的总和，因而 $WS(R_i)$应该尽可能大，而 Cut_2 表示 2 个区域间的相似度，应尽可能小。因整个图中 WS 的总和是定值，当 $R_1,\cdots,R_k$ 划分得不均衡时，例如某个区域只含很少的节点，则该类的 $WS(R_i)$会特别小，导致总的 $NCut$ 值变大。当 $WS(R_i)$趋向均匀时，$NCut$ 值会变小。因而该问题转变为求解优化问题。

$$opt=\min_{R_1,\cdots,R_k} NCut(R_1,\cdots,R_k) \tag{17}$$

注意到 $\boldsymbol{W}$ 是相似度矩阵，是一个对称矩阵。因此，引入拉普拉斯矩阵 $\boldsymbol{L}$。

$$\boldsymbol{L}=\boldsymbol{M}-\boldsymbol{W} \tag{18}$$

此处 $\boldsymbol{M}$ 是一个对角矩阵，元素值为 $\boldsymbol{W}$ 的行和，有

$$M_{i,i}=\sum_{j=1}^{n}w_{i,j},\qquad i=1,\cdots,n \tag{19}$$

$$WS(R_j)=\sum_{i\in R_j}M_{i,i},\quad j=1,\cdots,k \tag{20}$$

为了对网络节点进行有效的聚类，需要为每个节点赋予一个向量来指示该节点的类别特征。设该向量为 $\boldsymbol{h}$，长度为 k，n 个节点的向量堆叠可组成一个矩阵 $\boldsymbol{H}$。参照谱聚类的方法[36]，定义

$$h_{ij}=\begin{cases}\dfrac{1}{\sqrt{\sum_{i\in R_j}M_{ii}}},v_i\in R_j,i=1,\cdots,n\\ 0,v_i\notin R_j,j=1,\cdots,k\end{cases} \tag{21}$$

由于 $\boldsymbol{M}$ 是对角阵，可推导得出

$$\boldsymbol{h}_i'\boldsymbol{M}\boldsymbol{h}_i=1 \tag{22}$$

$$\boldsymbol{h}_i'\boldsymbol{L}\boldsymbol{h}_i=\frac{Cut_2(R_i,\overline{R_i})}{WS(R_i)} \tag{23}$$

则有

$$\begin{aligned}NCut(R_1,\cdots,R_k)&=\sum_{i=1}^{k}\boldsymbol{h}_i'\boldsymbol{L}\boldsymbol{h}_i\\&=\sum_{i=1}^{k}(\boldsymbol{H}'\boldsymbol{L}\boldsymbol{H})_{ii}\\&=Tr(\boldsymbol{H}'\boldsymbol{L}\boldsymbol{H})\end{aligned} \tag{24}$$

则原问题转化为

$$\begin{aligned}&opt=\min_{R_1,\cdots,R_k}Tr(\boldsymbol{H}'\boldsymbol{L}\boldsymbol{H})\\&\text{s.t. }\boldsymbol{H}'\boldsymbol{M}\boldsymbol{H}=I_k\end{aligned} \tag{25}$$

由于 $\boldsymbol{H}$ 中向量的取值受到限制条件的影响，这个问题的求解是 NP-Hard 的，为了求解这个最小值，将其松弛，将 $\boldsymbol{h}_i$ 的取值放宽到实数范围。并定义

$$\boldsymbol{X}=\boldsymbol{M}^{\frac{1}{2}}\boldsymbol{H} \tag{26}$$

则问题松弛为

$$\begin{aligned}&opt=\min_{X\in R^{n\times k}}Tr(\boldsymbol{X}'\boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{L}\boldsymbol{M}^{\frac{1}{2}}\boldsymbol{X})\\&\text{s.t. }\boldsymbol{X}'\boldsymbol{X}=I_k\end{aligned} \tag{27}$$

令

$$\boldsymbol{A}=\boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{L}\boldsymbol{M}^{\frac{1}{2}} \tag{28}$$

则问题转化为

$$\begin{aligned}&opt=\min_{X\in R^{n\times k}}(Tr(\boldsymbol{X}'\boldsymbol{A}\boldsymbol{X}))\\&\text{s.t. }\boldsymbol{X}'\boldsymbol{X}=I_k\end{aligned} \tag{29}$$

由于 $\boldsymbol{L}$ 和 $\boldsymbol{M}$ 是对称的，因而 $\boldsymbol{A}$ 也是对称的，在实数范围内，由 Rayleigh-Ritz 定理可知，对称矩阵具有这样的特性。

若 $\boldsymbol{A}_{n\times n}$ 为实对称矩阵，其特征值为 $\lambda_1\leqslant\cdots\leqslant\lambda_n$ 且其对应的特征向量 $\boldsymbol{v}_1,\cdots,\boldsymbol{v}_n$ 正交，则

$$\min\{Tr(\boldsymbol{X}'\boldsymbol{A}\boldsymbol{X}):\boldsymbol{X}(n\times k)real,\boldsymbol{X}'\boldsymbol{X}=\boldsymbol{I}_k\}=\lambda_1+\cdots+\lambda_k \tag{30}$$

其最小值对应的矩阵为 $\boldsymbol{X}=[\boldsymbol{v}_1,\cdots,\boldsymbol{v}_k]$。

因此，最小实数解出现在矩阵 $\boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{L}\boldsymbol{M}^{\frac{1}{2}}$ 前 k 个最小特征根对应的特征向量组成的矩阵 $\boldsymbol{X}$ 里。此时划分的指示矩阵 $\boldsymbol{H}$ 为

$$\boldsymbol{H}=\boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{X} \tag{31}$$

这里，$\boldsymbol{H}$ 矩阵由 n 个 k 维行向量组成，其分别表示 n 个节点的 k 维坐标。每个节点的 k 维坐标分别代表该节点的 k 种属性，越相似的节点其 k 维属

性越相近。将在 k 维坐标系中将距离相近的节点归为一类，下面本文用一个均衡型的 k-means 算法来改进谱聚类中原有的 k-means 算法，实现均衡的谱聚类，来计算这些节点的域划分。

4 控制器部署算法

在经典的 k-means 算法中，每个节点都由一个多维向量表示。要将这些节点划分为 k 个类，首先可随机选出 k 个初始点作为类中心，分别计算各节点到这些类中心的距离，然后将各节点归入最近的类中心，当所有的节点都分类完成后，再次在各个类中计算该类新的中心位置，此为一轮迭代。在接下来的每一轮迭代中，都重新计算各点到新的类中心的距离，再将节点归入最近的类中心。每一轮迭代都使得各类中内部节点的距离变得更小。经过多轮迭代后，若后一轮迭代的分类结果与上一轮迭代的结果一致，即各节点的分类情况不再发生变化时，则计算结束。在经典的 k-means 算法中，其分类目标是为了使类内部的距离变小，而没有考虑各个类的节点数量。为了实现均衡的部署，本文提出一个 k 控制器均衡部署（kCBP, k controllers balanced placement）算法来求取在给定 k 值的情况下如何进行控制器的部署和交换机的分配。在本文算法中，需要同时考虑类内部节点间的距离和各类节点的数量。为此，在每一轮迭代结束时，加入一次节点类别的调整。设 SA_j 为第 j 个域的节点集合，x_i 为第 i 节点的坐标，第 i 节点所在类的中心坐标为 xc_{Label_i}，根据时延最小和负载均衡的指标函数，这里引入一个节点调整的目标函数为

$$obj=\frac{\alpha}{k}\sum_{j=1}^{k}\left(\left|\,|SA_j|-\frac{n}{k}\right|\right)+\frac{1-\alpha}{n}\mathrm{punIsol}(G)\sum_{i=1,j=Label_i}^{n}\left\|x_i-xc_j\right\|^2 \tag{32}$$

目标函数 obj 中的参数 $\alpha\in[0,1]$。节点调整的目的是为了使目标函数值降低。目标函数的等式右边包括 2 个部分。函数 punIsol 为一个惩罚函数，对孤立节点进行惩罚。函数通过在每个类中从一个节点开始，将邻居的同类节点着色，染色完成后，若存在未被染色的节点，则说明当前图中存在孤立节点，此时 punIsol 函数将返回一个系数使整体时延加倍，本文中该系数取值为 2 倍，若没有孤立节点则函数返回 1。

$$\mathrm{punIsol}(G)=\begin{cases}2, \text{若}G\text{存在孤立点}\\ 1, \text{若}G\text{不存在孤立点}\end{cases} \tag{33}$$

式(32)右侧第一部分优化负载均衡，其负责各类大小的差异度计算。因总节点数是定值，若各个类的节点数目越接近平均值，则第一部分的值越小。第二部分优化最小时延，其负责各节点到类中心的距离，若各节点到类中心的距离越近，则第二部分的值越小。但若出现孤立点，受惩罚函数的影响，则该部分值会变大。两部分的比重用参数 α 调节，当 α 增大时，目标函数偏重于注重均衡度，当 α 减小时，目标函数偏重于注重查询时延。

4.1 k 控制器均衡部署 KCBP 算法

本文改造原有的 k-means 算法，在每轮迭代中计算完各节点的分类情况后，在计算下一轮的类中心之前，增加一个基于模拟退火算法的调整步骤。调整时，首先对各节点使用模拟退火方法进行调整。首先记录当前分类作为最好的结果，再将类中某个节点尝试归入某个邻居不同类节点的类，并计算目标函数，若调整后使目标函数值降低，则保留调整结果作为新的分类并更新最好结果，若调整后使目标函数值升高，则以某概率值决定保留原样还是采用新类。同时，按退火方式计算下一次的概率值。经过连续多次退火后，结束调整，以最好结果作为该轮迭代结果，再进入下一轮迭代。在迭代多次后，若连续 2 次分类结果不再变化则总体计算结束，输出分类结果。当求得各节点类别后对每个类将离类中心坐标距离最近的节点作为该类的控制器位置。

由于文本的算法在原有的 k-means 算法上在每轮迭代中增加了一次类别调整。且调整的目标是 obj 所示的函数。因此，每轮迭代的结果不仅使类内部节点的距离变小，而是综合考虑了类内部节点的距离变小和各类的大小均衡 2 个方面的因素。而类内部节点的距离变小和各类的大小均衡 2 个因素之间的比重由参数 α 来决定。

k 控制器均衡部署算法用于在给定区域个数 k 时，如何在已有的拓扑上进行划分，算法如下。

算法 1 k 控制器均衡部署 KCBP 算法

输入 图 $G(V, E, D)$，最大时延 T, 控制器数 k，系数 α

输出 控制器位置集 S, 交换机分配指示向量 ***Labels***

1) $\boldsymbol{PATH} \leftarrow Floyd\text{-}Warshall(G)$
2) for each $i,j \in V$ do
3) $CPRD_{i,j} \leftarrow Eq(10)$
4) if $CPRD_{i,j} < T$ then $w_{i,j} \leftarrow Eq(14)$
5) end for
6) $\boldsymbol{M} \leftarrow Eq(19)$
7) $\boldsymbol{L} \leftarrow \boldsymbol{M}-\boldsymbol{W}$
8) $A \leftarrow \boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{L}\boldsymbol{M}^{\frac{1}{2}}$
9) $(eigVals, eigVects) \leftarrow eig(A)$
10) $\boldsymbol{X} \leftarrow k$ *eigVects* for smallest *eigVals*
11) $\boldsymbol{H} \leftarrow \boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{X}$
12) $\boldsymbol{XC} \leftarrow InitKCenters(\boldsymbol{H}, k)$
13) for each $i \in V$ do $Labels_i \leftarrow$ argmin ($Distance$ (i, $\boldsymbol{XC}$))
14) $\boldsymbol{lastLabels} \leftarrow \boldsymbol{Labels}$
15) while (true) do
16) for each $i \in V$ do $Labels_i \leftarrow$ argmin ($Distance$ (i, $\boldsymbol{XC}$))
17) $\boldsymbol{newLabels} \leftarrow \boldsymbol{bestLabels} \leftarrow \boldsymbol{Labels}$
18) $SA \leftarrow getAssigns(\boldsymbol{Labels})$
19) $value \leftarrow obj(\boldsymbol{Labels})$; $fk \leftarrow 0$; $tk \leftarrow t0$,
20) while $tk > ts$ do
21) for h=1 to inL do
22) $i \leftarrow$ random select a node from V
23) $j \leftarrow$ random select a node from $neighbors(i)$
24) $newLabels_i \leftarrow newLabels_j$
25) if $obj(\boldsymbol{newLabels}) < obj(\boldsymbol{Labels})$ then
26) $\boldsymbol{Labels} \leftarrow \boldsymbol{newLabels}$
27) if $obj(\boldsymbol{Labels}) < obj$ ($\boldsymbol{bestLabels}$)
28) $\boldsymbol{bestLabels} \leftarrow \boldsymbol{Labels}$
29) continue
30) end if
31) $P \leftarrow \mathrm{e}^{-\frac{obj(\boldsymbol{newLabels})-obj(\boldsymbol{Labels})}{tk}}$
32) if random(0,1) < P then $\boldsymbol{Labels} \leftarrow \boldsymbol{newLabels}$
33) end for
34) $tk \leftarrow \dfrac{t0}{I+fk}$; $fk \leftarrow fk+1$
35) end while
36) $SA \leftarrow getAssigns(\boldsymbol{bestLabels})$
37) if $\boldsymbol{lastLabels} == \boldsymbol{bestLabels}$ then break
38) $\boldsymbol{lastLabels} \leftarrow \boldsymbol{bestLabels}$
39) for each $i \in (1,\cdots,k)$ do $XC_i \leftarrow calcCenter(H_j \mid j \in SA_i)$
40) for each $i \in V$ do $Labels_i \leftarrow$ argmin ($Distance$ (i, $\boldsymbol{XC}$))
41) end while
42) for each $i \in (1,\cdots,k)$ do
43) $CV_i \leftarrow$ argmin($Distance(XC_i, SA_i)$
44) return (CV, $\boldsymbol{Labels}$)

在 KCBP 算法中，*Floyd-Warshall* 函数返回矩阵 $\boldsymbol{PATH}$，该矩阵指示了图中从任一节点到另一节点最短路径的下一跳。第 3)到第 6)行的 $Eq(10)$、$Eq(14)$和 $Eq(19)$分别表示式(10)、式(14)和式(19)。第 9)行的函数 $eig()$为对矩阵进行特征分解得到 n 个 n 维向量。第 10)行选出其中特征值最小的 k 个向量组成一个 $n \times k$ 矩阵 $\boldsymbol{X}$。第 12 行的 *InitKCenters* 函数是从 $\boldsymbol{H}$ 矩阵中选出 k 行组成 $k \times k$ 矩阵 $\boldsymbol{XC}$，作为初始类中心的坐标。第 14)行的 *Distance* 函数是求解一个节点到各中心的距离。第 16)行的 *getAssigns* 函数进行从 $\boldsymbol{Labels}$ 到 SA 集合的转换。从第 20)行到第 35)行是基于模拟退火的类别调整。第 39)行的 *calcCenter* 函数从 $\boldsymbol{H}$ 中选出属于各类节点的行向量进行中心点坐标的计算。在 k-means 的算法中初始点的选择对最终结果有着较大的影响，过于接近的初始点会使计算容易陷入局部最优而导致聚类结果变差，因此初始的类中心应尽量的分散。本文采用的初始点选择方法是重复随机选取多次，并计算每次各初始点相互距离的乘积，取乘积最大的一次作为初始中心，该方法可以获得分散度较好的初始点集合。

定理 1　KCBP 算法是收敛的。

证明　在KCBP算法中每轮迭代都会计算$obj()$的函数值。KCBP 算法包括 2 部分：计算类中心和类别划分部分。假设当前轮迭代 *obj* 函数没有达到最小，则下一轮迭代中首先可以在保证 *obj* 函数第一部分不变的同时优化第二部分，在 x_i 和分类不变的情况下重新计算 xc_j，使第二部分比原来更小。接下来，在固定 xc_j 的情况下，节点先根据最小距离分类，再采用拟退火算法进行类别调整。由于模拟退火算法能绕过局部最优寻找当前 xc_j 下全局最优的能力，则调整结束时，*obj* 函数值与调整前的 *obj*

函数值相比将相等或更小。因此，在整个迭代过程中，*obj* 函数是单调递减的。对于确定的节点数 n，节点的组合是有限的，因而 *obj* 函数只可能存在有限个值。因此，KCBP 算法将在有限步内收敛于一个定值。证毕。

定理 2 设 n 为节点个数，t 为 k-means 迭代次数，dm 为节点最大度值，$outL$ 和 inL 为模拟退火算法的外层和内层循环次数，则 KCBP 算法的时间复杂度为 $O(n^3+t\cdot dm\cdot outL\cdot inL\cdot n)$。

证明 KCBP 算法在开始用 *Floyd-Warshall* 算法计算任意两点的最短路径，其时间复杂度为 $O(n^3)$；计算矩阵 $\boldsymbol{W}$、$\boldsymbol{M}$、$\boldsymbol{L}$ 的时间复杂度各为 $O(n^2)$；计算矩阵 $\boldsymbol{A}$ 的时间复杂度为 $O(n^3)$；计算特征向量的时间复杂度为 $O(n^3)$；由于前面已经计算出了最短路径，选取 k 类的初始中心时间复杂度为 $O(rk^2)$，r 为随机选取的次数；*obj* 函数的时间复杂度为 $O(2n+dmn)$；while 循环中第 18)行求解节点分配的时间复杂度为 $O(n)$，while 循环中的模拟退火调整部分时间复杂度为 $O(dm\cdot outL\cdot inLn)$，整个 k-means 部分总的时间复杂度为 $O(t(dm\cdot outL\cdot\ inL\cdot n+nk))$；最后求取控制器节点的时间复杂度为 $O(nk)$；因此，该算法总的时间复杂度为 $O(n^3+t\cdot dm\cdot outL\cdot inLn)$。

证毕。

在算法中出现了参数 α。参数 α 用于调节各控制器时延和负载差异度之间的比重，当 α 增大时，算法由注重时延向注重均衡度转变。随着 α 的增大，节点数小的域对节点数多的域产生更大的拉动作用，将其节点拉到自己域中来。

KCBP 是 k 控制器部署算法，该算法是在指定控制器个数情况（个数为 k）下求取各控制器的部署位置和各交换机的分配。但对于求解实际部署中所需的最少控制器数问题，KCBP 算法无法解决。下面提出一个均衡的控制器部署（BCP, balanced controller placement）算法，在 KCBP 的基础上，实现对最小 k 值的搜索。

4.2 均衡的控制器部署 BCP 算法

在 BCP 算法中通过向 KCBP 传入不同的控制器个数 k 值，及控制时延与均衡性比例的系数 α，多次调用了 KCBP 算法。控制器部署 BCP 算法的解包括控制器个数、各控制器的部署位置和各交换机的控制器分配 3 个部分。该算法根据时延和控制器容量的限制条件，返回在成本优先情况下，分别偏重时延最小和偏重负载最均衡情况下满足限制条件的一系列解。

因受到最大时延和最大控制器容量的限制，划分网络时应保证每个子域内交换机到控制器的时延不超过最大容忍时延 T，单控制器管理的最大交换机数不超过 B。完整的控制器部署算法 BCP 包括一个 k 值的搜索过程。若不考虑搜索的时间，可以让 k 从 1 开始逐渐增大，直到满足限制条件的解出现，则停止搜索。对每一个 k 值，若无法满足限制条件有可能是因为最大时延条件不满足也有可能是控制器容量条件不满足，因而算法对 α 进行一个调整，使在同一个 k 值下算法可以分别权衡最大时延和控制器容量来进行部署。算法的输入参数中提供了 α 的上界和下界。为了加快搜索速度，本文的算法首先根据限制条件计算 k 值的下限和上限，随后在上下限之间通过二进制搜索测试每一个 k 值，调用 KCBP 算法进行网络划分并判断在该 k 值下能否满足时延和容量要求，并缩小控制器数 k 的上下限范围，最后得到最终所需的控制器 k 值。

均衡的控制器部署算法 BCP 如下。

算法 2 均衡的控制器部署 BCP 算法

输入 图 $G(V, E, D)$，最大时延 T, 单控制器最大允许交换机数 B，控制器数 k，系数 $\alpha_{\min}$，$\alpha_{\max}$，α_{step}

输出 控制器个数，控制器位置集 CV, 交换机分配指示向量 ***Labels***

1) $minK\leftarrow\frac{n}{B}$，$maxK\leftarrow\frac{n}{B}$, $flag\leftarrow 0$，$k\leftarrow 0$，$res\leftarrow\varphi$
2) while (true) do
3) $maxK\leftarrow maxK\cdot 2$, $flag\leftarrow 0$
4) for $\alpha\leftarrow$ from $\alpha_{\min}$ to $\alpha_{\max}$ step α_{step} do
5) $(CV, \boldsymbol{Labels})\leftarrow KCBP(G, T, maxK, \alpha)$
6) $maxSwitchNum\leftarrow calcMaxSA$ (***Labels***)
7) $maxCPRD\leftarrow calcMaxCPRD(G, CV,$ ***Labels***)
8) if ($maxCPRD\leqslant T$) and ($maxSwitchNum\leqslant B$) then
9) flag $\leftarrow$ 1, break
10) end for
11) if ($flag$ = = 1) then break
12) end while
13) while ($minK\leqslant maxK$) do

14) $k \leftarrow \frac{minK + maxK}{2}$, $flag \leftarrow 0$，$res \leftarrow \phi$

15) for $\alpha \leftarrow$ from $\alpha_{\min}$ to $\alpha_{\max}$ step α_{step} do

16) $(CV, \textbf{\textit{Labels}}) \leftarrow KCBP(G, T, k, \alpha)$

17) $maxSwitchNum \leftarrow calcMaxSA$ (***Labels***)

18) $maxCPRD \leftarrow calcMaxCPRD(G, CV,$ ***Labels***)

19) if ($maxCPRD \leqslant T$) and ($maxSwitchNum \leqslant B$) then

20) $flag \leftarrow 1$, $res \leftarrow res \cup \{(k, CV,$ ***Labels***$)\}$

21) end for

22) if ($flag$ = = 1) then $maxK \leftarrow k$

23) else $minK \leftarrow k$+1

24) end while

25) return res

算法中初始的 k 值下界取值为总节点数除以单控制器的最大容量，随后以二进制指数形式寻找 k 值上界，同时下界跟随着上次未成功的上界上升。当最终上下界确定后，再以二分法在上下界之间搜索 k 值。当 k 值确定后，BCP 算法输出的是优化了最小距离和负载差异度 2 个指标并且满足限制条件的一组解。各解依次由偏重考虑时延向偏重考虑均衡性过渡。

4.3　控制器的动态均衡部署

本文的控制器部署算法可用于网络控制器的初始部署，经过简单的扩展，也可用于动态的均衡部署。设在单位时间中进入交换机 i 的新未知数据流数量为 u_i，单位时间内控制器对新流请求的处理能力为 c，则式(33)可改为式(34)。

$$obj = \frac{\alpha}{k}\sum_{j=1}^{k}(c - \sum_{i \in SA_j} u_i) + \frac{1-\alpha}{n}\text{punIsol}(G)\sum_{i=1, j=Label_i}^{n}\left\|x_i - xc_j\right\|^2 \tag{34}$$

网络系统中的控制器采用虚拟机部署，同时有一个监控模块来查询网络的状态。监控模块通过控制器的 RESTful 接口从控制器获取信息及向控制器推送命令。监控模块定期统计各控制器的实时负载 u_i，当负载超过给定的阈值并持续超过规定的时间时，将触发部署算法重新进行计算，在新位置部署控制器并向新控制器推送信息，包括拓扑知识、管辖的交换机列表、流表信息等。新控制器向管辖范围内的所有交换机发送 OFPT_ROLE_REQUEST 消息，将其在交换机中的角色从 OFPCR_ROLE_SLAVE 变更为 OFPCR_ROLE_MASTER，此时，交换机会以新控制器作为主控制器，当新流到来时，交换机将不再询问原有控制器，而是向新控制器发送请求。

尽管本文的算法可应用于控制器的动态部署，但在实际应用中，动态的控制器部署还要考虑更多的因素，如控制器动态迁移的代价等，这些内容可留待今后做进一步研究。

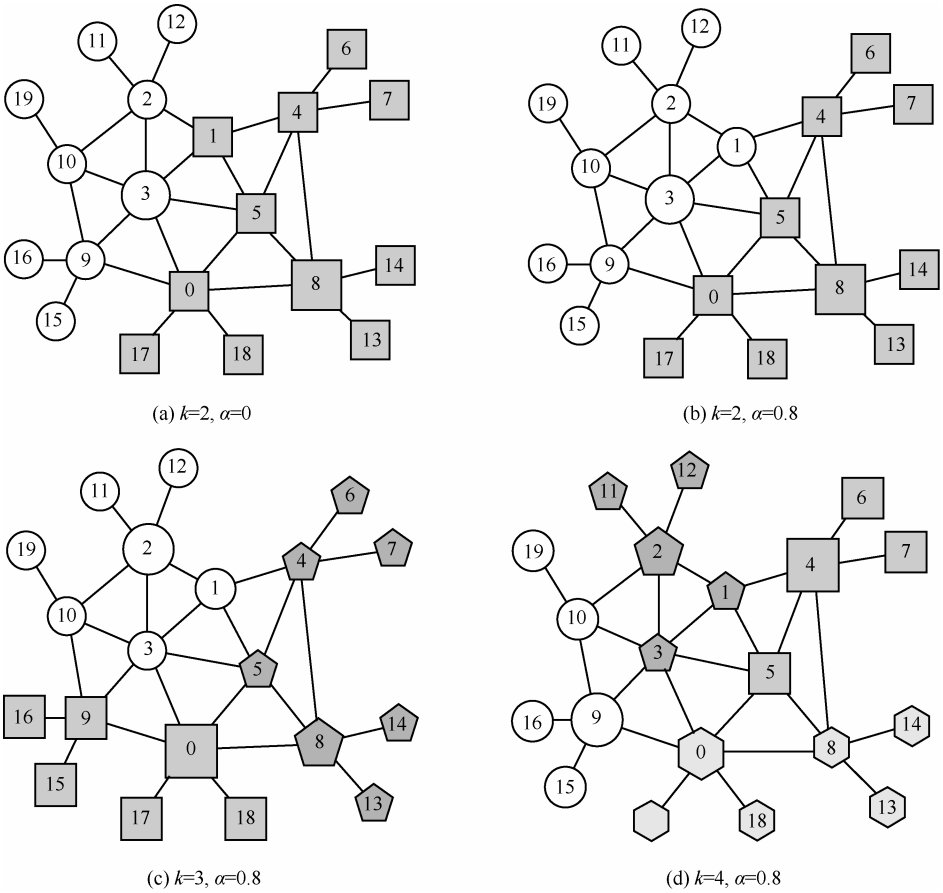
5　仿真实验

为了评估本文提出的控制器部署算法，本文分别使用了模拟网络和真实的网络拓扑来进行实验仿真。仿真程序由 Python 编译运行。

第一个仿真实验是对 KCBP 算法输入不同的 k 值以划分出 k 个子区域，观察其划分结果。该实验在一个由 20 个节点构成的模拟网络上进行。在该网络拓扑上使用控制器部署算法分别对其进行了 k=2、k=3、k=4 的仿真，网络的划分和控制器部署结果如图 3 所示。在图 3 中圆圈和多边形代表交换机节点，由 KCBP 算法划分出的不同子网由不同颜色和形状的节点表示，其中，计算出的控制器位置用大尺寸的节点表示。

在图 3(a)中，使用参数 k=2、α=0 将网络划分成 2 个部分，当 α=0 时相当于目标函数的前半部分没有起作用，此时与普通的 k-means 算法基本相同。可以看到，划分出的 2 个子网以节点 2、3、9 和节点 1、5、0 之间的边为分界，此时两域间割边最小，第一个域包括 9 个节点，第二个域包括 11 个节点。图 3(b)中使用的参数为 k=2，α=0.8。此时算法划分出的各子网大小均为 10 个节点，2 个控制器的负载均为 10 台交换机。图 3(c)和图 3(d)的参数为 k=3，α=0.8 和 k=4，α=0.8，分别将网络划分成 3 个和 4 个区域，此时划分出的子网大小均较为均匀。

第二个仿真实验为比较 BCP 算法与其他控制器部署算法在给定单控制器最大能管理的交换机数目时所需的控制器数量。该实验基于真实网络拓扑，实验中采用的网络拓扑结构为美国 Internet2 上的 Advanced Layer 2 设备拓扑，Internet2 项目组在上面部署了 SDN 设备进行实验，拓扑如图 4 所示。拓扑中链路间的时延根据节点的地理位置进行计算，设光纤中的信号传播速度为真空中光速的 $\frac{2}{3}$，



(a) k=2, α=0　　(b) k=2, α=0.8

(c) k=3, α=0.8　　(d) k=4, α=0.8

图 3　KCBP 算法在使用不同参数的部署情况

同时每 100 km 链路引入 1 ms 的光信号中继延迟。设交换机上平均的排队转发时延 t_f 为 0.1 ms，控制器的处理时延 t_c 为 0.01 ms。

图 4　Internet2 网络 Advanced Layer 2 拓扑结构

本文对比的基准算法分别为随机部署算法、基于节点度的贪心（Greedy-Degree）算法和谱聚类算法。基于节点度的贪心算法常用于复杂网络中的影响力最大化，计算方法为先取度最大的节点作为第一个控制器，分配交换机后，再从剩余节点取度最大的节点作为第二个控制器位置，依次计算直到分配完全部节点。谱聚类算法中采用分割规模差不多且割边最小的 RatioCut，该方法为文献[19]中采用的方法。随机算法为随机取第一个控制器位置，分配交换机后，再从剩余节点随机取第二个控制器位置，依次计算直到分配完全部节点，共计算 20 次求平均值。在本实验和后续实验中，$\alpha_{\min}$ 参数的取值均为 0，$\alpha_{\max}$ 参数的取值均为 1，α_{step} 的步进值均为 0.1。本实验中，给定最大允许时延为 300 ms，在给定不同的单控制器最大允许的交换机数目条件下，整个网络所需的控制器个数如图 5 所示。

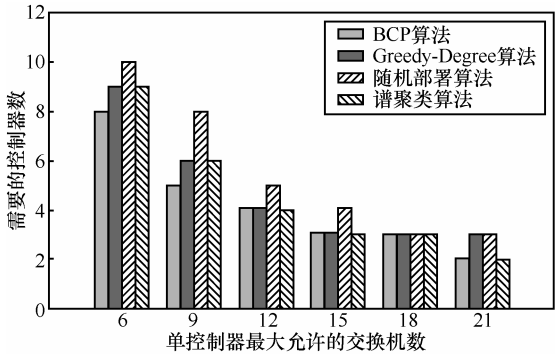


图 5　限定容量时需要的控制器数

从图 5 中可以看到，BCP 算法在同等条件下所需控制器个数在各阶段均少于其他算法。随机算法所需控制器最多。贪心算法在单控制器允许的交换机数较少时与 BCP 算法差距较大，在允许的交换机数超过 12 后与 BCP 差距不大，在超过 21 时 BCP 再次显出优势。由标准谱聚类算法得到的控制器数与 BCP 算法在大于 12 的情况下一致，允许的交换机数小于 12 时比 BCP 算法多需要一个控制器。分析其原因，由于标准谱聚类的划分结果中均衡度仍然不够理想，当允许的单个网络区域较小时，标准谱聚类结果中较大的类别超出了允许的最大容量，因而不得不再次将大类进行划分。在单台控制器容量较大时，标准谱聚类结果比贪心算法要好，其余结果两者基本一致。

第三个仿真实验为比较在网络拥有相同数量的控制器时，各算法下全网络的交换机向控制器查询的平均时延。在实验中设定允许的时延为一个较大的数，调节控制器的容量，可以模拟用指定的控制器数管理整个网络时不同算法的部署情况，然后计算交换机到控制器的平均时延，其中，随机算法的结果为计算 20 次选取平均值。由于 BCP 算法输出的是一组解，本实验选取的是其偏重时延一侧的数据。实验结果如图 6 所示，在拥有同等数量的控制器时，BCP 算法在几种算法中有最小的平均时延。标准谱聚类在控制器个数少时数据与 BCP 算法接近，但当控制器个数增大到 7 以后，时延随控制器增多而与 BCP 算法的差距拉大。贪心算法呈现不均匀的变化。随机算法总体最差。在实验中，在保持 $\alpha_{\min}$ 和 $\alpha_{\max}$ 参数不变的情况下，尝试调整 α_{step} 参数的值，将其从 0.1 调整至 0.05、0.02 以及 0.01，实验结果显示，尽管采用更小的 α_{step} 能进行更细致的搜索，但同该参数值为 0.1 时相比精度的提升有限，参数值为 0.1 已经能够较好地体现整体的走势。

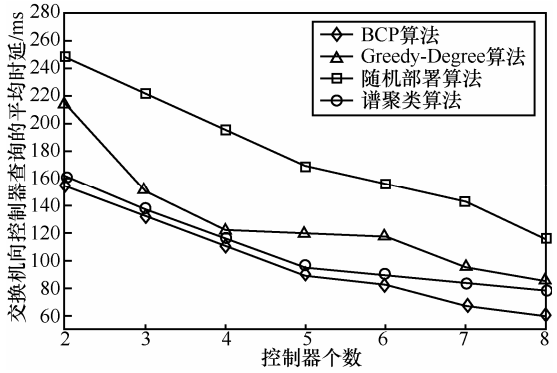


图 6　不同算法中交换机向控制器查询的平均时延

第四个仿真实验为比较不同算法下各控制器在网络运行情况下的动态负载的差别状况。假设网络中单位时间内每 2 个节点间发起新流的次数满足泊松分布，本文使用 Python 编制流量发生器来模拟新流来临时交换机对控制器的请求。实验比较使用 BCP 算法、随机部署算法、贪心算法和谱聚类算法在部署不同数目的控制器时，各控制器负载的标准差情况，该指标能较好地反映负载的均衡情况。由于 BCP 算法输出的是一组解，本实验选取的是其偏重负载均衡度一侧的数据。实验结果如图 7 所示，BCP 算法和谱聚类算法的控制器负载标准差明显低于其他算法，其中，BCP 算法又更低一些。说明谱聚类算法虽然采用 RatioCut 来划分网络，已经具有一定的均衡作用，但仍然不及经过优化 *k*-means 后的 BCP 算法。随机算法中随着控制器个数的增多其负载的标准差也渐渐下降。贪心算法的数据呈现波动性，其部分数据甚至不如随机算法，显示出贪心算法由于优先寻找度大的节点，很容易导致负载不均衡。

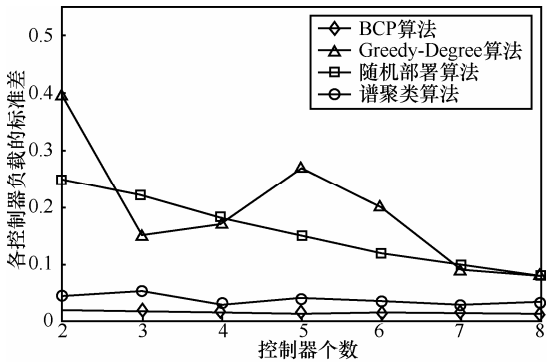


图 7　不同算法中各控制器负载的标准差

6　结束语

本文使用聚类算法来实现 SDN 网络控制器的部署。通过构造辅助拓扑图，并给出节点间相似性的度量公式，构造归一化的最小割来进行网络的划分。本文提出了一种基于谱聚类的 SDN 控制器部署算法，该算法改进了谱聚类中的 *k*-means，保证了域内的连通性，实现了时延和容量限制下的均衡部署。通过在模拟网络和真实因特网拓扑上进行仿真，对比了基于度的贪心算法和标准谱聚类算法，实验结果表明本文提出的控制器部署算法具有节省控制器、时延小、均衡度高的优点。

参考文献：

[1] FARHADY H, LEE H Y, NAKAO A. Software-defined networking: a survey[J]. Computer Networks, 2015, 81: 79-95.

[2] CURTIS A R, MOGUL J C, TOURRILHES J, et al. DevoFlow: scaling flow management for high-performance networks[J]. ACM SIGCOMM Computer Communication Review, 2011, 41(4): 254-265.

[3] YU M, REXFORD J, FREEDMAN M J, et al. Scalable flow-based networking with DIFANE[J]. ACM SIGCOMM Computer Communication Review, 2010, 40(4): 351-362.

[4] KOPONEN T, CASADO M, GUDE N, et al. Onix: a distributed control platform for large-scale production networks[C]//OSDI. 2010, 10: 1-6.

[5] TOOTOONCHIAN A, GANJALI Y. Hyperflow: a distributed control plane for OpenFlow[C]//Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking. 2010: 3-3.

[6] HASSAS Y S, GANJALI Y. Kandoo: a framework for efficient and scalable offloading of control applications[C]//Proceedings of the First Workshop on Hot Topics in Software Defined Networks. ACM, 2012: 19-24.

[7] TOOTOONCHIAN A, GORBUNOV S, GANJALI Y, et al. On controller performance in software-defined networks[C]//Presented as part of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services. 2012.

[8] FERNANDEZ M. Evaluating OpenFlow controller paradigms[C]// ICN 2013, The Twelfth International Conference on Networks. 2013: 151-157.

[9] TAVAKOLI A, CASADO M, KOPONEN T, et al. Applying NOX to the datacenter[C]//HotNets. 2009.

[10] HELLER B, SHERWOOD R, MCKEOWN N. The controller placement problem[C]//Proceedings of the First Workshop on Hot Topics in Software Defined Networks. ACM, 2012: 7-12.

[11] SALLAHI A, ST-HILAIRE M. Optimal model for the controller placement problem in software defined networks[J]. Communications Letters, IEEE, 2015, 19(1): 30-33.

[12] ISHIGAKI G, SHINOMIYA N. Controller placement algorithm to alleviate burdens on communication nodes[C]//2016 International Conference on Computing, Networking and Communications (ICNC). IEEE, 2016: 1-5.

[13] JIMÉNEZ Y, CERVELLÓ-PASTOR C, GARCIA A J. On the controller placement for designing a distributed SDN control layer[C]// Networking Conference, 2014 IFIP. IEEE, 2014: 1-9.

[14] WANG G, ZHAO Y, HUANG J, et al. A *k*-means-based network partition algorithm for controller placement in software defined network[C]//2016 IEEE International Conference on Communications (ICC). IEEE, 2016: 1-6.

[15] CHENG T Y, WANG M, JIA X. QoS-guaranteed controller placement in SDN[C]//2015 IEEE Global Communications Conference (GLOBECOM). IEEE, 2015: 1-6.

[16] YAO G, BI J, LI Y, et al. On the capacitated controller placement problem in software defined networks[J]. Communications Letters, IEEE, 2014, 18(8): 1339-1342.

[17] GAO C, WANG H, ZHU F, et al. A particle swarm optimization algorithm for controller placement problem in software defined network[C]//International Conference on Algorithms and Architectures for Parallel Processing. Springer International Publishing, 2015: 44-54.

[18] LIU S, WANG H, YI S, et al. NCPSO: a solution of the controller placement problem in software defined networks[C]//International Conference on Algorithms and Architectures for Parallel Processing. Springer International Publishing, 2015: 213-225.

[19] XIAO P, QU W, QI H, et al. The SDN controller placement problem for WAN[C]//Communications in China (ICCC), 2014 IEEE/CIC International Conference on IEEE. 2014: 220-224.

[20] BEHESHTI N, ZHANG Y. Fast failover for control traffic in software-defined networks[C]//Global Communications Conference (GLOBECOM), 2012 IEEE. IEEE, 2012: 2665-2670.

[21] MÜLLER L F, OLIVEIRA R R, LUIZELLI M C, et al. Survivor: an enhanced controller placement strategy for improving SDN survivability[C]//2014 IEEE Global Communications Conference. IEEE, 2014: 1909-1915.

[22] HU Y, WANG W, GONG X, et al. On the placement of controllers in software-defined networks[J]. Journal of China Universities of Posts and Telecommunications, 2012, 19: 92-171.

[23] HU Y, WENDONG W, GONG X, et al. Reliability-aware controller placement for software-defined networks[C]//2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013). IEEE, 2013: 672-675.

[24] HU Y, WANG W, GONG X, et al. On reliability-optimized controller placement for software-defined networks[J]. Communications, China, 2014, 11(2): 38-54.

[25] GUO M, BHATTACHARYA P. Controller placement for improving resilience of software-defined networks[C]//2013 Fourth International Conference on Networking and Distributed Computing. IEEE, 2013: 23-27.

[26] GUO S, YANG S, LI Q, et al. Towards controller placement for robust software-defined networks[C]//2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC). IEEE, 2015: 1-8.

[27] HOCK D, HARTMANN M, GEBERT S, et al. Pareto-optimal resilient controller placement in SDN-based core networks[C]//Teletraffic Congress (ITC), 2013 25th International. 2013: 1-9.

[28] HOCK D, GEBERT S, HARTMANN M, et al. POCO-framework for Pareto-optimal resilient controller placement in SDN-based core networks[C]//Network Operations and Management Symposium (NOMS), 2014 IEEE. 2014: 1-2.

[29] HOCK D, HARTMANN M, GEBERT S, et al. POCO-PLC: enabling dynamic pareto-optimal resilient controller placement in SDN networks[C]//Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on IEEE. 2014: 115-116.

[30] LANGE S, GEBERT S, ZINNER T, et al. Heuristic approaches to the controller placement problem in large scale SDN networks[J]. Network and Service Management, IEEE Transactions. 2015, 12(1): 4-17.

[31] AHMADI V, JALILI A, KHORRAMIZADEH S M, et al. A hybrid NSGA-II for solving multiobjective controller placement in SDN[C]//2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI). 2015: 663-669.

[32] JALILI A, AHMADI V, KESHTGARI M, et al. Controller placement in software-defined WAN using multi objective genetic algorithm[C]//2015 2nd International Conference on Knowledge-Based

Engineering and Innovation (KBEI). 2015: 656-662.

[33] YAO L, HONG P, ZHANG W, et al. Controller placement and flow based dynamic management problem towards SDN[C]// Communication Workshop (ICCW), 2015 IEEE International Conference. 2015: 363-368.

[34] BARI M F, ROY A R, CHOWDHURY S R, et al. Dynamic controller provisioning in software defined networks[C]//Network and Service Management (CNSM), 2013 9th International Conference. 2013: 18-25.

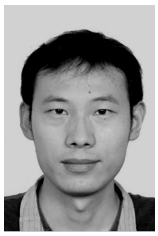
[35] RATH H K, REVOORI V, NADAF S M, et al. Optimal controller placement in software defined networks (SDN) using a non-zero-sum game[C]//World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on IEEE. 2014: 1-6.

[36] VON LUXBURG U. A tutorial on spectral clustering[J]. Statistics and Computing, 2007, 17(4): 395-416.

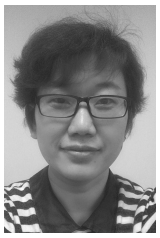
作者简介：

覃匡宇（1974-），男，壮族，广西马山人，武汉大学博士生，主要研究方向为软件定义网络（SDN）、网络管理。

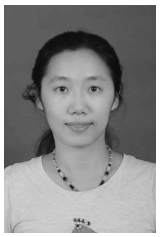
黄传河（1963-），男，湖北随州人，武汉大学教授、博士生导师，主要研究方向为计算机网络(移动互联网、移动 ad hoc 网络、无线传感器网络、未来互联网)、物联网、网络安全、高性能计算。



王才华（1987-），男，湖北咸宁人，武汉大学博士生，主要研究方向为机器学习、生物信息学。



史姣丽（1979-），女，山西运城人，武汉大学博士生，九江学院副教授，主要研究方向为计算机视觉、网络安全。



吴笛（1987-），女，湖北天门人，武汉大学博士生，主要研究方向为无线传感网、压缩感知、数据降维。

陈希（1988-），男，江苏南通人，武汉大学博士生，主要研究方向为无线网络 MAC 协议设计、协议优化等。