文章编号: 1671-6833(2021)03-0019-07

# 基于时延和负载均衡的多控制器部署策略

刘振鹏[1,2], 王鑫鹏[1], 李 明[1], 任少松[1], 李小菲[2]

(1.河北大学 电子信息工程学院,河北 保定 071002; 2.河北大学 信息技术中心,河北 保定 071002)

摘 要: 针对在软件定义网络(SDN)中多个控制器部署时面对的时间延迟和负载均衡问题,在保证负载均衡的基础上以降低整个网络的时间延迟和提高整个网络性能为目标,提出一种多控制器部署算法。针对传统粒子群算法收敛速度慢的缺点,采用改进的粒子群算法对 SDN 控制器进行部署,在考虑平衡控制器负载能力的同时,最小化交换机和控制器之间的传播时延。仿真实验结果表明: 采用改进的粒子群算法,可以在保证较高的负载均衡性能的基础上,以较低的时间延迟代价获得较优的网络总体性能,网络的合适度达到 0.05; 与传统粒子群算法相比,改进的粒子群算法在种群的收敛速度上提升约 6.3%。

关键词: 软件定义网络; 控制器部署,时间延迟; 负载均衡; 粒子群优化

中图分类号: TP393 文献标志码: A doi: 10.13705/j.issn.1671-6833.2021.03.004

## 0 引言

随着网络规模的增大,如何优化网络性能成为至关重要的问题。软件定义网络(software define network,SDN)是近年来新兴的一种网络结构,它将网络的控制层和数据层解耦,利用集中式的控制器对底层设备进行可编程化管理,进而实现对网络资源的灵活调配[1]。然而,随着大数据时代的到来,在大范围的 SDN 网络中部署健壮的控制平面仍然是一个主要的挑战。为了解决这些问题,引入了一个逻辑上集中、物理上分布的控制平面[2]。这种类型的控制体系结构通常由多个控制器组成,这些控制器之间彼此相互通信,Kandoo[3]就是这种类型的一个例子。但在引入这一体系的同时,也出现了多个控制器如何部署的问题。

目前,已经有很多对 SDN 控制器部署研究的文献。文献[4]首次提出控制器部署问题是一个 NP 问题,以平均时延和最大时延为指标,选择控制器位置。文献[5-8]提出了几种基于群体的智能优化算法,通过考虑时延作为单一目标参数来求解控制器最优部署位置。以上文献在研究控制器时延问题上均取得不错的效果,但在实际 SDN 环境下,控制器的负载能力也是一个不可忽略的因素。文献[9]提出了基于控制器的负载平衡因子,在已知控制器数量的情况下,通过最小化负载平衡因子和总流请求代价来获得特定控制器的位置。文献[10]为了保证控制器负载均衡和相对较低的时延,提出了一种基于层次 $K$-means 算法的多控制器部署方案,但该方案只考虑控制器子域如何分配交换机数量,在单个控制器的负载能力上未作考虑。文献[11]提出一个控制器位置-分配模型,引入公平负载分配函数来平衡控制器负载,并减少控制器之间的时延,但该方案未对整个网络总体性能进行分析。

为此,本文综合考虑 SDN 环境下控制器的时延和负载问题,提出控制器部署算法(MCP),并用改进的粒子群算法(APSO)和 MCP 算法结合的 APSO_MCP 算法解决多控制器部署问题。

## 1 控制器部署模型

### 1.1 问题描述

在 SDN 架构中,控制平面的控制器作为决策和信息收集的角色管理着数据平面的交换机,交换机与控制器之间的请求时延和处理时间影响了 SDN 网络的整体性能。数据包到达控制器和完

收稿日期: 2020-10-15; 修订日期: 2020-12-09

基金项目: 河北省自然科学基金资助项目(F2019201427); 教育部“云数融合 科教创新”基金资助项目(2017A20004)

作者简介: 刘振鹏(1966—),男,河北保定人,河北大学教授,博士,主要从事网络信息安全、隐私保护、软件定义网络等研究,E-mail: lzp@hbu.edu.cn。

成处理返回的时间由2个因素共同决定,分别是交换机与控制器之间的距离和控制器管理的交换机数量。交换机与控制器之间的距离决定了传播时延;而控制器管理的交换机数量决定了控制器的负载容量,即决定了流量转发请求处理的时间。

时间延迟是影响网络性能的一个重要因素。在SDN网络中,时间延迟包括4个方面:传输、传播、排队和处理时延。对广域网而言,在控制器不过载的前提下,传播时延远超过其他时延,其他3种时延影响很小,可以忽略[12]。因此,在考虑时间延迟因素时选择控制器和交换机之间的传播时延。

负载是控制器的重要指标。如果控制器负载过重就会导致其无法掌握实时的网络状态,造成网络性能瓶颈。因此,将SDN的广域网划分为多个子域的网络,合理分配各个子域内的交换机数量,可以提高控制器的服务质量,并且还可以限制广播风暴。

因此,本文考虑在平衡控制器负载能力的同时,最小化交换机和控制器之间的传播时延。

### 1.2 系统模型

用无向图 $G(V,E)$ 来表示物理网络拓扑,$V$ 和 $E$ 分别表示节点集合和边的集合。$n=|V|$ 表示节点的个数。节点集合的每个节点都是交换机的位置,则有交换机集合 $S=\{s_1,s_2,\cdots,s_n\}$。假设节点集合中的每个节点都可能是控制器部署的位置,需要在网络中部署 $k(k<n)$ 个控制器,即将网络拓扑划分成 $k$ 个控制器子域 $D=\{D_1,D_2,\cdots,D_k\}$。每个控制器子域只有一个控制器,则有部署的控制器集合 $C=\{c_1,c_2,\cdots,c_k\}$。各个控制器子域的控制器相互不覆盖,则有:

$$D_p\cap D_q=\varnothing,\forall p,q\in[1,k],p\neq q。\tag{1}$$

每个交换机只能由一个控制器控制,即

$$\sum_{j\in C}\delta_{S_iC_j}=1,\forall i\in S。\tag{2}$$

其中,当交换机 $S_i$ 由控制器 $C_j$ 控制时,$\delta_{S_iC_j}=1$,否则 $\delta_{S_iC_j}=0$。

每个控制器可以同时管理多个交换机,每个交换机只在一个控制器子域内,则有:

$$\sum_{i\in S}\delta_{S_iC_j}=m,m\in[1,n),\forall j\in C;\tag{3}$$

$$\sum_{j\in C}\sum_{i\in S}\delta_{S_iC_j}=n。\tag{4}$$

式中:$m$ 为单个控制器管理的交换机个数;$n$ 为 $k$ 个控制器管理的交换机个数之和。

#### 1.2.1 时间延迟

交换机和控制器之间的时延在控制器部署问题中是一种常用的度量方式,单个交换机和控制器之间的传播时延可用式(5)表示:

$$T_{S_iC_j}=\frac{d_{S_iC_j}}{vc},\forall i\in S,j\in C。\tag{5}$$

式中:$d_{S_iC_j}$ 为交换机 $S_i$ 和控制器 $C_j$ 之间的传输距离;$vc$ 为数据在链路中的传输速度。

各个控制器子域的传播时延和整个网络的总时延可由式(6)和式(7)分别表示:

$$T_{C_j}=\sum_{i=1}^{n}T_{S_iC_j}\cdot\delta_{S_iC_j};\tag{6}$$

$$T_{\text{total}}=\sum_{j=1}^{k}T_{C_j}。\tag{7}$$

为了衡量整个网络的控制器部署在时延方面的合理性,引入传播时延标准差 $\xi_T$ 来评判各个控制器子域的传播时延与整个网络的总传播时延之间的差异,由式(8)表示:

$$\xi_T=\frac{1}{k}\cdot\sqrt{\sum_{j=1}^{k}(T_{C_j}-T_{\text{total}})^2}。\tag{8}$$

引入归一化方法以解决数据指标之间的不可比性。原始数据经过数据标准化处理后,各指标处于同一数量级,适合进行综合对比评价。$\xi_T^*$ 归一化方程如下:

$$\xi_T^*=\frac{\xi_T-\xi_{T\min}}{\xi_{T\max}-\xi_{T\min}},\xi_T^*\in[0,1]。\tag{9}$$

式中:$\xi_{T\max}$ 为最差传播时延标准差;$\xi_{T\min}$ 为最优传播时延标准差。

#### 1.2.2 负载均衡

在网络 $G(V,E)$ 中,存在 $n$ 个交换机,每个交换机的负载请求表示为 $\lambda_{S_iC_j}$,则每个控制器子域的负载请求 $\lambda_{C_j}$ 为:

$$\lambda_{C_j}=\sum_{i=1}^{n}\lambda_{S_iC_j}\cdot\delta_{S_iC_j}。\tag{10}$$

其中,每个控制器子域内的负载 $\lambda_{C_j}$ 不能大于控制器 $C_j$ 自身的最大负载能力 $\lambda_{C_{j\max}}$,即满足:

$$\lambda_{C_j}\leqslant\lambda_{C_{j\max}}。\tag{11}$$

整个网络的总负载和平均负载可由式(12)和式(13)表示:

$$L_{\text{total}}=\sum_{j=1}^{k}\lambda_{C_j};\tag{12}$$

$$L_{\text{avg}}=\frac{L_{\text{total}}}{k}。\tag{13}$$

为了衡量整个网络在负载方面的合理性,引入了负载均衡标准差 $\xi_L$ 来评判各个控制器子域

的负载与整个网络的平均负载的差异,如下所示:

$$\xi_L = \frac{1}{k} \cdot \sqrt{\sum_{j=1}^{k} \left(\lambda_{C_j} - L_{\text{avg}}\right)^2}。\tag{14}$$

负载均衡标准差 $\xi_L$ 的归一化方程 $\xi_L^*$ 如下所示:

$$\xi_L^* = \frac{\xi_L - \xi_{L\min}}{\xi_{L\max} - \xi_{L\min}}, \xi_L^* \in [0,1]。\tag{15}$$

1.2.3　控制器部署模型

针对以上以传播时延差异性和负载差异性为指标的多控制器部署问题,可以找到一个合适的部署方案,即使以上指标最小化,为此提出控制器部署模型为

$$\min F = \alpha \cdot \xi_T^* + \beta \cdot \xi_L^*,\tag{16}$$

其中,

$$\alpha + \beta = 1, 0 \leqslant \alpha, \beta \leqslant 1。\tag{17}$$

式中: $F$ 为优化目标,是整个网络的性能; 参数 $\alpha$、$\beta$ 为调节目标函数的影响权重,参数 $\alpha$ 越大,越注重网络整体的传播时延差异性,参数 $\beta$ 越大,越注重网络整体的负载差异性。

## 2　控制器部署和改进的粒子群算法

### 2.1　控制器部署算法(MCP)

为解决上述问题,根据建立的模型提出多控制器部署算法(multi-controller placement,MCP)。MCP 算法的目的是在保证整个网络较高的负载均衡性能的基础上降低各个控制器子域的传播时延,从而保证整个网络的总时延相对较低。

控制器部署算法分为 3 个阶段: ①网络拓扑数据预处理(Step 1~3); ②选取控制器部署位置(Step 4); ③控制器子域部署交换机(Step 5~18)。在网络拓扑数据预处理阶段,将无向图 $G(V,E)$ 中节点和边的数据矩阵转换成距离矩阵,并通过 Johnson 全源最短路径算法可以计算出每个节点到其他所有节点的最短距离。在选取控制器部署位置阶段,采用粒子群算法初始化控制器部署位置。在控制器子域部署交换机阶段,以距离最近原则开始构建子网(Step 5~10),当各个控制器子域管理的交换机个数相同且均等于 $\lfloor n/k \rfloor$ 时,如果网络中还有未被分配的交换机节点,对剩下的交换机节点继续按照距离最近原则进行分配,直到所有控制器节点都被分配(Step 11~18)。至此,$k$ 个负载均衡的控制器子域分配结束。

通过改进的粒子群算法初始化控制器部署位置,确定控制器管理的交换机数量 $\lfloor n/k \rfloor$,这样可以保证整个网络的负载均衡,接着选择与各个控制器距离最近的交换机进入各控制器子域,这样可以保证每个控制器子域的传播时延尽可能小。当各个控制器子域管理的交换机个数相同且均等于 $\lfloor n/k \rfloor$ 时,优先按照距离最近原则分配交换机。这样可以保证剩余的交换机和控制器间的传播时延,牺牲了部分负载均衡性能。整个过程 2 种目标互相折中,从而使得整个网络的性能达到最优。

由于整个网络系统的负载不均衡主要集中在初始分配时期,使用 MCP 算法可以更好地确定控制器的位置和各控制器子域管理的交换机,使得整个网络的初始分配时期获得一个更良好的布局,可以有效降低动态网络中多控制器部署中出现的阈值过载现象。MCP 算法的描述如下。

**算法 1**　控制器部署(MCP)。

**输入:** 网络拓扑 $G(V,E)$,控制器个数 $k$;

**输出:** 控制器部署矩阵 ***place_value***。

**Step 1**　使用 Haversine 公式[13] 计算出拓扑上节点间距离;

**Step 2**　使用 Johnson 全源最短路径算法计算出单个节点到所有节点的最短距离;

**Step 3**　使用式(5)计算出单个节点到所有节点的最小时延;

**Step 4**　使用粒子群算法选择 $k$ 个控制器的初始位置作为被部署的控制器集合 $C = \{c_1, c_2, \cdots, c_k\}$;

**Step 5**　找到控制器集合 $C = \{c_1, c_2, \cdots, c_k\}$ 与所有交换机集合 $S = \{s_1, s_2, \cdots, s_n\}$ 之间的最小时延;

**Step 6**　将最小时延放到控制器部署矩阵 ***place_value***$(n,k)$ 对应位置;

**Step 7**　将最小时延对应的交换机节点 $s_i$ 放到被挑选的交换机集合 $select\_switch\{s_i\}$;

**Step 8**　将最小时延对应的控制器节点 $c_j$ 放到被挑选的交换机集合 $select\_controller\{c_j\}$;

**Step 9**　统计 $select\_controller$ 集合中 $c_1, c_2, \cdots, c_k$ 的个数;

**Step 10**　重复 Step 5~9 直到 $select\_controller$ 集合中 $c_1, c_2, \cdots, c_k$ 的个数都等于 $\lfloor n/k \rfloor$;

**Step 11**　从交换机集合 $S$ 中选择 $select\_switch$ 集合中没有出现的元素 $s_i$,形成集合 $S' = \{s_i\}$;

**Step 12**　找到交换机 $S' = \{s_i\}$ 集合与控制器集合 $C = \{c_1, c_2, \cdots, c_k\}$ 之间的最小时延;

**Step 13**　将最小时延放到控制器部署矩阵

*place_value*($n$,$k$) 对应位置;

**Step 14** 将最小时延对应的交换机节点放到被挑选的交换机集合 *select_switch*{$s_i$};

**Step 15** 将最小时延对应的交换机节点 $s_i$ 从 $S'$ 集合中移除;

**Step 16** 将最小时延对应的控制器节点 $c_j$ 从 $C=\{c_1,c_2,\cdots,c_k\}$ 集合中移除;

**Step 17** 统计 *select_switch* 集合中所有元素的个数;

**Step 18** 重复 Step 12~17,直到 *select_switch* 集合中所有元素的个数等于 $n$;

**Step 19** 返回控制器部署矩阵 *place_value*。

## 2.2 改进的粒子群算法

### 2.2.1 粒子群算法(PSO)

粒子群算法(particle swarm optimization,PSO)是一种源于自然的基于种群的随机优化算法[14]。粒子群算法在群体中寻找最优解,并从两种学习方式中获益:基于个体历史的认知学习和基于群体历史的社会学习[15]。初始阶段,认知学习需要确定最优位置,在获取局部最优后,社会学习来决定全局最优位置。

假设有 $n$ 个群体表示潜在的解决方案,每个群体都可以用 $d$ 维向量表示。在粒子群算法中,每个粒子包含 2 个分量:速度和位置[16]。当前的粒子群位置向量用 $\boldsymbol{X}_i=[x_{i1},x_{i2},\cdots,x_{id}]$($i=1,2,\cdots,n$)表示。当前的速度向量用 $\boldsymbol{V}_i=[v_{i1},v_{i2},\cdots,v_{id}]$ 表示。粒子群的局部最优位置为 $\boldsymbol{P}_i=[p_{i1},p_{i2},\cdots,p_{id}]$,$\boldsymbol{P}_{\mathrm{gb}}$ 为全局最优位置向量。

在 $t+1$ 时刻,更新粒子群的位置 $X_{id}$ 和速度 $V_{id}$ 分别表示为:

$$V_{id}^{t+1}=\omega V_{id}^{t}+c1r1(\boldsymbol{P}_{id}-X_{id}^{t})+c2r2(\boldsymbol{P}_{\mathrm{gb}}-X_{id}^{t}); \quad (18)$$

$$X_{id}^{t+1}=X_{id}^{t}+V_{id}^{t+1}。 \quad (19)$$

式中:$\omega$ 为惯性权重;$r1$ 和 $r2$ 为 [0,1] 的随机数;$c1$ 和 $c2$ 分别为调整个体部分的局部学习因子和调整社会部分的全局学习因子。

一个大的惯性权重 $\omega$ 有利于展开全局寻优,而一个小的惯性权重值有利于局部寻优。因此,采用时变权重,在迭代过程中采用线性递减惯性权值,则粒子群算法在开始时具有良好的全局搜索性能,能够迅速定位到接近全局最优点的区域,而在后期具有良好的局部搜索性能,能够精确地得到全局最优解。时变权重的表达式如下:

$$\omega=\omega_{\max}-((\omega_{\max}-\omega_{\min})/I_{\max})\times I。 \quad (20)$$

式中:$I_{\max}$ 为最大迭代;$I$ 为当前迭代;通常 $\omega_{\max}$ 和 $\omega_{\min}$ 分别取 0.9 和 0.1。

每个粒子的最优位置向量为

$$\boldsymbol{P}_i(t+1)=\begin{cases}\boldsymbol{P}_i(t), & F(\boldsymbol{X}_i(t+1))\geqslant F(\boldsymbol{P}_i(t));\\ \boldsymbol{X}_i(t+1), & \text{其他}。\end{cases} \quad (21)$$

### 2.2.2 改进的粒子群算法

针对 PSO 算法出现的收敛速度慢的缺点,采用改进的粒子群优化算法(advanced particle swarm optimization,APSO)以改善种群的收敛速度。

由于 PSO 算法的搜索过程是一个非线性的复杂过程,采用惯性权重线性过渡的方法并不能正确地反映真实的搜索过程。因此引入文献[17]中收敛因子概念,从提高种群的收敛速度方面对粒子群算法进行改进。其中收敛因子表达式为

$$\chi=\frac{2}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|},\varphi=c1+c2,\varphi>4。 \quad (22)$$

对粒子群算法的速度方程稍加改进,可得到:

$$V_{id}^{t+1}=\chi[V_{id}^{t}+c1r1(\boldsymbol{P}_{id}-X_{id}^{t})+c2r2(\boldsymbol{P}_{\mathrm{gb}}-X_{id}^{t})]。 \quad (23)$$

根据式(23)可以更新粒子的飞行速度。文献[18]证明了采用收敛因子的优化算法相比于采用惯性权重的优化算法有更好的收敛速度。

### 2.2.3 改进的粒子群算法实现控制器部署(APSO_MCP)

在改进的粒子群算法中,通过迭代,使多个粒子并行搜索最优解,并确定最小合适度时控制器的位置。控制器位置向量是上述目标函数的次优解,每个个体都表示控制器部署的一种方式。在给定控制器数量为 $k$ 个的情况下,单个粒子的位置向量为 $\boldsymbol{P}_i=[p_{i1},p_{i2},\cdots,p_{ik}]$。APSO_MCP 算法将所有节点的经纬度 *latlon*、需要被部署的控制器数量 $k$、最大迭代 *maxIt* 作为输入,最后得到控制器的最终部署位置 $\boldsymbol{P}_{\mathrm{gbest}}$,整个网络的最优合适度 $F(\boldsymbol{P}_{\mathrm{gbest}})$。APSO_MCP 算法的描述如下。

**算法 2** APSO_MCP 算法。

**输入:** 所有节点的经纬度 *latlon*,被部署的控制器数量 $k$,最大迭代次数 *maxIt*;

**输出:** 最终的控制器部署位置 $\boldsymbol{P}_{\mathrm{gbest}}$,全局最优合适度值 $F(\boldsymbol{P}_{\mathrm{gbest}})$。

(1) 初始化阶段:

①初始化学习因子 $c1$,$c2$,收敛因子 $\chi$,粒子群 *nPop*,将粒子群全局最优合适度 $F(\boldsymbol{P}_{\mathrm{gbest}})$ 设置为∞;

②for $i=1$ to *nPop*

③ 随机部署 $k$ 个控制器部署位置 $\boldsymbol{P}_{\mathrm{position}}$,并将当前位置设为当前最优位置 $\boldsymbol{P}_{\mathrm{pbest}}$;

④ 初始化粒子的速度 $\boldsymbol{P}_{\mathrm{velocity}}$ 为 0;

⑤　调用控制器部署算法(MCP)，调用合适度函数 $F$;

⑥　if( $F(\boldsymbol{P}_{\text{pbest}}) < F(\boldsymbol{P}_{\text{gbest}})$ )

⑦　　将局部最优位置 $\boldsymbol{P}_{\text{pbest}}$ 设为全局最优位置 $\boldsymbol{P}_{\text{gbest}}$;

⑧　　将局部最优合适度 $F(\boldsymbol{P}_{\text{pbest}})$ 设为全局最优合适度 $F(\boldsymbol{P}_{\text{gbest}})$;

⑨endif

⑩endfor

(2) 优化阶段:

⑪for $it = 1$ to $maxIt$

⑫　for $i = 1$ to $nPop$

⑬　　使用公式(18)更新粒子 $\boldsymbol{P}$ 的速度，使用公式(21)更新粒子 $\boldsymbol{P}$ 的位置;

⑭　　调用控制器部署算法(MCP)，调用合适度函数 $F$;

⑮　　if ( $F(\boldsymbol{P}_{\text{position}}) < F(\boldsymbol{P}_{\text{pbest}})$ )

⑯　　将当前位置 $\boldsymbol{P}_{\text{position}}$ 设为局部最优位置 $\boldsymbol{P}_{\text{pbest}}$;

⑰　　if ( $F(\boldsymbol{P}_{\text{pbest}}) < F(\boldsymbol{P}_{\text{gbest}})$ )

⑱　　将局部最优位置 $\boldsymbol{P}_{\text{pbest}}$ 设为全局最优位置 $\boldsymbol{P}_{\text{gbest}}$;

⑲　　将局部最优合适度 $F(\boldsymbol{P}_{\text{pbest}})$ 设为全局最优合适度 $F(\boldsymbol{P}_{\text{gbest}})$;

⑳　　endif

㉑　　endif

㉒　endfor

㉓endfor

㉔return 最终的控制器部署位置 $\boldsymbol{P}_{\text{gbest}}$，全局最优合适度值 $F(\boldsymbol{P}_{\text{gbest}})$。

粒子合适度函数 $F$ 通过式(5)~(17)来计算。

## 3　仿真实验

### 3.1　实验设置

相关算法均在 MATLAB 中运行，选择的仿真环境是 MATLAB 2018a，系统配置是 Windows 10 64 位系统，i5 处理器，8 GB 内存。实验采用的网络拓扑选自于网络拓扑动物园的 Xspedius 网络。Xspedius 网络有 34 个节点和 49 条边。设置控制器的最大负载为 1 600 flows/s，各个交换机的流请求相互独立且为 100 flows/s[19]。设置 $vc$ 为 $3\times10^8$ /s，设置粒子群种群数量为 50，迭代次数为 300。设置控制器部署模型权重因子 $\alpha$ 为 0.5，$\beta$ 为 0.5。

为了使实验仿真更具有说服力，选用随机算法和 $K$-means 算法做对比。随机算法是面对大量的数据，从其中随机选取一些数据来进行分析。$K$-means 算法是输入聚类个数 $k$，以及包含 $n$ 个数据对象的数据库，输出满足标准方差最小的 $k$ 个聚类的一种算法。将 APSO_MCP 算法、随机算法[7]、$K$-means 算法[20]和 PSO_MCP 算法[7]在网络总时延、负载均衡比、总体性能以及运行时间方面进行比较。

### 3.2　实验结果及分析

实验 1　验证随机算法、$K$-means 算法、PSO_MCP 算法和 APSO_MCP 算法在控制器传播时延上的差异。如图 1 所示，由于改进的粒子群算法是在收敛速度的基础上进行改进，故 APSO_MCP 算法和 PSO_MCP 算法在控制器传播时延上的差异不大；在控制器数量为 3、4、5 时，APSO_MCP 算法和 $K$-means 在总时延的差别较大；随着控制器数量的增加，APSO_MCP 算法在总传播时延方面接近于 $K$-means 算法。出现这种情况的原因是 $K$-means 算法在部署控制器时根据最小距离原则选择和更新控制器位置，而本文算法选择增加部分传播时延来平衡负载均衡指标。

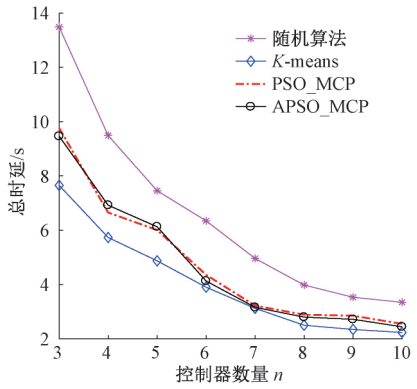


图 1　网络总时延比较

Figure 1　The comparison about the total latency of the network

实验 2　验证随机算法、$K$-means 算法、PSO_MCP 算法和 APSO_MCP 算法在整个网络最大负载控制器子域和最小负载控制器子域之间的差异。实验结果如图 2 所示，APSO_MCP 算法和 PSO_MCP 算法的实验结果相近，始终接近于理想参数 1，优于 $K$-means 算法和随机算法。分析其原因是：网络拓扑中节点分布不均匀，小部分分散，大部分相对集中，而 $K$-means 是以减少传播时延为目标，在划分控制器子域时未考虑控制器子域的负载，从而导致部分控制器子域内的负载大幅增大。

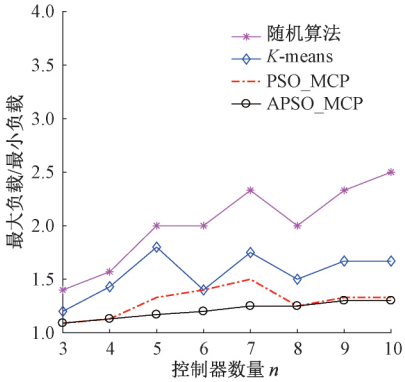


图2 网络负载均衡比比较

Figure 2 The comparison about the load balance ratio of the network

实验3 验证随机算法、$K$-means 算法、PSO_MCP 算法和 APSO_MCP 算法在整个网络的性能上的差异。实验结果如图3所示,APSO_MCP 算法和 PSO_MCP 算法在整个网络的总体性能上相近。在控制器数量为3、4、5时,随机算法、$K$-means 和 APSO_MCP 在整个网络的性能的差别较大。出现这种现象的原因是网络拓扑结构中节点分布不均匀,使用 $K$-means 算法划分子域后,各个控制器子域的传播时延和整个网络的总时延、各个控制器子域之间的负载和整个网络的平均负载相差较大,从而导致整个网络的性能降低。随着控制器数量的增加,$K$-means 算法和随机算法在整个网络的性能方面接近于 APSO_MCP。分析其原因是随着 $K$-means 算法划分的控制器子域数量的增多,每个子域管理的交换机数量逐渐减少,整个网络被划分成的控制器子域在传播时延和负载均衡方面越来越合理,整个网络的性能越来越稳定。

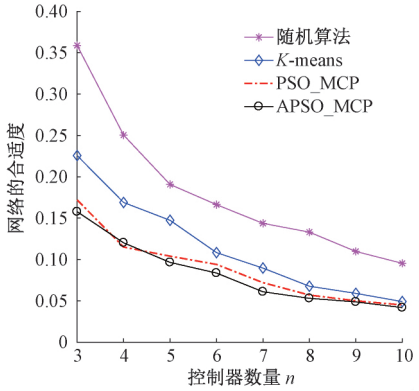


图3 网络的总体性能比较

Figure 3 The comparison about the overall performance of the network

实验4 验证随机算法、$K$-means 算法、PSO_MCP 算法和 APSO_MCP 算法在运行时间上的差异。如图4所示,随机算法的运行时间低于 $K$-means 算法、PSO_MCP 算法和 APSO_MCP 算法。随机算法没有迭代过程,故运行时间最短。$K$-means 算法随着控制器数量、迭代次数增加,运行时间逐渐增长。而 APSO_MCP 算法由于引入收敛因子,种群的收敛速度加快,在运行时间方面优于 PSO_MCP 算法,种群的收敛速度提升约6.3%,并且随着控制器数量的增加,APSO_MCP 算法的收敛速度快的优势越来越明显。

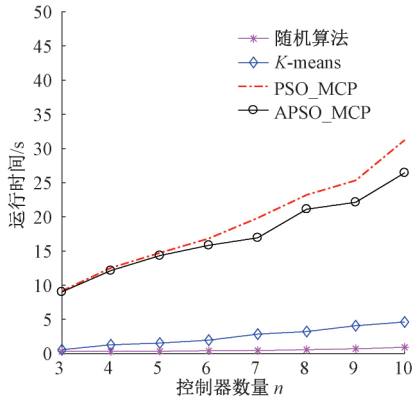


图4 运行时间比较

Figure 4 Comparison of the computing time

## 4 结论

(1) 针对软件定义网络中多控制器部署问题,以时间延迟和负载为双重优化目标,建立控制器部署模型,提出 MCP 算法。结果表明该算法可以保证多控制器子域之间的负载均衡性能,并且有效降低整个网络的时间延迟,从而确保了整个网络的总体性能处于最优的状态。

(2) 针对传统粒子群算法收敛速度慢问题,引入收敛因子概念,提出 APSO 算法。结合 APSO 算法和 MCP 算法来部署多控制器位置。结果表明 APSO_MCP 算法和 PSO_MCP 算法相比,有效地缩短了算法的运行时间,验证了本文提出的 APSO_MCP 算法可以有效地提高种群的收敛速度。

(3) 合理部署控制器位置的算法可以提升网络性能[21]。然而本文提出的算法还有一些不足,提出的是一种静态的部署算法,在实际网络中,网络状态不断变化,还需要考虑动态网络中的负载情况。

## 参考文献:

[1] LIU Y F, ZHAO B, ZHAO P Y, et al. A survey: typ-

ical security issues of software-defined networking [J]. China communications, 2019, 16(7): 13-31.

[2] KILLI B P R, RAO S V. Controller placement in software defined networks: a comprehensive survey [J]. Computer networks, 2019,163: 106883.

[3] YEGANEH S H, GANJALI Y. Kandoo: a framework for efficient and scalable offloading of control applications [C] // Proceedings of Workshop on Hot Topics in Software Defined Networks. New York: ACM, 2012: 19-24.

[4] HELLER B, SHERWOOD R, MCKEOWN N. The controller placement problem [J]. ACM SIGCOMM computer communication review, 2012, 42(4): 473-478.

[5] LIAO L X, LEUNG V C M. Genetic algorithms with particle swarm optimization based mutation for distributed controller placement in SDNs [C] // Proceedings of 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN). Piscataway: IEEE, 2017: 1-6.

[6] ATEYA A A, MUTHANNA A, VYBORNOVA A, et al. Chaotic salp swarm algorithm for SDN multi-controller networks [J]. Engineering science and technology, an international journal, 2019, 22(4): 1001-1012.

[7] SAHOO K S, SARKAR A, SAHOO S, et al. On the placement of controllers for designing a wide area software defined networks [C] // Proceedings of 2017 IEEE Region 10 Conference (TENCON 2017). Piscataway: IEEE, 2017: 3123-3228.

[8] JALILI A, AHMADI V, KESHTGARI M, et al. Controller placement in software-defined wan using multi-objective genetic algorithm [C] //Proceedings of International Conference on Knowledge-Based Engineering and Innovation. Piscataway: IEEE, 2015: 656-662.

[9] TAO P Y, YING C, SUN Z, et al. The controller placement of software-defined networks based on minimum delay and load balancing [C] // Proceedings of 4th International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress. Piscataway: IEEE, 2018: 310-313.

[10] KUANG H, QIU Y, LI R, et al. A hierarchical *K*-means algorithm for controller placement in SDN-based wan architecture [C] // Proceedings of 2018 10th International Conference on Measuring Technology & Mechatronics Automation. Piscataway: IEEE, 2018: 263-267.

[11] KHORRAMIZADEH M, AHMADI V. Capacity and load-aware software-defined network controller placement in heterogeneous environments [J]. Computer communications, 2018, 129(9): 226-247.

[12] 田家翼. 基于 SDN 的多级多域流量动态协同调度机制研究 [D]. 北京: 北京邮电大学, 2019.

[13] SAHOO K S, PUTHAL D, OBAIDAT M S, et al. On the placement of controllers in software-Defined-WAN using meta-heuristic approach [J]. Journal of systems and software, 2018,145: 180-194.

[14] GAO C G, WANG H, ZHU F J, et al. A particle swarm optimization algorithm for controller placement problem in software defined network [C] // Proceedings of 15th International Conference of Algorithms and Architectures for Parallel Processing (ICA3PP 2015). Berlin: Springer,2015: 44-54.

[15] PEHLIVANOGLU Y V. A new particle swarm optimization method enhanced with a periodic mutation strategy and neural networks [J]. IEEE transactions on evolutionary computation, 2013, 17(3): 436-452.

[16] 程适,王锐,伍国华,等.群体智能优化算法 [J].郑州大学学报(工学版),2018,39(6): 1-2.

[17] CLERC M. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization [C] // Proceedings of International Conference on Evolutionary Computation. Piscataway: IEEE, 1999: 51-57.

[18] 徐霜,万强,余琍.基于学习理论的改进粒子群优化算法 [J].郑州大学学报(工学版),2019,40(2): 29-34.

[19] 史久根,谢熠君,孙立,等. 软件定义网络中面向时延和负载的多控制器放置策略 [J]. 电子与信息学报, 2019,41(8): 1869-1876.

[20] GONG W R, PANG L H, WANG J, et al. A social-aware *K* means clustering algorithm for D2D multicast communication under SDN architecture [J]. AEU-international journal of electronics and communications, 2021,132: 153610.

[21] SHI J G, XIE Y J, SUN L, et al. Multi-controller placement strategy based on latency and load in software defined network [J]. Journal of electronics and information technology, 2019,41(8): 1869-1876.

(下转第 32 页)

## Dense Depth Interpolation for 3D Human Pose Estimation

CHEN Mengting, WANG Xinggang, LIU Wenyu

(School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan 430074, China)

**Abstract:** The 3D human pose estimation is a challenging task in computer vision. Due to the difficulty of annotation, only some disperse key-point data form limited scenes are available, which makes 3D prediction a big challenge. In this paper, the human body is deemed as a flexible structure, but a specific limb can be viewed as a rigid-body. Given depths of two points on both ends, the depths of the whole limb can be estimated by dense interpretation. Therefore, this paper proposes a method that can take the dense depth interpretation feature map as middle supervision. It provides a denser and more structured target, instead of regression for disperse key-points directly. The *MPJPG* on Human3.6M reaches 50.9 mm with only a simple network structure. The cross-domain experiments on dataset MPI-INF-3DHP further show the generalization ability of the proposed method.

**Key words:** 3D vision; human pose estimation; dense depth interpolation; cross-domain generalization

(上接第25页)

## Multi-controller Deployment Strategy Based on Delay and Load Balancing

LIU Zhenpeng[1,2], WANG Xinpeng[1], LI Ming[1], REN Shaosong[1], LI Xiaofei[2]

(1. School of Electronic Information Engineering, Hebei University, Baoding 071002, China; 2. Center for Information Technology, Hebei University, Baoding 071002, China)

**Abstract:** To address the time delay and load balancing problems faced by multiple controllers deployed in the software definition network (SDN), in this paper, a multi-controller placement algorithm is proposed to reduce the time delay between controllers, and improve the network performance on the basis of load balancing. Aiming at the slow convergence speed of traditional particle swarm optimization algorithm, this paper proposes an improved particle swarm optimization algorithm to deploy the SDN controller. The improved particle swarm optimization algorithm is used to deploy the SDN controller to minimize the propagation delay between the switch and the controller while considering the load balance of the controller. The simulation results show that the improved particle swarm optimization algorithm for controller deployment can guarantee high load balancing performance and the better overall network performance by acquire fitness about 0.05. And compared with the traditional particle swarm optimization algorithm, the improved particle swarm optimization algorithm can improve the convergence speed of the whole network about 6.3% with lower time delay.

**Key words:** software defined network; controller placement; latency; load balancing; particle swarm optimization