SFCSim 类设计

BUPT

2020 年 1 月 1 日

目 录

vnf_type 类

1 类概述

vnf_type 类。包含 vnf 名称，节点属性，vnf 属性对应的资源，vnf_type 类可以作为只表示 vnf 类型的实例，不分配任何资源，也可以作为 node 中的 vnf 实例，占用实际资源。一旦确定类型，其中处理单位流量占用的资源通过 resource_coefficient 计算，目前采用线性计算法。对任意大小输入流量f_{in}所需资源资源类型 i($i \in \{cpu, memory, storage, \dots\}$)公式与流出流量大小公式$f_{out}$如下：

$$r_i = k_i * f_{in} \quad (i \in \{cpu, memory, storage, \dots\})$$

$$f_{out} = ratio * f_{in} \ (ratio > 0)$$

2 类属性

名称		描述	类型	默认值
name		vnf 类型名称	string	‘vnf_type1’
atts	cpu	vnf 实例分配得到的资源	{}	{'cpu':0,'memory':0,'storage':0}
	memory			
	storage			
ratio		vnf 处理后流量伸缩系数	Int	1
resource_coefficient		vnf 处理单位带宽流量所需资源系数	{}	{'cpu':1,'memory':1,'storage':1}
remain_resource	cpu	vnf 实例的剩余资源	{}	atts
	memory			
	storage			

3 类方法

名称	参数类型	描述
__init__(name,atts,radio,resource_coefficient)		类内部构造函数
set_name(name)	str	设置 vnf 名称
get_name		获取 vnf 类型名称
set_atts(atts_value)	{}	设置 vnf 属性
get_atts		获取 vnf 属性

set_ratio(ratio)	str	设置带宽伸缩系数
get_ratio		获取带宽伸缩系数
set_coeff(coeff)	{}	设置属性处理系数
get_coeff		获取属性处理系数
set_remain_resource(resource)	{}	设置剩余资源
get_remain_resource()		获取剩余资源
is_idle()		判断 vnf 是否有 nf
show()		输出 vnf_type 信息

vnf_types 类

1 类概述

vnf_types 类。包含一组不分配实体资源的 vnf_type 实例，表示 vnf 类型的集合，包含网络中所有的 vnf 类型。

2 类属性

名称	描述	类型	默认值
number	Vnf 类型总数	int	0
vnf_types	Vnf_type 实例，但不分配资源	vnf_type 数组	[]

3 类方法

名称	参数类型	描述
__init__(number,vnf_types)	int,[]	类内部构造函数
get_number		返回存储 vnf type 数量 number
generate(number)	int	内部简单生成 number 个 type 的 vnf_type
search_vnf_type(name)	str	类中是否存在当前类型 vnf
add_vnf_type(vnf_type)	vnf_type	加入一个 vnf_type
delete_vnf_type(name)	str	删除一个 vnf_type 并放回值
delete_vnf_type(name)	[]	删除多个个 vnf_type 并放回值

get_vnf_type(name)	str	返回对应 vnf_types
get_vnf_types()		返回 vnf_types
set_ratio(name,ratio)	str,str	设置带宽伸缩系数
get_ratio(name)	str	获取带宽伸缩系数
set_coeff(name,coeff)	str,{}	设置属性处理系数
get_coeff(name)	str	获取属性处理系数
show		输出 vnf_type 信息

node 类

1 类概述

节点类。包含节点 id，节点属性，属性对应的资源，节点开启的 vnf，剩余资源。节点类只记录节点的属性、节点剩余资源以及节能内部开启的 VNF。

当在节能内部加入一个 vnf(add_vnf)，以及删除 vnf(delete_vnf)时，节点会自动减少和增加剩余资源值，并把这部分资源记录在 vnf 的 atts 中。注意一旦节点中存在 vnf(is_idle()函数放回 False)，则不可改变节点属性值。

2 类属性

名称		描述	类型	默认值
id		节点 id	string	‘node1’
atts	cpu	节点占有的资源属性以及是否是接入节点	{}	{'cpu':0,'memory':0, 'storage':0,’access’:False}
	memorys			
	storage			
	access			
vnfs		内部包含的 vnf 实例数组	vnf_type 类数组	[]
remain_resource	cpu	节点剩余资源	{}	{'cpu':0,'memory':0, 'storage':0}
	memorys			
	storage			

3 类方法

名称	参数类型	描述

__init__(uuid,atts)	Str,{}	内部构造函数
set_id(id)	str	设置节点 id
get_id		获取节点 id
set_atts(atts)	{}	设置属性资源值
get_atts		获取属性资源
set_access_node()		将节点设置成接入节点
get_vnf(name)	str	获取名称为 name 的开启的 vnf
get_vnfs()		获取开启的 vnf
search_vnf_type(name)	str	查找 vnf 类型是否存在
add_vnf(vnf)	vnf_type	在节点中开启一个 vnf
delete_vnf(name)	str	删除节点中的一个 vnf
scale_in_vnf(name)	str	缩小分配给 vnf 资源至最小值
scale_out_vnf(name,atts)	Str,{}	扩大分配给 vnf 资源， 添加 atts 大小
get_remain_resource		获取剩余资源
is_access		判断节点是否未接入节点
is_idle		判断节点是否被使用
Show		显示 node

nodes 类

1 类概述

节点组合类。包含多个 node 类实例 nodes，以及为了区分节点类型自动划分的服务节点实例__server_nodes 和接入节点实例__access_nodes，由于这两个子实例为私有变量，不要使用类方法之外的手段修改其属性。

一个网络中应该只包含一个全局的 nodes 类实例，代表网络的所有节点集合，其上的每一个 node 为 networkx 库的一个节点。

2 类属性

名称	描述	类型	默认值
number	节点总数	int	0

__access_number	接入节点总数	Int	0
__server_number	服务节点总数	int	0
nodes	节点列表	node 数组	[]
__server_nodes	服务器节点列表	node 数组	[]
__access_nodes	接入节点列表	node 数组	[]

3 类方法

名称	参数类型	描述
__init__(nodes)	[node]	内部构造函数
get_number()		获取 nodes 实例数量
get_nodes()		获取 nodes 实例
get_access_nodes()		获取接入 nodes 实例
get_server_nodes()		获取服务 nodes 实例
search_node(uuid)	str	查找节点
get_node(uuid)	str	返回 id 为 uuid 的节点
search_access_node(uuid)	str	查找接入节点
search_server_node(uuid)	str	查找服务节点
add_node()	node	向类中加入一个 node 实例
add_nodes()	nodes	向类中添加多个 node 实例
delete_node(uuid)	str	在类中删除一个 node 实例并返回
set_access_nodes([uuid])	str	将多个节点设置成接入节点
get_atts(uuid)	str	获取 uuid 节点属性资源
set_atts(uuid,atts)	Str,{}	设置 uuid 节点属性值
get_vnfs(uuid)	str	获取 uuid 节点开启的 vnf
get_remain_resource(uuid)	str	获取 uuid 节点剩余资源
add_vnf(uuid,vnf)	Str,vnf_type	在 uuid 节点中开启一个 vnf
delete_vnf(uuid,name)	Str,name	在 uuid 节点中删除的一个 vnf
Show()		打印节点所有信息
show_access()		打印接入节点所有信息
show_server()		打印服务节点所有信息

link 类(暂时不需要)

1 类概述

链路类。包含链路 id，链路属性，属性对应的资源，链路剩余资源，由于networkx 本身采用邻接链表存储网络，对链路友好，暂不实现此部分代码，可以直接使用 networkx 代替链路类

2 类属性

名称	描述	类型	默认值
id	链路 id	string	‘link1’
nodes	连接节点 id	()	(0,0)
atts	链路属性值	{}	{'bandwidth':0,'delay':0}
remain_resource	剩余资源	{}	atts

3 类方法

名称	描述
set_atts	设置属性资源值
get_atts	获取属性资源
get_remain_resource	获取剩余资源
get_id	获取链路 id
get_nodes	获取链路两端节点

network 类

1 类概述

底层网络类。network 类继承 nodes 类，用于存储节点属性。同时包含一个底层网络 G(networkx 类型)，用于存储链路属性和一些内部算法库，其中 G 的节点为 nodes 实例中的节点 node 实例。

此方案将 node 实例引用作为 G 的节点(别的方案如将 node 的 id 作为 G 的节点也行)，能够方便节点属性的一些操作，如 G.nodes 直接等于 nodes.nodes。此时 sfc 类中 in_node 和 out_node 对应 node 实例引用较好。

由于需要将 nodes 实例与 G 实例关联，切记不要使用 networkx 原生的修改

属性算法(如 G.add_edge()，G.add_node())，而使用下表给出的修改属性方法。但一些不修改属性的方法可以任意使用

2 类属性

名称	描述	类型	默认值
nodes	nodes 类实例	string	0
G	networkx 实例	networkx	无

3 类方法

名称	参数类型	描述
名称	参数类型	描述
__init__(nodes)	[node]	内部构造函数
nodes 类原有方法(红色表示此方法已重载)		
get_number()		获取 nodes 实例数量
get_nodes()		获取 nodes 实例
get_access_nodes()		获取接入 nodes 实例
get_server_nodes()		获取服务 nodes 实例
search_node(uuid)	str	查找节点
search_access_node(uuid)	str	查找接入节点
search_server_node(uuid)	str	查找服务节点
add_node(node)	node	向类中加入一个 node 实例
add_nodes()	nodes	向类中添加多个 node 实例
delete_node(uuid)	str	在类中删除一个 node 实例
set_access_nodes([uuid])	str	将多个节点设置成接入节点
get_atts(uuid)	str	获取 uuid 节点属性资源
set_atts(uuid,atts)	str,{}	设置 uuid 节点属性值
get_vnfs(uuid)	str	获取 uuid 节点开启的 vnf
get_remain_resource(uuid)	str	获取 uuid 节点剩余资源
add_vnf(uuid,vnf)	Str,vnf_type	在 uuid 节点中开启一个 vnf
delete_vnf(uuid,name)	Str,name	在 uuid 节点中删除的一个 vnf
show()改成 show_node		打印节点所有信息
show_access()		打印接入节点所有信息

show_server()		打印服务节点所有信息
新方法		
generate ()		将 nodes 实例与 G 实例关联，生成网络，一般不再外部调用
add_edge(node1,node2,**link)	node or str,node or str,key=value	向网络中添加链路
add_edge_from(edges)	[[node1,node2,atts],…]	向网络中加入多条边
add_edges(edges)	[[node1,node2,atts],…]	向网络中加入多条边，实际上是上方法的另一个名称
delete_edge(node1,node2)		删除链路
delete_edge_from(edges)	[[node1,node2],…]	删除多条链路
delete_edges(edges)	[[node1,node2],…]	删除多条链路的另一种方法
set_edge_atts(node1,node2,atts)		设置链路属性
set_edges_atts(atts)	[[node1,node2,atts],…]	设置所多条链路属性
get_edge_atts()		获得链路属性
get_edges_atts()		获得所有链路的属性
show_edges()		打印所有链路
show()		打印所有节点链路
draw(pos, ,node_size=85000,node_fone_size=15,link_fone_size=25,node_shape='H')	{node:[x,y],…},int,int,int	画图

sfc 类

1 类概述

服务功能链类。其属性值包含 sfc 的 id、输入输出节点 in_node、out_node、需要的网络功能集合 nfs、初始带宽、延迟需求、利润、开始时间、持续时间、类型(基础 sfc 类型为 0，不用设置)。初始化 sfc 还需要知道 vnf_types 类实例，用于计算 sfc 每条虚拟链路占用的链路资源以及每个 nf 占用的节点资源

其中 nf 集合 nfs 只需要包含 vnf 类型名称数组，通过查找 vnf_types 计算后续链路带宽以及每个 nf 所需资源得到属性 nfs_detail

nfs: ['nfs_type1','nfs_type2',…]

nfs_detail:{ 'nfs_type1':{'cpu':10,…},'nfs_type2':{'cpu':20,…},…}

2 类属性

名称		描述	类型	默认值
id		sfc 标志 id	string	无
atts	in_node	sfc 的输入节点	str	无
	out_node	sfc 的输出节点	str	无
	nfs	nf 名称集合	[]	[]
	bandwidths	Nf 链路带宽集合	[]	[]
	delay	延迟需求	int	0
	duration	持续时间	int	0
	service_time	服务时间，不需要自己设置	int	0
	profit	利润	int	0
	nfs_detail	nf 的细节	{}	无
vnf_types		全局的 vnf_types 实例	vnf_types	[]
type		sfc 类型	int	0

3 类方法

名称	参数类型	描述
__init__(uuid,in_node,out_node ,nfs,bandwidth,delay duration,profit,vnf_types)	str,str,str,[], int,int,int,int,vnf_types	类构造函数
__calculate_link_bw	内部函数	根据初始带宽和 vnf 伸缩因子计算虚拟链路带宽
__calculate_resource	内部函数	根据 vnf 属性计算每个 nf 占用资源
get_id		获取 id
set_id		设置 id
get_length		sfc 链的长度
set_atts	{}	设置属性
get_atts		获取所有属性
set_vnf_types(vnf_types)	vnf_types	设置 vnf_types
get_vnf_types		获取 vnf_types
get_nfs		获取 nfs 集合

get_nfs_detail		获取 nfs 详细信息
get_bandwidths		获取虚拟链路带宽集合
get_in_node		获取输入节点
get_out_node		获取输出节点
get_duration		获取持续时间
get_sevice_time		获取已经服务时间
next_cycle		动态调度，每一周期执行一次
is_life_end		判断是否到达生命周期
get_delay		获取延迟
get_profit		获取利润
get_type		获取类型
show		打印 sfc 所有属性

扩展 sfc 类

1 扩展 sfc 分类

服务功能链扩展类，继承 sfc 类，为了满足各种特殊要求定制的 sfc，如移动 sfc，流量时刻改变的 sfc 等，其对应的 type 不为 0，各种类型 sfc 的 type 分配如下：

场景	类名称	Type 值	功能
对应移动用户的移动 sfc	mobile_sfc	1	输入节点周期改变或者非周期改变
流量时刻改变的动态 sfc	dynamic_traffic_sfc	2	sfc 流量周期或非周期动态改变
后续注册	……	……	……

2 扩展 sfc 描述

2.1 mobile_sfc

类概述

type 值为 1，输入节点 in_node 为一个数组，数组每一项代表之后用户会移

动到的位置，如：

in_node=[node1,node2,node3]

表示之后会相继移动到 node1 节点附近，node2 节点附近，node3 节点附近。若用户周期移动对应的持续时间 duration 如果设置成固定值，若非周期移动，duration 设置成数组，且长度与 in_node 相同，表示每个位置停留时间，如

duration=[3,4,0]

表示在第一个节点停留 3 个时间单位，第二个节点停留 4 个时间单位，最后一直停留在第三个节点(0 表示一直停留)

类属性

名称		描述	类型	默认值
id		sfc 标志 id	string	无
atts	in_node	sfc 的单前输入节点	str	无
	in_nodes	所有会移动到的输入节点	[]	无
	out_node	sfc 的输出节点	str	无
	nfs	nf 名称集合	[]	[]
	bandwidths	Nf 链路带宽集合	[]	[]
	delay	延迟需求	int	0
	duration	持续时间	[],int	0
	profit	利润	int	0
	service_time	服务时间，不需要自己设置	int	0
	nfs_detail	nf 的细节	{}	无
vnf_types		全局的 vnf_types 实例	vnf_types	[]
type		sfc 类型	int	1

类方法

红色表示这些方法已经重载或者为新方法

名称	参数类型	描述
__init__(uuid,in_nodes,out_node ,nfs,bandwidth,delay duration,profit,vnf_types)	str,str or [],str,[], int,int,int or [],int,vnf_types	类构造函数
__calculate_link_bw	内部函数	根据初始带宽和 vnf 伸缩因子计算虚拟链路带宽
__calculate_resource	内部函数	根据 vnf 属性计算每个 nf 占用资源
get_id		获取 id

set_id		设置 id
get_length		sfc 链的长度
set_atts	{}	设置属性
get_atts		获取所有属性
set_vnf_types(vnf_types)	vnf_types	设置 vnf_types
get_vnf_types		获取 vnf_types
get_nfs		获取 nfs 集合
get_nfs_detail		获取 nfs 详细信息
get_bandwidths		获取虚拟链路带宽集合
get_in_node		获取输入节点
get_in_nodes		获取所有输入节点
get_out_node		获取输出节点
get_duration		获取持续时间
get_sevice_time		获取已经服务时间
next_cycle		动态调度，每一周期执行一次 1. service_time 增加一 2. 检查并更新 in_node
is_life_end		判断是否到达生命周期
get_delay		获取延迟
get_profit		获取利润
get_type		获取类型
show		打印 sfc 所有属性

2.2 dynamic_traffic_sfc

类概述

type 值为 2，流量 bandwidth 为一个数组，数组每一项代表之后的流量变化，如：

bandwidth=[1,2,3,4,5,6]

表示之后流量会依此变化为 1，2，3，4，5，6。若用户流量周期变化，持续时间 duration 设置成固定值，若非周期变化，duration 设置成数组，且长度与 bandwidth 相同，表示每个流量停留时间，如

duration=[3,4,5,6,7,0]

表示在流量 1 停留 3 个时间单位，流量 2 停留 4 个时间单位…，最后流量保持为 6

类属性

名称		描述	类型	默认值
id		sfc 标志 id	string	无
atts	in_node	sfc 的单前输入节点	str	无
	out_node	sfc 的输出节点	str	无
	nfs	nf 名称集合	[]	[]
	bandwidth	流量变化集合	[]	[]
	bandwidths	Nf 链路单前带宽集合	[]	[]
	delay	延迟需求	int	0
	duration	持续时间	[],int	0
	profit	利润	int	0
	service_time	服务时间，不需要自己设置	int	0
	nfs_detail	nf 的细节	{}	无
vnf_types		全局的 vnf_types 实例	vnf_types	[]
type		sfc 类型	int	2

类方法

红色表示这些方法已经重载或者为新方法

名称	参数类型	描述
__init__(uuid,in_nodes,out_node ,nfs,bandwidth,delay duration,profit,vnf_types)	str,str,str,[], int,[] ,int or [],int,vnf_types	类构造函数
__calculate_link_bw	内部函数	根据初始带宽和 vnf 伸缩因子计算虚拟链路带宽
__calculate_resource	内部函数	根据 vnf 属性计算每个 nf 占用资源
get_id		获取 id
set_id		设置 id
get_length		sfc 链的长度
set_atts	{}	设置属性
get_atts		获取所有属性
set_vnf_types(vnf_types)	vnf_types	设置 vnf_types
get_vnf_types		获取 vnf_types
get_nfs		获取 nfs 集合
get_nfs_detail		获取 nfs 详细信息
get_bandwidth		获取带宽集合

get_bandwidths		获取当前虚拟链路带宽集合
get_in_node		获取输入节点
get_in_nodes		获取所有输入节点
get_out_node		获取输出节点
get_duration		获取持续时间
get_sevice_time		获取已经服务时间
next_cycle		动态调度，每一周期执行一次 1. service_time 增加一 2. 检查并更新 bandwidth 3. 如果当前带宽改变，则更新所有虚拟节点、虚拟链路需求资源
is_life_end		判断是否到达生命周期
get_delay		获取延迟
get_profit		获取利润
get_type		获取类型
show		打印 sfc 所有属性

sfcs 类

1 类概述

服务功能链集合类。其内部包含多条 sfcs 以及 sfc 生成算法

2 类属性

名称	描述	类型	默认值
number	类中包含 sfc 实例数	string	0
sfcs	Sfc 类实例集合	Sfc 类数组	[]

3 类方法

名称	参数类型	描述

get_number	int	获取 sfc 数目
get_sfcs		获取 sfcs 数组
add_sfc(sfc)	sfc	向 sfcs 中增加一个 sfc
search_sfc(uuid)	int	查找 sfc 是否存在
delete_sfc(id)	str	从 sfcs 中删除一个 sfc
get_sfc(id)	str	从 sfcs 中取出但不删除一个 sfc
pop_sfc(id)	str	从 sfcs 中取出并删除 sfc
get_length(id)	str	sfc 链的长度
set_atts(id,att)	str,{}	设置属性
get_atts(id)	str	获取所有属性
set_vnf_types(uuid,vnf_types)	vnf_types	设置 vnf_types
get_vnf_types(uuid)	str	获取 vnf_types
get_nfs(id)	str	获取 nfs 集合
get_nfs_detail(id)	str	获取 nfs 详细信息
get_bandwidths(id)	str	获取虚拟链路带宽集合
get_in_node(uuid)	str	获取输入节点
get_out_node(uuid)	str	获取输出节点
get_duration(uuid)	str	获取持续时间
get_delay(uuid)	str	获取延迟
get_profit(uuid)	str	获取利润
get_type(uuid)	str	获取 sfc 类型
show()		打印 sfcs 所有 sfc
show_detail()		打印 sfcs 所有 sfc 以及 nf 细节

scheduler 类

1 类概述

Scheduler 类中、将 sfc 中的 nf 加入到 node 的 vnf 中，虚拟链路添加到 edge 中、同时实现记录功能

nf 记录方式：{1:node,2:node,…}，通过字典的方式记录网络功能对应部署的节点

虚拟链路记录方式：{1:[node1,node2,…,nodeN],2:[noden,nodeN+1,…]…}，通过节点数组的方式记录虚拟链路经过的节点(应为一条虚拟链路可能经过多个节

点，如中间的中继节点)，其中前一条虚拟链路最后一个节点必须是后一条虚拟链路起点

scheduler 类应该是利于继承扩展的，所以只实现基础功能，后续使用者可以继承此类创建出多种调度器，实现各种算法，如 radom_scheduler（随机调度）、shortest_path_schedular（最短路径调度）等。

2 类属性

名称	描述	类型	默认值
__record	记录 sfc 部署情况	{sfc_id:{‘node’:{1:node,…}, ‘edge’:{1:[node_id,node_id,…],…} },…}	0
log	是否输出部署信息	Bool	Ture

3 类方法

名称	参数类型	描述
__init__		
get_record()		获取记录
__deploy_nf(name,resoruce,node)	str,{} ,node	在 node 上部署 nf
deploy_nf(sfc,node,i)	sfc,node,int	在 node 上部署 sfc 的第 i 个 nf
__deploy_nf_scale_out(name,resoruce,node,vnf_types)	str,{} ,node,vnf_types	在 node 上部署 nf，资源不够则扩大资源
deploy_nf_scale_out (sfc,node,i,vnf_types)	sfc,node,int,vnf_types	在 node 上部署 sfc 的第 i 个 nf，资源不够则扩大

		资源
__remove_nf(name,resource,node)	str,{},node	移除 node 上的 nf 占用的资源
remove_nf(sfc, i)	sfc,int	移除已经部署的 sfc 的第 i 个 nf
__deploy_link(network, node1,node2,att)	network,node,node,{}	在链路(node1,node2)上部署虚拟链路
deploy_link(sfc,i,network,nodes)	Sfc,int,network,[node]	部署 nf 的第 i 条链路(应该是先确定了节点位置，才确定链路位置)
__remove_link(network,node1, node2,att)	network,node,node,{}	移除网络(node1,node2)上的虚拟链路
remove_link(sfc,i,network)	sfc,int,network	移除 sfc 已经部署的第 i 条虚拟链路
remove_sfc(sfc,network)	Sfc,network	
show()		显示记录的部署情况

dynamic_scheduler 类

1 类概述

dynamic_scheduler 继承 scheduler 类，将符合部署条件(部署时间条件的)sfc

中的 nf 加入到 node 的 vnf 中，虚拟链路添加到 edge 中、同时实现记录功能(只记录生命周期之内的 sfc)，在此基础之上加入对历史 sfc 的记录(记录那些超过生命周期的 sfc)

nf 记录方式：{1:node,2:node,…}，通过字典的方式记录网络功能对应部署的节点

虚拟链路记录方式：{1:[node1,node2,…,nodeN],2:[noden,nodeN+1,…]…}，通过节点数组的方式记录虚拟链路经过的节点(应为一条虚拟链路可能经过多个节点，如中间的中继节点)，其中前一条虚拟链路最后一个节点必须是后一条虚拟链路起点

scheduler 增加基本属性计时属性 T，记录当前周期（由于动态 sfc 一般都是周期部署的），每当经过一个周期后，T 自增

动态调度器通过对节点和链路占用资源的方式增加对增加对迁移的支持，提供 occupy_node_resources()和 occupy_edge_resources()方法模拟迁移对资源占用过程。同时资源占用被记录到 node_occupy_records，edge_occupy_records 中。

- 节点资源占用记录方式：

[{‘id’:’node1’,’name’:’vnf1’,’atts’:{‘cpu’:10,’memory’:10,…, },’time’:10,’remain_time’:10}]

如果’name’属性为空，则表示占节点资源，而不是占节点部署 vnf 资源

- 链路资源占用记录方式：

[{‘id’:[’node1’,’node2’],’atts’:{‘bandwidth’:10 },’time’:10, ’remain_time’:10}]

scheduler 类应该是利于继承扩展的，所以只实现基础功能，后续使用者可以继承此类创建出多种调度器，实现各种算法，如 radom_scheduler（随机调度）、shortest_path_schedular（最短路径调度）等。

2 类属性

名称	描述	类型	默认值
__record	记录 sfc 部署情况	{sfc_id:{‘node’:{1:node,…}, ‘edge’:{1:[node_id,node_id,…],…} ‘other’:{start_time: ,duration: ,}},…}	0
log	是否输出部署信息	Bool	Ture
__history_record	记录 sfc 部署情况	[{sfc_id:{‘node’:{1:node,…}, ‘edge’:{1:[node_id,node_id,…],…} },…}](数组索引表示 sfc 结束时间)	0

__node_occupy_records	记录节点占用情况	[{‘id’:’node1’,’name’:’vnf1’,’atts’:{‘cpu’:10,’memory’:10,…, },’time’:10,’remain_time’:10}]	[]
__edge_occupy_records	记录链路占用情况	[{‘id’:[’node1’,’node2’],’atts’:{‘bandwidth’:10 }, ’time’:10},’remain_time’:10]	[]
T	周期数	int	0

3 类方法

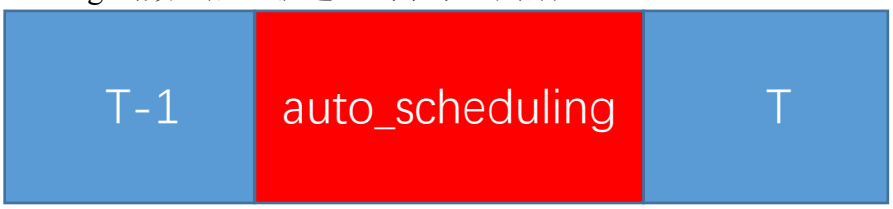
名称	参数类型	描述
__init__		
scheduler 类原有方法(红色表示此方法已重载)		
get_record()		获取记录
__deploy_nf(name,resoruce,node)	str,{} ,node	在 node 上部署 nf
deploy_nf(sfc,node,i)	sfc,node,int	在 node 上部署 sfc 的第 i 个 nf
__deploy_nf_scale_out(name,resoruce,node)	str,{} ,node	在 node 上部署 nf，资源不够则扩大资源
deploy_nf_scale_out (sfc,node,i)	sfc,node,int	在 node 上部署 sfc 的第 i 个 nf，资源不够则扩大资源
__remove_nf(name,resource,node)	str,{},node	移除 node 上的 nf 占用的资源
remove_nf(sfc, i)	sfc,int	移除已经部署的 sfc 的第 i 个 nf
__deploy_link(network, node1,node2,att)	network,node,node,{}	在链路(node1,node2)上部署虚拟链路
deploy_link(sfc,i,network,nodes)	Sfc,int,network,[node]	部署 nf 的第 i 条链路(应该是先确定了节点位置，才确定链路位置)
__remove_link(network,node1, node2,att)	network,node,node,{}	移除网络(node1,node2)上的

		虚拟链路
remove_link(sfc,i,network)	sfc,int,network	移除 sfc 已经部署的第 i 条虚拟链路
show()		显示记录的部署情况
新方法		
get_history_record()		获取历史记录
remove_sfc(uuid)		清空当前部署的 id 为 uuid 的 sfc
auto_scheduling()	自动调度，每个周期开始都应该执行该函数，功能如下： 1 用于自增 T 2 释放占用时间到期的资源 3 删除服务到期(有变动)的 sfc、删除记录、添加历史记录	
get_node_occupy_records()	获取节点占用记录	
get_edge_occupy_records()	获取链路占用记录	
occupy_node(network,node,name,att,time,vnf_types)	Netwrok 类,str,str,{},int,vnf_types	占用节点资源 time 个周期，如果不提供 name 则表示占用节点资源，也不许提供 vnf_types
occupy_edge(network,edge,atts,time)	Netwrok 类,[str,str…] ,{},int	占用链路资源 time 个周期
show_occupy_records()		显示占用的记录
show_histroy_records()		显示历史记录情况
show_sfc_reocrds()		显示所有已部署 sfc
custom_process		自定义处理

关于 auto_scheduling 说明

auto_scheduling 发生时间

auto_scheduling 发生在上一周期结束到下一周期的开始，及执行过了 auto_scheduling 函数之后，就进入到了下一周期。



auto_scheduling 调度流程

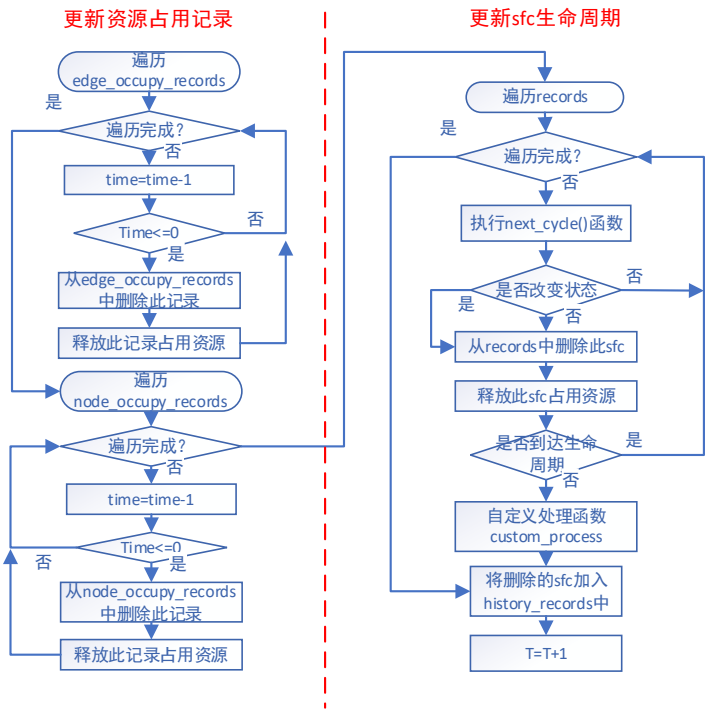
调度流程如下：

1. 释放占用的节点边资源

遍历 node_occupy_records 和 edge_occupy_records，检查 time 属性并减一，如果减少到 0，则释放对应资源，删除这条记录

2. 删除服务到期(有变动)的 sfc、删除记录、添加历史记录

执行 sfc 对应 next_cycle()函数，如果状态改变(静态 sfc 永远不会发生状态改变)并达到生命周期，调度器会简单的取消这条 sfc 的部署，如果状态改变没有达到生命周期，调度器也会取消这条链的部署，同时执行用户自定义的 custom_process 函数，如果状态没有改变，也不用改变。最后调度器进入下一周期。


更新资源占用记录
遍历
edge_occupy_records
是
遍历完成?
否
time=time-1
Time<=0
否
是
从edge_occupy_records
中删除此记录
释放此记录占用资源
遍历
node_occupy_records
遍历完成?
否
time=time-1
Time<=0
否
是
从node_occupy_records
中删除此记录
释放此记录占用资源
更新sfc生命周期
遍历records
是
遍历完成?
否
执行next_cycle()函数
是否改变状态
否
是
否
从records中删除此sfc
释放此sfc占用资源
是否到达生命
周期
是
否
自定义处理函数
custom_process
将删除的sfc加入
history_records中
T=T+1