**TDRS系统开发设计报告**

**浙江大学计算机学院**

**2013-04-18**

修订记录

| 日期 | 修改描述 | 作者 |
| --- | --- | --- |
| 2013-04-18 | 初稿模板 | 陈建海 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**目录**

## 引言

## 概述

本文档用来记录TDRS设计编码阶段需要说明，包括设计编码规范，设计框架等一些需要遵循的规则。设计编码中的一些细节说明。

## 术语

## 参考资料

## 开发准备

## 环境

基于windows开发，选用vistudio.net 2008.

然后后期，移植到linux上。

测试环境：服务器。

## 编码规范

**统一采用Linux内核编码规范，基本准则如下：**

**1、宏定义名称用大写**

如：#define DIMENSION 8

**2、结构体加前缀**

1）数据结构体定义名称加前缀st\_；

2）类结构体表示以功能调用为主的结构体，加前缀cls\_st\_；

**3、变量名用小写，多个单词用下划线隔开**

**4、注释：每个函数头，文件头，疑难程序段，尽可能用英文描述**

文件头：

/\*///////////////////////////////////////////////////////////////////////////////

@文件名/功能描述：Initplace module

@Created by Jianhai Chen at 2013-04-18

@All rights reserved.

////////////////////////////////////////////////////////////////////////////////\*/

**函数头：**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

@函数名称：<功能描述>

@参数：

@Created by### at ####-##-##

@modified by #### at ####-##-##

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**5. 类的设计规范**

类用结构体实现，成员包含属性成员变量，函数指针指向具体的函数实现，结构定义在头文件中。

样例：类cls\_st1的定义。

**头文件clsst1.h：**

typedefstruct{

int a; /\*属性成员\*/

void (\*hello)(); /\*函数指针成员\*/

} cls\_st1;

typedef cls\_st1\* cls\_st1Ptr; /\*指针\*/

static cls\_st1Ptr \_st1this;/\*定义指向当前结构体类的对象this指针\*/

void init\_st1(cls\_st1Ptr \_this);/\*用于初始化类对象的函数\*/

void st1hello(); /\*其他函数声明\*/

**文件clsst1.c:函数的实现：**

#include"clsst1.h" /\*包含类的头文件\*/

/\*初始化类对象相关函数和类的内部指针\*/

void init\_st1(cls\_st1Ptr THIS){

THIS->hello=st1hello;/\*初始化函数指针\*/

\_st1this=THIS; /\*初始化类内部指针\*/

}

/\*函数的实现\*/

void st1hello(){

printf("This is hello in clsst1!----a:%d\n",\_st1this->a);

}

/\*函数的实现\*/

void st1how(){

printf("This is hello in clsst1!----a:%d\n",\_st1this->a);

}

## 业务功能模型

见需求分析说明书。

需求变更：

物理拓扑不变，物理机之间的时延由一个时延矩阵来表示，只反映物理机之间，不考虑端口之间的时延。

物理机之间的网络带宽维度的判断，由多块网卡的总和来给出。

代码已经粗粗浏览了一下，基本完成了 PM，VM，和 VM流量关系的解析，基本完成了初始放置算法的实现。

当前的代码还存在以下问题，导致无法集成，希望能在本周整改一下，再给一个版本，以便支持双方联调：

1. 接口友好性整改：
   1. 当前华为云平台是采用python实现，可以调用C函数；当前的交付代码中，如果要调用初始放置算法，似乎需要声明类变量，对变量初始化，再调用其中的多个类方法才能实现此功能，过于复杂，不利于集成和双方的界面清晰解耦，希望将此部分全部封装成一个函数（类似   initplace(“xml1”,”xml2”,”xml3”, output) 的形式），以便华为云平台不用感知算法的内部实现。

没有问题：开放调度命令。

通过包装成函数库。

* 1. 当前的初始放置结果，是用打印的方式来输出的，接口定义过于粗糙，华为云平台完全没有办法获取此结果。

提供XML结果文件。

1. 功能补齐
   1. XML文件的解析函数当前已实现，XML文件的封装函数还没有提供，建议提供相关的XML文件封装API接口并给出一个生成XML文件的调用例子。
   2. 当前的代码实现中，没有看到维护物理机间拓扑结构的代码，这部分代码缺失。PM之间的时延应该是从物理拓扑中解析出来的，当前的实现是直接从 XML 文件中读取，与需求不符。
   3. 放置结果输出中，代码针对每组虚拟机组，只给出了一个可放置的物理机，之前的需求是针对每组虚拟机组，给出多个可放置的物理机，并按照一定的规则进行优先级排序后输出，当前实现与需求不符。

## 需求

/\*调度算法处理逻辑\*/

/\*

根据流量进行分组，有流量的放置在一起。

数据中：不包括冲突逻辑，如VM1,VM2不同板，同时又同板；

根据亲和性再次进行分组。

规则：同板、不同板的亲和性与不亲和性跟流量关系一起处理。

1）不同板的亲和性

VM1,VM2 不同板：若VM1,VM2有流量关系，则优先将其分割；

2）同板的亲和性

VM1,VM2同板，VM1,VM2有流量关系，则放置在一组；

3）不同框的亲和性

对不同框亲和性的每条边，依次判断VM所在物理板是否在同一框，如果在不同框，则不用处理，否则需要进行策略调整；

4）同框的亲和性

对同框亲和性的每条边，依次判断VM所在物理板的框，如果已经同框，不处理，否则需要进行策略调整，调整其中一个VM使其放置在不同框

5）同组的亲和性

6）不同组的亲和性

\*/

## API接口规范

**1. 云平台与DRS系统的接口规范。**

DRS系统提供函数库，供云平台调用。

调度决策的调用方式，统一由调度控制器类，提供了一个执行调度的接口。

Drscontroller:

execute(xml1, xml2)

调度请求XML

## 标准数据规范

## 调度请求标准数据：XML1.xml

**调度请求，是云平台向调度系统发出的请求信息，本文件给出了具体算法调用的格式。**

**初始放置：输入一组待放置的VM，可用的PM及其拓扑关系，输出调度决策的结果。**

**drs\_request.xml**

<?xml version="1.0" encoding="UTF-8"?>

<CONTROLOR VERSION="0.1" PREDICT="0">

<REQUEST TYPE="PLACE" ENABLE="1">

<ALGORITHM NAME="AlgInitplace">

<PARAMETER NAME="hot" TYPE="DOUBLE" VALUE="0.6"/>

<PARAMETER NAME="warm" TYPE="DOUBLE" VALUE="0.4"/>

<PARAMETER NAME="cold" TYPE="DOUBLE" VALUE="0.2"/>

</ALGORITHM>

</REQUEST>

<REQUEST TYPE="BALANCE" ENABLE="1">

<ALGORITHM NAME="AlgGreedy1">

<PARAMETER NAME="hot" TYPE="DOUBLE" VALUE="0.8"/>

<PARAMETER NAME="cold" TYPE="DOUBLE" VALUE="0.2"/>

<PARAMETER NAME="balance" TYPE="DOUBLE" VALUE=".175"/>

<PARAMETER NAME="maxcount" TYPE="DOUBLE" VALUE="200"/>

</ALGORITHM>

</REQUEST>

<REQUEST TYPE="CONSOLIDATE" ENABLE="1">

<ALGORITHM NAME="AlgConsolidate">

<!--<ALGORITHM NAME="AlgAstar">-->

<PARAMETER NAME="hot" TYPE="DOUBLE" VALUE="0.8"/>

<PARAMETER NAME="cold" TYPE="DOUBLE" VALUE="0.2"/>

<PARAMETER NAME="binpack" TYPE="DOUBLE" VALUE="3"/>

</ALGORITHM>

</REQUEST>

<REQUEST TYPE="HOTSPOTSOLVE" ENABLE="1">

<ALGORITHM NAME="AlgHotSolve">

<PARAMETER NAME="hot" TYPE="DOUBLE" VALUE="0.8"/>

<PARAMETER NAME="cold" TYPE="DOUBLE" VALUE="0.2"/>

</ALGORITHM>

</REQUEST>

<REQUEST TYPE="PREPROC" ENABLE="1">

<ALGORITHM NAME="AlgPreprocess">

<PARAMETER NAME="hot" TYPE="DOUBLE" VALUE="0.8"/>

<PARAMETER NAME="cold" TYPE="DOUBLE" VALUE="0.2"/>

<PARAMETER NAME="method" TYPE="DOUBLE" VALUE="0"/>

<!--<PARAMETER NAME="reserved\_pm" TYPE="DOUBLE" VALUE="2"/>-->

</ALGORITHM>

</REQUEST>

<!--

<REQUEST TYPE="PREDICT">

<ALGORITHM NAME="NN">

</ALGORITHM>

</REQUEST>

-->

</CONTROLOR>

## 负载数据规范：XML2.xml

包含了PM信息，VM信息。

ID的编码规则：pm+外部云平台中的PM内码；

VM的ID的编码规则：vm+外部云平台分配的VM内码，如 vm0~vm100。

STAT: 对物理机PM，STAT=0表示空载物理机，STAT=1表示上电物理机。

IP: 默认为第0块网卡的地址，如果多块网卡IP，可以用;分隔开。

<PM ID="pm0" IP="10.10.82.174" STAT="0">

<METRIC NAME="cpu" USED="0" TOTAL="20.8" RESERVED="0"/>

<METRIC NAME="mem" USED="0" TOTAL="19.7" RESERVED="0"/>

<METRIC NAME="eth0" USED="0" TOTAL="0" RESERVED="0"/>

<METRIC NAME="eth1" USED="0" TOTAL="0" RESERVED="0"/>

<METRIC NAME="eth2" USED="0" TOTAL="0" RESERVED="0"/>

<METRIC NAME="eth3" USED="13" TOTAL="100" RESERVED="0" />

<METRIC NAME="diskio" USED="0" TOTAL="100" RESERVED="0" />

<METRIC NAME="fpga" USED="0" TOTAL="100" RESERVED="0" />

</PM>

<VM ID="vm0" IP="192.168.244.140" AT="pm6" STAT="0">

<METRIC NAME="cpu" USED="2.2" TOTAL="2.2"/>

<METRIC NAME="mem" USED="4.3" TOTAL="4.3"/>

<METRIC NAME="diskio" USED="0.0725107" TOTAL="0.0725107"/>

<METRIC NAME="fpga" USED="1" TOTAL="1"/>

<METRIC NAME="eth0" USED="0" TOTAL="0"/>

<METRIC NAME="eth1" USED="0" TOTAL="0"/>

<METRIC NAME="eth2" USED="0" TOTAL="0"/>

<METRIC NAME="eth3" USED="13" TOTAL="100"/>

</VM>

调度系统根据负载情况，确定做那种类型的调度（热点消除、负载均衡、冷点整合、初始放置），然后选择具体的调度算法，给出调度决策结果。

初始放置需要决策出开启多少台PM，

1）如果PM都已经开启，VM请求直接进行放置决策。

2）

## 数据结构设计

本系统涉及的数据结构定义如下：

## 资源类型数据结构

定义及说明如下：

typedefstruct {

char metric\_name[40]; //资源维度名称

double used; // 已用量

double total; // 总量

double reserved; // 预留量

} st\_metric;

该结构用于记录物理机、虚拟机和虚拟机分组的负载信息。

## 虚拟机结构

定义及说明如下：

（在算法中未使用的数据成员不加以说明）

typedefstruct{

int id; //虚拟机的ID

OBJECTID sid; //字符串形式的虚拟机ID/名称

st\_metric load[DIMENSION]; //虚拟机的8项负载

char vnic\_deviceid[20];

char ip[30];

int link\_port;

OBJECTID at;

int at\_partid;//虚拟机所在的组号

int at\_pmid;//虚拟机所在的物理机

st\_net\_element net\_elm; /\*所属网元\*/

int TenaID;/\*所属租户ID\*/

int affinity\_type;/\*亲和类型\*/

int affinity\_at;/\*所在PM\*/

} st\_vm;

该结构用于记录虚拟机的各项信息。

## 物理机结构

定义及说明如下：

（在算法中未使用的数据成员不加粗）

typedefstruct{

**int id;//物理机ID**

**OBJECTID sid;//字符串形式的物理机ID/名称**

**st\_metric load[DIMENSION]; //物理机的8项负载类型**

char ovsmac[30]; /\*物理机的MAC信息\*/

char nicmac[PORT\_NUM][30];/\*每块实际物理网卡的MAC\*/

char hostip[30]; /\*主机IP\*/

char ip[PORT\_NUM][30]; /\*每块物理网卡的IP\*/

int stat; /\*物理机状态\*/

int rack\_id;/\*所在机架\*//\*预留字段\*/

int group\_id;/\*所在组\*/

int frame\_id;/\*所在框\*/

} st\_pm;

该结构用于记录物理机的各项信息。

## 1.4 虚拟机到虚拟机的流量

定义及说明如下：

typedefstruct{

**int id;//该组流量关系的ID**

**st\_vm\* src\_vm;//指向该流量关系中起点虚拟机的指针**

**st\_vm\* dst\_vm;//指向该流量关系中终点虚拟机的指针**

**double bandwidth; /\*预设带宽需求\*/**

**double trafficrate;//流量大小，折算为带宽需求**

double delay;/\*预留字段：表示时延要求\*/

int affinity\_type;

int affinity\_at;

OBJECTID at;

} st\_vm2vm;

该结构用于记录虚拟机之间的流量、带宽、时延等信息。

## 虚拟机分组结构

定义及说明如下：

typedefstruct {

int groupid;//该组的组号

int vm\_id\_list[MAX\_VM\_COUNT\_PER\_GROUP];//该组所有虚拟机的集合

int edge\_id\_list[MAX\_EDGE\_PER\_GROUP];//该组所有边的序号的集合

int vm\_total;//该组中虚拟机的个数

int edge\_count; //该组中边的个数

int at\_pmid;//该组所在的物理机编号

st\_metric load[DIMENSION]; //该组的负载信息

} st\_part\_vm\_group;

该结构用于记录虚拟机进行分组与切割之后所得到的虚拟机组。

## 邻接表节点

定义及说明如下：

（在算法中未使用的数据成员不加以说明）

typedefstruct{

int dst;//终点虚拟机ID

double traffic;//流量大小

}st\_adj\_list\_node;

该结构是之后将要提到的邻接表结构的一个节点。

## 邻接表结构

定义及说明如下：

（在算法中未使用的数据成员不加以说明）

typedefstruct{

int vm\_id;//当前虚拟机ID

st\_adj\_list\_node adj\_vm\_id\_list[MAX\_VM\_COUNT\_PER\_GROUP];//邻接表节点数组，即与该虚拟机有边的虚拟机以及该边表示的流量大小

int adj\_vm\_id\_count;//邻接表节点个数

} st\_adj\_vmid\_list;

该结构在分组算法中用于读取虚拟机之间是否存在联系，在切割算法中还需要从这个结构中得到流量信息并进行相应的修改与合并操作。

## 系统详细设计

## DRS面向对象框架

## 调度系统框架

基于面向对象方法设计了调度系统对象框架结构，如图1所示:



图1调度系统框架

1. **系统入口主程序main.c**

提供了一个界面实现了系统入口。

1. **Drscontroller模块**

负责调度系统的控制。实现了调度规则与策略。

1. **Dataprocess模块：**

Xmlprocessor:处理XML数据，包括：

1. 读取外部XML数据文件到共享数据结构；read\_vm\_pm\_from\_xml…
2. 从共享数据结构中读取数据转出生成XML外部文件: save\_vm\_pm\_to\_Xml…
3. **Predictor模块**

实现了负载预测功能。

1. **Scheduler模块**

实现了各调度算法相关模块。

1. **共享数据结构**

以数组或邻接表，包括调度系统接受输入的虚拟机列表、PM列表、虚拟机流量关系列表、物理机拓扑结构、物理机时延矩阵、带宽矩阵、VM放置PM的决策结果列表。

1. **公共对象操作函数库**

针对基本对象, 系统sys, 主机 host, 物理机PM, 虚拟机VM, 虚拟机组VMG,等进行操作的函数库。

## 系统分层架构

数据层：drsdata.h

数据访问层：drslib.h

业务逻辑层：调度算法

用户界面层：drsctrl.c

## 主要类对象设计

|  |  |  |
| --- | --- | --- |
| **类名称** | **类指针别名** | **功能描述** |
| cls\_st\_drscontrol | cls\_drscontrolPtr | 系统控制器类 |
| cls\_st\_xmlprocess | cls\_xmlprocessPtr | XML数据处理类 |
| cls\_st\_datagenerator | cls\_generatorPtr | 模拟数据生成类 |
| cls\_st\_scheduler | cls\_schedulerPtr | 算法调度类，负责调度执行相关调度算法 |
| cls\_st\_vm\_group | cls\_vmgroupPtr | 分组、切割算法类，负责基于流量之间的分组、切割 |
| cls\_st\_initplace | cls\_initplacePtr | 初试放置算法类 |
| cls\_st\_loadbalance | cls\_loadbalancePtr | 负载均衡算法类 |
| cls\_st\_consolidate | cls\_consolidatePtr | 整合算法类 |
| cls\_st\_hotsolve | cls\_hotsolvePtr | 热点消除算法类 |

系统操作类

## 主入口程序drsctrl.c(main)

实现系统界面窗口。

1. 全局数据定义。

Drsdata.h: 公共数据结构定义

Global.h：全局变量定义

Drsdef.h：常量数据定义

Drsfunc.h：公共函数定义

1. Main函数

提供一个界面提示窗口，供用于选择要进行的操作。

## Drscontroller(DC-控制器)

控制器接收外部云平台发过来的调度请求、负载数据、物理机拓扑数据，然后进行分析，调用相应的调度算法，并返回调度算法的决策结果。控制器实现了调度策略与调度规则。

系统给出一个调度控制器类：cls\_st\_drscontroller，封装了调度系统调度相关函数。

控制器类结构体对象：cls\_st\_drscontroller。

类结构体如下：

typedef struct {

//主要属性：

sharedataPtr sd;

//主要函数：

int (get\_sysstat\*)(); //获取（判断）系统状态；

void (\*read\_xml\_vm\_pm\_data)(); //读取XML负载数据，包括vm and pm；

void (\*read\_xml\_vm2vm)(); //读取XML虚拟机流量关系数据；

void(\*read\_xml\_pm2pm\_delay)(); //获取XML物理机拓扑关系数据；

void(\*do\_schedule)(); //调用调度算法

} cls\_st\_drscontroller;

1. 读取外部数据：如解析XML；load\_data\_from\_XMLfile(char \*filename);
2. 初始化数据；initial\_Data();
3. 更新数据；update\_data();
4. 显示决策结果：show\_result();
5. 调用决策算法；do\_schedule();

## Dataprocess(DA模块)

1、模拟数据生成

2、xml数据获取与处理

## Scheduler(SCHED模块)

调度器：

schedule.h:

typedef struct \_sched{

void (\*initplace)(); /\*do initplace\*/

void (\*hotsolve)(); /\*do hotsolve \*/

void (\*loadbalance)(); /\*do loadbalance \*/

void (\*consolidation)(); /\*do consolidation \*/

int (\*ask\_netctrl\_for\_net\_bw)(pvmgPtr p\_pvmg, pmPtr p\_pm);//成功返回1，否则返回0

} sched;

void init\_sched\_algorithm(sched \*THIS){

THIS->initplace=do\_initplace;

THIS->hotsolve =do\_hotsolve;

THIS->initplace=do\_loadbalance;

THIS->initplace=do\_consolidation;

THIS->ask\_netctrl\_for\_net\_bw =ask\_netctrl\_for\_net\_bw ;

}

## 调度算法

按照调度功能对算法进行分类，每类调度算法单独实现为一个文件，含一个头文件与实现文件。

每个调度算法的基本属性：算法类型（初始放置、负载均衡、热点消除等），算法名称，算法参数（列表）。

数据结构：算法参数，算法

st\_algparameter{

char paraname[30];

char paratype[20];

char paravalue[20];

}

## 分组切割：

分组切割类的设计：

|  |  |  |
| --- | --- | --- |
| cls\_st\_vm\_group | cls\_vmgroupPtr | 分组、切割算法类，负责基于流量之间的分组、切割 |

对虚拟机进行分组切割。

根据虚拟机之间的带宽需求，进行分组。

文件：Alg\_groupcut.h、Alg\_groupcut.c

/\*分组算法的类结构体\*/

typedefstruct{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*属性定义\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

sharedataPtr sd;/\*共享数据结构指针\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*类内部公共函数\*/

void (\*generate\_vm\_group)();/\*生成虚拟机组函数指针\*/

double (\*group\_cut)(int part\_no);/\*组的切割函数\*/

void (\*output\_parts)();/\*输出组的结果\*/

} cls\_st\_vm\_group;

/\*定义类对象的指针\*/

typedef cls\_st\_vm\_group\* cls\_vmgroupPtr;

/\*指向类对象的指针\*/

cls\_vmgroupPtr \_gcthis;

/\*初始化分组类指针\*/

void init\_vmgroup(cls\_vmgroupPtr THIS);

## 初试放置

Alg\_Initplace.h

Alg\_initplace.c

拓扑结构的表示：描述为多颗树构成的森林。

**初试放置方案设计**

**1.初试放置的目标及使用要求**

将一组具有流量关系的虚拟机，在满足容量、时延、带宽约束的前提下，决策将VM放置在一组异构的PM中，获得最小的东西流量、成本或最佳性能。

每个VM可以运行应用APP的作业JOB，占用各种物理资源。约定两点要求：1）针对通信流量问题，为简化决策规模，约定一个VM只有一个APP的JOB与外部通信，通信路径只占用物理单板的一个网口。2）保证有足够的异构PM资源能放置VM。3）两个有通信的VM放置在同一个PM，网络通信要求分配PM的同一个网口，并要求不对外产生流量开销。

给定的PM无法放置到来的VM，需要开启新的能放置VM的PM，并指定PM的规格。

PM的状态有两种：0-关闭状态，1-开启状态，2-整合下电。

**放置过程中的要求：**

1、多维度资源的约束，按照CPU, MEM,DISKIO,FPGA, ETH0~ETH3，共8个维度，其中网络资源维度ETH是逻辑网络资源的平面维度，容量单位是带宽，折算为4个网络平面的带宽。外部云平台可以根据资源分配的平面将物理网卡资源映射到各个平面，得到网络资源的容量。对于VM的网络资源需求，可以给出各个平面对应资源的需求。进行网络资源分配时，依据资源维度容量，判断是否可以放置得下。

2、带宽约束问题：通过访问控制器来决定是否能放得下；与网络控制器的接口形式，判断一个VM组是否放置得下PM，将其VM组及流量关系的信息，即VM集合与边的集合信息，一起提交给网络控制器，然后控制器根据带宽请求信息，给出能否放置得下的结果，如果成功，返回1，否则返回0。

3、时延矩阵：为每对PM的多个网络路径，给出一种综合时延，计算方法：多个路径取最小值，多个路径时延各设置一个权值，通过加权求和获得一个值。



图物理机的拓扑关系

**时延矩阵：**

**0 1 1 3 3 3 3 3 3 3**

**1 0 1 3 3 3 3 3 3 3**

**1 1 0 3 3 3 3 3 3 3**

**3 3 3 0 3 3 3 3 3 3**

**3 3 3 1 0 3 3 3 3 3**

**3 3 3 3 3 0 1 1 1 1**

**3 3 3 3 3 1 0 1 1 1**

**3 3 3 3 3 1 1 0 1 1**

**3 3 3 3 3 1 1 1 0 1**

**3 3 3 3 3 3 1 1 1 0**

**3.算法输入、处理过程与输出**

**1）算法输入**

一组虚拟机VM（小于12K），具有多个维度资源的容量属性，虚拟机之间流量大小（折算为带宽需求）和通信时延要求的约束条件。足够多的PM（小于4K）及多个维度的资源容量属性，资源维度中，CPU、MEM、DISKIO、FPGA资源，称为基本维度资源，简称CMDF资源，四块网卡维度资源，为ETH0~ETH3,称为NET资源。PM之间通过网络链接，形成特定的拓扑结构，如树或FAT-TREE等。本方案基于树和FAT-TRESS。

各维度容量说明如下：

1. CPU：单位为%，核的个数\* 100（每个核折算为100%），如16个核，容量为1600；
2. MEM：单位为MB，1GB，折算为1024MB；
3. DISKIO：单位%，范围：0~100；
4. FPGA：单位为%，范围：0~100；
5. NET：单位Mbps，规格（10M，100M，1GB，10GB），1GB折算为1024MB；

**2）处理过程**

（1）分组。按照有无流量关系进行对VM分组，得到一个VM组的集合，称为“带宽依赖VM组”或需要集中放置VM组，简称VM组。

（2）亲和性分类。根据有无放置亲和性，将VM组分为两类，按照先处理有亲和性的VM组，后无亲和性的VM组的顺序，逐个处理VM组。对PM亲和性的VM，直接加入PM部署列表。亲和性：分为同板（PM），同框，同组。**冲突如何处理？**

**同板问题：（亲和）**

**不同板问题：（互斥）**

同板一定同框。

冲突1: A与B同板，A与B互斥。

亲和性校验。

如果同板：则将流量关系设置为“无穷大”。

如果互斥：则将流量关系设置为 -1 。

没有亲和关系：则流量关系为0。

（3）选取VM组。采用贪心策略，分为随机、大组优先和小组优先的策略。

（4）选取PM。对每个VM组决策放置的PM，规则基于亲和性约束、容量、时延、带宽，采用贪心随机、大PM优先或小PM优先的策略。选取一组时延最少的PM能够放得下VM组。

以VM组的大小为基础，找出一组距离近的能放置VM组的多个PM。

（5）放置决策。如果找不到PM能放置VM组，则基于最小化东西流量进行VM组的切割，得到多个VM组，然后，将新的VM组，转（3），重新进行放置决策。

（6）VM放置。对VM组中的VM放置，按照随机顺序进行放置PM，生成放置结果。

**3）算法输出**

输出基于VM放置PM的对应关系的VM待放置列表信息。形式：

VMID=1，PMDIST=”PM01”。

效果：

1）东西流量的大小：total\_outraffic：在不同PM之间的流量；total\_itraffic：PM内部VM间的流量之和。

2）算法的执行时间。

**4.算法的性能要求**

在最大规模场景下，在初始放置和动态调度时，一个网元内所有逻辑单板的初始放置决策时间<=2s ;逻辑单板的动态放置决策时间<=300 ms。

网元的最大规模是单逻辑网元512VM，涉及300物理板。目前场景有一个网元含13物理框\*28物理板，共计364逻辑板。场景二：20框\*13板=260逻辑板。

**5. 测试场景**

给出测试场景来验证算法的性能，测试场景的规模由以下几个参数确定，不同的规模给出不同的场景。

**场景参数：**

N: VM数; E：V2V(流量关系)的数目;

M: PM数; P2P矩阵的规模[4\*M,4\*M];

**测试场景一：512VM场景。**

N=512， M=1024，E=N\*0.2(,…0.5);

P2P矩阵：[4\*M,4\*M];

**测试场景二：364VM场景。**

N=364， M=1024，E=N\*0.2(,…0.5);

P2P 矩阵：[4\*M,4\*M];

放置：

动态：冷点、热点、均衡

///////////////////////////////////////////  
2013-05-24会议纪要：

全局配置开关参数: 组内流量，组外流量；

算在哪个网卡上：

VM 通过哪个网卡往外走？动态。

4块网卡带宽的总和，作为容量，虚拟机带宽需求，四块网卡的总和。

放置算法：分组放置？

每个VM分别描述每条边的请求。

换物理机，切割

带宽：通过访问控制器来决定是否能放得下；

时延矩阵：每个路径时延权值，加权求和获得一个值；

资源维度： CPU\MEM\IO\FPGA，NET(基于块网卡的总和)。

外部输入——时延放置要求；//时延放置约束；

5月28日，电话交流

计划6月份，跑通，云平台与调度系统。

网元：按照流量关系分组

组：可以放在哪几个PM上，组内VM 与组外交互的流量

按照流量从大到小分组排序。

组的流量：组内流量计算1次流量，要么不算，算的2倍算。

组内流量：大的先放；组内流量大的先放；

组外流量：

放置过程：

选物理机策略：FF、 。

按照带宽维度计算的大小排序。

按照多个维度计算的总分大小排序。 CPU资源权重，组内流量占内存。

FPGA：1。VM以个为单位独占FPGA。

组放置到一半，重新发送请求。

什么时候，启动物理机。

//////////////////////////////////////////////////////////////////////////////////////////////

**初试放置算法实现**

**输入数据的表示**

1. **VM的CMDF资源与NET资源**
2. **PM的CMDF资源与NET资源**
3. **VM关系、带宽要求与VM的时延要求**
4. **PM网口间的时延距离**
5. **PM网口间的带宽容量**

**算法相关功能的支持**

1. 最小东西流量支持

给出一个分组算法：对所有VM根据流量关系，构建成图，如果没有关系，自成一个图，将虚拟机分成若干个连通图，每个图划分为一组。然后，基于分组进行放置决策。

算法一：最大化流量权重边的分组。VM看做顶点，流量关系看做边，只要有边相连，则加入同一个组。

算法二：基于容量的分组：在一个给定的VM组中，选取流量边权重最大的多个VM构成的子组，使得这个子组的容量不超过给定容量约束。

1. 基于VM分组的放置决策支持

放置过程主要包括两个阶段：VM组的选取和PM的选取决策

**阶段一：VM分组选取**

选取一个VM分组，对该分组，选取PM进行相关约束条件的判断，最终决策出每个VM对应放置的PM。

**阶段二：PM选取**

基于三个约束：

* 1. 基于容量约束的PM选取算法
  2. *基于时延约束的PM选取算法*





从PMLIST， P2PLIST

* 1. 基于带宽约束的PM选取算法

接口：Canpack\_by\_bandwidth(VM组与边)

ask\_netcontrol\_for\_bandwidth()

2

***约束：dictPreplaceSrc结构中源虚拟机所在单板OVS的MAC信息需唯一，即dictPreplaceSrc结构中一个虚拟机只有一个所在单板OVS的MAC信息***

**输出：**

VM与PM放置结果信息。

VM放置的数据结构：

**放置策略：**

1）MAX\_MAX策略。VM组从大到小排序依次做放置决策，大PM优先选取策略。

2）MAX\_MIN策略。大VM组优先被放置，小PM优先被选取策略。

3）MIN\_MIN策略。小VM组先放置，小PM优先被选取的策略。

优化指标：最小东西流量

**算法流程：**

第一步：选取一个组，计算资源需求总量，基本维度资源需求、组内带宽需求、组外带宽需求；

第二步：基于基本维度资源需求，对所有PM进行遍历，选取PM做预放置处理；

第三步：判断是否存在能放置的PM。如果存在PM能放置，则按照FF策略，用第一个能放入的PM，放入该VM组，转第五步；如果不存在PM可以放置，则说明当前VM组资源需求太大，则需要放置在多台PM中；转第四步；

第四步：遍历PM拓扑树的高层节点，依次遍历1层，2层节点，获取多台PM构成的PM组，将该VM组通过最小东西流量的切割分为两个组，记录组外流量，组内流量，然后对每个组放置在该PM组。

第五步：判断该VM组的外部网络带宽需求是否满足。对每个具有外部带宽流量的VM，获得与该VM通信的其他VM，取得其放置所在的PM及网口信息，遍历当前PM单板的各网络端口，选取剩余容量最大的网络端口，将该VM的网络放置端口设置为该网口，同时，将该端口信息与另一个VM所放置的PM及网口，向网络控制器发送请求。如果控制器返回可放置信息，则将该VM放入该PM，配置为当前网络端口，否则换当前PM的另一端口，继续判断是否可放置，如果不存在路径可放，则换另一PM继续进行。

第六步：VM组放置完毕。转第一步，继续。直到所有的VM组放置完毕为止。

亲和性放置：

VM具有PM亲和性的，直接放入指定PM。

条件：PM是否有足够资源？是否存在有效的网络带宽资源？

分析：

基本资源需求：cpu,mem,diskio,fpga

基于流量的网络资源维度的需求：

从给定的PM提供多个网络端口中，确定一个可满足VM通信带宽需求的端口，将VM网络连接配置为该端口。

**初试放置基本策略：分两步。**

1. 基于基本维度资源需求目标的放置：根据基本资源需求确定是否能放置进去；
2. 基于网络带宽维度需求目标的放置：查看PM各网络端口的带宽容量情况，选择能够放置该VM带宽需求的端口，确定该配置给那块物理网卡；VM的带宽需求分为两部分：内部带宽需求（该VM同本物理机中的其他VM之间的带宽需求），外部带宽需求（该VM与其他物理机中的带宽需求）。外部带宽需要配置给具体的单板的物理网络端口。内部带宽不占用所在物理单板的网络带宽。

**选择PM的策略：**

**初试放置原理：**

输入一组VM虚拟机序列，每个VM具有一定的资源容量，将其放置到一组物理机中。

1、同构异构问题

1）同构下：目标在满足资源容量放得下的前提下，使物理机的数量尽可能少；

2）异构下：

（1）成本目标：在满足容量限制的情况下，使所耗物理机资源的成本最少；

（2）性能目标：在满足容量限制的情况下，给定能够放得下的所耗物理机

资源，各物理机能达到负载均衡；

**2、基本要求：**

1）虚拟机除满足自身负载资源需求（CPU、mem、磁盘IO），还要满足网络流量需求，VM之间存在通信，使相互通信的虚拟机能够集中放置——分组。

2）物理机之间存在网络上的通信距离：距离的远近由经过的路由器节点的个数确定；部署的目标要求通信开销最小，即传输给定的数据，所有的时延最小（时延=数据量/带宽）

3）部署原则：两个具有频繁通信的VM，被放置在两个不同的PM，要求选取PM时使PM之间的

距离要最小。

4）PM拓扑网络表示为一棵树，每个节点是PM或交换机。

PM上不同的网络端口，可能被连接到不同的交换机，不同PM之间的网络端口，存在一条路径；路径的性能指标：路径时延，经过该路径上交换机的个数；用Link连接表示相邻两个端口之间的联接。

3、相关算法：

1）分组算法

根据VM及关联，对VM进行分组；根据VM之间的亲和性，对VM进行分组；

2）放置算法

选择VM策略：

**随机策略：随机选取一个VM组，进行放置；**

**大VM组先放置的策略：**将VM组按容量需求从大到小排序，FFD；

**小VM组先放置的策略：**将VM组按容量需求从小到大排序，进行放置；

选择物理机的策略：

**最短距离策略：按照物理机的距离，选取一组距离相近的物理机；**

容量策略：选取能够放得下的物理机；

小物理机优先策略：为能使到来的需求量的VM放得下，选择容量小的先放下；

最优时延或最小东西流量的策略：

1）分组；2）按组FFD；3）多个物理机按最短距离选取；4）将VM逐一放入PM；

# 分组切割及初始放置功能说明

## 数据结构说明

在实现分组切割以及初始放置的过程中使用到的数据结构体如下：

### 1.1 资源类型数据结构

定义及说明如下：

typedefstruct {

char metric\_name[40]; //资源维度名称

double used; // 已用量

double total; // 总量

double reserved; // 预留量

} st\_metric;

该结构用于记录物理机、虚拟机和虚拟机分组的负载信息。

### 1.2 虚拟机结构

定义及说明如下：

（在算法中未使用的数据成员不加以说明）

typedefstruct{

int id; //虚拟机的ID

OBJECTID sid; //字符串形式的虚拟机ID/名称

st\_metric load[DIMENSION]; //虚拟机的8项负载

char vnic\_deviceid[20];

char ip[30];

int link\_port;

OBJECTID at;

int at\_partid;//虚拟机所在的组号

int at\_pmid;//虚拟机所在的物理机

st\_net\_element net\_elm; /\*所属网元\*/

int TenaID;/\*所属租户ID\*/

int affinity\_type;/\*亲和类型\*/

int affinity\_at;/\*所在PM\*/

} st\_vm;

该结构用于记录虚拟机的各项信息。

### 1.3 物理机结构

定义及说明如下：

（在算法中未使用的数据成员不加以说明）

typedefstruct{

int id;//物理机ID

OBJECTID sid;//字符串形式的物理机ID/名称

st\_metric metricArr[DIMENSION]; //物理机的8项负载

char ovsmac[30];

char nicmac[PORT\_NUM][30];

char hostip[30];

char ip[PORT\_NUM][30];

int stat;

int rack\_id;

int group\_id;

int frame\_id;

} st\_pm;

该结构用于记录物理机的各项信息。

### 1.4 虚拟机到虚拟机的流量

定义及说明如下：

（在算法中未使用的数据成员不加以说明）

typedefstruct{

int id;//该组流量关系的ID

st\_vm\* src\_vm;//指向该流量关系中起点虚拟机的指针

st\_vm\* dst\_vm;//指向该流量关系中终点虚拟机的指针

double bandwidth; /\*预设带宽需求\*/

double trafficrate;//流量大小，折算为带宽需求

double delay;/\*预留字段：表示时延要求\*/

int affinity\_type;

int affinity\_at;

OBJECTID at;

} st\_vm2vm;

该结构用于记录虚拟机之间的流量、带宽、时延等信息。

### 1.5 虚拟机分组结构

定义及说明如下：

（在算法中未使用的数据成员不加以说明）

typedefstruct {

int groupid;//该组的组号

int vm\_id\_list[MAX\_VM\_COUNT\_PER\_GROUP];//该组所有虚拟机的集合

int edge\_id\_list[MAX\_EDGE\_PER\_GROUP];//该组所有边的序号的集合

int vm\_total;//该组中虚拟机的个数

int edge\_count; //该组中边的个数

int at\_pmid;//该组所在的物理机编号

st\_metric load[DIMENSION]; //该组的负载信息

} st\_part\_vm\_group;

该结构用于记录虚拟机进行分组与切割之后所得到的虚拟机组。

### 1.6 邻接表节点

定义及说明如下：

（在算法中未使用的数据成员不加以说明）

typedefstruct{

int dst;//终点虚拟机ID

double traffic;//流量大小

}st\_adj\_list\_node;

该结构是之后将要提到的邻接表结构的一个节点。

### 1.7 邻接表结构

定义及说明如下：

（在算法中未使用的数据成员不加以说明）

typedefstruct{

int vm\_id;//当前虚拟机ID

st\_adj\_list\_node adj\_vm\_id\_list[MAX\_VM\_COUNT\_PER\_GROUP];//邻接表节点数组，即与该虚拟机有边的虚拟机以及该边表示的流量大小

int adj\_vm\_id\_count;//邻接表节点个数

} st\_adj\_vmid\_list;

该结构在分组算法中用于读取虚拟机之间是否存在联系，在切割算法中还需要从这个结构中得到流量信息并进行相应的修改与合并操作。

## 主要函数说明

### 2.1 邻接表生成函数

函数声明如下：

void create\_adj\_vm\_list();

该函数的作用是从记录虚拟机到虚拟机流量的数组 g\_v2vlist 中获取所有流量信息，为实行分组算法做准备。

### 2.2 分组算法函数

函数声明如下：

void part\_component();

该函数的作用是遍历所有虚拟机节点，找出其中没有访问标记的节点，然后用深度优先搜索算法（DFS）找出与该虚拟机直接或间接有流量关系的虚拟机节点，并将这些节点合为一组，并设置访问标记。当所有虚拟机都被访问过之后，结束算法。

### 2.3 虚拟机组装箱算法函数

函数声明如下：

void pack\_group(int part\_no);

该函数的作用是找到能装下该组虚拟机并且VOL指标最大（即最适合装该组虚拟机）的物理机，并将该组内的全部虚拟机放入找到的物理机。如不存在任意一台物理机能够放置该组虚拟机，则调用2.4中将要提到的分组切割算法，将当前组分割为两个组，然后对分割结果中的第一组重复进行以上操作，直到被切割的足够小并能放入某一物理机为止。

注：VOL指标（part\_no为该虚拟机组的ID，i为物理机ID），只考虑前四个基本负载维度。



### 2.4 分组切割算法函数

函数声明如下：

double group\_cut(int part\_no);

该函数使用了计算无向图最小割的Stoer-Wagner算法。该算法的基本思想是：

1.最小割值记为min（其初始值为INF，无穷大），并固定一个顶点P。

2.将记录每个点的总流量的数组wage[i]初始值设置为0，然后找出和P点相连的所有顶点，将其对应的wage值增加P到该顶点的流量值。

3.找出当前wage值最大且未访问过的的顶点U，然后找出和U点相连的所有顶点，将其对应的wage值增加U到该顶点的流量值。重复该步骤直到该组中所有的顶点都已经被访问过。在这个过程中，需要记录最后访问的两个顶点S与T，并将T从该组中切除。

4.计算最后访问的顶点T的切割值（即与此顶点相连的所有边权和，即流量总和wage[T]），若该流量总和比min小，则更新min，同时将目前已从该组中切除的所有顶点加入新的分组。

5.合并最后访问的两个顶点S,T为一个顶点（这样的操作并不会影响最终的切割结果）。

6.转到2，S与T合并N-1次后结束

7.min即为所求的最小割，即分割出的两个组的最小东西流量。

基于这种算法，我们将一个虚拟机组切割成了具有最小东西流量的两个组。

### 2.5 输出函数

函数声明如下：

void output\_parts(VMGPtr ppart);

该函数将初始放置的结果进行输出。输出以分组算法和分组切割算法所得到的虚拟机组为单位来输出。首先输出总分组数，然后输出每一组所包含的虚拟机编号、流量关系、负载以及该组所在的物理机。

## 处理流程

/\*调度算法处理逻辑\*/

/\*

根据流量进行分组，有流量的放置在一起。

数据中：不包括冲突逻辑，如VM1,VM2不同板，同时又同板；

根据亲和性再次进行分组。

规则：同板、不同板的亲和性与不亲和性跟流量关系一起处理。

1）不同板的亲和性

VM1,VM2 不同板：若VM1,VM2有流量关系，则优先将其分割；

2）同板的亲和性：

VM1,VM2同板，VM1,VM2有流量关系，则放置在一组；

3）不同框的亲和性

对不同框亲和性的每条边，依次判断VM所在物理板是否在同一框，如果在不同框，则不用处理，否则需要进行策略调整；

4）同框的亲和性

对同框亲和性的每条边，依次判断VM所在物理板的框，如果已经同框，不处理，否则需要进行策略调整，调整其中一个VM使其放置在不同框

5）同组的亲和性

6）不同组的亲和性

\*/

从文件“xml2.xml”中读取物理机和虚拟机的信息。

1. 从文件“xml22.xml”中读取虚拟机之间的流量关系。
2. 调用邻接表生成函数，根据是否有流量关系构建一个虚拟机邻接表。
3. 调用分组算法函数，根据有无流量关系将虚拟机分为若干个组。
4. 调用虚拟机组装箱算法函数放置虚拟机组，若虚拟机组无法放下则调用分组切割算法函数，并对新分组重复以上操作，直到该虚拟机组可以被放置到某一物理机。

调用输出函数，按组输出虚拟机放置结果。

## 负载均衡

Alg\_loadbalance.h

Alg\_loadbalance.c

## 热点消除

输入一组VM，流量关系VM2VM，PM，及PM之间的时延关系，VM2PM的部署。

## 动态整合

Alg\_consolidate.h

Alg\_consolidate.c

## 代码文件组织

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **目录文件名和级别** | | |  | **备注说明** |
| **tdrs\_src** |  | | 根目录 |  |
|  | drsctrl.c | | 调度器主文件,含main() |  |
|  | drsdef.h | | 系统公共定义文件，extern全局变量，在主程序文件中引入 |  |
|  | drsglobal.h | | 系统公共文件，定义在主程序中用到的全局变量 |  |
|  | drscontroller.h | | 控制器结构体封装文件 |  |
|  | drscontroller.c | | 控制器结构体函数实现文件 |  |
|  | **Scheduler** | |  |  |
|  |  | alg\_groupcut.h  alg\_groupcut.c | 分组切割算法及实现 |  |
|  |  | alg\_initplace.h, alg\_initplace.c | 初试放置算法及实现 |  |
|  |  | alg\_loadbalance.h, alg\_loadbalance.c | 负载均衡算法及实现 |  |
|  |  | alg\_hotsolve.h  alg\_hotsolve.c | 热点消除算法及实现 |  |
|  |  | alg\_consolidate.h  alg\_consolidate.c | 动态整合算法及实现 |  |
|  |  | … |  |  |
|  |  | … |  |  |
|  |  | … |  |  |
|  | **Include** | | 公共头文件 |  |
|  |  | drs\_data.h | 数据结构定义文件 |  |
|  |  |  |  |  |
|  | **Datagenerator** | | 模拟数据生成器 |  |
|  |  | **generator.h**  **generator.c** |  |  |
|  |  |  |  |  |
|  | **Xmldataprocess** | | 数据处理 |  |
|  |  | **xmlprocess.h**  **xmlprocess.c** |  |  |
|  |  |  |  |  |
|  | **predict** |  | 预测模块 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

根目录：tdrs\_src

drsctrl.c: 调度器主文件,含main()

drs\_def.h——系统公共定义文件，全局变量

drscontroller.h——控制器结构体封装文件

drscontroller.c——控制器结构体函数实现文件

子目录

(1)scheduler ——调度器目录

初试放置

alg\_initplace.h, alg\_initplace.c

负载均衡

loadbalance.h, loadbalance.c

热点消除

alg\_hotsolve.h, alg\_hotsolve.c

整合算法

alg\_consolidate.h, alg\_consolidate.c

(2)datagenerator——模拟数据生成器

data 中间保存数据

--input：输入数据文件所在路径。

--output：决策结果输出文件路径。

(3)predict——预测

(4)include——公用模块

存放公用模块，如基本数据结构定义文件，系统配置文件。

drs\_data.h——基本数据结构定义文件

## 数据结构设计

虚拟机：

/\*5)define the data structure of logic board

(which is also named virtual machine)\*/

typedefstruct{

int id;

OBJECTID sid; //denote string id or name for VM

st\_metric metricArr[DIMENSION]; /\*metricArray\*, \*denote vmCpu, vmMem, vmDiskio, vmFpga.\*/

char vnic\_deviceid[20];/\*only one VNIC for this VM\*/

char ip[30]; /\*ip address of VM\*/

int link\_port; /\*vm port\_no for linking to other vm port\*/

OBJECTID at; //<!—放置位置--> if s\_at is null then means for initial placement

int at\_pmid;//The PM where VM located. if null means initialplace

int neID; //The net element which VM belongs to

int TenaID; //the vm's tenant.

} st\_vm;

物理机结构：

/\*define the rescource data of physical machine\*/

/\*6)define PM\*/

typedefstruct{

int id;

OBJECTID sid;

st\_metric metricArr[DIMENSION]; /\*metricArray\*, \*denote vmCpu, vmMem, vmDiskio, vmFpga.\*/

char ovsmac[30]; /\*denote the mac of current ovs or pm \*/

charnicmac[PORT\_NUM][30]; /\*denote all mac corresponding to the PM NICs\*/

charhostip[30]; //this ip is used for access

char ip[PORT\_NUM][30]; /\*4 ip for 4 vNICs\*/

int stat; //state：-表示待初试放置，此时，ovs\_mac为空

int rack\_id; //denotes which rack the PM belongs to

int group\_id; //denotes which group the PM belongs to

int frame\_id; //denotes the frame where PM located

} st\_pm;

样例：VM之间的关系数据：XML22.XML

<?xml version="1.0" encoding="UTF-8"?>

<DATA VERSION="0.1">

<VEDEGE SRCID="1" DSTID="5" TRAFFIC="15" DELAY="1" AFFTYPE="1" AT="pm01"/>

<VEDEGE SRCID="2" DSTID="5" TRAFFIC="15" DELAY="3" AFFTYPE="1" AT="pm01"/>

<VEDEGE SRCID="3" DSTID="6" TRAFFIC="15" DELAY="1" AFFTYPE="1" AT="pm01"/>

<VEDEGE SRCID="1" DSTID="7" TRAFFIC="15" DELAY="3" AFFTYPE="1" AT="pm01"/>

<VEDEGE SRCID="2" DSTID="6" TRAFFIC="15" DELAY="3" AFFTYPE="1" AT="pm01"/>

<VEDEGE SRCID="8" DSTID="15" TRAFFIC="15" DELAY="3" AFFTYPE="1" AT="pm01"/>

<VEDEGE SRCID="8" DSTID="17" TRAFFIC="15" DELAY="3" AFFTYPE="1" AT="pm01"/>

<VEDEGE SRCID="9" DSTID="20" TRAFFIC="15" DELAY="3" AFFTYPE="1" AT="pm01"/>

<VEDEGE SRCID="11" DSTID="18" TRAFFIC="15" DELAY="3" AFFTYPE="1" AT="pm01"/>

<VEDEGE SRCID="11" DSTID="30" TRAFFIC="15" DELAY="3" AFFTYPE="1" AT="pm01"/>

<VEDEGE SRCID="16" DSTID="21" TRAFFIC="15" DELAY="3" AFFTYPE="1" AT="pm01"/>

<VEDEGE SRCID="18" DSTID="19" TRAFFIC="15" DELAY="3" AFFTYPE="1" AT="pm01"/>

<VEDEGE SRCID="19" DSTID="25" TRAFFIC="15" DELAY="3" AFFTYPE="1" AT="pm01"/>

<VEDEGE SRCID="19" DSTID="27" TRAFFIC="15" DELAY="3" AFFTYPE="1" AT="pm01"/>

</DATA>

资源额度的度量：

cpu: 核的个数 16，每个记为100%，按照1600记录容量。

## 与外部云平台接口说明

1、（浙大）定义好调度算法与云平台的API接口，并实现 PM，VM，物理拓扑和 VM流量关系的解析等API和初始放置算法API，其他API可以先定义，下一阶段再实现。

1）数据

PM列表：

VM列表：

V2V列表：

P2P列表----由网络控制器决策网络带宽。

2）调度控制器类

外部云平台直接访问调度控制器类：cls\_st\_drscontroller

2、（浙大）XML文件的解析函数当前已实现，XML文件的封装函数还没有提供，建议提供相关的XML文件封装API接口并给出一个调用例子。

3、（华为）进行PM，VM 实时数据监控功能的开发，拉通调度算法周边配合模块的开发（依赖于调度算法定义的API接口）。

## 附件

## 附件一：编码规范

**统一采用Linux内核编码规范，基本准则如下：**

1、宏定义名称用大写

如：#define DIMENSION 8

2、结构体加前缀

1）数据结构体定义名称加前缀st\_；

2）类结构体表示以功能调用为主的结构体，加前缀cls\_st\_；

3、变量名用小写，多个单词用下划线隔开

4、注释：每个函数头，文件头，疑难程序段，尽可能用英文描述

文件头：

/\*

Function: Initplace module

Created by Jianhai Chen at 2013-04-18

All rights reserved.

\*/

函数头：

/\*

Func：功能

Created by### at ####-##-##

Parameters:

\*/

/\* 1. 普通变量命名, 采用小驼峰的形式 \*/

int aSimpleVar;

/\* 2. 类、枚举类型、typedef类型名采用大驼峰的形式 \*/

class Person;

enmum Weekday { .. };

typedefint ElementType;

/\* 3. 常量名保持大写 \*/

#define BUF\_SIZE 4096

constint MAX\_LENGTH = 1024;

enum WeekDay { MON, TUE, ... };

/\* 4. 为类定义一个指针类型 \*/

class Person;

typedef Person \*PersonPtr;

/\* 5. 为长类型名字定义一个简称 \*/

typedef map<string, strign> ParamMap;

/\* 6. 语句编程风格：括号、空格 \*/

int a = b + c; // 双目运算符两边分别添加一个空格

/\* 6.1 统一指针和引用的声明方式 \*/

int \*p;

int&p;

// for if while 与 ( 之间有一个空格

// for if while 与 { 之间有一个空格，并且与其保持在同一行

// 标点符号之后要加一个空格

for (int i = 0; i < 10; i++) {

}

if (cond) {

...

} else {

...

}

while (cond) {

...

}

class Person {

...

}

void funcion(int a, int b)

{

...

}

/\* 7. 语句块之间添加空行以示区分 \*/

int a;

int b;

int c;

int d;

for (...) {

...

}

/\* 8. 注释 \*/

/\* 短注释 \*/

int count; // counter

/\* 长注释 \*/

/\* ............................. \*/

void function(int a, int b);

/\* 文件注释 \*/

/\*

\* @file ganglia.c parse ganglia xml

\* @created by tuantuan <dangoakachan@foxmail.com>

\* @modified by tuantuan at 2011.7.19

\*/

/\* 函数注释 \*/

/\*

\* @fn int connectServer(const char \*host, int port)

\* Try to onnect to server specified by host:port.

\* @param host either ip address or hostname of the server

\* @param port the port number

\* @return the socket file descriptor if sucessfully, otherwise -1

\* @created by tuantuan at 2011.7.01

\* @modified by tuantuan at 2011.7.19

\*/

/\* 9. 缩进 \*/

/\* 缩进要统一，一个tab \*/

/\* 10. 头文件 \*/

/\* 要设置头文件保护，以避免多次引用 \*/

#ifndef \_HEADERNAME\_H

#define \_HEADERNAME\_H

...

#endif

/\* 减少引用的头文件数目，以降低依赖 \*/

/\* 原则是只引用用到的头文件 \*/

/\* 头文件中不要使用using namespace 或者 using std::cout 等\*/

/\* person.h \*/

#ifndef \_PERSON\_H

#define \_PERSON\_H

#include<string>

//using namespace std; // 不要使用，以名名字空间混乱

class Person {

int age;

std::string name;

}

#endif

/\* person.c \*/

#include"person.h"// 与此文件相关的头文件，放在最前面

#include<iostream>

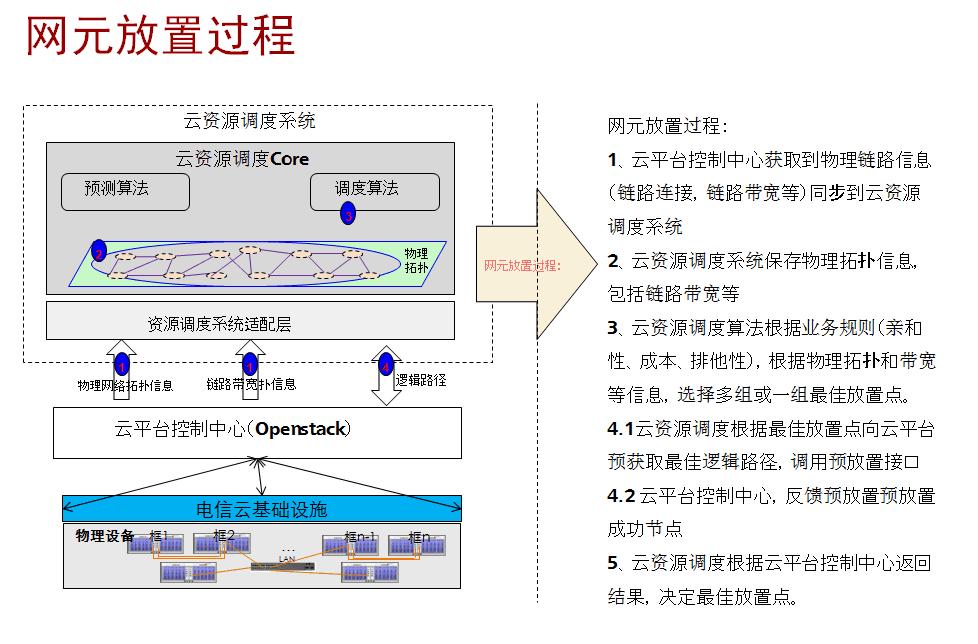
int main()

{

...

}

## 附件二：网元放置过程-华为云平台



**以放置虚拟机C为例说明接口如下（假设有4个物理机（分别是1,2,3,4），A，B虚拟机已经在运行中， A虚拟机跑在 1 物理机上，B虚拟机跑在 2 物理机上）**

**云资源调度算法认为C可以放在物理机 3 或物理机 4 上，其中要求 A/C 间带宽为500M，B/C 间带宽为 1000M，调用如下接口，将放置需求发给网络控制器。**

网元预放置接口：

curl -v -H "Content-type: application/json" -d '{"dictPreplaceSrc":{"00:00:00:00:00:00:00:01": "500"，"00:00:00:00:00:00:00:02": "1000"},"dstList":["00:00:00:00:00:00:00:03", "00:00:00:00:00:00:00:04"]}' http://10.67.128.110:8080/networkService/v1.1/tenants/ed85cbfb030344d09a429c8dca65b756/NE/preplace

其中 -d 后面是预放置json的格式，格式为：

{

"dictPreplaceSrc": {

"00:00:00:00:00:00:00:01": "500",

"00:00:00:00:00:00:00:02": "1000"

},

"dstList": ["00:00:00:00:00:00:00:03", "00:00:00:00:00:00:00:04"]

}

"00:00:00:00:00:00:00:01"为源虚拟机A所在单板OVS的MAC信息，"500"为源虚拟机A与目的虚拟机间的带宽要求，"00:00:00:00:00:00:00:02"为源虚拟机B所在单板OVS的MAC信息，"1000"为源虚拟机B与目的虚拟机间的带宽要求；

"00:00:00:00:00:00:00:03"， "00:00:00:00:00:00:00:04"为目的虚拟机C可能放置的单板OVS的MAC信息

10.67.128.110:8080为控制器(floodlight)的IP和端口信息

ed85cbfb030344d09a429c8dca65b756为租户id(tenant\_id)

约束：dictPreplaceSrc结构中源虚拟机所在单板OVS的MAC信息需唯一，即dictPreplaceSrc结构中一个虚拟机只有一个所在单板OVS的MAC信息

**网络控制器返回满足带宽要求的放置点，如下**

网元预放置接口返回值信息：

{"dstList":["00:00:00:00:00:00:00:03"]} "00:00:00:00:00:00:00:03"为目的虚拟机C能够放置的单板OVS的MAC信息

**云资源调度算法从可放置点中选择一个最终放置点，将放置请求发给云平台控制中心OpenStack。**

**OpenStack调用如下接口放置虚拟机，同时发送一个请求给网络控制器，网络控制器完成规划整网路径，将路径带宽预留给 C。**

网元放置接口：

curl -v -H "Content-type: application/json" -d '{"dictPlaceSrc":{"6a:4b:d3:2a:2f:7a": {"srcMac":"00:00:00:00:00:00:00:01","srcPort":"1","bandwidth":"500"}，"1a:29:41:c9:78:33": {"srcMac":"00:00:00:00:00:00:00:02","srcPort":"2","bandwidth":"1000"}},"dictDst":{"dstDeviceId":"92:e4:2e:48:f2:f9","dstMac":"00:00:00:00:00:00:00:03","dstPort":"3"}}' http://10.67.128.110:8080/networkService/v1.1/tenants/ed85cbfb030344d09a429c8dca65b756/NE/place

其中 -d 后面是放置json格式，具体格式为：

{

"dictPlaceSrc": {

"6a:4b:d3:2a:2f:7a": {

"srcMac":"00:00:00:00:00:00:00:01",

"srcPort":"1",

"bandwidth":"500"

},

"1a:29:41:c9:78:33": {

"srcMac":"00:00:00:00:00:00:00:02",

"srcPort":"2",

"bandwidth":"1000"

}

},

"dictDst": {

"dstDeviceId": "92:e4:2e:48:f2:f9",

"dstMac": "00:00:00:00:00:00:00:03",

"dstPort":"3"

}

}

"6a:4b:d3:2a:2f:7a"为源虚拟机A的VNIC信息，"srcMac":"00:00:00:00:00:00:00:01"为源虚拟机A所在的单板OVS的MAC信息，"srcPort":"1"为源虚拟机A的VNIC所连接的OVS上vport的端口号，"bandwidth":"500"为虚拟机A与目的虚拟机间的带宽要求

"1a:29:41:c9:78:33"为源虚拟机B的VNIC信息，"srcMac":"00:00:00:00:00:00:00:02"为源虚拟机B所在的单板OVS的MAC信息，"srcPort":"2"为源虚拟机B的VNIC所连接的OVS上vport的端口号，"bandwidth":"1000"为虚拟机A与目的虚拟机间的带宽要求

"dstDeviceId": "92:e4:2e:48:f2:f9",为目的虚拟机C的VNIC信息，"dstMac": "00:00:00:00:00:00:00:03",为目的虚拟机C所在的单板OVS的MAC信息，"dstPort":"3"为目的虚拟机C的VNIC所连接的OVS上vport的端口号

10.67.128.110:8080为控制器(floodlight)的IP和端口信息

ed85cbfb030344d09a429c8dca65b756为租户id(tenant\_id)

**网络控制器返回虚拟机占用的物理网卡的带宽信息，OpenStack保存起来，云资源调度系统可从OpenStack中获取此信息。**

网元放置接口返回值信息：

{

"bandwidthlist": [

//////////////////////////////////////////////////////////////////

{

"dictSrc": {

"srcDeviceId":"6a:4b:d3:2a:2f:7a", //虚拟机的VNIC

"srcMac":"00:00:00:00:00:00:00:01", //逻辑单板的访问MAC地址

"outPort":"4", //本虚拟机通信出口号

"outPortmac":"e0:24:7f:00:24:9a", //本虚拟机出口网卡MAC地址

"bandwidth":"500" //本虚拟机通信带宽

},

"dictDst": {

"dstDeviceId": "92:e4:2e:48:f2:f9", //目标虚拟机VNIC

"dstMac": "00:00:00:00:00:00:00:03", //目标VM所在逻辑单板MAC地址

"inPort":"6" //目标VM通信入口端口号

"inPortmac":"e0:24:7f:00:24:9c" //目标VM通信入口网卡MAC地址

}

},

//////////////////////////////////////////////////////////////////

{

"dictSrc": {

"srcDeviceId":"1a:29:41:c9:78:33",

"srcMac":"00:00:00:00:00:00:00:02",

"outPort":"5",

"outPortmac":"e0:24:7f:00:24:9b",

"bandwidth":"1000"

},

"dictDst": {

"dstDeviceId": "92:e4:2e:48:f2:f9",

"dstMac": "00:00:00:00:00:00:00:03",

"inPort":"6"

"inPortmac":"e0:24:7f:00:24:9c"

}

}

//////////////////////////////////////////////////////////////////

]

}

网元放置接口返回带宽信息的list，每个元素包括源虚拟机信息及目的虚拟机信息如下所示：

{

"dictSrc": {

"srcDeviceId":"6a:4b:d3:2a:2f:7a",

"srcMac":"00:00:00:00:00:00:00:01",-----标示PMID PM01

"outPort":"4",**-----PM01上的4号端口**

"outPortmac":"e0:24:7f:00:24:9a",**----PM01上的对应网卡MAC地址**

"bandwidth":"1000"

},

"dictDst": {

"dstDeviceId": "92:e4:2e:48:f2:f9",

"dstMac": "00:00:00:00:00:00:00:03",-----标示PMID PM03

"inPort":"6"

"inPortmac":"e0:24:7f:00:24:9c"

}

**}----表明：两个VM间的通信关系**

dictSrc为源虚拟机信息，包括"srcDeviceId":"6a:4b:d3:2a:2f:7a"为源虚拟机A的VNIC信息，"srcMac":"00:00:00:00:00:00:00:01"为源虚拟机A所在的单板OVS的MAC信息，

"outPort":"4"为源虚拟机A与目的虚拟机间通信的出接口在OVS上的端口号，"outPortmac":"e0:24:7f:00:24:9a"为源虚拟机A与目的虚拟机间通信的出接口的MAC **（物理接口）**

"bandwidth":"1000"为虚拟机A与目的虚拟机间的带宽要求

dictDst为目的虚拟机C信息，包括 "dstDeviceId": "92:e4:2e:48:f2:f9"为目的虚拟机C的VNIC信息，"dstMac": "00:00:00:00:00:00:00:03"为目的虚拟机C所在的单板OVS的MAC信息，

"inPort":"6"为源虚拟机与目的虚拟机C间通信的入接口在OVS上的端口号，"inPortmac":"e0:24:7f:00:24:9c"为源虚拟机与目的虚拟机C间通信的出接口的MAC**（物理接口）**

2012-04-12 22:35

## Libxml2库的编译和使用

http://blog.chinaunix.net/uid-25885064-id-3141923.html

Libxml2库提供了C语言解析和构造xml文档的接口，为后台C语言处理程序和前台应用程序提供了一种通用的通迅方式。

本文以libxml2-2.6.30版本来说明Libxml2库的使用方法。

1.    编译库文件

libxml2-2.6.30.tar.gz文件解压后，进入libxml2-2.6.30文件夹，顺序执行以下命令：

chmod +x ./configure

./configure

make

make install

“chmod +x ./configure”命令增加configure脚本的可执行权限；

“./configure”脚本根据当前编译系统的实际情况生成相应的makefile文件；

“make”命令执行上一命令中生成的makefile文件生成相应的目标文件；

“make install”命令主要把目标文件拷贝到/usr/local目录下，

/usr/local/lib目录下为以下库文件:

libxml2.a  libxml2.la  libxml2.so  libxml2.so.2  libxml2.so.2.6.30 pkgconfig  xml2Conf.sh

/usr/local/include/libxml2目录是Libxml库使用时需要的头文件，包含在libxml子目录下；

2.    使用Libxml2库

Libxml2库的api参考可以从http://www.xmlsoft.org/html/index.html查询。下面以解析一个简单的xml文件为例，给出一个完整的例子。

Xml文档：

<ioMsg><type>she</type><subtype><st1>123</st1><st2>563</st2></subtype></ioMsg>

C解析代码xmltest.c：

#include <libxml/parser.h>

#include <libxml/tree.h>

int main(int argc, char\* argv[])

{

 xmlDocPtr doc; //定义解析文档指针

    xmlNodePtr curNode; //定义结点指针(你需要它为了在各个结点间移动)

    xmlChar \*szKey; //临时字符串变量

    char \*szDocName;

    if (argc <= 1)

    {

   printf("Usage: %s docname/n", argv[0]);

       return(0);

    }

    szDocName = argv[1];

    doc = xmlReadFile(szDocName,"GB2312",XML\_PARSE\_RECOVER); //解析文件

    if (NULL == doc)

    {

       printf("Document not parsed successfully/n");

       return -1;

    }

    curNode = xmlDocGetRootElement(doc); //确定文档根元素

    if (NULL == curNode)

    {

       printf("empty document/n");

       xmlFreeDoc(doc);

       return -1;

    }

    if (xmlStrcmp(curNode->name, BAD\_CAST "ioMsg"))

    {

       printf("document of the wrong type, root node != ioMsg/n");

       xmlFreeDoc(doc);

       return -1;

    }

    curNode = curNode->children;

    while(curNode != NULL)

    {

       //取出节点中的内容

       szKey = xmlNodeGetContent(curNode);

       printf("Content value =%s/n", szKey);

       curNode = curNode->next;

     }

     xmlFreeDoc(doc);

    return 0;

}

3.    编译xml解析程序

假设Libxml2库是按步骤1的编译方式，其库文件和头文件分别位于/usr/local/lib和/usr/local/include/libxml2目录下。

动态库编译方式：

cc -o xmltest -I/usr/local/include/libxml2 -L/usr/local/lib -lxml2 xmltest.c

“-I/usr/local/include/libxml2”指定Libxml2库的头文件所在的路径，“-L/usr/local/lib”指定动态库所在路径。

[**【ZZ】最小割集Stoer-Wagner算法**](http://www.cnblogs.com/ylfdrib/archive/2010/08/17/1801784.html)

一个无向连通网络，去掉一个边集可以使其变成两个连通分量则这个边集就是割集；最小割集当然就权和最小的割集。

可以用最小切割最大流定理：

1.min=MAXINT,确定一个源点

2.枚举汇点

3.计算最大流，并确定当前源汇的最小割集，若比min小更新min

4.转到2直到枚举完毕

5.min即为所求输出min

不难看出复杂度很高：枚举汇点要O(n)，最短增广路最大流算法求最大流是O((n^2)m)复杂度，在复杂网络中O(m)=O(n^2)，算法总复杂度就是O(n^5)；哪怕采用最高标号预进流算法求最大流O((n^2)(m^0.5))，算法总复杂度也要O(n^4)

所以用网络流算法求解最小割集复杂度不会低于O(n^4)。

－－－－－－－－－

prim算法不仅仅可以求最小生成树，也可以求“最大生成树”。最小割集Stoer-Wagner算法就是典型的应用实例。

求解最小割集普遍采用Stoer-Wagner算法，不提供此算法证明和代码，只提供算法思路：

1.min=MAXINT，固定一个顶点P

2.从点P用“类似”prim的s算法扩展出“最大生成树”，记录最后扩展的顶点和最后扩展的边

3.计算最后扩展到的顶点的切割值（即与此顶点相连的所有边权和），若比min小更新min

4.合并最后扩展的那条边的两个端点为一个顶点（当然他们的边也要合并，这个好理解吧？）

5.转到2，合并N-1次后结束

6.min即为所求，输出min

prim本身复杂度是O(n^2)，合并n-1次，算法复杂度即为O(n^3)

如果在prim中加堆优化，复杂度会降为O((n^2)logn)

这个Stoer-Wagner算法可以参见这篇paper(http://docs.google.com/fileview?id=0BwxLvD9mcDNtMjk3MWVkMTAtZjMzNi00ZWE3LTkxYjQtYTQwNzcyZTk3Njk2&hl=en), 其核心思想是迭代缩小规模, 算法基于这样一个事实:

对于图中任意两点s和t, 它们要么属于最小割的两个不同集中, 要么属于同一个集.

如果是后者, 那么合并s和t后并不影响最小割. 基于这么个思想, 如果每次能求出图中某两点之间的最小割, 然后更新答案后合并它们再继续求最小割, 就得到最终答案了. 算法步骤如下:

1. 设最小割cut=INF, 任选一个点s到集合A中, 定义W(A, p)为A中的所有点到A外一点p的权总和.

2. 对刚才选定的s, 更新W(A,p)(该值递增).

3. 选出A外一点p, 且W(A,p)最大的作为新的s, 若A!=G(V), 则继续2.

4. 把最后进入A的两点记为s和t, 用W(A,t)更新cut.

5. 新建顶点u, 边权w(u, v)=w(s, v)+w(t, v), 删除顶点s和t, 以及与它们相连的边.

6. 若|V|!=1则继续1.

看起来很简单, 每次像做最大生成树一样选最大"边"(注意, 这里其实不是边, 而是已经累计的权值之和, 就当是加权的度好了), 然后把最后进入的两个点缩到一块就可以了. 合并点最多有n-1次, 而不加堆优化的prim是O(n^2)的, 所以最终复杂度O(n^3), 要是你有心情敲一大坨代码, 还可以在稀疏图上用Fibonacci Heap优化一下, 不过网上转了一圈, 大多都是说能用Fibonacci Heap优化到怎样怎样的复杂度, 真正能自己写出来的恐怕也没几个, 看看uoregon(俄勒冈大学)的一大坨代码就有点寒. (http://resnet.uoregon.edu/~gurney\_j/jmpc/fib.html)

特别注意几个地方, 网上的好几个Stoer-Wagner版本都存在一些小错误:

1. 算法在做"最大生成树"时更新的不是普通意义上的最大边, 而是与之相连的边的权值和, 当所有边都是单位权值时就是累计度.

2. "最后进入A的两点记为s和t", 网上对s有两种解释, 一是在t之前一个加进去的点, 二是t的前趋节点, 也就是最后选择的那条边的另一端. 正解是第一种!

3. 对于稠密图, 比如这题, 我用堆, 映射二分堆, 或者STL的优先队列都会TLE, 还不如老老实实O(n^3).

另一篇论文：

最小割 Stoer-Wagner 算法  
Etrnls 2007-4-15   
Stoer-Wagner 算法用来求无向图 G=(V, E)的全局最小割。  
  
算法基于这样一个定理：对于任意s, t V ∈，全局最小割或者等于原图的s-t 最小割，或者等于将原图进行 Contract(s, t)操作所得的图的全局最小割。  
  
算法框架：  
1. 设当前找到的最小割MinCut 为+∞   
2. 在 G中求出任意 s-t 最小割 c，MinCut = min(MinCut, c)   
3. 对 G作 Contract(s, t)操作，得到 G'=(V', E')，若|V'| > 1，则G=G'并转 2，否则MinCut 为原图的全局最小割  
  
Contract 操作定义：  
若不存在边(p, q)，则定义边(p, q)权值w(p, q) = 0   
Contract(a, b): 删掉点 a, b 及边(a, b)，加入新节点 c，对于任意 v V ∈，w(v, c) = w(c, v) = w(a, v) + w(b, v)   
  
求 G=(V, E)中任意 s-t 最小割的算法：  
定义w(A, x) = ∑w(v[i], x)，v[i] A ∈  
定义 Ax 为在x 前加入 A 的所有点的集合（不包括 x）  
1. 令集合 A={a}，a为 V中任意点  
2. 选取 V - A中的 w(A, x)最大的点 x加入集合 A   
3. 若|A|=|V|，结束  
令倒数第二个加入 A的点为 s，最后一个加入 A的点为 t，则s-t 最小割为 w(At, t)

再加一份模板

pku2914[Minimum Cut](http://124.205.79.250/JudgeOnline/problem?id=2914)

[?](http://www.cnblogs.com/ylfdrib/archive/2010/08/17/1801784.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68 | /\*pku 2914 \*/  #include<stdio.h>  #include<string.h>  #define NN 504  #define INF 1 << 30  intvis[NN];  intwet[NN];  intcombine[NN];  intmap[NN][NN];  intS, T, minCut, N;  voidSearch(){  inti, j, Max, tmp;  memset(vis, 0, sizeof(vis));  memset(wet, 0, sizeof(wet));  S = T = -1;  for(i = 0; i < N; i++){  Max = -INF;  for(j = 0; j < N; j++){  if(!combine[j] && !vis[j] && wet[j] > Max){  tmp = j;  Max = wet[j];  }  }  if(T == tmp) return;  S = T; T = tmp;  minCut = Max;  vis[tmp] = 1;  for(j = 0; j < N; j++){  if(!combine[j] && !vis[j]){  wet[j] += map[tmp][j];  }  }  }  }  intStoer\_Wagner(){  inti, j;  memset(combine, 0, sizeof(combine));  intans = INF;  for(i = 0; i < N - 1; i++){  Search();  if(minCut < ans) ans = minCut;  if(ans == 0) return0;  combine[T] = 1;  for(j = 0; j < N; j++){  if(!combine[j]){  map[S][j] += map[T][j];  map[j][S] += map[j][T];  }  }  }  returnans;  }  intmain()  {  inta, b, c, M;  while(scanf("%d%d", &N, &M) != EOF){  memset(map, 0, sizeof(map));  while(M--){  scanf("%d%d%d", &a, &b, &c);  map[a][b] += c;  map[b][a] += c;  }  printf("%d\n", Stoer\_Wagner());  }  return0;  } |

代码已经粗粗浏览了一下，基本完成了 PM，VM，物理拓扑和 VM流量关系的解析，部分完成了初始放置算法的实现，代码实现风格良好，易于扩展。

为了进一步联调的顺利执行，建议以下工作可以优先完成

1. （浙大）定义好调度算法与云平台的API接口，并实现 PM，VM，物理拓扑和 VM流量关系的解析等API和初始放置算法API，其他API可以先定义，下一阶段再实现。
2. （浙大）XML文件的解析函数当前已实现，XML文件的封装函数还没有提供，建议提供相关的XML文件封装API接口并给出一个调用例子。

3、（华为）进行PM，VM 实时数据监控功能的开发，拉通调度算法周边配合模块的开发（依赖于调度算法定义的API接口）。

**5月23日，上午10:00，罗文通电话交流。**

讨论了调度算法与外部系统的接口。

虚拟机放置，支持多个虚拟网口。

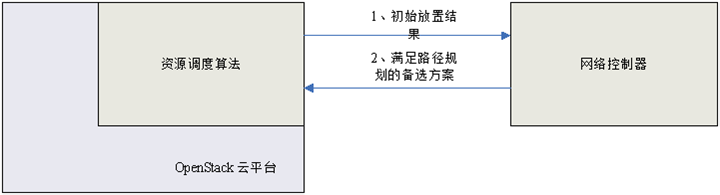
放置规则，按照8个维度来判断资源是否放置得下。

然后基于流量关系判断带宽是否满足。

6月7日，讨论

整个系统的整体架构简化如下所示，大致思路是：

OpenStack 平台（华为提供）集成 资源调度算法模块（浙大提供），初始放置时 资源调度算法模块 先计算得出初始放置结果，然后以此结果为基础与 网络控制器 交互，得到几组可行的放置方案，最后调度算法选择一种最优放置方案输出给 OpenStack 平台。



细化过程：

1、新的网元（含多个虚拟机）放置请求到来，OpenStack 平台 调用 资源调度算法模块 进行初始放置，传入的 XML 文件中含有当前的物理拓扑，当前已有的虚拟机放置状态, 待放置的虚拟机资源需求以及其中的流量需求。（原有的XML文件定义没有变化）

2、资源调度算法模块 完成虚拟机组的切割，对每一组虚拟机给出多种可能的物理机放置方案**（满足虚拟机间亲和性要求，满足 CPU/内存/FPGP/硬盘 要求）**，将此结果通过 XML文件 的方式告知 网络控制器，XML文件内容示例如下

   （VM1，VM3）--> (H3,H5,H8,H20,H34)     表示虚拟机组 （VM1，VM3）可以放置在 (H3,H5,H8,H20,H34) 任意一个host上

   （VM2，VM5）--> (H1,H2,H3,H5,H6)       表示虚拟机组 （VM2，VM5）可以放置在 (H1,H2,H3,H5,H6) 任意一个host上，但已经被放置过的host不能重复放置（如假设 H3 已被 （VM1，VM3） 占用，则（VM2，VM5）不能选用 H3，下同）？

   （VM4，VM6）--> (H4,H6,H18,H20,H32)

          VM7  --> (H3,H4,H8,H21,H30)

分组依据：

**VM组，PM组**

可以做的切割，根据流量关系得到分组，寻找PM组可以放置。

基于PM组，按照PM剩余容量从大到小排序，选择PM逐一进行放置，根据PM容量的大小，从VM组中切割一组，完成对PM1的放置，接着选择PM2，切割一组VM。

**由谁来决定VM该放在哪个PM？**

这里的H是同构还是异构？

**上述的初始放置结果，应满足虚拟机间亲和性要求，例如**

   1、要求**同板**的虚拟机必然属于同一个虚拟机组。

***如果找不到PM放入，则新开PM.***

***同板VM组优先放置。***

   2、要求**不同板**的虚拟机，其可放置的Host集合不能有交集，如 VM2 和 VM3 要求不同板，则 VM2 --> (H3,H5,H8,H20,H34) ; VM3 --> (H4,H6,H18,H20,H32) 是不允许的，因为 H20 出现在两个集合中，会导致后续的方案错误。

**不同板优先：VM2.PM!=VM3.PM。**

**放置规则：1）获取所有的互斥对的VM；2）对每个VM，选择一个PM放置，查找与其互斥的VM是否在该PM中，若存在，则选择另一个PM；否则，根据容量放入；**

3、要求**同框**的虚拟机，其可放置的Host集合都只能在同一个框内，如 VM2 和 VM3 要求同框，则  VM2 --> (H3,H5,H8,H20,H34) ; VM3 --> (H4,H6,H18,H20,H32) 这里所列出的所有 Host 必须是同框的。

**比较VM2.FRAME==VM3.FRAME.**

4、要求**不同框**的虚拟机，其可放置的**Host集合的框属性**不能有交集，如 VM2 和 VM3 要求不同框，则 VM2 --> (H3,H5,H8,H20,H34) ; VM3 --> (H4,H6,H18,H24,H32)，那么  (H3,H5,H8,H20,H34) 属于 框1和框3， (H4,H6,H18,H24,H32) 属于框2和框4，这是可以的。 如果 H5 属于框1，H6 也属于框1，则是不允许的。

**比较VM2.FRAME!=VM3.FRAME.**

  此处由网络控制器提供接口 int NetCtrl\_ApplyPathForNE("XML文件名"); 给 资源调度算法调用，其中，“XML文件名” 指明上述结果所在的 XML 文件。  此交互对应图中交互1。

3、网络控制器以上述放置方案为基础，申请相关网络资源进行路径规划，如果资源申请成功，则网络控制器返回满足流量关系的结果，如下

    方案1

   （VM1，VM3）--> H3

   （VM2，VM5）--> H1

   （VM4，VM6）--> H4

          VM7  --> H8

    方案2

   （VM1，VM3）--> H5

   （VM2，VM5）--> H2

   （VM4，VM6）--> H4

          VM7  --> H21

    方案3

   （VM1，VM3）--> H8

   （VM2，VM5）--> H1

   （VM4，VM6）--> H20

          VM7  --> H4

   如果网络规划失败，返回给资源调度算法，资源调度算法**重新做切割（此处隐含资源调度算法要保存当前的虚拟机组切割结果，重新切割的时候在现有切割方案上继续进行。建议在上述 XML 文件中定义一个会话ID进行会话识别）**。

***调度算法不保存回话。***

   此处由 资源调度算法 提供回调函数int TDRS\_ChooseBestSolution("XML文件名", int hasPath)，由网络控制器调用。如果路径规划成功，则 hasPath = 1，“XML文件名”指明可行放置方案的 XML 文件路径。如果路径规划失败，则 hasPath = 0， "XML文件名" 为空。 此处的交互对应图中交互2。

局部调整

4、如果网络规划成功，资源调度算法根据权值，在上述方案中选择一种最优方案，以 XML文件格式输出最终结果，与现有流程一致。

   特殊说明：

   在非SDN场景下，不存在网络控制器，也不需要规划路径（所有VM间的通信路径都是唯一的），此时第三步返回的放置方案为特殊值 “ALL”，表明虚拟机任意放置都可以满足流量关系，此时 资源调度算法 可在第二步的基础上自行计算出最优放置方案，并输出。

已知一组请求，进行全局决策

全局最优：

局部最优：

调度算法给出一种近似最优的决策结果，网络控制器根据带宽需求进行路径规划和分配带宽，如果遇到带宽无法满足，则记录无法放置的VM机，将这些VM信息汇总重新向调度系统进行调整决策。

调整VM放置。

2013-06-19

下述问题请老师先看看，给个答复，如果需要的话，我们再电话沟通一下。

关于XML2

1、XML2文件中针对PM和VM，每个属性的USED和TOTAL是按照使用量 (资源的实际的使用量和资源的总量) 来计算的还是按照使用率（资源实际使用量占总量的比率和资源总量的比率）。

PM:

1）TOTAL 是指资源总量，CPU 按照核的个数\*100,如8个核，总量就是800%；内存mem取16G.

2）USED：物理机实际已用的资源利用率；

VM：

1）TOTAL是指VM所占用资源的最大值；（不用）

2）used是指VM已用的资源：CPU 利用率，MEM 内存大小， FPGA，利用率；

2、针对PM和VM中的eth口的USED和TOTAL是按照什么维度来统计的？按照实际的带宽和物理网卡的带宽？网卡一般都是全双工的，每个网卡都是有收发两个方向，eth口的USED和TOTAL如何体现收发报文情况？

按照实际的带宽和物理网卡的带宽。ETH，采用利用率来。

关于XML22

1、xml22中描述的是虚拟机之间的亲和性关系，在

代码给的例子中SRCID和DSTID都是用数字表示的，如下所示：

<VEDEGE SRCID="0" DSTID="1" TRAFFIC="5" DELAY="1" AFFTYPE="1" AT="pm01"/>

但是在XML2中每个VM的ID是vm+数字，如下所示

<VM ID="vm7" IP="192.168.203.213" AT="pm6" STAT="0">

这两个xml文件中虚拟机的ID不需要统一么？

2、代码中对这个ID是不是有什么要求，我们提供的虚拟机的ID是一个字符串