# 调度系统代码框架与初试放置方案

# 调度系统设计

# 框架图

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。外部Cloud系统可以通过与main入口程序进程间通信进行交互，也可以直接访问基于C实现的面向对象接口代码，将代码与实际系统直接集成。



**图1 调度系统框架**

代码框架的核心包括6个部件。说明如下：

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. **数据处理函数库（drslib）与共享数据结构(sharedata)**

数据处理函数库，包含一些基本对象的封装，host、pm、vm、vmg、sys。

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。

# 逻辑分层系统架构

**系统分为四层：数据层、数据处理层（访问层）、业务逻辑层、用户层。**



**图2 系统分层的逻辑架构图**

# 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 |

## 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

drsGlobal.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。

typedef struct

{

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

/\*一、主要属性\*/

sharedataPtr sd;/\*指向共享数据结构的指针\*/

/\* Predict if true \*/

int predict;

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

/\*二、主要函数\*/

/\*\*\*1.XML解析有关的类\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Read xml from src (from either file or memory) \*/

int (\*readXML)(int type, char \*src, char \*dst, char \*dir);

/\* Parse xml1 to requestList \*/

void (\*parseXML1)(char \*filename);

/\* Parse xml2 to VMList and PMList \*/

void (\*parseXML2)(char \*filename);

/\* Parse xml21 to p2pList 时延-距离-矩阵\*/

void (\*parseXML21)(char \*filename);

/\* Parse xml22 to v2vList VM流量-矩阵\*/

void (\*parseXML22)(char \*filename);

/\* Parse xml3 to actionList 调度决策实施操作列表\*/

void (\*parseXML3)(char \*filename);

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

/\* Construct xml2 based on vmid2vm and pmid2pm = "data\_output.xml"\*/

void (\*constructXML2)(char \*filename);

/\* Construct xml21 based on p2pList 时延-距离-矩阵\*/

void (\*constructXML21)(char \*filename);

/\* Construct xml22 based on v2vList VM流量-矩阵\*/

void (\*constructXML22)(char \*filename);

/\* Construct xml3 based on the actionList \*/

void (\*constructXML3)(char \*filename);

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

/\*\*||||||2.生成模拟数据有关的函数||||||||||||||||||||||\*/

/\* Generate PM list data \*/

void (\*generate\_pmlist)(int pm\_num, pmPtr lstpm);

/\* Generate VM list data \*/

void (\*generate\_vmlist)(int vm\_num, vmPtr lstvm);

/\* Generate v2v list data -虚拟机流量数据\*/

void (\*generate\_v2vlist)(int vm\_num, float vr);

/\* Generate p2p list data -物理机时延距离数据\*/

void (\*generate\_p2plist)(int pm\_num);

/\* Generate all data \*/

void (\*generator\_all)(void);

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

/\*\*||||||3.调度系统有关的函数||||||||||||||||||||||\*/

/\* Set default request \*/

void (\*setDefaultRequest)();

/\*||执行调度算法\*/

void (\*exec\_sched)(int calc\_util);

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

/\*获取（判断）系统状态；\*/

int (\*get\_sysstat)();

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

/\*\*||||||其他相关函数||||||||||||||||||||||\*/

/\*控制器行为\*/

void (\*do\_action)(int action);

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；

/\* Append action list to existing list \*/

/\* Update request and data

void (\*updateRequest)(const char \*user);

int (\*updateData)(const char \*user);\*/

/\* Process all request successively \*/

void (\*processRequests)();

/\* Call algorithm \*/

/\* Debug data contained in controllor \*/

void (\*debugData)();

} cls\_st\_drscontrol;

# 初试放置方案

**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。



图物理机的拓扑关系

**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。

**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。

（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。

（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。

**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];

# 初始放置详细设计

# 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：

# 资源类型数据结构

定义及说明如下：

typedef struct {

char metric\_name[40]; // metric name

double used; // used value

double total; // total value

double reserved; // reserved value

} 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。

# 主机host结构

typedef struct{

int id; /\*Inner single coder\*/

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

char ip[30]; /\*this ip is used for access\*/

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

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

} st\_host;

用于表示vm、pm的共同属性部分。

# 虚拟机vm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）

typedef struct{

st\_host host;

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\*/

int at\_partid;

int 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

st\_net\_element net\_elm; //The net element which VM belongs to

int TenaID; //the vm's tenant.

} 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以说明）

typedef struct{

st\_host host;

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

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

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

st\_vm vmlist[MAX\_VM\_COUNT\_PER\_PM]; /\*vmlist in this PM 100\*/

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;

该结构用于记录物理机的各项信息。

# 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）

/\*VM traffic rate and affinity property\*/

typedef struct{

int id;

vmPtr p\_src\_vm;

vmPtr p\_dst\_vm;

double bandwidth;

double trafficrate;//流量大小

double delay;//时延

int affinity\_type; /\*aff\_type表示亲和性类型,包括4个值：0~4。

“0”-表示一般流量关系，不考虑亲和性；

“1”-表示定物理板属性，两个VM必须部署在同一个PM上，affinity\_at属性说明放置的具体PM；

“2”-表示同框亲和属性，affinity\_at为空，表示只要同框即可，若affinity\_at不为空，表示要放置的具体框ID；

“3”-表示同组亲和性，affinity\_at为空，表示只要同组即可，若affinity\_at不为空，表示要放置的具体组ID。

-->\*/

int affinity\_at;/\* 表示亲和性所在的PMID，框ID，或组ID\*/

OBJECTID at;/\* 表示亲和性所在的PMID，框ID，或组ID\*/

} 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：

（在算法中未使用的数据成员不加以说明）

/\*VM组，记录ID构成的组，用于基于流量关系的VM分组\*/

typedef struct {

int groupid;

int vm\_id\_list[MAX\_VM\_COUNT\_PER\_GROUP];/\*该组所有点的集合\*/

int edge\_id\_list[MAX\_EDGE\_PER\_GROUP];/\*该组的边集合,从g\_v2vlist[N]数组中对应的边的序号\*/

int vm\_total;//该组VM 的个数

int edge\_count; //该组边的个数

double itraffic;//组内流量

double otraffic;//组外流量

int at\_pmid;/\*初试放置决策后获得的pmdi\*/

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

} st\_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。

# 主要文件模块

1)alg\_initplace.c, alg\_initplace.h

2)cls\_vmgroup.c, cls\_vmgroup.h

# 主要函数说明

# 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。

# 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。

# 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。



# 物理机选取决策函数

/\*FindMinStar算法寻找适合的物理机组\*/

void find\_min\_star(int grp\_id);

基于最小时延寻找PM的方法。

void pack\_group(int part\_no,vmgPtr ppart);

将虚拟机组进行逐一安装。

# 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。

# 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

1. 从文件“xml2.xml”中读取物理机和虚拟机的信息。
2. 从文件“xml22.xml”中读取虚拟机之间的流量关系。
3. 从文件“xml21.xml”中读取PM之间的时延矩阵。
4. 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。
5. 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。然后计算每个组的组内流量。
6. 调用虚拟机组装箱算法函数放置虚拟机组，先将VM组按照内部流量大小排序。
7. 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。