Scheduler模块在openstack中的作用就是决策虚拟机创建在哪个主机上，调度仅支持计算节点.

1 主机过滤

如上图，Filter Scheduler首先得到未经过滤的主机列表，然后根据过滤属性，选择主机创建指定数目的虚拟机。

目前，openstack默认支持几种过滤策略，开发者也可以根据需要实现自己的过滤策略。在nova.scheduler.filters包中的过滤器有以下几种：

l AllHostsFilter – 不做任何过滤，直接返回所有可用的主机列表。

l AvailabilityZoneFilter – 返回创建虚拟机参数指定的集群内的主机。

l ComputeFilter – 根据创建虚拟机规格属性选择主机。

l CoreFilter – 根据CPU数过滤主机。

l IsolatedHostsFilter – 根据 “image\_isolated” 和 “host\_isolated” 标志选择主机。

l JsonFilter – 根据简单的JSON字符串指定的规则选择主机。

l RamFilter – 根据指定的RAM值选择资源足够的主机。

l SimpleCIDRAffinityFilter – 选择在同一IP段内的主机。

l DifferentHostFilter – 选择与一组虚拟机不同位置的主机。

l SameHostFilter – 选择与一组虚拟机相同位置的主机。

下面是Essex版本中可以使用的过滤器：

需要在nova.conf文件中配置以下两项：

l scheduler\_available\_filters – 指定所有可用过滤器（默认是nova.scheduler.filters.standard\_filters（一个函数），该函数返回nova.scheduler.filters包中所有的过滤器类）

l scheduler\_default\_filters – 指定默认使用的过滤器列表

如果要实现自己的过滤器，可以继承自BaseHostFilter类，重写host\_passes方法，返回True表示主机可用。然后在配置文件中的添加自己过滤器。

2 权值计算

经过主机过滤后，需要对主机进行权值的计算，根据策略选择相应的某一台主机（对于每一个要创建的虚拟机而言）。

尝试在一台不适合的主机上创建虚拟机的代价比在一台合适主机上创建的代价要高，比如说在一台高性能主机上创建一台功能简单的普通虚拟机的代价是高的。

openstack对权值的计算需要一个或多个（weight值，代价函数）的组合，然后对每一个经过过滤的主机调用代价函数进行计算，将得到的值与weight值乘积，得到最终的权值。openstack将在权值最小的主机上创建一台虚拟机。

openstack默认只有一个代价函数：

简单的返回主机剩余的内存。默认的weight值为-1.0（在配置文件nova.conf文件中是以代价函数名称加\_weight表示）。开发者可以实现自己的代价函数，设置自己的weight值来更精确的、利用更加复杂的算法选择主机。对于openstack提供的默认值来说，主机拥有的剩余内存越多，权值越小，被选择在其上创建虚拟机的可能性就越大。

3 代码分析

以创建虚拟机为例（其实scheduler模块目前也只是用在创建虚拟机）。

scheduler（nova.scheduler.manager.py::SchedulerManager类）模块收到RPC消息，调用run\_instance()函数：

如上图，在run\_instance()方法中调用了driver对象的schedule\_run\_instance()方法。那我们来看下driver对象是什么？找到SchedulerManager类的初始化函数\_\_init\_\_():

可见driver对象是配置中的scheduler\_driver字段表示的类对象。通过配置文件看到这里默认是nova.scheduler.muti.MultiScheduler类。

MultiScheduler::schedule\_run\_instance()

接着在MultiScheduler类的初始化函数中找到drivers字典的定义，发现其“compute”键所对应的值是nova.scheduler.filter\_scheduler.FilterScheduler类对象。

FilterScheduler::schedule\_run\_instance()方法做了两件事：

a) 首先调用了\_schedule()方法。该方法的功能是获取所有的计算节点主机，对每一个要创建的虚拟机进行循环：

i. 对主机进行过滤。FilterManager类继承自Scheduler类。在Scheduler类的初始化中，加载了所有可用的filter类。根据配置文件中scheduler\_default\_filters字段的定义选择默认使用的一个或多个filter。依次对每个主机调用filter类的host\_passes()方法，如果返回都为True，则主机通过过滤。

ii. 对所有通过过滤的主机计算权值。openstack默认是算法是-1\*(主机剩余的内存值)，选择权值最小的主机，即：选择剩余内存最大的主机创建虚拟机。（这里的策略即负载均衡，如果把默认的weight值改为1，策略就调整为优先选择一个主机创建，直至该主机内存不足）

iii. 对选择的主机资源进行更新，以便在下一次循环中使用最新的数据计算。

最终，\_schedule()方法会得到与创建虚拟机个数一致的主机列表。

b) 对每一个要创建的虚拟机调用：

instance = self.\_provision\_resource(elevated, weighted\_host,request\_spec, reservations, kwargs)，在该方法中：

i. 向数据库中添加虚拟机信息

ii. 向对应的compute主机发送创建虚拟机的异步RPC请求

循环结束后，返回虚拟机信息，调用链依次返回，给用户响应。

4 个人总结

之前一直说openstack的调度功能很简单，其实个人感觉openstack给了开发者最大的自由度，让开发者能够根据自己项目的需求自由定制很多环节的处理。openstack的框架很简单，因为它把复杂的部分（也是依个人需求而不同的部分）交给了开发者定制。它的接口设计、函数定制以及动态加载机制都是值得我们学习的地方。