A new mechanism for nova-scheduler : Policy-based Scheduling

(Draft)

Khanh-Toan Tran (khanh-toan.tran@cloudwatt.com)
Jérôme Gallard (gallard.jerome@gmail.com)

Blueprint: https://blueprints.launchpad.net/nova/+spec/policy-based-scheduler

1. Introduction

Nova-scheduler default driver, Filter_Scheduler, uses Filters and Weighers to help choose the host best suited for any requested VM. However, Filter_Scheduler has several limits that make it unable to make the best use of its Filter and Weigher catalog and provide business-level services to clients. Among others:

- Static policy: admin cannot change the placement policy in runtime without restarting nova-scheduler. Most of Filters and Weighers use parameters from configuration and thus also require restarting nova-scheduler.
- Lack of client context: the same filters and weighers are applied with the same parameters regardless of clients. In this situation, it is difficult to provide different qualities of services to clients who sign different contracts.
- Lack of local policy: the same filters and weighers are applied to all hosts. Even though Openstack defines aggregates for regrouping a set of hosts with similar characteristics, it still does not allow admin to define different policies for these aggregates (even with some efforts done to customize Ram/Core OverProvisioning per aggregates).

Consider the following usecases:

1. Pre-selected availability zone: For regulatory and security reasons, a company wants all their VMs to be hosted in a particular availability zone. No user from the company can create a VM outside the availability zone that the company selected.
2. Enforced service class: A company signs a contract with gold service class. With this contract its VMs will be hosted in the aggregate where all high-quality hosts are regrouped, regardless of flavors its users choose.
3. Runtime modification: Admin can select any set of filters and weighers from the catalog available in Openstack to execute without restarting nova-scheduler.
4. Local policy vs global policy: Admin wants to define a Consolidation policy in each aggregate to minimize the number of active hosts, and a global Load Balancing policy to share the workload among aggregates.

At its current state, Filter_Scheduler cannot realize these scenarios due to lack of client context (Usecase 1 & 2), dynamic policy (Usecase 3) and local policy (Usecase 4).

Therefore, there are needs for overcoming the shortcoming of Filter_Scheduler. Our objective is to provide a scheduling driver that is capable of providing:

- Dynamic scheduling: Admin could dynamically change the policy at any moment without service disruption.
- Client context consideration: The new driver would take into account the context of each and every client, so that their contracts are enforced.
- Fine granularity: Admin could define one policy per group of resources to make the best use of the latter.
- Extensible architecture: The new driver would be flexible; it could incorporate different policies determined by admin. In addition, the new driver would be generic in order to allow admin to integrate other solutions in it.

2. General design

a. Policy and Rule Concept

The scheduling decision is determined according to rules defined by the administrator. Rules allow admin to dynamically define and change objectives of the system and enforce customers' contracts. The administrator's policy can be realized by defining a set of rules.

Each rule defines an action to certain objects. A general form of a rule is composed of three parts: Target – Effect – Condition.

- Target: defines the perimeter in which the rule would be applied. It can be requests from a particular tenant, or a group of resources (e.g. an aggregate).
- Effect: defines the action that the administrator wants to apply inside the perimeter determined by Target.
- Condition: precises the condition in which the rule is applied. It can be certain time during a day, exceptional situation (e.g. maintenance), etc.

Following the type of target, the rules can be defined as "tenant rules" whose perimeter contains a particular client, or "admin rules" whose perimeter contains a group of resources.

For instance, the following rule:

```
 {
   “target”: “client_1”,
   “effect”: {“class_service”: “gold”},
   “condition”: “all”
 }
```

ensures that client "client_1" always has his VM hosted in gold area (e.g. "aggregate-gold"), while

```
 {
   “target”: “aggregate-gold”,
   “effect”: “LoadBalancing”,
  "condition": "all"
 }
```
enforces the policy LoadBalancing for all hosts in the aggregate “aggregate-gold”, i.e. the requested VM will be hosted in the less-loaded host in “aggregate-gold”.

b. Architecture

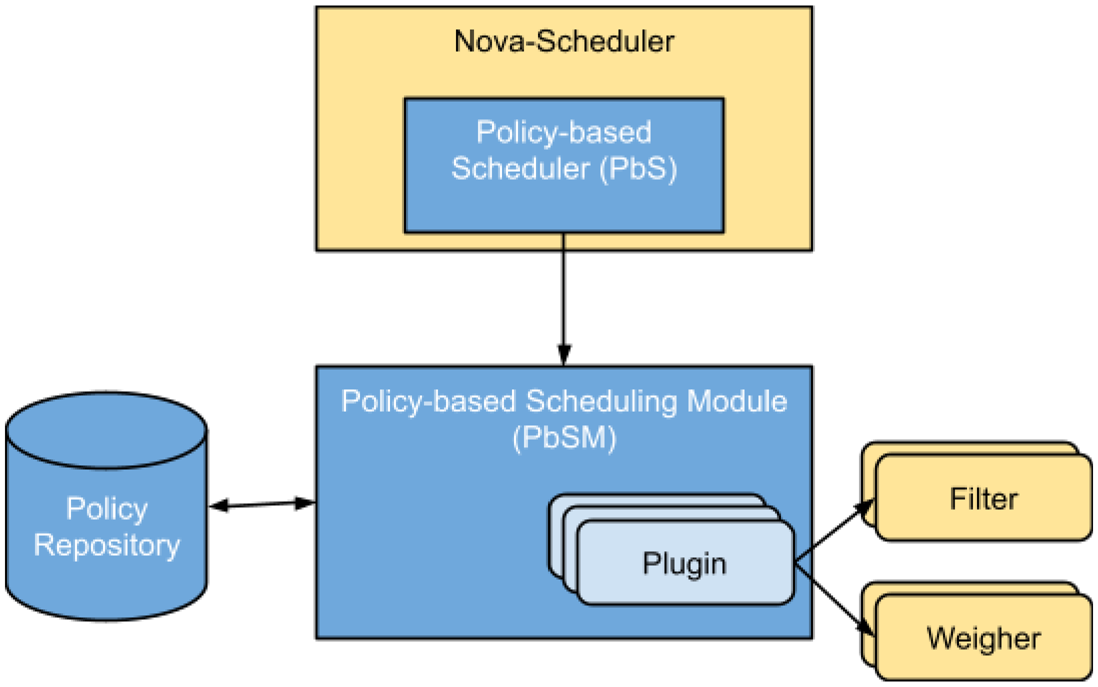
Nova-Scheduler
Policy-based Scheduler (PbS)
Policy-based Scheduling Module (PbSM)
Policy Repository
Plugin
Filter
Weigher

Figure 1: Policy-based Scheduling Module architecture

The Policy-based Scheduling Module architecture is composed of 3 components:

- **Policy Repository**: contains the rules that control the scheduling decisions. Rules can be created and updated at runtime.
- **Policy-based Scheduling Module (PbSM)**: is responsible for selecting the compute nodes to host the requested VMs. It reads rules from Policy Repository and applies rules using **Plugins**. Each plugin is capable of realizing the effect of a rule. The Plugins are capable of using the existing Filters and Weighers to fulfil the rules’ effects. A Plugin can be as simple as running a Filter to select a group of suitable hosts, or executing a combination of Filters and Weighers, or as complex as applying optimization algorithms to make the best use of the infrastructure resources. Admin can implement new provisioning solutions in form of Plugins and easily integrate them into the Policy-based Scheduling Module, making it extensible and flexible to use.
- **Policy-based Scheduler (PbS)**: is a driver in Nova-scheduler that plays the role of interface between Nova-scheduler and Policy-based Scheduling Module.

Upon receiving a user's request, Nova-scheduler calls PbSM (through PbS) to process the request. PbSM consults Policy Repository for the rules and uses the associated Plugins to schedule the requested VMs. Existing Filters and Weighers can be used in the Plugins to make the scheduling decision. The scheduling result will be eventually returned to Nova-scheduler.

3. Usecases

This section presents 4 usecases. Each of them describes a specific configuration for both users and administrators.

3.1.Availability zone according to time

In this scenario, we consider a cloud provider with two datacenters: one is placed in western hemisphere, the other is placed in eastern hemisphere (12 hours of time difference between the two datacenters). This cloud provider offers discount according to the time of the day: instance prices during the night are lower than during the day (when the workload is higher). In this scenario the datacenters are set up as availability zones.

Let's consider a company who wants to take advantage of the discount offered by the cloud provider. The company wants that their users always create instances in the western datacenter during nighttime (western time) and in the eastern datacenter during daytime to benefit from the discount.

When a company employee creates an instance, the instance has to be created in the correct availability_zone. If the employee manually specifies an availability_zone not authorized because of the time of the day, the PbSM will return a "permission denied" error.

The following table summarizes the context:

Targets	Rules	Time (western time)
company_A	availability_zone = Western	20:00 → 08:00
company_A	availability_zone = Eastern	08:00 → 20:00

Following is the development of the scenario:
The administrator defines a rule for that tenant (target = company_A):

```
[
      {  "target": "company_A",
         "effect": {"availability_zone" : "eastern"},
         "condition" : {"time": "08:00-20:00"}
      },
      {  “target”: “company_A”,
         “effect”: {“availability_zone” : “western”},
         “condition” : {“time”: “20:00-08:00”}
      }
]
```

For each instance creation requested by a user of company_A, PbSM will decide as follow:
- if no availability_zone was requested by the user, PbSM will automatically choose the correct availability_zone to deploy in, according to the “time condition”.
- if an availability_zone is specified, the PbSM will deploy the instance in the specified availability_zone if the condition is met, or return an error otherwise.

3.2.Service_class

We consider a cloud provider providing two classes of services to its customers: silver class and gold class. In this example, both classes are represented by aggregates (“aggregate-silver” and “aggregate-gold”) inside the datacenter.

We consider a client (client_B) contracting for a gold service class only: in this usecase, all VMs requested by cilent_B have to be deployed in “aggregate-gold”.

Following is the development of the scenario:

1. The administrator creates an aggregate with a metadata “gold”:

```
nova aggregate-create aggregate-gold
nova aggregate-add-host aggregate-gold <some hosts>
nova aggregate-set-metadata aggregate-gold service_class=gold
```

2. The administrator defines a rule for client_B:

```
{
       “target”: “client_B”,
       “effect” : {“service_class”: “gold”},
       “condition”  : “all”
}
```

For each instance creation requested by client_B, PbSM will deploy the instance only in the aggregate “gold”.

3.3.Runtime modification

This scenario demonstrates the use of OpenStack filters and weighers with PbSM: it allows an administrator to call filters and/or weighers directly as they would be called by filter_scheduler. With this method, filters and weighers can be added or removed dynamically without the need to restart nova-scheduler.

We consider a scenario where the admin wants to call the availability_zone_filter, the disk_filter and the ram_weigher. In that case, the admin specifies in the list of rules (target = the whole infrastructure):

```
{
      “target”: “all”,
      “effect” : {“generic_plugin”: [[“AvailabilityZoneFilter”,
      “DiskFilter”], [“RAMWeigher”]],
      “condition”  : “all”
}
```

In this scenario, all compute hosts will be processed through the availability_zone_filter, disk_filter and ram weigher before being selected to hosts the VMs. Filters and weighers are called with default parameters.

3.4. Local policy vs Global policy

Let’s consider a cloud provider who has in its infrastructure two aggregates (aggregate-1 and aggregate-2). Admin wants to apply two types of policies to his infrastructure:

- Local policy: Consolidation.
  Admin wants that inside each aggregate, resources have to be RAM-consolidated (based on the RAM, the minimum of computes have to be used inside an aggregate).
- Global policy: LoadBalancing.
  In addition, admin wants to balance the load between the two aggregates (the load balancing will be based on the available RAM of the compute hosts).

The admin defines the following rules:

```
[ {
       “target”: “aggregate-1”,
       “effect” : {“Consolidation”: “Ram”},
       “condition”  : “all”
  },
  {
       “target”: “aggregate-2”,
       “effect” : {“Consolidation”: “Ram”},
       “condition”  : “all”
  },
  {
       “target”: “all”,
       “effect” : {“LoadBalancing”: “Ram”},
       “condition”  : “all”
  }
]
```

In this scenario, the 3 rules are interpreted by PbSM: inside the aggregates, the consolidation policies are applied and between the two aggregates, the load balancing policy is applied.

4. Related Blueprints

4.1. Schedulers

- Unified Resource Placement Module for OpenStack
  https://blueprints.launchpad.net/nova/+spec/unified-rpm
  This blueprint proposes a high level scheduling module that sits on top of nova,cinder and neutron to provide scheduling service for different types of resources (volume, compute and network) at the same time.

- Add resource optimization service to OpenStack
  https://blueprints.launchpad.net/nova/+spec/resource-optimization-service
  This blueprint proposes to add a resource optimization service to monitor and balance resource. It can first create a service framework then add optimization service one by one, such as load balance policy, high availability policy, power management policy etc. Each specified policy can be a plug-in driver.

- SolverScheduler - Complex constraint based resource placement
  https://blueprints.launchpad.net/nova/+spec/solver-scheduler
  This blueprint proposes to add a new pluggable scheduler that leverages existing solver (e.g. PULP, CVXOPT, COIN_OR) to solve the placement problem with linear constraints.

- Support for multiple active scheduler drivers
  https://blueprints.launchpad.net/nova/+spec/multiple-scheduler-drivers
  This blueprints proposes to apply different scheduling policies in different host aggregates. This could be different drivers, or even a same driver with different configurations (e.g., FilterScheduler with different sets of filters/weights and/or different parameters of particular filters/weights).

4.2. Filters and Weighers

- Normalize Scheduler Weights
  https://blueprints.launchpad.net/nova/+spec/normalize-scheduler-weights
  This blueprint aims to introduce weight normalization so that one can apply multiple weighers easily. All the weights will be normalized between 0.0 and 1.0.

- Support scheduler filter per host aggregate
  https://blueprints.launchpad.net/nova/+spec/aggregate-scheduler-filter
  This blueprint's objective is to enable different scheduler filters in different aggregates.