Scheduling in Openstack
The nova-scheduler

- Nova-scheduler is the component responsible to decide which compute node should be used when a VM instance is to be launched
- It interacts with other components through queue
  - For scheduling, queue is the essential communications hub
- nova-scheduler makes decisions by collecting information about compute resources
- It has a number of configuration options that can be accessed and modified in the configuration file “nova.conf”
- FilterScheduler is the default
The nova-scheduler
The nova-scheduler

The scheduler process is divided into the following phases:

- **Getting the current state of all the compute nodes:** It will generate a list of hosts.
- **Filtering phase** will generate a list of suitable hosts by applying filters.
- **Weighting phase** will sort the hosts according to their weighted cost scores, which are given by applying some cost functions.

- The sorted list of hosts is candidated to fulfill the user's request.
The user request

2013-12-17 17:10:30.895 3032 DEBUG qpid.messaging.io.raw [-] READ[3e8ea28]:...
The nova-scheduler

- User requests are processed sequentially (FIFO scheduling)
  - nova-scheduler doesn't provide any dynamic priority strategy algorithm
- User requests not satisfied (e.g. resource not available) fails and will be lost
  - on that scenario, nova-scheduler doesn't provide queuing of the requests
- OpenStack simply provides a partitioning of resources among more projects / experiments (use of quotas)
  - if a project has free quota (underutilized its resources), and another project instead has consumed its quota, the only solution is to change the related quotas by the Cloud Administrator

Nova-scheduler is mainly missing of:

- queuing of the requests
- fair-share algorithm in the resources provisioning
### The nova-scheduler

Resources unavailable: any new request fails and is lost.
Fair-share algorithm

**The priority** is an integer and the larger the number, the higher the job will be positioned in the queue, and the sooner the job will be scheduled.

**The fair-share** is a component of the job's priority that influences the order in which a user's queued jobs are scheduled to run.

It guarantees the usage of the resources is equally distributed among users and groups by considering the portion of the resources allocated to them (i.e. share) and the resources they already consumed:

- historical resource utilization information is incorporated into priority decisions

We analyzed the fair-share algorithms implemented by the most relevant LRMS:

- Selected **SLURM's Priority MultiFactor** strategy, a sophisticated and complete fair-share algorithm
- [https://computing.llnl.gov/linux/slurm/priority_multifactor.html](https://computing.llnl.gov/linux/slurm/priority_multifactor.html)
The SLURM fair-share formula

\[ F = 2^{\frac{-U_e}{S}} \]

\( U_e \): user's effective usage

\( S \): user's normalized share

Consider account = tenant

The share values must be defined in the nova.conf
**FairShareScheduler**, a pluggable scheduler with the objective to extend the existing OpenStack scheduler by integrating a (batch like) dynamic priority algorithm has been developed by INFN-PD

- Selected the “**Multifactor Priority**” SLURM algorithm
- FairShareScheduler will assign dynamically the proper priority to every user request
- The priority at any given time will be a weighted sum of these factors (configurable): **age** and **fair-share**

  \[
  \text{priority} = (\text{PriorityWeightAge}) \times (\text{age\_factor}) + \\
  (\text{PriorityWeightVCPUFairshare}) \times (\text{fair-share\_vcpu\_factor}) + \\
  (\text{PriorityWeightMemoryFairshare}) \times (\text{fair-share\_memory\_factor})
  \]

- The weight expresses the interest for a specific factor

  - Example: you can configure fair-share-vcpu to be the dominant factor (say 80%), set the age factor to contribute 20%, and set the fair-share-memory influences to zero in the priority decision
• all user requests will be inserted in a (persistent) priority queue and then processed asynchronously by the dedicated process (filtering + weighting phase) when compute resources are available

• From the client point of view the queued requests remain in “Scheduling” state till the compute resources are available
  • No new states added: this prevents any possible interaction issue with the Openstack clients

• User requests are dequeued by a pool of WorkerThreads (configurable)
  ♦ no sequential processing of the requests

• the failed requests at filtering + weighting phase may be inserted again in the queue for n-times (configurable)

• the priority of the queued requests will be recalculated periodically (see age_factor)
FairShareScheduler: High level architecture

1. user req. (nova api)
2. AMQP
3. AMQP client
4. PriorityQueue Manager
5. FairShare Manager
6. PQueue
7. filtering + weighting phases
8. nova, keystone
9. nova-compute
10. nova.conf

nova, keystone
FairShareScheduler

Resources unavailable: new requests remain Scheduling (no host, no ip address)
**FairShareScheduler**

Resources unavailable. Terminate one instance.
### FairShareScheduler

The previous scheduled request goes Running (assigned both host and ip address).
**QUACK** (Queues in Openstack)

CCR Cloud Working Group task (Bari, CNAF, Padova)

Objective: to implement the integration of Cloud in an existing Grid environment (LRMS)

- use the same resources for Cloud computing and Grid computing (**no partitioning**)
- All requests for resources (batch jobs, Cloud) will result in requests for allocations of VMs to OpenStack
- FairShareScheduler provides both queuing of the requests and fair-share algorithm in the resources provisioning

*Work in progress*
Assessing a possible evolution of the CREAM architecture to submit jobs directly to IaaS (with FairShareScheduler)

- CREAM responsible of the VM's life-cycle management
  - jobs executed in the VMs
- The batch system is not more necessary
  - fair-share and queueing provided by the FairShareScheduler
- Nothing changes for the Grid users
  - No updates of the WS interfaces are foreseen

Virgo experiment has shown interest in this type of architecture

- for further details, please see the yesterday Virgo presentation
Conclusion and next steps

- FairShareScheduler source code (for HAVANA) available in github:
  - https://github.com/CloudPadovana/openstack-fairshare-scheduler
  - Authors: Eric Frizziero (INFN-PD), Lisa Zangrando (INFN-PD)
- Testing in progress in Bari's Cloud Testbed
- Installation of the FairShareScheduler in the “Cloud Area Padovana”
- Started the integration process of the FairShareScheduler in IceHouse Openstack release
- Evaluate how to integrate this scheduler in the official OpenStack distribution
- FairshareScheduler's live demo at coffee break area
Thank you for your attention!

Questions?