Unified Resource Placement Module

Authors:

*Debo Dutta (dedutta@cisco.com), Senhua Huang (senhua.huang@gmail.com)***

*** Started the doc, left Cisco and might not be working on this*

Link to the slide deck on this topic:
https://docs.google.com/presentation/d/1ErOvJ5WFHqNctWf5vkKqW-7O62Z6ZDIV4ZQd7kuT-Vo/edit?usp=sharing

Why do we need it (Isn't current Nova+Cinder enough)?

Right level of abstraction

Coordination among different type of resources do not belong to a module/service that handles a particular kind of resource. If we treat each type of resource independently from each other which is what is done today in OpenStack, we are most likely not able to achieve any sensible global optimization. This is especially true because:

1. The data center resources (computing and storage) are connected and the connection itself (i.e., network bandwidth and connections) is constrained. While it is easy to scale out compute and storage resources by just simply adding more servers and/or hard disks, it is much harder to scale out the network resources.
2. The tenants often care not only about individual compute or storage resources but also how these resources can work closely with each other.

However, if we look at currently available resource selection procedure in OpenStack (in particular the Nova-scheduler and Cinder-scheduler), we can quickly observe that the scheduling is contained within their own compute and storage domain respectively. Specifically, Nova-scheduler chooses the compute host to boot a VM using a variety of filters while Cinder-scheduler chooses the volume host to create a volume for tenants. This is not likely to lead to optimal resource usage since doing their own job well does not necessarily lead to a good global solution. For example, if two hosts selected for VM and volume are not in the same rack, the traffic between the VM and the volume will consume a good amount of the bandwidth and at least some forms of tunneling resource (e.g., VLAN id) at the ToR switches. To avoid this situation, we need something above the Nova and Cinder to coordinate the placement of compute and storage resources.

*[footnote:Nevertheless, the separation of Nova-volume from Nova is a great advance within the OpenStack in that the two projects (namely Nova and Cinder) can better focus on its domain specific tasks and thus it is much easier for developers in these two domains come up with better abstraction and widen the choices of backend techniques within each domain.]

Architecture

Overall description

The unified resource placement module (u-rpm) has 3 major components as shown in the figure: resource selection engine, capability cache, and orchestrator.

- Resource selection engine (RSE) is responsible for selecting a group of physical resources to satisfy the requests from the tenant. The selection is performed by some resource selection algorithm based on the configured parameters provided by the admin, the available resources on each physical host in the data center, and potentially the performance measurement results.
- Capability cache is an optional component that keeps an in-memory record of the available resources of each host in the data center (e.g., vcpu core, free memory, available hard disk). Such information is calculated based on the capability update from the OpenStack services such as Nova/Cinder/Quantum.
- The orchestrator is probably the most complicated part of the module. It consumes the resource selection results calculated by the RSE and actually maintains states of each provisioning stage (this is probably already one by NTT data/Yahoo!) so that all the VMs, storage and networks are successfully created on the selected hosts. When failure happens, this component can probably retry until timeout and report the status back to the RSE. RSE can then find an alternative automatically without any user intervention and resend the selection (maybe in the form of delta) to the Orchestrator. It is possible to separate this component out for scalability purpose since a lot of states will be managed by this component.

The u-rpm module exposes a tenant facing API that allows tenants to specify their requests in the forms of virtual clusters. The details of the API are discussed in Section. ?.

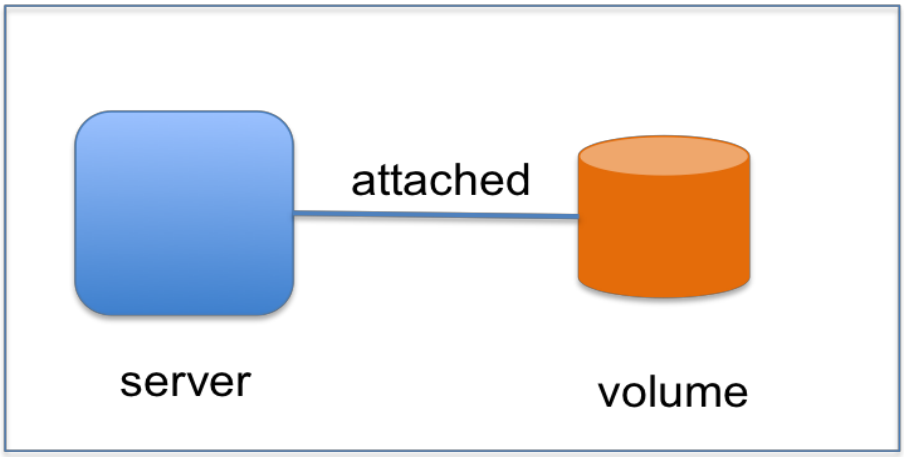
Introduction

This document introduces the concept of a unified resource placement into OpenStack. The responsibility of this module is to select compute, storage, and network and other Infrastructure resources together to satisfy the infrastructure requests from tenants. In other word, this module decides on which physical machine(s) to boot the request virtual machines, to create the extra storage, and to set up the traffic flow to reach these virtual machines. Different algorithms within the module can be chosen by the admin (or the owner of the OpenStack infrastructure) can be used to make such a decision. Tenants can specify their infrastructure requirements that include compute, storage, and networking in one shot, e.g., a connected virtual computing cluster with persistent data storages and potentially QoS requirements on the links using the REST APIs exposed by this module. The unified resource

placement module will select the corresponding physical machines (i.e., hosts) based on some configured criterion and the available resources in the data center and potentially their topology information. The output of the resource selection can be fed into a provisioning engine (e.g., orchestrator, coordinator, workflow manager) that will actually call the API's of Nova, Cinder, Swift, Quantum to instantiate the VM's and storage requested by the user/tenant.

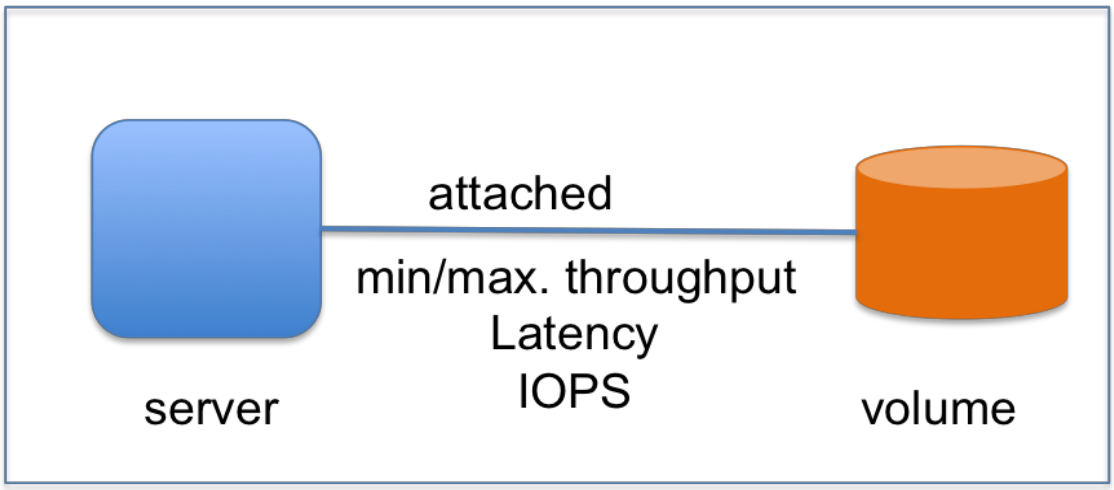
A simple use case would be to select the host where the volume is created to be "closed" to the host where the VM is booted. In this use case, a tenant is deploying a database service. This service has the following components:

- A server that serves requests. It has its owned specs on image, flavor, and network setup.
- An extra persistent volume with a particular size that stores the data.

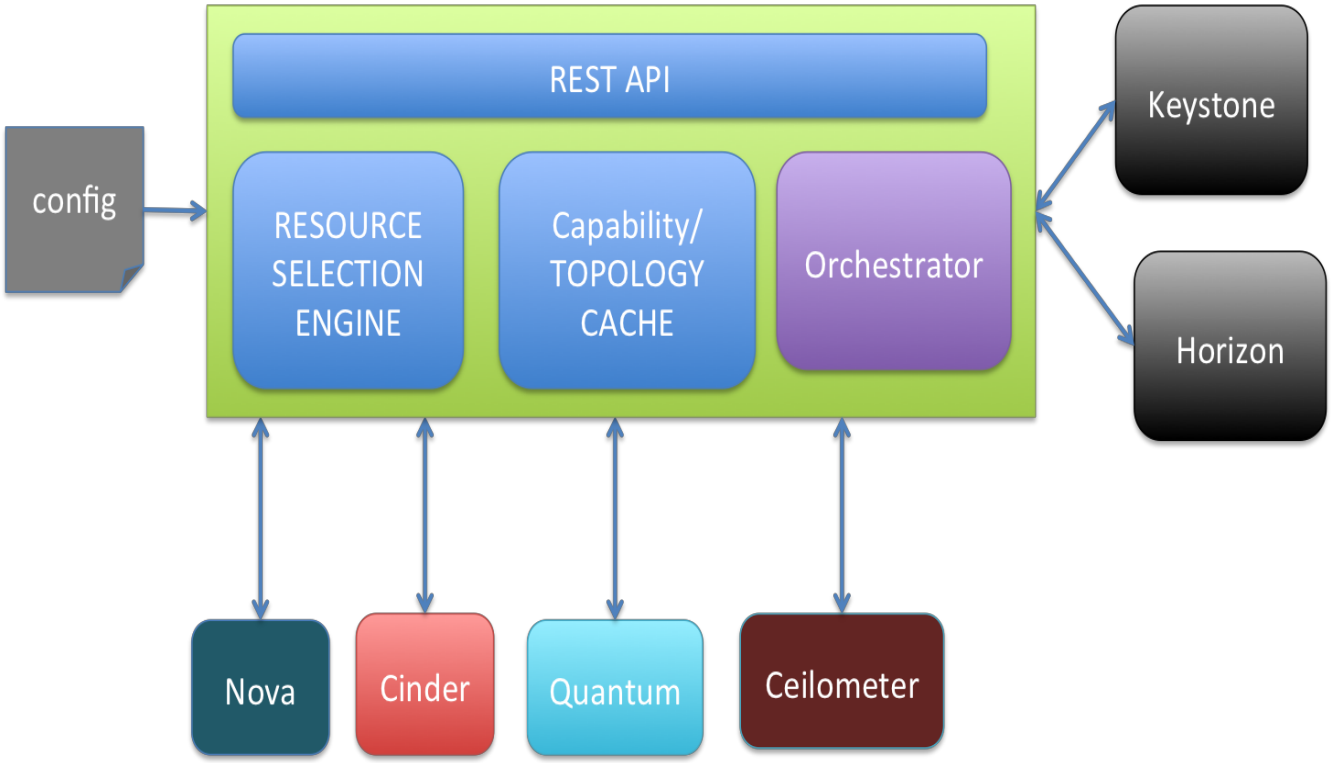


For this simple request, supposed there are N commodity hosts in the data center managed by OpenStack, there are roughly NxN choices of placing the requested (server, volume) pair on the hosts. The unified resource placement module is to select among all these possible choices the one most appropriate based on the need of the data center owner/admin. For example, to reduce network traffic, the admin could ask the module to place these two requests as "closed" as together, with the closeness measured by the number of hops between the two hosts. Ideally, if the same host is chosen for both server and volume, it will consume no network bandwidth. This leaves us roughly N choices. Other criterion could kick in and influence the module to make the final choice.

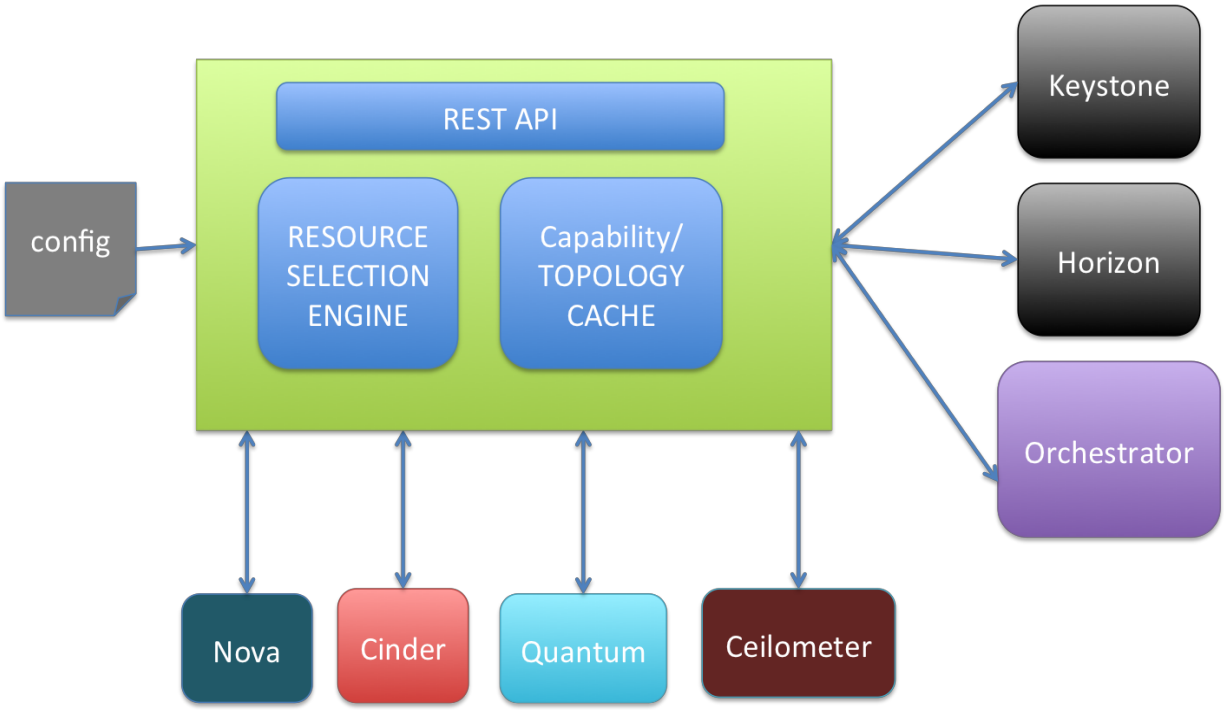
A slightly more complicated user request can include some QoS requirements on the "link" between the virtual server and volume as shown in the following figure. Then the current node-based capacity filtering need to be extended to account for the link-based capacity filtering.



More complicated use cases would include multiple virtual servers, and potentially multiple volumes. This is the case with large scale computations over a large set of data.



Another likely architecture is to separate the Orchestrator from the u-rpm.



RSE

Resource Selection Algorithms

There have been a lot of interesting work (both academic research and industrial proposals) on how to select resources intelligently for the optimal or sub-optimal resource usage. All the work has shown the advantage of using a more sophisticated selection algorithm than the simple round-robin or random algorithms.

This document is not about proposing new algorithms. Instead, we try to propose an architecture with which the service providers could take advantage of these existing algorithms and even develop their own secret sauces easily on top of the OpenStack APIs to achieve different goals. The following is a far from complete list of such goals.

- Minimize the amount of hardware resources needed for a given requested work loads
- Minimize the energy usage subject to the SLA guarantee to its tenants
- Maximize the performance of particular work loads subject to minimum requirements for others
- Maximize the resilience to single-x failures.

Capability/Topology cache

In order for the RSE to work, capability information about the compute/storage hosts and the network need to be fed to the u-rpm module. Part of the information is already available today for admin. For example, REST APIs are available to list all the hosts that provide service to

tenants (e.g., compute, volume, keystone, network) and the total/used vCPU, memory, disk of a particular host. This type of APIs are very handy in that i) it decouples the internal mechanism of Nova ii) developers are free to use any other language or framework. We are expecting similar APIs are available for storage manager such as Cinder or Swift so that we can retrieve the location of a particular volume, the residual storage capacity of a particular volume host conveniently. Note that some users might still bypass the u-rpm module and request VM/volume services using the existing APIs. This is why it might not enough for the u-rpm module to keep track of its own decision history for the purpose of calculating the available resources in the data center.

The topology information is very useful to make network-aware resource placement decision. However, it is also not easy to obtain without considerable effort on enhancing the Quantum or the agent running on each host and network element. As a first step, we could assume the topology is pretty stable and only incremental updates on broken network links or nodes are needed for the connectivity information. Therefore, the admin could manually specify the data center topology using a configuration file. However, if we want to achieve decent QoS guarantee with the advanced resource selection algorithms, then i) Quantum or other new service needs to be able to report the QoS account information to u-rpm; 2) or we have a persistent data store to maintain the topology and network resource consumption at the u-rpm. This should be planned for the future phases of the u-rpm module.

An alternative to REST API, we could also use the currently used RPC/messaging mechanism for the communication between u-rpm and other OpenStack components. The benefit of this choice is that the u-rpm will be more tightly coupled with the whole eco-system of OpenStack and probably more efficient in terms of communication overhead.

As a community, we should decide which way is the best to go.

Orchestrator

[This should be basically covered by the awesome work done by Yahoo!/NTT Data. Just reference to that work.]

Tenant facing API

With the help of a centralized resource placement module, we could enable more sophisticated requests from tenants. In particular, tenants could request virtual clusters (vcluster) rather than individual VMs or volumes. The detailed spec of the APIs of vcluster are TBD. In a nutshell, it should allow the tenant to create, update, list and delete their vcluster. Each vcluster has a set of virtual instances of known types, with each type corresponding to a resource category. Currently available types include server and volume. Each virtual instance has a dictionary of resource requests, which are the same as the currently supported

Nova/Cinder APIs. In particular, tenants could specify the image, flavor, nic and other properties for the requested server. Tenants could also specify the size, type, and other properties for the requested volume. For the first phase, only one server and one volume will be allowed in a virtual cluster.

The response will identify each virtual cluster with a virtual cluster UUID and return the provisioning state of the virtual cluster.

In the future phase, we expect that tenants are able to specify multiple servers and volumes in a virtual cluster as well as the QoS of the connectivity between server and volume. The service provider could provide some templates (similarly to the flavor of server) for the virtual clusters. For example, mapreduce vcluster, logging vcluster, video vcluster etc. In this way, the service provider could focus on optimizing/tuning its resource selection engine to handle these particular requests.

Interfaces with other OpenStack components

The u-rpm needs to obtain information about the running services on each cloud resource such as compute hosts and storage hosts as well as networking resources. This is via the communication with other OpenStack components. In particular, the u-rpm module needs to talk to Nova to know the following information:

- the identities of all compute hosts and their available compute resources,
- the identities of all storage hosts (could be the same as the compute hosts) and their availability resources,
- the identity of the compute host where the server is booted,
- the identity of the storage host where the volume is created.
- the states of resource reservation on these hosts

All this communication should not visible to tenants and tenants should not care about this. There are at least two ways to perform the communication.

- Using REST APIs defined for admin usage

If this approach is adopted, we need to extend the current REST API's provided by Nova and Cinder. Nova has already provided some admin APIs to list the compute hosts and describe the capabilities of each host. We could see how much changes are needed.

- Using RPC calls

Interfaces with external modules

This interface is mainly used to get the topology and network resource information for the resource selection algorithms. Maybe we could have a layer of drivers so that we could get this information from different sources. For the first phase, we can just use some static configuration files to provide the connectivity information.

Design goals

- Pluggable
- Extensible
- Scalability

Any service provider can write or use their own customized resource selection algorithms.

Dependency

Keystone is used as the authentication and authorization for the u-rpm service. Horizon could be used as the GUI for the u-rpm.

Client

References

[1] Topology-Aware Resource Allocation for Data-Intensive Workloads http://conferences.sigcomm.org/sigcomm/2010/papers/apsys/p1.pdf

[2] Yang Guo, Alexander L. Stolyar, and Anwar Walid, “Shadow-Routing Based Dynamic Algorithms for Virtual Machine Placement in a Network Cloud,” INFOCOM 2013.

[3] Zamanifar, K.; Nasri, N.; Nadimi-Shahraki, M., "Data-Aware Virtual Machine Placement and Rate Allocation in Cloud Environment," *Advanced Computing & Communication Technologies (ACCT), 2012 Second International Conference on* , vol., no., pp.357,360, 7-8 Jan. 2012

[4] K. Mills, J. Filliben and C. Dabrowski, “Comparing VM-Placement Algorithms for On-Demand Clouds,” Information Technology Laboratory. NIST. Gaithersburg, MD

[5] Bin packing problem: http://en.wikipedia.org/wiki/Bin_packing_problem