Business Rules and Policies driven Constraints-based Smart Resource Placement in Openstack

Yathiraj Udupi (yudupi@cisco.com), Debo Dutta(dedutta@cisco.com)

INTRODUCTION

Openstack currently supports independent resource placement (i.e., scheduling) decisions internally among the individual components such as Nova, Cinder, and Neutron. For example, while provisioning VMs, Nova-scheduler determines the host, based on certain filter criteria such as available disk space, ram, cpu cores, etc. Cinder-scheduler determines the volume hosts to create volumes. Because of the independent decisions made by the Nova-scheduler and the Cinder-scheduler, there is a good possibility for the two hosts selected for VM and Volume to reside in different racks and hence consuming a good amount of traffic bandwidth, leading to a non-optimal resource placement. The filter and weight based solutions currently supported by the FilterScheduler are capable of handling simple constraints in both Nova or Cinder scenarios. The filters are implemented in python and are run sequentially to filter out the hosts, and then eventually sorted based on the supported weight functions.

The above observations of independently done resource placement decisions, and filter and weight based scheduling mechanisms currently supported in Openstack leads to no guarantees of providing a globally optimal solution and capabilities to handle complex optimization constraints that could not only involve state variables local to the service, but also from the other services. It is very important for the resource placement decisions to consider all the data center resources such as compute, storage, and network.

Tenants can have complex business rules and policies that govern the data center resources, and the resource placement decisions should satisfy the complex constraints. For example, tenants may expect all the storage to reside locally where the compute is, or may expect to minimize the network bandwidth usage. There could be also be cost-related business rules on what what kinds of instances to schedule depending on the time. Tenant policies may also request to minimize the distance between VMs, VMs and storage, etc that also rely on the network topology for making resource placement decisions.

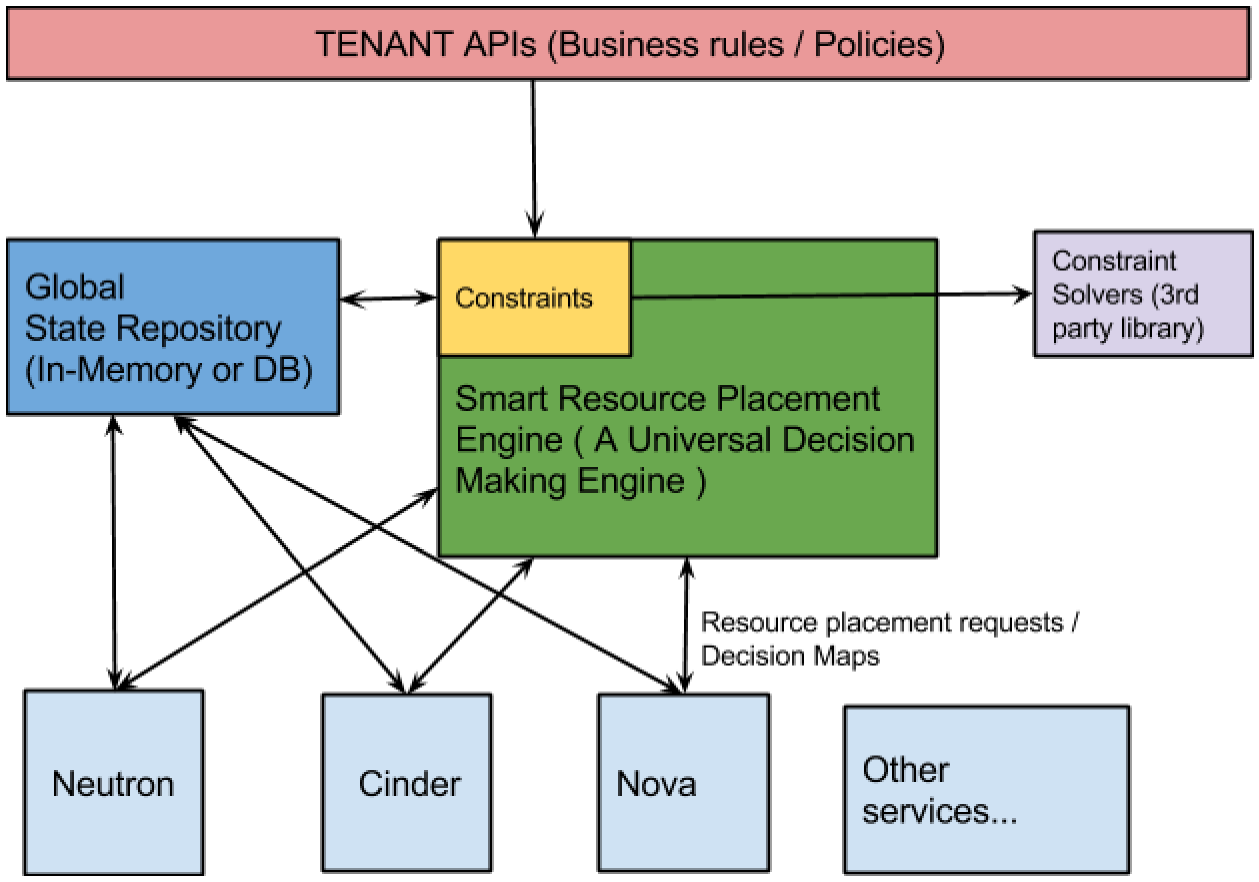
To facilitate a smart, global, optimal resource placement decision making in Openstack, we feel the following solutions are needed:
  - A smart resource placement decision making engine that is universally applicable for all kinds of resource placement decisions, and can communicate with all the openstack services. The underlying purpose of this decision making engine is to solve for minimizing (or

maximizing) certain optimization metrics while satisfying a set of constraints.
- A support for handling business rules and policies of the tenants that eventually translate to the complex constraints and feed into the decision making engine
- All services need to be able to communicate with the universal decision making engine to request for decisions and get decision results.
- A resource state repository for near-real-time updated states of all resources. The decision engine will be able to get the latest states while solving the constraints to make smart resource placement decisions.

In this document, we will propose the high-level vision of the architecture that is required to be able achieve this smart resource placement. The rest of the document will focus on providing additional design-level details of the universal decision making engine. We will refer to certain existing IceHouse blueprints that are related to the other aspects essential for this smart resource placement vision.

HIGH-LEVEL ARCHITECTURE

The architecture diagram below presents the high-level components that are essential.


TENANT APIs (Business rules / Policies)
Global State Repository (In-Memory or DB)
Constraints
Smart Resource Placement Engine (A Universal Decision Making Engine)
Constraint Solvers (3rd party library)
Resource placement requests / Decision Maps
Neutron
Cinder
Nova
Other services...

Before getting into the further details on the Universal Decision Making Engine component, we would like to list the other components and the blueprints that are proposed to achieve a similar purpose. Our vision is to collaborate with all the key players in the Openstack community and get this overall vision implemented.

1. Tenant APIs for Business rules, Policies

This support is key to this architecture, where a tenant should be able to push business rules, policies that affects the scheduling of resources. The following blueprints are related to this topic and address some of the requirements.:

a. Instance-group-api-extension (Debo Dutta)
https://blueprints.launchpad.net/nova/+spec/instance-group-api-extension
This proposes to allow for scheduling of VM groups that adhere to a common policy such as "anti-affinity", "network-proximity". The requirement is to find an optimal way of scheduling this entire group of VMs to satisfies the given policies.
*How it relates to this architecture***:** The APIs should support specifying policies and business rules and the Universal decision making engine should translate these policies and rules into some constraints that can be computationally used by the engine to make decisions.

b. Multiple Scheduler Policies (Alex Glikson)
https://blueprints.launchpad.net/nova/+spec/multiple-scheduler-drivers
https://wiki.openstack.org/wiki/Nova/MultipleSchedulerPolicies
This addresses the need to have different policies under different situations.
*How it relates to this architecture***:** It should be possible for the tenant to switch the policies at any time, and hence the constraints that are used to make resource placement decisions will be pluggable and modifiable at any time.

2. Global State Repository (In-Memory or DB)

Our smart resource placement architecture assumes that in order to make intelligent decisions for getting a globally optimum solution, we need to be get near real-time states of all applicable resources from all the relevant openstack services. Any decision making and constraints solving engine needs to have the computational metrics available. The state metrics should be easily accessible via APIs and will be consumed while creating the constraints. Currently all the state information is stored locally in a DB within each of the individual Openstack services such as Nova, Cinder, and Neutron.

The following blueprints address to bring the state information together and accessible in a fast manner:

a. No-DB scheduler (Boris Pavlovic)
https://blueprints.launchpad.net/nova/+spec/no-db-scheduler
https://docs.google.com/a/mirantis.com/document/d/1_DRv7it_mwalEZzLy5WO92TJcummpmWL4NWsWf0UWiQ/edit

This blueprint proposes a In-memory way of storing all the host states bypassing the DB, to make the access faster and improve scheduling performance.
How it relates to this architecture: This proposal promises an efficient way to get real-time access to the state information, which will benefit the resource placement decision engine to make quick decisions. Have the state in-memory, also provides a quick way to consolidate state information from across the services.

b. Scheduler metrics and Ceilometer (Paul Murray)

https://blueprints.launchpad.net/nova/+spec/network-bandwidth-entitlement
https://blueprints.launchpad.net/nova/+spec/cpu-entitlement
https://blueprints.launchpad.net/nova/+spec/utilization-aware-scheduling

These proposal addresses a new mechanism to provide additional metrics to the scheduler.
How it relates to this architecture: Our architecture can consume state information from Ceilometer and this way we will benefit from more metric information while solving for the complex constraints.

c. Scheduling across Services (Boris Pavlovic)

This discussion started on Etherpad also addresses about using metrics across services.
How it relates to this architecture: The solution will feed into the constraints with resources that are either local to a service or available from across other services. Hence the design should be such that it can feed into the constraints.

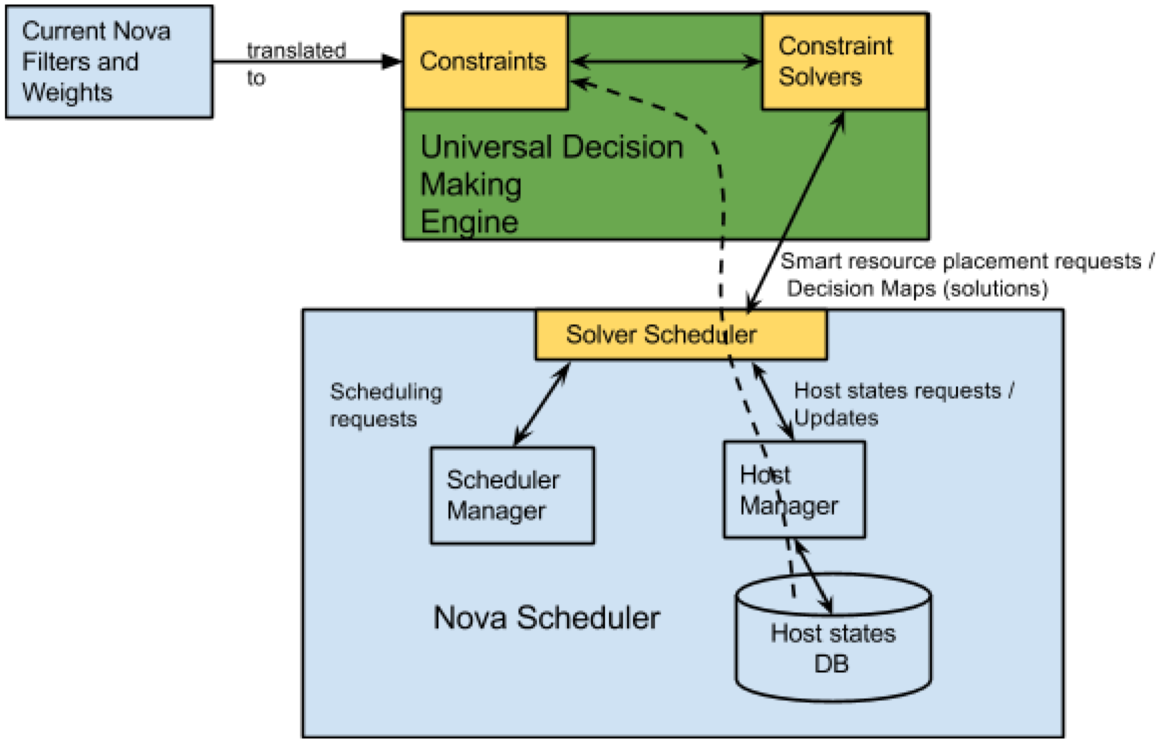
3. Smart Resource Placement Engine (A Universal Decision Making Engine)

The key purpose of this component is to solve for constraints and make resource placement decisions. The existing openstack services will plugin to this module and utilize the decisions in the provisioning of resources. When we talk about resource placement decisions, we are working with a set of constraints that have to be satisfied, and if there are multiple options that are feasible, our goal is to find the best matching solution that optimizes a cost metric (could be a minimization metric or a maximization metric).

OpenStack scheduler has nice options like filters which allow resource selection based on simple constraints e.g. Don't put instances in a set on the same host, or find me a host that satisfies these disk, ram requirements.
However for complex constraints, building a filter would be as complex as building a real constraint solver e.g. place VMs while minimizing average (VM-storage-bandwidth) with complex constraints. On the other hand, complex solvers based on decades of research are available in open source (PULP, CVXOPT, COIN_OR), and the underlying solvers have fast and efficient C implementations. This forms the key motivation for us to design a solution that leverages existing constraint solvers.

The following blueprint provides an initial proposal of the smart resource placement engine:
SolverScheduler - Complex constraint based resource placement
https://blueprints.launchpad.net/nova/+spec/solver-scheduler
Key Design Points:

a. Not disruptive, and works with the current Nova architecture: The key aspect of this solution is that it is not disruptive of the existing Nova scheduler. This constraint based Solver scheduler fits into the existing Nova scheduling process utilizing the host states from DB. The requests to provision VMs can be made to be routed (by changing the scheduler driver configuration) via the Solver scheduler, as opposed to the Filter scheduler. The existing Filters can be implemented as constraints and fed to the solving engine, and the weights can be listed as a minimization or maximization cost metric, and the solver will compute the whole problem providing a decision map with instance, host tuples. This will be used by the Nova Scheduler component to complete the VM provisioning.


Current Nova Filters and Weights
translated to
Constraints
Constraint Solvers
Universal Decision Making Engine
Smart resource placement requests / Decision Maps (solutions)
Solver Scheduler
Scheduling requests
Host states requests / Updates
Scheduler Manager
Host Manager
Nova Scheduler
Host states DB

How it fits in the current Nova Scheduling Architecture

b. Using pluggable Solvers: Currently we have pushed a POC code (https://review.openstack.org/#/c/46588/) that relies on a LP-based solver written using the PULP solver. Our architecture will support pluggable solvers, and there could be multiple implementations catering to different scenarios of constraints.

c. Can consume resources from across services: With the support of an additional component that provides a global state repository, we can consume resources from one or all of the openstack services that share state information relevant to making resource placement decisions.

d. APIs to talk to external (tenants) as well as internal services (Nova, Cinder, etc.): Tenant APIs talk to the decision engine to feed business rules / policies, that translate to constraints and optimization metrics used to take resource placement decisions.

The smart resource placement engine with the help of pluggable constraint solvers aims to provide a intelligent way of scheduling in Openstack, and enables solving for complex constraints. Complex constraints are efficiently solved using the underlying open source solvers, which have faster C-based implementations, and thereby avoids the situation of implementing complex solving techniques in Python within Openstack.

References:

1. Unified Resource Placement Module:
https://docs.google.com/document/d/1cR3Fw9QPDVnqp4pMSusMwqNuB_6t-t_neFqgXA98-Ls/edit?pli=1#
2. Solver Scheduler: Complex constraint based Resource Placement - Blueprint
https://blueprints.launchpad.net/nova/+spec/solver-scheduler
3. POC code that provides PULP-based Solver Scheduler: (Under review)
https://review.openstack.org/#/c/46588/
4. Instance Group API extension - Blueprint:
https://blueprints.launchpad.net/nova/+spec/instance-group-api-extension