3D visualization in an OpenStack cloud: a tasteful recipe

Carlo Impagliazzo, Muriel Cabianca

CRS4 - Center for Advanced Studies, Research and Development in Sardinia

{carlo.imp,muriel.c}@crs4.it

Purpose

We present a method that allows users to capitalize on the availability of cloud computing resources to visualize complex 3D graphical models on simple – even mobile – thin clients. The portability and form factor of thin clients makes them ideal companions in meetings and work locations away from the traditional desk, while being well suited for mobile model visualization. Unfortunately, to achieve their mobility, these devices sacrifice computing power, thus limiting the complexity of models that can be rendered on board. We address this problem by integrating a cloud-based rendering infrastructure with a 3D visualization system on thin clients, allowing the visualization platform to take advantage of remote graphics acceleration, while optimizing the resources and sharing them with other HPC tasks.

How

The system consists of three main components:

* a web interface that manages application sessions;
* a custom orchestrator that manages cloud-based resources and virtual application instances;
* a visualization front-end that runs on clients.

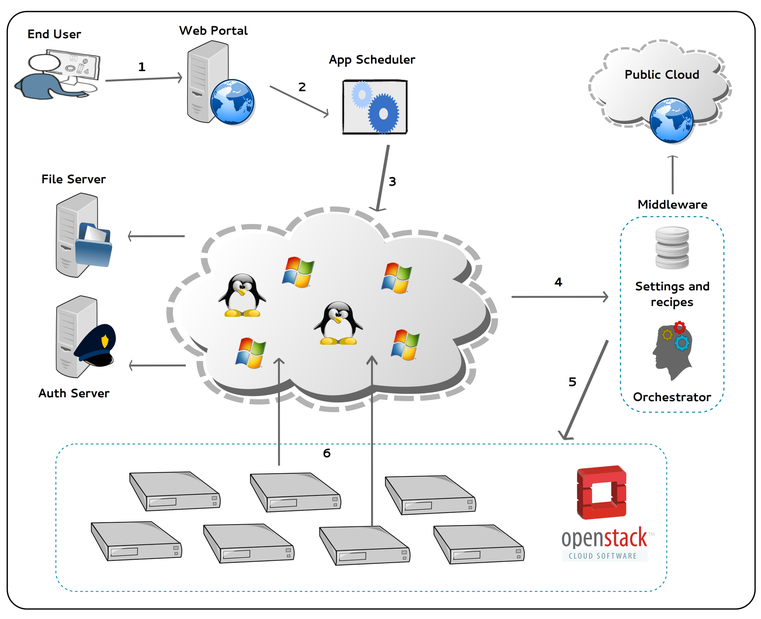
The web-based front-end provides web services to access accelerated applications that run in the data center. It also allows sessions to be shared among users.

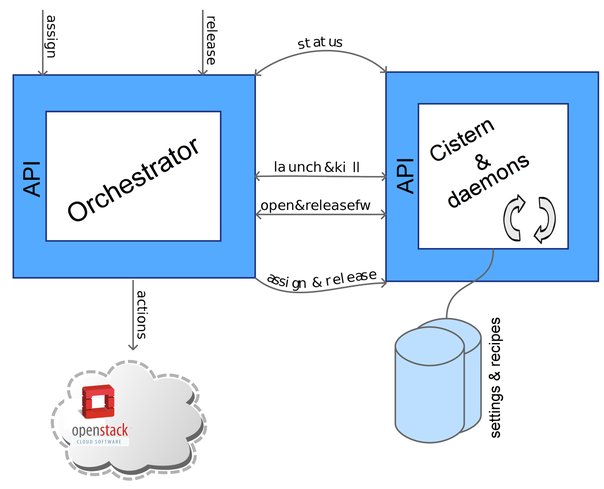
The orchestrator provides an abstraction layer to manage instances of application appliances, also provides the frontend with the parameters to communicate with the applications running on them. To manage these virtual computing resources, the orchestrator uses the **OpenStack** cloud computing platform. To acquire hardware resources on which to instantiate the virtual machines, **OpenStack** negotiates with **GridEngine** through a custom scheduler extension.

All the custom API layers have been developed using the **Flask** microframework and **SQLAlchemy** to interact with the DB, while **Python** was used to implement the middleware.

The system also includes a communication and synchronization layer between its three main components.

Schema





The aim of the middleware is to maintain a constant number of VMs per recipe (a set of resource specifications for an application).

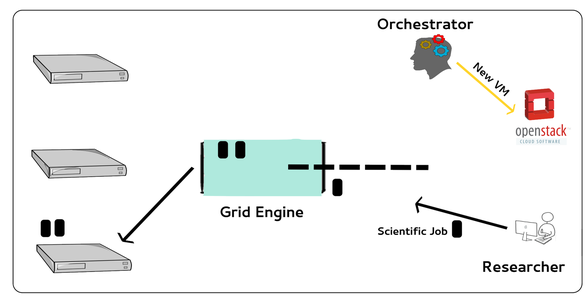
Through recipes is possible to specify the flavor and how many VMs is desirable to keep alive. Within the M*iddleware* two cores work to provide facilities: *Orchestrator*, *Cistern*.

As its name implies, the orchestrator coordinates virtual resources to ensure that session requests are satisfied as quickly and efficiently as possible, while hiding details from the upper layers. The use of **Libcloud** to merge different API dialects allows it to support different cloud providers. In our server farm, we ran our tests with **OpenStack**. **Flask** has been used to provide a practical access and management point as well as ensure reliability.

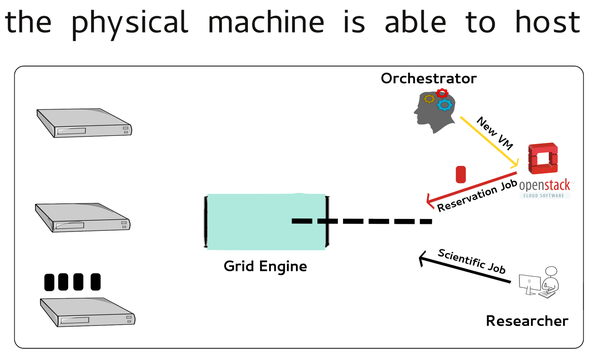
Cistern is the codename of the DB interface based on **Flask**. Besides acting as a web service interface, it shares database models with two independent daemons, *Loops* and *Cook*. The former communicates with the orchestrator to keep specs synced on the DB, while the latter synchronizes user information between the orchestrator and the DB. Depending on the recipe, Cook also calls orchestration routines to launch new VMs.

Flow

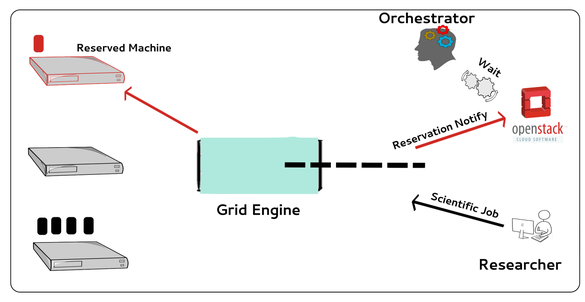
In our typical scenario, resources are shared between **OpenStack** and other HPC applications, so we need to use an additional scheduler to reserve physical machines for **OpenStack**.



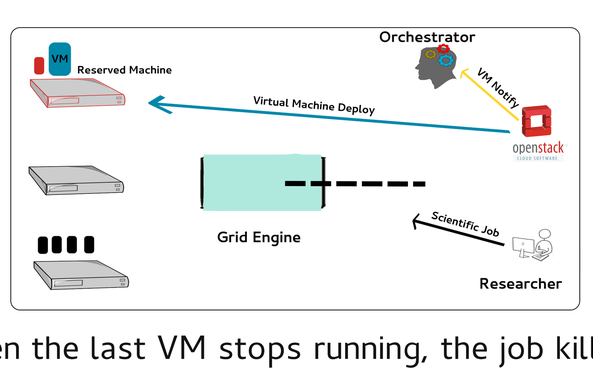
When a new request comes from the orchestrator, **OpenStack** checks if it has enough resources to launch the requested VM. If it doesn't, it submits a special job to Grid Engine and starts a monitor process. Before submitting the job, **OpenStack** maps the resource needed to launch the VM to the resource schema used by Grid Engine, to ensure that the physical machine is able to host the VM.



Grid Engine launches a "reservation job" in the first machine that fits the requested VM specs; the monitor process becomes aware of which machine the job is running on, configures the **OpenStack** environment accordingly and prevents other jobs to interfere with it.



Now **OpenStack** can launch the VM and all those resources in the machine are dedicated to **OpenStack** VMs.



When the last VM stops running, the job kills itself, and the **OpenStack** monitor marks the server as unavailable.

Scenario

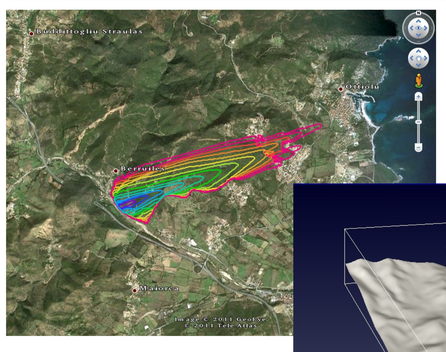


Fig A

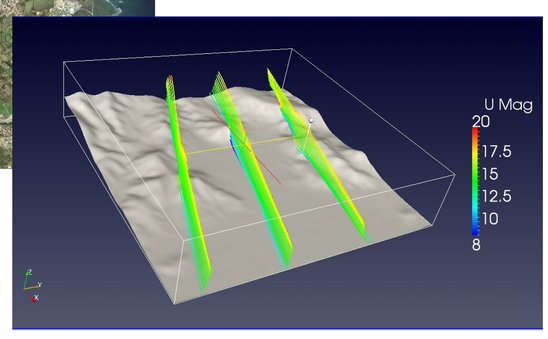


Fig B

Courtesy of CRS4's E&E Sector

The application used to test the infrastructure consists in performing a multiple run of a regional weather forecast model. A fast fluid dynamics mass-consistent flow model (based on OpenFOAM) produces a highly resolved wind in a smaller and finer domain. A ParaView's screenshot, displaying some downscaled 3D wind streamlines at grid resolution of 20m, is shown in figure (B). With those high resolution weather conditions it is possible to drive a forest fire simulation module and evaluate fire evolution scenarios for that domain. Fig (A)

PIA - Pacchetti Integrati di Agevolazioni "Industria, Artigianato e Servizi" (Annualita' 2010)
Programmazione Unitaria 2007/2013
P.O. FESR 2007/2013
REGIONE AUTONOMA DELLA SARDEGNA