PROCEEDINGS OF SCIENCE

Synergy: a service for optimising the resource allocation in the cloud based environments

Lisa Zangrando[1]
INFN, Sezione di Padova
Via Marzolo 8, I-35131 Padova, Italy
E-mail: Lisa.Zangrando@pd.infn.it

Federica Fanzago
INFN, Sezione di Padova
Via Marzolo 8, I-35131 Padova, Italy
E-mail: Federica.Fanzago@pd.infn.it

Marco Verlato
INFN, Sezione di Padova
Via Marzolo 8, I-35131 Padova, Italy
E-mail: Marco.Verlato@pd.infn.it

Massimo Sgaravatto
INFN, Sezione di Padova
Via Marzolo 8, I-35131 Padova, Italy
E-mail: Massimo.Sgaravatto@pd.infn.it

In OpenStack, the current resources allocation model provides to each user group a fixed amount of resources. This model based on fixed quotas, accurately reflects the economic model, pay-per-use, on which the Cloud paradigm is built. However it is not pretty suited to the computational model of the scientific computing whose demands of resources consumption can not be predetermined, but vary greatly in time. Usually the size of the quota is agreed with the Cloud Infrastructure manager, contextually with the creation of a new project and it just rarely changes over the time. The main limitation due to the static partitioning of resources occurs mainly in a scenario of full quota utilization. In this context, the project can not exceed its own quota even if, in the cloud infrastructure, there are several unused resources but assigned to different groups. It follows that the overall efficiency in a Data Centre is often rather low.

The European project INDIGO DataCloud is addressing this issue with “Synergy”, a new service that provides to OpenStack an advanced provisioning model based on scheduling algorithms known by the name of “fair-share”. In addition to maximizing the usage, the fair-share ensures that these resources are equitably distributed between users and groups.

[1]Speaker

PoS(ISGC 2016)032

In this paper will be discussed the solution offered by INDIGO with Synergy, by describing its features, architecture and the selected algorithm limitations confirmed by the preliminary results of tests performed in the Padua testbed integrated with EGI Federated Cloud.

International Symposium on Grids and Clouds 2016
13-18 March 2016
Academia Sinica, Taipei, Taiwan

1. Introduction

Computing activities performed by user groups in the Public Research and in the Public Administrations are usually not constant over long periods of time. Therefore, the amount of computing resources effectively used by such user teams may vary in a quite significant way. Often these teams make specific contracts with the Data Centres and the main requirement concerns the provisioning of an average computing capacity to be guaranteed during a long period which typically covers the whole year. This is strongly preferred to an agreed amount of computing resources that should be available at any given time. So, in these Data Centres new hardware is acquired according to the user's best estimates of the foreseen annual computing usage needed for their activities and then partitioned among them. The partitioning policy is defined in terms of fractions of average usage of the total available capacity that is the percentage of the Data Centre computing resources that each team has the right to use averaging over a fixed time window. Therefore the administrator must make sure that each stakeholder team, at the end of the year, has enjoyed its agreed average number of resources. Moreover, since the request for resources is typically much greater than the amount of the available resources, it becomes necessary to seek to maximize their utilization by adopting a proper resource sharing model.

In the current OpenStack model, the resource allocation to the user teams, namely the projects, can be done only by granting fixed quotas. Such amount of resources cannot be exceeded by one project even if there are unused resources allocated to different projects. Therefore in a scenario of full resource usage for a specific project, new requests are simply rejected. It follows that, when resources are statically partitioned among user teams, the global efficiency in the Data Centre's resource usage can be quite low.

In the past the same problem has been solved by the batch systems and today, the INDIGO DataCloud project [1] is addressing the same issue through “Synergy”, a new advanced scheduling service to be integrated in the OpenStack Cloud Management Framework. Synergy adopts a resources provisioning model based on a fair-share algorithm in order to maximize the resources usage as well as to guarantee that resources are equally distributed among users and groups. This can be done by considering the portion of the resources allocated to them and the resources already consumed. Moreover it provides a persistent priority queuing mechanism for handling user requests that can't be immediately fulfilled, so that they can be processed later, when the required resources become available.

Starting from the list of the selected requirements which Synergy has to satisfy, this paper provides details about the service architecture design and its implementation, focusing on the main integration and interoperability aspects with the existing OpenStack components. It also discusses the chosen algorithm by highlighting its limitations confirmed by the preliminary results of tests performed in the Padua instance of the EGI Federated Cloud.

2. Requirements analisys

The scheduling capability is a set of quite complex activities that advanced schedulers must meet. The most relevant features that were chosen when designing Synergy are::

- allow the IaaS administrators to allocate a subset of resources, referred in this document as “dynamic resources”, to be shared among different user projects, besides the ones statically partitioned by fixed quotas per project;
- provision of an automatism which (re)calculates the size of the dynamic resources set, considering the total number of resources and the ones allocated as “static resources”;
- provision of a queuing mechanism for handling the user requests that cannot be immediately fulfilled so that they can be served as the required resources are available;
- provision of a resources allocation mechanism based on a fair-share scheduling algorithm;
- with respect to the OpenStack projects, allow the IaaS administrators to:
  - enable the project to act only in the standard OpenStack mode: the requests are processed according the First Come First Served (FCFS) model; the ones that can’t be immediately satisfied are simply rejected and forgotten;
  - enable a project to consume the either static or dynamic resources in a fair-share mode: the user requests are processed basing on the priority order calculated by the fair-share algorithm. The ones that can't be immediately satisfied are stored in a persistent priority queue and served only when the required resources are available. The resource usage is controlled by a set of fair-share policies which define:
    - the type of resource to be accessed: static or dynamic. The static resources are limited by the project's quota
    - a positive integer representing the number of shares of Cloud resources assigned to that project
    - a maximum lifetime for the its Virtual Machines (VM) after which the VM shall be destroyed to prevent it from running indefinitely
    - optionally the definition also of different shares among users belonging to the same project. If not specified, it is assumed that all users have the same share
- provision of an automatic killing mechanism which destroys all VMs running for more than the associated maximum lifetime. Without this enforcement the fair-share based resource allocation model is not realizable;
- allow the IaaS administrator to drain the queue of requests, needed for e.g. for maintenance activities;
- provision of proper command line tools to allow the IaaS administrator to manage, at runtime, the fair-share policies and the queue of requests.

3. The Synergy service

Synergy is not a scheduler. It is, indeed, a new extensible general purpose management service to be integrated in OpenStack. Its capabilities are implemented by a collection of managers which are specific and independent pluggable tasks, executed periodically, like the cron jobs, or interactively through a RESTful API. Different managers can coexist and they can interact with each other in a loosely coupled way by implementing any even complex business logic (figure 1).

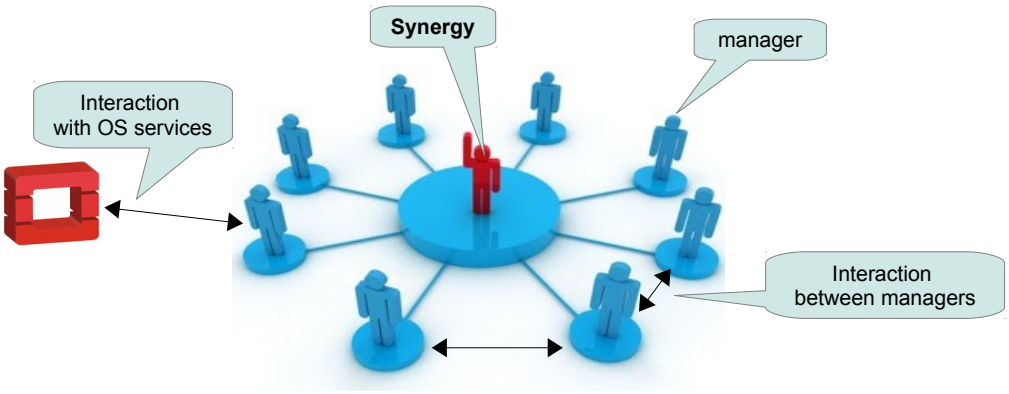


Figure 1: the schema shows Synergy represented by the red man which handles a set of managers (the blue men). The arrows are the interactions among managers and/or external services.

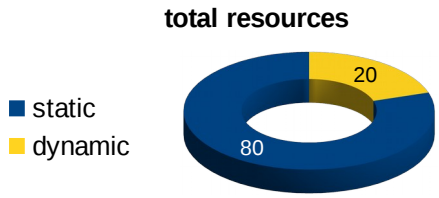
New managers can be easily implemented by extending the python abstract base class provided by the Synergy python API:

```
class Manager(Thread):
   def getName(self): # return the manager name
   def getStatus(self): # return the manager status
   def isAutoStart(self): # is AutoStart enabled or disabled?
   def setup(self): # allows custom initialization
   def destroy(self): # invoked before destroying
   def execute(self, cmd): # executes user command synchronously
   def task(self): # executed periodically at fixed rate
```

In particular the last two methods, “execute” and “task”, allow developers to implement synchronous or asynchronous activities.

3.1. Resource allocation

With Synergy the OpenStack administrator can allocate a new kind of resources, named dynamic resources to be shared among different projects. Such dynamic resources are a subset of the total resources not statically allocated with fixed quotas. Static and dynamic models can coexist. In particular the static resources are consumed according to the standard OpenStack model, i.e. according to a First Come First Served (FCFS) policy, whereas the dynamic ones are handled by Synergy through a specific set of policies defined by the administrator, as for example:



- the definition of the list of projects allowed to access to the dynamic resources
- the definition of shares on resource usages for the relevant projects

- the maximum lifetime for Virtual Machines which is a sort of killing functionality. This functionality prevents Virtual Machines to be running for a very long time on the dynamic resources and it is needed to enforce the fair-sharing.

4. Advanced scheduling with Synergy

Advanced scheduling is a complex set of functionality primarily a fair-share algorithm and the priority queuing mechanism. The fair-share algorithm maximizes the resources usage and guarantees that resources are distributed among users following the fair-share policies defined by the administrator. Instead the priority queuing mechanism is needed for handling the user requests that cannot be immediately fulfilled. Five managers implement the fair-share based scheduling model as shown in figure 2:

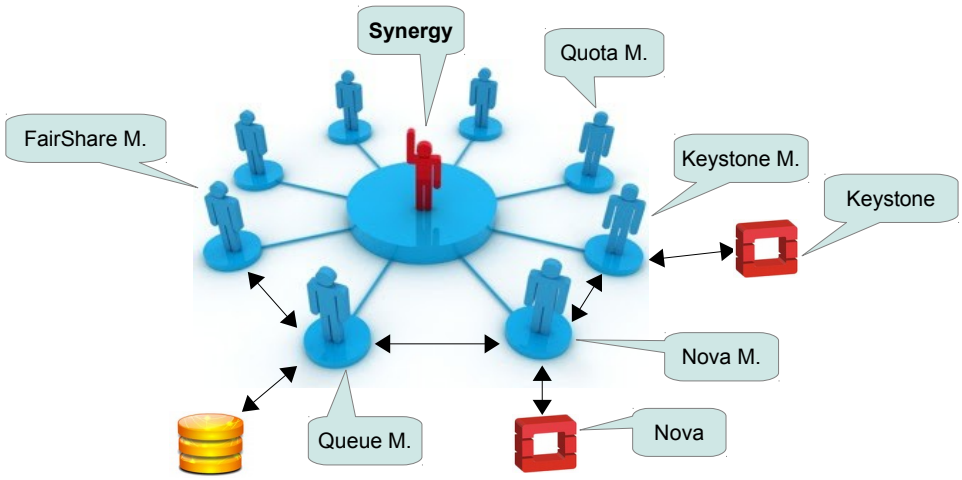


Figure 2: the high level Synergy architecture

- **FairShare-Manager**: implements the main fair-share scheduling logic. In particular it dynamically assigns the proper priority value to every user request. At any given time the priority is a weighted sum of factors: age and fair-share expressed in terms of CPU, memory and disk. Moreover the weight expresses the interest for a specific factor: for example the administrator might want to make the CPU factor dominant with respect to other parameters. The selected fair-share algorithm is based on the Priority Multifactor Strategy of SLURM [2]. The possibility to consider pluggable mechanisms for applying different algorithms will be explored in the future.
- **Queue-Manager**: provides a persistent priority queue service. It can handle several queues. All user requests involved in the fair-share computation are stored in a specific queue and processed asynchronously by the FairShare-Manager by taking into account the priority order. A queue contains relevant information as: full user request, timestamp, priority, retry count, trusted token.
- **Quota-Manager**: this manager is in charge of handling the quota of all projects. In particular it calculates periodically the dynamic quota size. In case of quota saturation, the dedicated scheduling process will be blocked by this manager until the required resources become again available. Moreover the Quota-Manager handles the set of

Virtual Machine lifetime policies for dynamic resources. So that, whenever a Virtual Machine exceeds the limit, this manager invokes the Nova-Manager for destroying it.

- **Nova-Manager** and **Keystone-Manager** interact with the related OpenStack services.

Synergy is not intended to replace any existing OpenStack service (e.g. Nova-Scheduler), but it may pretty complement their functionality as an independent Openstack service. Moreover Synergy doesn't require any changes in the existing OpenStack components and it doesn't force the utilization of its resource allocation model.

4.1 The low level architecture

The figure below illustrates the low level architecture of Synergy. In particular it shows Synergy and its integration within the existing OpenStack architecture:

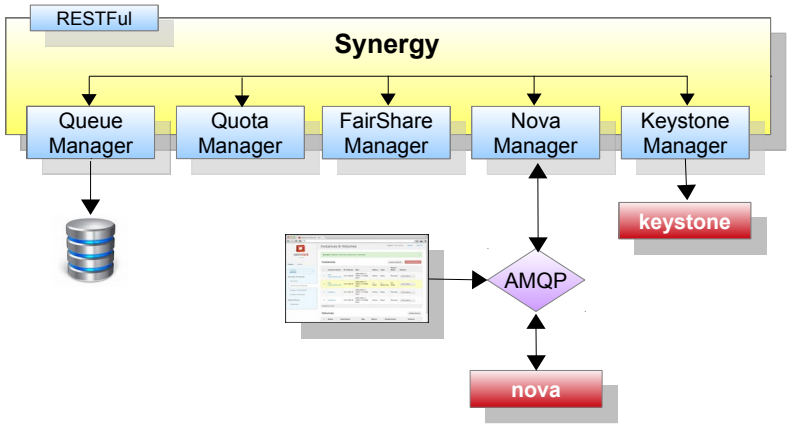


Figure 3: the low level Synergy architecture

The Nova-Manager intercepts all user requests, coming from the Horizon Dashboard or command line while the FairShare-Manager calculates and assigns to each of them the proper priority value. Such requests are immediately inserted in a persistent priority queue by the Queue-Manager. Moreover the FairShare-Manager fetches from the queue the request having the higher priority, processes it and finally sends it to the Nova-Scheduler through Nova-Manager by using the AMQP messaging system.

In the scenario of full resource utilization for a specific quota, the Quota-Manager advises the FairShare-Manager to wait until the compute resources become again available. In case of failure, instead, the FairShare-Manager provides a retrying mechanism which handles the failed requests by inserting them again into the queue (up to n retries). Moreover the priority of the queued requests is periodically recalculated for giving a chance to the older requests to rise up in the queue.

To complete, the KeyStone-Manager provides Synergy with all user and projects information and handles the security accesses.

To prevent any possible interaction issue with the existing OpenStack clients, no new states have been added so that from the client point of view the queued requests remain in "Scheduling" state till the compute resources are available.

5. Implementation

Whenever applicable Synergy has been developed by adopting the full OpenStack style in according to the specific development guide lines, including the coding style and the technologies to be used: the source code id in launchpad [3][4] while the integration with the OpenStack Continuous Integration System is being finalized.

Synergy is a full python service accessed by clients through a well defined RESTful API which provides either management commands for handling the plugged managers (e.g. activate manager, suspend, get status, etc) or executing specific commands on the selected manager (e.g. get/set quota, list queues, etc). The API can be extended for including new specific interfaces. For what concerns the internal interfaces, Synergy interacts with the OpenStack services as Nova and Keystone, by invoking the exposed RESTful interfaces or using the internal RPC APIs.

All Synergy default configuration options are defined in the synergy.conf file but they may be then changed at runtime (i.e. without the need of restarting any service) by the IaaS administrator. Moreover some information about the configuration on Nova are retrieved accessing directly the nova.conf file.

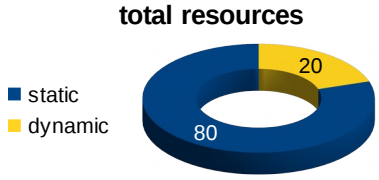
The historical resource usage required by the fairshare scheduling algorithm is retrieved accessing the Nova database through the provided API. This approach guarantees good performance although there are some alternatives to be explored as the Ceilometer metering service.

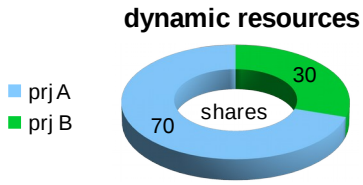
Finally, persistent data, as the priority queues, are stored in the same storage backend used by OpenStack.

6. Testing

The main aim was to test Synergy in a real Cloud production environment in order to verify its functionalities as the fair-share computation algorithm, the robustness as well as its stability under different real usage conditions. Therefore the Synergy prototype was deployed at the Padua Cloud production infrastructure which is integrated with the EGI Federated Cloud. The testbed was composed by one server acting as Controller&Network node while six servers were acting as Compute nodes. The total capacity was: 144VCPUs with 283GB of RAM and 3.7TB of block storage. All servers ran on Ubuntu 14.04 based operating system equipped with the KVM hypervisor.

The Synergy service was installed on the Controller&Network node. The dynamic resources were configured to be the 20% of the total capacity while the remaining 80% was statically partitioned among seven projects, to support seven Virtual Organisations of the EGI Federated Cloud.





Different tests were run in 30 days. Two projects were set up in fair-share mode with 70% and 30% of share respectively. In particular the share of users within the same project was tested in two different configurations: either by defining the same share value for all users or using different values per user.

Tests were submitted using an automatic robot which requested the instantiation of VMs at fixed rate on both projects by using different users. A set of scripts were used to analyze the results by considering both the information provided by the Synergy command-interface as "get_quota" and "get_priority", and the accounting information in terms of VCPU and memory usage. Several cycles of tests were carried out. In each cycle, more than 20,000 VMs were executed over two days, using Cirros images with different flavors. The VM lifetime was limited to 5 min to speed up testing. The project resource usage accounted at the end of each period was measured to be as expected (70% and 30%) within 1% of tolerance.

The results of tests configured with users with different share confirmed the expected limitation of the SLURM Multifactor algorithm as documented at [5], since the expected share of single users was never reached.

None service disruption has occurred during the tests. The performed tests coexisted and did not interfere or degrade the normal operations of other production projects/VOs (not involved in fair-share computation).

7. Next steps

The limitation of the SLURM Multifactor algorithm does not prevent the use of Synergy but requires all users have the same share. The Fair Tree [5] is a more sophisticated algorithm and fully solves the observed issue. Therefore improving Synergy with this new algorithm will be explored. A further improvement on the fair-share strategy could be achieved by changing the resource usage calculation. Currently the calculation is based on CPU wall-clock time but it can be optimized by considering the different CPU performance of the Compute nodes, measured with HEPSPEC 2006 (HS06) benchmark.

Other stress and scalability tests will be done: Synergy will the tested also in the bigger IN2P3 production site.

The OpenStack community operates around a six-month, time-based release cycle with frequent development milestones. The latest release version may differs from the previous one by a set of new features in addition to bug fixes. In particular the enhancements may imply changes on the RPC APIs as well as the database schemas and consequently Synergy must be updated. The ultimate goal is to have Synergy integrated in the Official OpenStack distribution. Contributing internally as OpenStack development team should minimize the development effort.

8. Conclusions

In this paper Synergy, a new service, developed in the context of the European INDIGO DataCloud project, was presented. It enhances OpenStack with an advanced resource provisioning model based on the fair-share scheduling algorithm. Its capabilities, architecture design and implementation have been discussed, also highlighting its limitations. The development process is not completed. The prototype must be consolidated and improved with

new features. The ultimate goal is to have Synergy integrated in the Official OpenStack distribution.

References

[1] https://www.indigo-datacloud.eu/

[2] https://computing.llnl.gov/linux/slurm/priority_multifactor.html

[3] https://launchpad.net/synergy-service

[4] https://launchpad.net/synergy-scheduler-manager

[5] http://slurm.schedmd.com/SUG14/fair_tree.pdf