PROCEEDINGS OF SCIENCE

Synergy, a new approach for optimizing the resource usage in OpenStack

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

Ervin Konomi
INFN, Sezione di Padova
via Marzolo 8, 35131 Padova, Italy
E-mail: ervin.konomi@pd.infn.it

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

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

Vincent Llorens
IN2P3 Computing Center - CNRS
21 AVE P. de Coubertin, 69100 Villeurbanne, France
E-mail: vincent.llorens@cc.in2p3.fr

Managing resource allocation in a Cloud-based data center serving multiple virtual organizations is a challenging issue. In fact, while LRMS (Local Resource Management Systems) are able to maximize the resource usage by fairly distributing computing resources among different user groups according to specific policies imposed by the data center administrator, this is not so straightforward in the most common Cloud management frameworks (e.g. OpenStack, OpenNebula). For example, the current OpenStack implementation provides a too simplistic scheduling model based on an immediate First Come, First Served paradigm. Therefore, a user request will be rejected if no resources are immediately available: it is then up to the user to later re-issue the same request. Moreover the resource provisioning is only limited to the static partitioning strategy. In particular, each project is assigned an agreed and fixed quota of resources that cannot be exceeded by one group even if there are unused resources allocated to other groups. The EU-funded INDIGO-DataCloud project is addressing this issue through 'Synergy', a new advanced scheduling and resource provisioning service targeted at OpenStack. With Synergy it is possible to maximize the resource utilization by allowing OpenStack projects to consume extra shared resources in addition to those statically allocated. Therefore such projects can now access two different kinds of quotas: the private quota and the shared one. The first one is the OpenStack quota operated in a standard OpenStack way. The shared quota is instead handled by Synergy and is composed of resources non statically allocated. Such shared resources are fairly distributed among users following the fair-share policies defined by the administrator. In case the user request cannot be immediately satisfied, it is not rejected but instead inserted in a persistent priority queue and

[1]Speaker

PoS(ISGC2017)012

scheduled later. We present the architecture of Synergy, the status of its implementation, some results demonstrating its functionalities and the foreseen evolution of the service.

International Symposium on Grids and Clouds 2017 -ISGC 2017-
5-10 March 2017
Academia Sinica, Taipei, Taiwan

1. Introduction

One of the problems common to all data centers, regardless of their size, concerns the efficiency of their hardware resources. The adoption of advanced strategies of resource distribution and management, makes it possible to optimize their use, achieving high levels of energy efficiency with low costs, all without compromising the quality of service (QoS) offered to its users. The best strategy, however, depends on several factors, including the number of users and their requirements. In the context of the Scientific Research, a number of international data centers of small and medium size provides the computational needs to the local scientific communities and experiments for executing their research activities. From a computational point of view, different activities are carried out concurrently in the same computing infrastructure. Moreover, they don't require a constant consumption of resources, but it can vary significantly in the medium and long term. In addition, peaks in usage requests can happen, saturating the entire data center. Therefore, to maximize the overall efficiency in terms of usage of computational resources, a common strategy is to adopt an adequate resource allocation model to ensure the research groups an average computing capacity in the long term rather than always guaranteeing a fixed quota of resources, that would be often unused. This strategy is usually implemented by batch systems that, with their sophisticated scheduling algorithms called fair-share, allow a dynamic partitioning of resources based on the resource sharing model. So the available resources are shared among the stakeholders on the basis of assignment policies defined by the administrator.

In these last few years, the growing interest of the scientific world for Cloud Computing, considered as a promising computing paradigm, has stimulated the deployment of many private Cloud infrastructures in data centers. These infrastructures based on open source Cloud Management Frameworks are small or medium-sized and intended exclusively for local scientific communities for their experiments. This technological choice entails a number of advantages to the benefit of both users and data center administrators. From the point of view of the experiments, the Cloud provides a high level of scalability, flexibility and simplicity of use such as to enable them to meet their functional requirements. The administrator instead benefits from these advanced techniques of resource management, especially the virtualization, which can increase the performance of physical servers by allowing to resize the data center resources with a clear economic advantage. Within the same infrastructure, the Cloud resources are managed in a centralized way, this promotes an easier and more efficient allocation to users. However, private Cloud infrastructures are not yet an ideal tool for scientific computing. Unlike the public Clouds where resources look infinite, enabling multi-tenant applications to scale, private Clouds for the scientific community often operate in a saturation regime. The main issue is the resource partitioning model adopted by the most renowned open source Cloud Management Frameworks such as OpenStack and OpenNebula. Both are widely used in the commercial/industrial world and in science. In fact, they offer just a static partitioning model, implemented through the mechanism of quotas: the administrator assigns to various groups of users a fixed number of resources. The static partitioning model fully reflects the economic model on which the full Cloud paradigm is built but is too rigid for scientific computing. This model strongly limits users, as they can only access a subset of the overall computing capacity.

It also decreases the global efficiency of data centers which aim to maximize the utilization of their own hardware for optimizing the cost in terms of energy and management.

Today the efficient management of Cloud resources is an open challenging issue. The EU-Funded INDIGO-DataCloud European project is addressing it through several, complementary approaches. One of them is the Synergy service.

2. Resource allocation and scheduling models in OpenStack

In OpenStack every kind of resources (e.g. computing, storage, networking, etc) is partitioned among projects. Each project is assigned a quota of resources agreed with the administrator. The quotas cannot be shared among different projects and are deliberately limited in size to prevent improper consumption of Cloud capacity. In fact, in Cloud computing there is no concept of "expiration time" and once allocated, the resource belongs to the user that requested it until its explicit release. This limitation happens whenever a project reaches its quota threshold: in this scenario, it cannot consume extra resources in addition to those allowed by the quota, even if there are unused resources allocated to other projects. This is due to the immediate processing on the basis of the First Come, First Served (FCFS) scheduling model, that is, the user requests are scheduled by considering their chronological order and if they cannot be immediately fulfilled are inexorably rejected.

Synergy is a Cloud service which aims to add to OpenStack a complementary resource allocation and scheduling model, besides the legacy one. The adopted approach is based on advanced batch system strategies and in the long term, it is able to ensure the research groups a higher average computing capacity than the one provided by fixed quotas.

3. How to maximize the global efficiency in OpenStack based IaaS

In scientific research computing, a common strategy to maximize the overall efficiency in terms of usage of a traditional computing cluster is to share resources among users and groups. Since the users compete for limited resources, to prevent any monopolization of them for a long time, it is usual to control how resources should be shared by competing users. This can be done by using the fair-share algorithm, a sophisticated scheduling logic implemented by the batch systems. A set of fair-share policies defined by the administrator assign a fixed number of shares to each user or group. These shares represent a fraction of the resources that are available in the cluster. The most important users or groups are the ones with the most shares, while users who have no shares cannot run jobs. To each user is assigned a dynamic priority calculated periodically by a priority formula which takes into account several factors as the user share and the resources its jobs have already consumed, with respect to the other users. All jobs are placed into a queue and are ordered by the users' priorities. Therefore the first job in the queue belongs to the user with the highest dynamic priority.

This resource sharing model can be applied in OpenStack by defining a pool of resources to be shared among projects and regulated by a set of fair-share policies. According to the OpenStack terminology, this pool can be considered as a “shared quota”.

4. The Synergy service and its managers

Synergy is a Cloud service designed to execute generic tasks in OpenStack. Its functionalities are provided by an extensible collection of independent and pluggable managers. Every manager performs specific actions executed periodically (like cron jobs), or interactively by users through a RESTful API. Different managers can coexist and cooperate together and with external services such as the OpenStack components, to implement even complex tasks. At startup, Synergy configures its managers by applying their configuration parameters defined by the administrator. Right after, it initializes them and checks for any problem that may arise during this phase. Finally, it starts the execution of the managers configured to automatically start. All the others can be started in a second time by the administrator through the command line tool which allows the control over the full life cycle of the managers. Synergy and its managers are fully written in Python and any new manager can be easily implemented by using its API.

We addressed the resource efficiency problem mentioned above by implementing six specific managers which, together, extend the Synergy capabilities. They provide a complementary resource allocation and scheduling mechanism which allows the administrator to define a pool of resources to be shared among projects. Such a pool is called “shared quota” and gives to the OpenStack projects the opportunity to consume extra resources in addition to those statically assigned (i.e. the “private quota”). The shared quota is composed of resources not statically allocated and are fairly distributed among the different users by Synergy. The size of such quota is calculated as the difference between the total amount of cloud resources and the total resources allocated to the private quotas (figure 1)

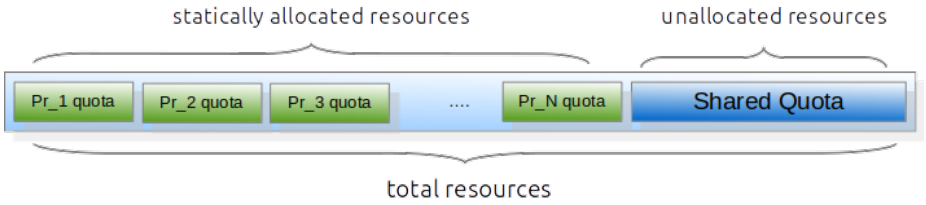

figure 1:The different kinds of quotas in the Synergy resource allocation model

Only the projects selected by the administrator can consume the shared resources and, unlike the private quota, the user requests that cannot be immediately satisfied are not rejected: they are instead inserted in a persistent priority queue. Synergy assigns to each user a dynamic priority, periodically calculated by considering the historical resource usage in a well defined time window, the project and user shares and some specific weights (e.g. age, decay, cpu usage, memory). The enqueued requests are processed according to their priority ordering. In case there are no resources available for the selected request, it is skipped and Synergy processes the next one in the queue (i.e. backfilling): this allows to maximize the resource utilization and, at the same time, avoid to block the queue for long time. Finally to prevent any monopolization of the shared resources, Synergy enforces a maximum allowed lifetime (e.g. 48 hours) of the

relevant instances as Virtual Machines and Docker containers. Therefore they are automatically destroyed right after the expiration time if not released explicitly by the user.

4.1 The high level architecture

The high level architecture is shown by the following schema (figure 2) which highlights the interactions among managers and the external OpenStack components:

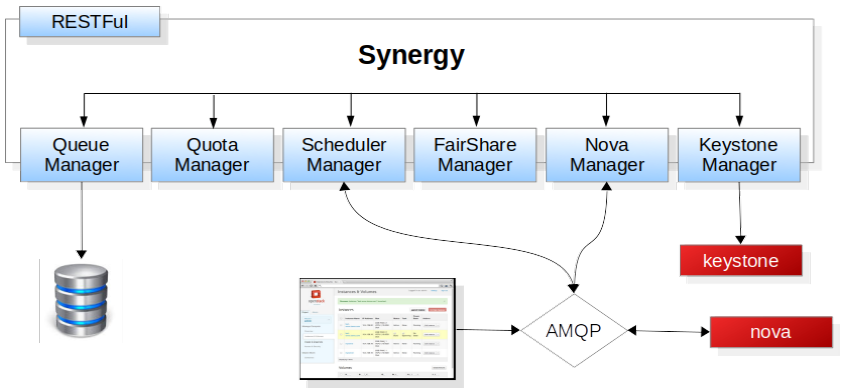

figure 2: The Synergy high level architecture

- Scheduler-Manager processes the user requests from Nova-Manager according to the private or shared quota policy;
- FairShare-Manager is designed to implement the main fair-share logic by assigning dynamically the right priority value to each user. The priority value is computed as a weighted sum of factors representing the past usage in terms of CPU, memory and disk. Weights can be tuned according to the importance assigned to a specific factor, allowing e.g. an administrator to make the CPU dominant with respect to other parameters. The fair-share algorithm based on the Priority Multifactor of SLURM [1] was selected for the first implementation;
- Queue-Manager takes care of the persistent priority queue which contains the user's requests for consuming the shared quota;
- Quota-Manager handles the shared quota and periodically computes its size. In case of quota saturation it blocks the related scheduling process until the required resources are again available. This manager also handles the VM/Container lifetime policies applied to the shared resources. It invokes the Nova-Manager for destroying the VM/Container whenever this exceeds the defined time limit;
- Nova-Manager interacts with the OpenStack Nova components, responsible for the management of VM/Container lifetime;
- Keystone-Manager interacts with the OpenStack Keystone service, responsible for all the authentication and authorization steps.

4.2 The Synergy integration in OpenStack

To provide the new functionalities described above, Synergy has to intercept all user requests for the instantiation of new servers (i.e. openstack server start). To do that it has to act as a "man in the middle" between the Nova-API and the Nova-Conductor services (figure 3):

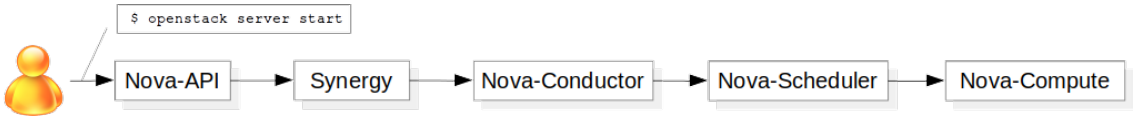

figure 3: Synergy acts as a "man in the middle" between the Nova-API and the Nova-Conductor

Nova-API uses the AMQP messaging system (e.g. rabbitmq, qpid, etc) to communicate with all Nova components which are all listening on specific topics (figure 4):

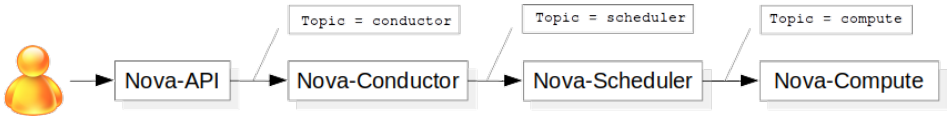

figure 4:The AMQP topics for communicating with the Nova components

To allow the interception, Nova-API must be configured in order to send the user requests directly to Synergy which listens on a specific new topic “synergy” (figure 5):

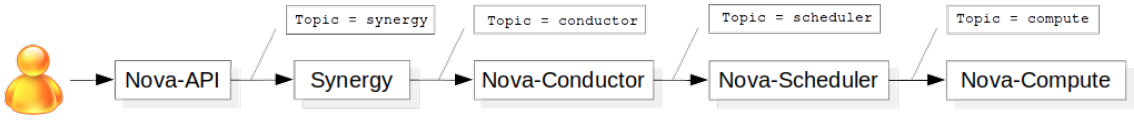

figure 5: The “synergy” topic

This setup requires just the creation of a new configuration file for Nova-API (i.e. /etc/nova/nova-api.conf) where the administrator specifies the new “synergy” topic in the conductor section (i.e. [conductor] topic=synergy) [2].

5. Development details

Since Synergy was growing as a software project, we faced new challenges to improve our development practices. Although Synergy is born in the context of the INDIGO-DataCloud project, it has been designed with the hope of being integrated into the official OpenStack distribution. Therefore the natural choice was to adopt the specific OpenStack development guidelines, including the coding style and the development tooling. This approach doesn't prevent to meet the SQA (Software Quality Assurance) requirements of INDIGO.

5.1 Taking advantage of the OpenStack development infrastructure

OpenStack provides a development infrastructure [3] that unifies the workflow for all the projects in its ecosystem, even for those not officially supported. The process to make Synergy part of it was quite complex and required the interaction with the OpenStack Infrastructure team. To make collaboration easier, the OpenStack infrastructure provides: a code collaboration platform [4], integration with the Launchpad issue tracker [5, 6], and automatic testing and building [7] upon code changes.

The code collaboration platform is running the code review tool Gerrit. Once one of the developers has made a commit to its local Git repository, he then makes the change public by submitting the commit for review using the `git review` command. Other developers -- be it Synergy developers or anyone having an OpenStack developer account, since the platform is open -- can then comment on the submitted change and state if he thinks the commit should be merged or not. If the commit has to be reworked, any developer can submit changes to it.

Follows another round of comments and acceptation or rejection of the commit until it reaches a good state. While the developers exchange about the commit, a continuous integration tool checks in the background that the submitted code changes pass unit tests and follow the OpenStack Python style guide. When the developers agree on the commit state and the automatic checks validate it, the commit is automatically merged.

The OpenStack Infrastructure team set up integration with the Launchpad issue tracker. It makes managing bugs easier: when we commit a bug fix we can reference the corresponding issue on Launchpad. When the bug fix is merged, Gerrit will automatically mark the issue as *Fix Committed*.

Using the OpenStack infrastructure allowed us to collaborate efficiently, getting feedback quickly and managing code changes effortlessly.

5.2 Turning packaging Synergy into a simple task

As Synergy evolved to a mature state, the requirement to make it easily installable by system administrators became very important. Thus we began working on the process of building system packages. We developed a system based on Docker to build packages, which allows to make the packaging task simple and fast. Since we target two operating systems (Ubuntu and CentOS) we prepared two Dockerfiles, one for each OS. Using Docker as a packaging system has added the benefit of packaging for a different OS than the one used by the developer.

Producing a Synergy package (a RPM in the following example) is easy and simple as running a single Docker command:

```
$ docker run -i -v /path/to/synergy-service:/tmp/synergy synergy-centos7-builder
```

The time between introducing a change in the Synergy code base and making it into a system package was dramatically shorten, therefore helping also to reduce the time needed to test such changes.

6. Testing

Synergy was deployed at the production INFN Padova Cloud infrastructure which is integrated with the EGI Federated Cloud. The goal was to thoroughly test the behavior and the stability of Synergy under real use conditions that only a production environment is able to provide. Therefore a test suite was released to validate Synergy under different aspects: *functionality tests* allowed to verify that each planned functionality has been implemented correctly, *load tests* were essential to evaluate the efficiency of the fair-share computation algorithm with a varying number of users and finally the *stress tests* have checked the recovery capability of the system after a failure in case of heavy load. Some ad-hoc tests were also performed. In particular we ran a 30 day test in a Cloud infrastructure composed of a controller and network node and six compute nodes with a total capacity of 144 VCPUs, 283 GB of RAM and 3.7 TB of block storage, of which 20% was reserved for the shared quota. The servers run the CentOS7 operating system equipped with KVM hypervisor. Synergy was installed on the controller and network node and configured to enable two projects to access the shared quota with 70% and 30% of share respectively (figure 6). Moreover the share of users within these

two projects was tested in two different configurations: same share for all users and different share for each user.

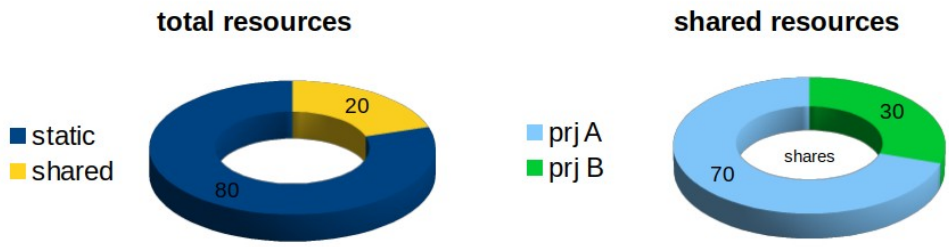

figure 6: The testbed configuration in terms of resource allocation

A Cron job executing a script was used as an automatic robot to request the instantiation of a reasonable number of VMs at a fixed rate from different users of both projects. The results were collected by using the Synergy client commands and the Nova API for the full accounting information of VCPU and memory usage. In each testing session more than 20K VMs over two days were instantiated using Cirros images with different flavors. In order to speed up the test, the VM lifetime was limited to 5 min. Several testing sessions with different configurations were carried out during the month. At the end of the time window period defined for the fair-share algorithm in the Synergy configuration, the project resource usage was always measured to be as expected (70% and 30%) within 1% of tolerance. When the individual users within a project were configured to have different shares, the results of the tests confirmed the expected limitation of the SLURM Multifactor Priority algorithm, as documented in [8]. Indeed, the expected share of the individual users was never achieved. The tests did not disrupt any OpenStack production service, that continued to be available and worked properly. The normal operations of other production projects, i.e. the ones not involved in fair-share computation, were neither affected nor degraded by the coexistence with Synergy.

7. Conclusions

In this article we described Synergy, the new Cloud service developed in the context of the European INDIGO DataCloud project. The issue of how to maximize the global efficiency in terms of resource usage of the private Clouds based on OpenStack has been firstly examined. Then we discussed how Synergy addresses the challenge by highlighting its functionalities, architecture design and implementation details. Moreover, we described how we managed to integrate Synergy into the OpenStack Infrastructure, which is a step forward making Synergy recognized as an OpenStack component. Streamlining and, where possible, automating the redundant tasks such as sharing code, testing or packaging it was highly beneficial for us because the time saved can be used for thinking and writing code.

References

[1] *The SLURM Multifactor Priority*, https://slurm.schedmd.com/priority_multifactor.html

[2] *The Synergy documentation*, https://indigo-dc.gitbooks.io/synergy-doc/

[3] *The OpenStack developer's guide*, https://docs.openstack.org/infra/manual/developers.html

[4] *The OpenStack collaboration platform*, https://review.openstack.org/

[5] *The Launchpad issue tracker for the Synergy-Service*, https://bugs.launchpad.net/synergy-service

[6] *The Launchpad issue tracker for the Synergy-Scheduler-Manager*, https://bugs.launchpad.net/synergy-scheduler-manager

[7] *The Zuul for automated testing and building of OpenStack projects*, http://status.openstack.org/zuul/

[8] *The SLURM Fair Tree algorithm*, https://slurm.schedmd.com/fair_tree.html