Optimizing the resource usage in Cloud based environments: the Synergy approach

L Zangrando¹, V Llorens², M Sgaravatto¹ and M Verlato¹

¹INFN, Sezione di Padova, Via Marzolo 8, I-35131 Padova, Italy
²IN2P3 Computing Center - CNRS, 21 Avenue Pierre de Coubertin, 69100 Villeurbanne, France

E-mail: Lisa.Zangrando@pd.infn.it

Abstract. Managing resource allocation in a cloud based data centre serving multiple virtual organizations is a challenging issue. In fact, while batch systems are able to allocate resources to different user groups according to specific shares imposed by the data centre administrator, without a static partitioning of such resources, this is not so straightforward in the most common cloud frameworks, e.g. OpenStack. In the current OpenStack implementation, it is only possible to grant fixed quotas to the different user groups and these resources cannot be exceeded by one group even if there are unused resources allocated to other groups. Moreover in the existing OpenStack implementation, when there aren’t resources available, new requests are simply rejected: it is then up to the client to later re-issue the request. The recently started EU-funded INDIGO-DataCloud project is addressing this issue through “Synergy”, a new advanced scheduling service targeted for OpenStack. Synergy adopts a fair-share model for resource provisioning which guarantees that resources are distributed among users following the fair-share policies defined by the administrator, taken also into account the past usage of such resources. We present the architecture of Synergy, the status of its implementation, some preliminary results and the foreseen evolution of the service.

1. Introduction

In OpenStack [1] based IaaS private clouds the computing and storage resources are statically partitioned among projects. A user typically is member of one project and each project has its own fixed quota of resources defined by the cloud administrator. A user request is rejected if the project quota has been already reached, even if unused resources allocated to other projects would be available. This rigid resource allocation model strongly limits the global efficiency of the data centres, which aim to fully utilise their resources for optimizing costs. In the traditional computing clusters the utilisation efficiency is maximized through the use of a batch system with sophisticated scheduling algorithms plugged in.

Today, the INDIGO DataCloud project [2] aims at addressing the cloud efficiency issue through Synergy, an advanced service interoperable with the OpenStack components, which implements a new resource provisioning model based on pluggable scheduling algorithms. It allows to maximize the resource usage, at the same time guaranteeing a “fair” distribution of resources among users and groups. The service also provides a persistent queuing mechanism for handling those user requests exceeding the current overall resource capacity. These requests are processed according to a priority defined by the scheduling algorithm, when the required resources become available.

This paper describes in detail the Synergy functionalities, the architectural design and its implementation, highlighting the integration aspects with the OpenStack cloud framework. It also reports about the results of some tests performed in the EGI Federated Cloud [3] and the current limitations of the chosen scheduling algorithm.
2. Synergy overview

2.1. Main functionalities

Synergy has been designed to provide a number of advanced functionalities not available in OpenStack. At first, the service allows the cloud administrators to define a subset of resources to be shared among different projects, besides the ones statically partitioned by fixing each project quota. This subset in the rest of the document is referred as “shared quota” or “shared resources”. The end user is then able to specify if his/her request is targeted to the shared resources or to the static ones.

Internally, Synergy is able to timely recalculate the size of the shared resources set, taking into account the overall resource capacity and its statically allocated fraction, which could change e.g. when adding new projects. Synergy also provides a persistent queuing mechanism for those user requests that cannot be instantly fulfilled, but can be served at a later stage as the required resources become available. It implements a resource allocation mechanism based on a fair-share scheduling algorithm.

An OpenStack IaaS controlled by Synergy allows its administrators to:

1. enable a project to consume both static and shared resources. While static resources are managed in the standard OpenStack mode, requests targeted to the shared ones are processed in an order based on a priority weight calculated by the fair-share algorithm. The ones that can’t be instantly satisfied hang in a persistent priority queue and are served only when the required resources are available. The shared resource usage control depends on a set of fair-share configurable policies that define:
   ○ a positive integer which represents the number of shares of cloud resources assigned to a given project. Higher the number, higher the fraction of resource usage is guaranteed on average in the defined time window
   ○ a maximum lifetime for the Virtual Machines (VM) or Containers, after which they are destroyed. This is needed to prevent VM/Container running indefinitely, that would make the fair-share algorithm application ineffective

2. enable the traditional OpenStack mode for a given project: in this case the requests are as usual processed according to the First Come First Served (FCFS) model. The ones that can’t be instantly satisfied are simply rejected and lost.

As required by the operation mode 1. described above, Synergy provides an automatic killing mechanism for destroying all the VMs or Containers, instantiated in the shared quota, running over their defined maximum lifetime. The fair-share based resource allocation model would not be working without this enforcement. Finally, Synergy provides its own command line tool, following the OpenStack style, to manage and control the operations of the service.

2.2. The managers

Synergy can be considered an extensible general purpose management service fully integrated in OpenStack. It is composed by a set of managers which are specific and independent pluggable tasks that implement the capabilities described in the previous section. The tasks are executed periodically like system cron jobs, or interactively by means of RESTful API.

The different managers can interact with each other in a loosely coupled manner, or coexist without interacting. The whole design, shown in the simplified picture of figure 1, allows the implementation of complex business logic.
Figure 1. The Synergy service (the red man) coordinates a set of managers (the blue men). The arrows represent the interactions among managers and/or external OpenStack services.

More precisely, the Synergy python API provides an abstract base class “Manager”, shown in figure 2. Any new manager can be easily implemented by extending this abstract base class.

```python
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
```

Figure 2. The Synergy abstract base class “Manager”.

In particular, synchronous and asynchronous activities are respectively implemented by the last two methods: “execute” and “task”.

2.3. Resource allocation
Synergy provides a new advanced allocation model based on a new kind of quota: “shared”. The shared quota is a subset of the total resources, can be shared among projects and allows to enqueue requests that cannot be immediately fulfilled. The size of such quota is calculated as the difference between the total amount of cloud resources and the total resources statically allocated to projects, as shown in figure 3.

Figure 3. The different kinds of quotas in the Synergy resource allocation model.

Its resources are fairly distributed according to a set of fair-share policies defined by the cloud administrator as:
- the shares for the different projects (e.g. project A=70%, project B=30%, as shown in section 5)
- the shares among different users of the same project (optional)
- the maximum allowed lifetime (e.g. 48 hours) of the relevant instances (Virtual Machines, Containers)

3. Synergy Architecture

3.1. High level architecture

The advanced resource allocation model has been implemented with a set of specific Synergy managers, as described in figure 4:

![Synergy high level architecture](image)

- 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 authorisation steps
- FairShare-Manager is designed to implement the main fair-share logic by assigning dynamically the right priority value to each user request. The priority value is computed as a weighted sum of factors representing the past usage and the fair-share 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 term dominant with respect to other parameters. The fair-share algorithm based on the Priority Multifactor Strategy of SLURM [4] was selected for the first implementation. However, the architecture allows to plug more advanced algorithms, that will be explored in the future
- Scheduler-Manager processes the requests from Nova-Manager and schedules them according to the private or shared quota policy
- Queue-Manager takes care of the persistent priority queues. Several queues can be managed. Any user request for which the fair-share calculation is required is in fact stored in a queue and asynchronously processed by the Scheduler-Manager, according to its priority. The relevant information entered in the queue are: full user request, timestamp, priority, retry count, trusted token
- Quota-Manager handles all project quotas. It periodically computes the size of the shared quota and, in case of quota saturation, 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 these exceeds the defined limit.

Synergy behaves as an independent OpenStack service, while not replacing any of the OpenStack components. They do not need to be modified in order to interact with Synergy, excepting small changes to the standard configuration files. Most importantly, Synergy doesn’t force the utilization of its resource allocation model and doesn’t interfere with the normal operations of the projects that do not require the shared quota.

3.2. Low level architecture
The low level architecture of Synergy is shown in figure 5. In particular, the interaction of Synergy with the existing OpenStack components is pointed out.

![Figure 5. Synergy low level architecture.](image)

All user requests, coming either from the Horizon Dashboard [5] or the command line, are intercepted by the Nova-Manager and delivered to the Scheduler-Manager. The Scheduler-Manager analyzes the request and decides whether to allocate the VM/Container instance to the private quota or to the shared one. The requests assigned to the private quota are immediately scheduled and processed by respecting the quota limits, whereas the ones for the shared quota are inserted in a persistent priority queue managed by the Queue-Manager.

The priority is computed by the FairShare-Manager and periodically recalculated by considering even request age, needed for guaranteeing a good QoS. The request with the highest priority is fetched from the queue by the Scheduler-Manager, which processes and sends it to the Nova-Scheduler by means of the Nova-Manager, using the AMQP messaging system [6]. In case there are no resources available for the selected request, it is skipped and the next one in the queue is processed (i.e. backfilling): this allows the maximization of the resource utilization and, at the same time, avoids blocking the queue for a long time.

A retry mechanism is also in place within the Scheduler-Manager that in case of failure keeps track of the failed requests and inserts them back into the queue for a configurable number of retries. Synergy delegates all tasks related to security and user and project information management to the Keystone-Manager.

It is worth to notice that from the user point of view the queued requests remain in the “Scheduling” state until enough computing resources are available again. No new states has been added to the VM/Container management to prevent any possible interaction issue with the existing OpenStack clients.
4. Implementation details
Synergy has been implemented following the OpenStack style and its specific development guidelines, which define the coding style and the technologies to be used. The source code is in launchpad [7][8] and its tracking system is being used. The OpenStack Continuous Integration system is also used.

Synergy exposes a well defined RESTful python API, which provides commands for e.g. activating a manager, suspending it, getting its status, etc., or executing specific commands on the manager, like get/set quota, list queues, etc. New specific interfaces can also be included by extending the API. Internally, Synergy interacts with the Nova and Keystone OpenStack services by invoking their RESTful interfaces or using the internal RPC APIs.

The synergy.conf file contains all the Synergy default configuration options [9]. Synergy accesses the Nova database through the provided API to retrieve the historical resource usage required to compute the priority factors defined in the fair-share scheduling algorithm. This has proven to guarantee good performance, while alternative solutions like adopting the OpenStack Ceilometer metering service are under study.

Lastly, for what concerns the storage of persistent data as the priority queues, Synergy adopts the same backend used by OpenStack.

5. Testing results in production environment
Synergy was first deployed at INFN-Padova OpenStack production site of the EGI Federated Cloud. The goal was to test its behaviour and stability under real usage conditions typical of a production environment. This cloud infrastructure consists of one server acting as Controller and Network node, and six nodes acting as Compute nodes. Its total capacity is of 144 VCPUs, 283 GB of RAM and 3.7 TB of block storage. The servers run the CentOS7 operating system and use KVM as hypervisor. For this test the resource allocation and the projects’ shares were defined as shown in figure 6.

![Figure 6. Resource allocation and share definition for the testing environment.](image)

Shared resources accounted for 20% of the total. Two projects were configured to access the shared quota, and assigned with 70% and 30% of share respectively. The user share within a project was tested in two configurations: 1) same share for all users; 2) different share for each user. A cron job executing a script was used as 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 exploiting Synergy client commands and the Nova API for the full accounting information of VCPU and memory usage. Each testing session was designed to execute more than 20,000 VMs over two days, instantiated from 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 one month.

In summary, at the end of the time window 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 algorithm, as documented in [10]. Indeed, the expected share of the individual users was never achieved.

About stability and robustness, the tests did not disrupt any OpenStack production service, that continued to be available. 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.

6. Conclusion and future work
We described in this article the new service - Synergy - developed in the context of the European INDIGO DataCloud project. It aims at filling a gap existing in the current open source cloud frameworks like OpenStack by integrating a fair-share policy based algorithm allowing for an advanced resource provisioning model that maximizes the resource usage of a data centre. We discussed the Synergy functionalities, its architectural design and the implementation, also highlighting the current limitations. The development process is ongoing and future work briefly summarized below has been already planned to consolidate and improve Synergy with new features. The tests have demonstrated that the expected limitation of the SLURM Multifactor algorithm does not prevent the use of Synergy, but requires that all users must have assigned the same share. A more sophisticated algorithm that fixes the observed issue has been already developed and available as fair-share algorithm for SLURM: the Fair Tree [10]. The improvement of Synergy with this new algorithm is planned in a next release. A further improvement on the fair-share strategy is possible by considering the different CPU performance of the Compute nodes, measured e.g. with the HEPSPSEC 2006 (HS06) benchmark. The current implementation in fact bases the resource usage calculations on just the CPU wall-clock time, but does not take into account the effective CPU power.

Deeper stress and scalability tests have been planned on bigger production sites of INFN and IN2P3. The fast development cycle adopted by the OpenStack community leads to a new major OpenStack release every six months. In addition to bug fixes, the latest release version often differs from the previous one by a set of new features or enhancements, which typically imply changes on the RPC APIs or the database schemas. This requires a continuous update of Synergy components. Thus, in order to minimize the development effort, we are actively working to have Synergy integrated in the official OpenStack distribution, opening it to contributions from the whole OpenStack community.

Acknowledgments
The authors want to acknowledge the support of the INDIGO-DataCloud (grant number 653549) project, funded by the European Commission's Horizon 2020 Framework Programme.

References
[1] OpenStack web site: www.openstack.org
[2] Salomoni D et al., arXiv:1603.09536v3 [cs.SE]
[3] Fernandez-del-Castillo E et al., Procedia Computer Science 68 (2015 ) 196 – 205
[4] Multifactor Priority Plugin, https://slurm.schedmd.com/priority_multifactor.html
[5] Horizon Dashboard project, https://docs.openstack.org/developer/horizon/
[6] AMQP web site, https://www.amqp.org/
[7] https://launchpad.net/synergy-service
[8] https://launchpad.net/synergy-scheduler-manager
[9] Synergy documentation: https://www.gitbook.com/book/indigo-dc/synergy-doc/details
[10] Fair Tree Fairshare Algorithm, https://slurm.schedmd.com/fair_tree.html