A FairShare Scheduling Service for OpenNebula

S Bagnasco\textsuperscript{1}, S Vallero\textsuperscript{1} and V Zaccolo\textsuperscript{1}
\textsuperscript{1} INFN Torino, Via Pietro Giuria 1, 10125 Torino, Italy
E-mail: sara.vallero@to.infn.it

Abstract. In the ideal limit of infinite resources, multi-tenant applications are able to scale in/out on a Cloud driven only by their functional requirements. While a large Public Cloud may be a reasonable approximation of this condition, small scientific computing centres usually work in a saturated regime. In this case, an advanced resource allocation policy is needed in order to optimize the use of the data centre. The general topic of advanced resource scheduling is addressed by several components of the EU-funded INDIGO-DataCloud project. In this contribution, we describe the FairShare Scheduler Service (FaSS) for OpenNebula (ONE). The service must satisfy resource requests according to an algorithm which prioritizes tasks according to an initial weight and to the historical resource usage of the project. The software was designed to be less intrusive as possible in the ONE code. We keep the original ONE scheduler implementation to match requests to available resources, but the queue of pending jobs to be processed is the one ordered according to priorities as delivered by the FaSS. The FaSS implementation is still being finalized and in this contribution we describe the functional and design requirements the module should satisfy, as well as its high-level architecture.

1. Introduction
In the ideal limit of infinite resources, multi-tenant applications are able to scale in/out on a Cloud driven only by their functional requirements. A large Public Cloud may be a reasonable approximation of this condition, where tenants are normally charged \textit{a posteriori} for their resource consumption. On the other hand, small scientific computing centres usually work in a saturated regime and tenants are charged \textit{a priori} for their computing needs by paying for a fraction of the computing/storage resources constituting the Cloud infrastructure. Within this context, an advanced resource allocation policy is needed in order to optimize the use of the data centre. We consider a scenario in which a configurable fraction of the available resources is statically assigned and partitioned among projects according to fixed shares. Additional assets are partitioned dynamically following the effective requests per project. Efficient and fair access to such resources must be granted to all projects. This is achieved by satisfying resource requests according to an algorithm which prioritizes tasks according to an initial weight and to the historical resource usage of the project, irrespective of the number of tasks she has running on the system.

The general topic of advanced resource scheduling is addressed by several components of the EU-funded INDIGO-DataCloud project [1]. In that context, dedicated services for the OpenNebula [2] and OpenStack [3] cloud management systems are addressed separately, because of the different internal architectures of the systems. In this contribution, we describe the FairShare Scheduler Service (FaSS) for OpenNebula (ONE).
2. Service requirements

2.1. Functionality
The functional requirements that the Fair Share Scheduling service should satisfy are:

- provide a dynamic resource partitioning model, which handles all unallocated resources (i.e. the dynamic fraction of resources) and shares them among projects
- guarantee the coexistence of the dynamic and static partitioning models
- for dynamic resources, provide an allocation mechanism based on a fair-share algorithm
- define a new kind of project/tenant quota, named *dynamic quota*, to be assigned to all projects wishing to access to the dynamic resources
- recalculate periodically the size of the dynamic quota in order to guarantee the amount of resources allocated to the static quotas, which can change in time
- provide a queuing mechanism for handling the requests that cannot be immediately fulfilled (contrary to the OpenStack case, this functionality is already built-in in OpenNebula)
- possibly apply the fair-share mechanism seamlessly also to non standard resources (e.g. GPUs)

The new service should process the following inputs:

- a queue of virtual machines to be deployed
- a set of priority values reflecting the initial shares
- the historical information on the resource usage by each project

The output produced should be a queue of virtual machines to be deployed, reordered according to recalculated priorities.

The data items to be stored within the new module developed (i.e. not already stored by the ONE system) are:

- the set of initial priority values
- the historical information on the resource usage
- the recalculated set of priority values

The first item can be changed manually by the IaaS administrator, the latter two are updated periodically by the module functionality.

2.2. Design
The software was designed to be less intrusive as possible in the ONE code. By keeping minimal dependencies on the ONE implementation details, we expect our code to be fairly independent on future ONE internals changes and developments. The scheduling service is structured as a self-contained module interacting only with the ONE XML-RPC interface.

Other design requirements that the Fair Share Scheduling service should satisfy are:

- keep the original ONE FIFO scheduler implementation for matching resources to requests
- order the requests queue according to priorities
- do not add new states to the VM life-cycle
- act in synergy with the ONE authentication, quotas, monitoring and accounting systems
Figure 1. High-level architecture of the FaSS service. White blocks are native ONE components, while colored blocks represent the newly developed FaSS service. New components are represented in blue, new interfaces/APIs in gray and new tools in green. The original ONE Sunstone GUI is extended to manage FaSS operations.

3. Architecture

The architecture of the FaSS service is depicted in Figure 1. It has been conceived following the ONE conceptual design. In the figure, white blocks are native ONE components, while the newly developed components are shown in blue. New tools are pictured in green, and interfaces/APIs in gray. The only modification required to the paired ONE instance, is to configure its FIFO scheduler to point to the FaSS XML-RPC server instance, instead of the original ONE endpoint. Communication to the original ONE XML-RPC server is handled solely by the FaSS module. This allows us to leave untouched the ONE scheduler implementation, by changing only a configuration parameter. The main components of the FaSS module are described in the remainder of this paragraph.

Priority Manager - It’s the core component of the FaSS module. The Priority Manager’s (PM) main task is to periodically calculate a set of priorities for queued jobs. In doing so, the manager interacts with a set of pluggable algorithms to calculate priorities. During each PM cycle, the list of pending virtual machines (VMs) is retrieved from the ONE endpoint via an instance of the ONE XML-RPC client. The set of initial priority values is given in the initial FaSS configuration. The historical information on the resource usage is an internal data of the FaSS module and it is stored in the FaSS database. The recalculated set of priority values is also stored in the database. The PM periodically outputs a re-ordered list of pending VMs, which is stored in the FaSS database in order to be processed by the ONE FIFO scheduler.

Pluggable algorithms - The algorithm used to calculate the fair-share priorities is related to the PM as a pluggable module. This allows the flexibility to study the performance of different algorithms and to choose the best suited algorithm for the case at hand. For instance, the SLURM FairTree [4] algorithm will be implemented as a default.

XML-RPC interface - The FaSS XML-RPC server is an independent service that runs asynchronously with respect to the PM. It catches the FIFO scheduler calls and provides the reordered queue of pending VMs, when requested. The reordered queue is periodically retrieved from the FaSS database. All other method calls are redirected to the original ONE XML-RPC
interface. This two-step procedure introduces a latency, which could be relevant in case ONE and FaSS run on different hosts. This consideration needs to be taken into account when planning the layout of the Cloud infrastructure.

**Database** - This component holds the modules’ internal data. Since ONE internally describes the properties of pending VMs (and of most of its internal objects) as complex XML data structures, we chose to utilize a NoSQL technology to facilitate search operations inside the database.

**Clients** - They are the interface to query and configure the FaSS service. They rely on a set of bindings analogous to the ONE Cloud API (OCA).

**Sunstone** - The original ONE GUI, named Sunstone, is extended in order to monitor and operate the FaSS service.

4. Implementation status
A prototype of the scheduler is being implemented [5]. Namely, it comprises the Priority Manager and the XML-RPC server. An instance of this prototype is currently running in a test Cloud infrastructure at the INFN Torino site. All the calls from the FIFO scheduler are redirected to ONE by the FaSS XML-RPC server. The only exception is the request of the list of VMs to be scheduled. In this case, the request is processed by FaSS and it returns a list reordered according to a dummy algorithm that simply inverts the queue ordering. The FaSS instance runs on the same host as the ONE main daemons and it has shown not to introduce any penalty in the functionality or performance of ONE. A fully featured prototype, including a realistic algorithm and the installation tools, will be made available for the next INDIGO project release, foreseen at the beginning of April 2017.

Acknowledgments
The authors would like to acknowledge the support of the INDIGO-DataCloud project, funded by the European Commission’s Horizon 2020 Research and Innovation Programme (grant agreement RIA 653549).

References
[1] Bagnasco S, Dal Pra S, Lopez Garcia A, Sgaravatto M, Taneja S, Vallero S and Zangrando L Improved Cloud resource allocation: how INDIGO-DataCloud is overcoming the current limitations in Cloud schedulers In these proceedings
[2] OpenNebula https://opennebula.org/
[3] OpenStack https://www.openstack.org/
[4] Cox R and Morrison L Fair tree: Fairshare algorithm for slurm slurm.schedmd.com/SUG14/fair_tree.pdf
[5] FaSS on GitHub https://github.com/indigo-dc/one-fass