EPJ Web of Conferences **214**, 07001 (2019) https://doi.org/10.1051/epjconf/201921407001
CHEP 2018

Fair Share Scheduler for OpenNebula

Implementation and performance tests

Stefano Bagnasco[1,*], *Sara* Vallero[1,**], and *Valentina* Zaccolo[2,***]

[1]Istituto Nazionale di Fisica Nucleare, Sezione di Torino, Via Pietro Giuria 1, 10125, Torino, Italy
[2]Università e INFN, Dipartimento di Fisica di Trieste, Via Alfonso Valerio 2, 34127 Trieste, Italy

Abstract. A small Cloud infrastructure for scientific computing likely operates in a saturated regime, which imposes to optimize the allocation of resources. Tenants typically pay *a priori* for a fraction of the overall resources. Within this business model, an advanced scheduling strategy is needed in order to optimize the data centre occupancy. FaSS, a Fair Share Scheduler service for OpenNebula, addresses this issue 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. In this proceedings, we are going to describe the implementation of FaSS Version 1.0, released in March 2017 as a product of the INDIGO-DataCloud project. We are also going to discuss the results of FaSS functional and stress tests performed at the Cloud infrastructure of the INFN-Torino computing centre.

1 Introduction

OpenNebula (ONE) [1] is a simple and flexible open-source tool to build and manage Private Clouds. The default OpenNebula scheduler is First-In-First-Out (FIFO) and based only on static resources partitioning among the projects. This scheduling strategy applies well to a large public Cloud, where applications can scale in/out freely since the resources are approximately infinite and tenants are charged accordingly *a posteriori*. The FIFO scheduler, instead, is not suitable for a scientific data Cloud, for which advance resource allocation is needed, given the fact that it often operates at a saturated regime and tenants are charged *a priori*. For this reason, we deployed the Fair Share Scheduler (FaSS) service [2]. In FaSS, task priorities are assigned according to an initial weight set by the user and the historical resource usage. The first version of FaSS was released with ElectricIndigo in April 2017, within the INDIGO-DataCloud project [3]. Version 1.0 contains a patch to run with ONE, while starting from v1.2 the patch has been backfed to the upstream ONE code and there is full compatibility with ONE 5.4. A demo of FaSS is available at [4].

2 High-level architecture

FaSS was designed to be less intrusive as possible in the ONE code, and interacts with ONE exclusively through its XML-RPC interface. The native ONE scheduler is preserved for

*e-mail: stefano.bagnasco@to.infn.it
**e-mail: sara.vallero@to.infn.it
***e-mail: valentina.zaccolo@ts.infn.it

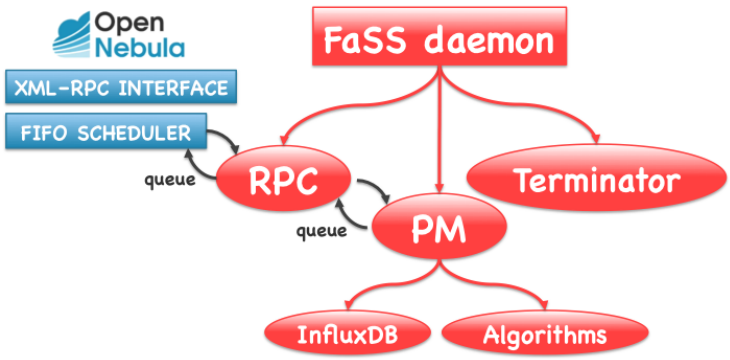

Figure 1. High-level architecture diagram of FaSS components is shown in red, while blue components are part of the ONE infrastructure.

matching requests to available resources. FaSS is stateless are there are no duplicated information in the database. It consists of five functional components: the Priority Manager (PM), a set of fair-share algorithms, the Terminator, the XML-RPC interface and the database, as shown in Fig. 1. The main modules, the RPC, the PM and the Terminator, are asynchronous. The PM periodically requests the list of pending Virtual Machines (VMs) to ONE and recalculates the priorities in the queue by interacting with an algorithm module of choice. In FaSS 1.0 the default algorithm is Slurm's MultiFactor [5]. Moreover, the PM synchronously kills pending VMs. The Terminator module is responsible for removing from the queue VMs in pending state for too long. The XML-RPC server intercepts the calls from the FIFO scheduler of ONE and sends back the reordered VMs queue. FaSS uses InfluxDB [6] (NoSQL system). It stores the initial and recalculated VM priorities and some additional information for accounting purposes, like the initial priorities, the historical usage and the recalculated priorities.

FaSS can also be used to keep the Cloud infrastructure clean [7]. Indeed, it is possible to set VMs to be dynamic and to be terminated after a specific Time-To-Live (TTL). Moreover, the VMs can also be powered-off, suspended or rebooted by the user. Another interesting feature is the one which allows to set different TTLs for each user.

3 Installation and usage

Detailed instructions on how to install FaSS can be found elsewhere [2]. The first prerequisite is to install ONE, noting that versions above 5.4 work with FaSS above v1.2. When running a previous version of ONE one needs FaSS v1.1 or before. The second requirement is to install InfluxDB and create the FaSS database. All other needed packages are installed automatically with the `.rpm` file, and can be found here [2]. After that, the installation of FaSS has to be done as root user. The last step is to adjust the configuration file of the ONE scheduler, to allow it to point at the FaSS endpoint. A detailed usage description can be found in GitHub [2]. An important step is to edit the initial shares for every user.

Figure 2. Top: FaSS performances example with three users. Bottom: FaSS stress test, one week of running around 22k VMs adding periodically random stress sources.

4 Performances

The first performance test was done with three users with different initial shares configured in `/etc/fass/shares.conf`. User 4, pictured in green in Fig. 2 top, had 50% of the share, while user 5 (orange) and user 6 (blue) had both 25% of the share. At the beginning of the simulation, all three users instantiated periodically VMs and, as it can be observed in the blue box in Fig. 2 top, user 4 had about double VMs running, with respect to the other two. Around the green arrow, user 6 stopped instantiating machines, and rapidly FaSS readjusted the running VMs in such a way to re-balance the resources usage among users 4 and 5. After about 10 minutes (indicated by the purple line in Fig. 2 top) all three users started to instantiate again and the equilibrium was rapidly reached where the red arrow is placed. This can be deducted looking at the fact that the VMs reflected again the initial shares configured, 50% for user 4 (green) and 25% for the other two.

The second simulation was a stress test shown in Fig. 2 bottom. FaSS was run for approximately 1 week and handled around 22k VMs. The performance was stable and balanced among the three users. At some points, indicated by the white vertical lines, stress sources were added in the form of a random number of VMs (among 0 and 300) instantiated by one of the three users, also selected randomly. What was observed was that during stress FaSS correctly favoured the user with the higher share, user 4 in green, while after the stress source was exhausted it went back to equilibrium.

5 Outlook

The high-level architecture of FaSS is not intrusive in the ONE code, and completely integrated with ONE 5.4 starting from v1.2. Optimally, we will gradually equip FaSS with several different algorithms for fair share or different advanced scheduling policies. FaSS can also be used to keep the Cloud infrastructure clean and its performances are promising

for usage in Cloud infrastructures for scientific computing. Therefore, we are planning to integrate it to the production infrastructure at the INFN-Torino computing centre.

References

[1] R.S. Montero, R. Moreno-Vozmediano, I.M. Llorente, Computer **45**, 65 (2012)
[2] INFN-Torino - INDIGO-DATACLOUD, *FairShare Scheduler for OpenNebula*, https://github.com/indigo-dc/one-fass (2018)
[3] D. Salomoni, I. Campos, L. Gaido, J.M. de Lucas, P. Solagna, J. Gomes, L. Matyska, P. Fuhrman, M. Hardt, G. Donvito et al., Journal of Grid Computing **16**, 381 (2018)
[4] S. Vallero for INFN-Torino, *Fair share scheduler for opennebula*, https://vimeo.com/216656868 (2017)
[5] SchedMD®, *Multifactor Priority Plugin*, https://slurm.schedmd.com/priority_multifactor.html (2013)
[6] InfluxData, *InfluxDB*, https://docs.influxdata.com/influxdb/v1.7/ (2016)
[7] V. Zaccolo for INFN-Torino, *Fass – fair share scheduler for opennebula*, https://opennebula.org/fass-fair-share-scheduler-for-opennebula (2018)