RESOURCE MANAGEMENT IN PRIVATE MULTI-SERVICE CLOUD ENVIRONMENTS

N. Balashov[1,a], N. Kutovskiy[1], N.Tsegelnik[2]

[1] *Meshcheryakov Laboratory of Information Technologies, Joint Institute for Nuclear Research, 6 Jolio-Curie st., Dubna, 141980, Russia*

[2] *Bogoliubov Laboratory of Theoretical Physics, Joint Institute for Nuclear Research, 6 Jolio-Curie st., Dubna, 141980, Russia*

E-mail: [a] balashov@jinr.ru

The JINR cloud infrastructure hosts a number of cloud services to facilitate scientific workflows of individual researchers and research groups. Although batch processing systems are still the major compute power consumers of the cloud, new auxiliary cloud services and tools are being adopted by researchers and are gradually changing the landscape of the cloud environment. While such services, in general, are not so demanding in terms of computational capacity, they can have spikes of demand and can dynamically scale to keep the service availability at a reasonable level. Moreover, these services may need to compete for resources due to the limited capacity of the underlying infrastructure. This paper discusses how resource distribution can be managed in such a dynamic environment with the help of a cloud meta-scheduler.

Keywords: cloud computing, virtualization, distributed computing

1. Introduction

The JINR cloud [1] is built on the OpenNebula platform, which implements the Infrastructure-as-a-Service model, and is used to provide virtual machines on an individual basis to users (who are mainly researchers and engineers from JINR and partner organizations), as well as to host some multi-user systems and provide them as cloud services. Examples of such cloud services include GitLab with its Continuous Integration tooling, the HTCondor batch cluster and the JupyterHub virtual cluster. The services consist of a number of virtual machines playing different roles in these systems, and their structure is shown in Figure 1.

The cloud provides two types of resources: shared resources, which are in common use by all JINR participants, and resources of the so-called Neutrino Platform, which are owned by several neutrino experiments JINR participates in.

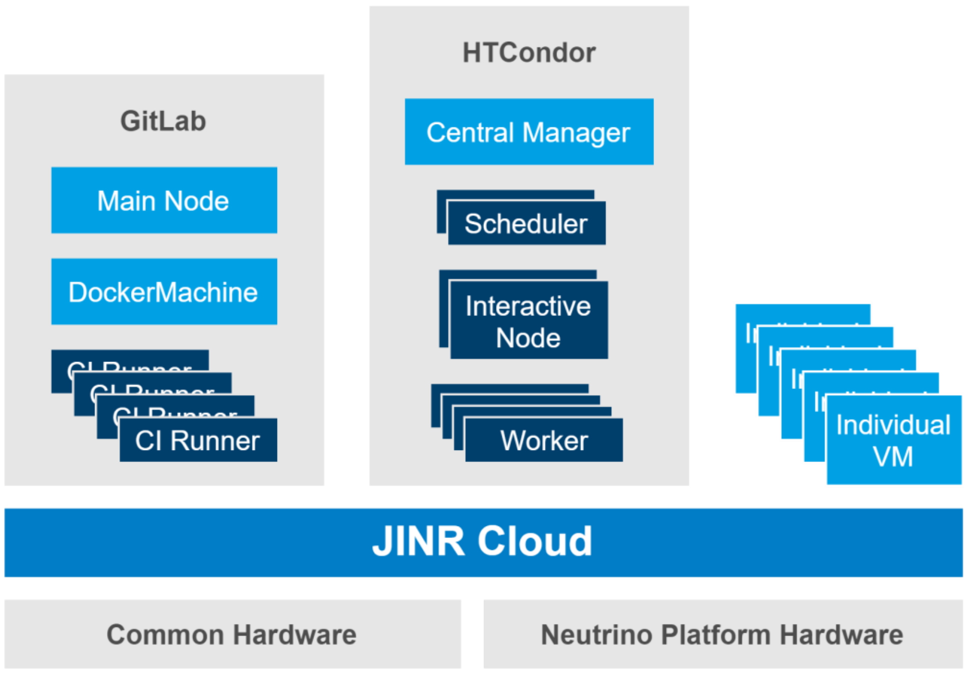

Figure 1. JINR cloud and example of the services structure

The abovementioned cloud services are sometimes underutilized [2] for different reasons, partially due to their different usage models. When services have a fixed amount of resources provided, the underutilization of resources in one service results in the underutilization of the underlying hardware, even though idle resources can be utilized by other services in such cases. In the following sections, we will describe a possible approach to dealing with cloud services resource underutilization using dynamic resource redistribution with the help of the Cloud Meta-scheduler we are developing.

2. Background on resources underutilization

As mentioned in the introductory part, the underutilization of cloud resources can occur for various reasons. For example, interactive services (like JupyterHub or interactive nodes of the HTCondor cluster) are usually underutilized at night or during holidays. Figure 2 illustrates a typical CPU usage profile of an interactive machine. It is clearly seen that this machine was not used at all at night and in the morning, then the load increased after lunch and dropped by the end of the working day. Sometimes people can work at night too.

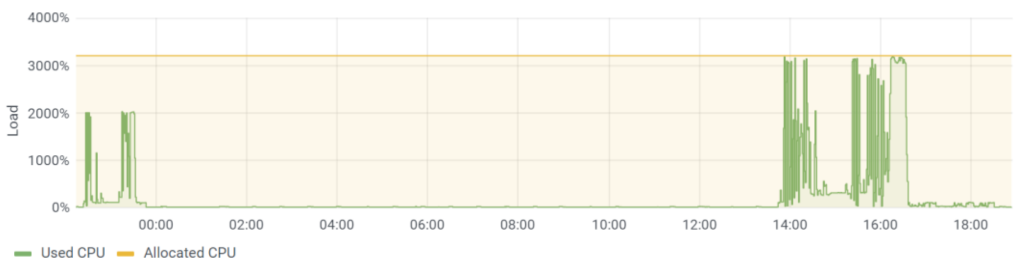

Figure 2. Typical CPU usage profile of an interactive machine

On the contrary, batch clusters are usually better loaded in terms of hardware usage since batch jobs do not need sleep and can run for hours or even days without a stop. However, resources dedicated to individual projects or experiments can also sometimes encounter periods of inactivity (for example, when data production stops during detector maintenance periods), and these periods can be quite long, up to a few weeks (Fig. 3).

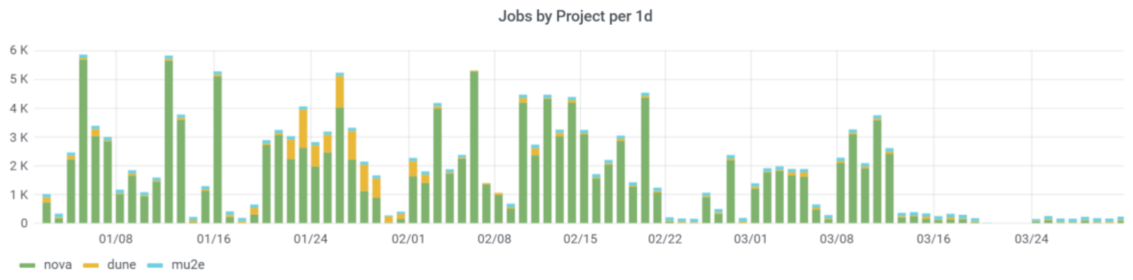

Figure 3. Grid jobs rate on the HTCondor instance of the JINR cloud

Thus, in most cases, underutilization can be considered normal (and expected) because the system efficiency can be defined in different ways depending on the purpose of the system in question. For example, with batch systems we usually want to maximize hardware utilization, while in the case of interactive systems like JupyterHub, we try to keep the system responsive and for this reason it is fine to keep a reasonable amount of resources idle and ready to serve incoming users.

In certain cases, hardware utilization can be easily improved by redistributing resources between different cloud services. For instance, at night, most of the interactive nodes can be stopped in favor of additional batch cluster worker nodes reverting everything back in the morning. The same applies to the owned resources in batch systems: when the experiment knows that its resources will not be used for a long period of time, these resources can be shared with other experiments using the same technique, i.e. scaling in unused resources and scaling out systems that need them. Nevertheless, standard cloud tools do not give us convenient control over the cloud services scaling, taking into account the interests of all services running in the environment, as well as the interests of different working groups that own some fraction of cloud resources. To deal with the issue, we started the Cloud Meta-scheduler project.

3. Cloud Meta-scheduler

The project goal is to provide resource managers and users of JINR’s cloud services with convenient tools for managing and monitoring resource distribution between the services and resource owners, as well as for creating and approving resource lease requests, with a scheduler component in its core, which handles the actual scaling of the services.

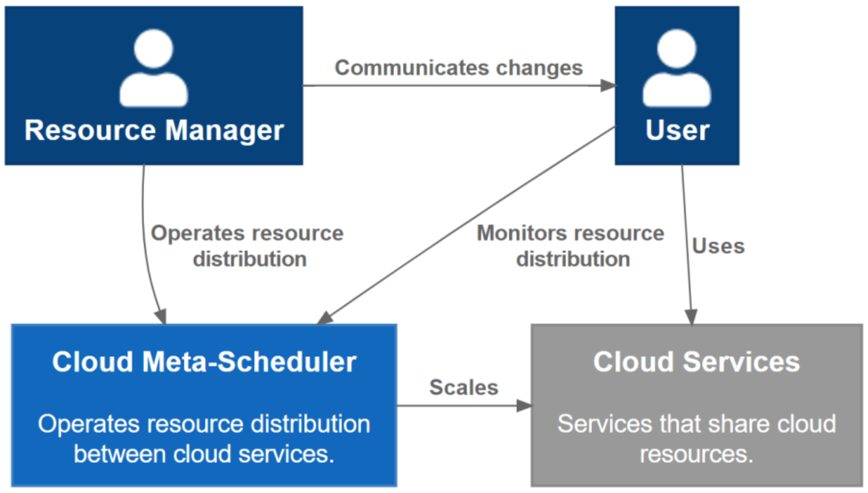
Resource Manager
Communicates changes
User
Operates resource distribution
Monitors resource distribution
Uses
Cloud Meta-Scheduler
Operates resource distribution between cloud services.
Scales
Cloud Services
Services that share cloud resources.

Figure 4. System context diagram of the project

Although the general idea of such a system seems simple, the implementation of the actual scaling of cloud services encounters intricate details, as different services (or even their different parts) can be scaled for different reasons. For example, three different components of the HTCondor cluster may be scaled for the following reasons:

- schedulers (virtual machines that operate the job queue) – to maximize the job submission rate;
- interactive nodes – to keep them responsive;
- worker nodes – to improve the throughput of the cluster.

For this reason, we started the development with the prototype of a meta-scheduler component to study possible technical solutions, to discover some potential pitfalls and better understand the requirements for the system under development.

Python was chosen as the primary development language because of its rapidly growing popularity in data science, which makes it possible to involve data science students in the development of the project with the potential to apply data analytics for incorporating more complex scheduling schemes [2-5]. To implement the microservices approach [6] in the prototype architecture, the Pyro library [7] was used. It wraps Python objects and allows using them in a distributed system as regular Python objects, while Pyro takes care of all the network communication. The main components of the developed prototype (Fig. 5) include:

- Scheduler daemon – runs the scheduling loop;
- HTCondor API microservice – implements communication with the HTCondor cluster;
- Cloud API microservice – implements communication with the JINR cloud.

The microservices approach gives the system additional flexibility that may be needed for large-scale deployments. For instance, the HTCondor API implemented as a service can be run on the same machine as the scheduler and can communicate HTCondor via SSH; however, it can also be run on the HTCondor scheduler machine, directly executing shell commands to talk to HTCondor and then communicate back the information to the scheduler over the network using the specialized Pyro wire protocol.

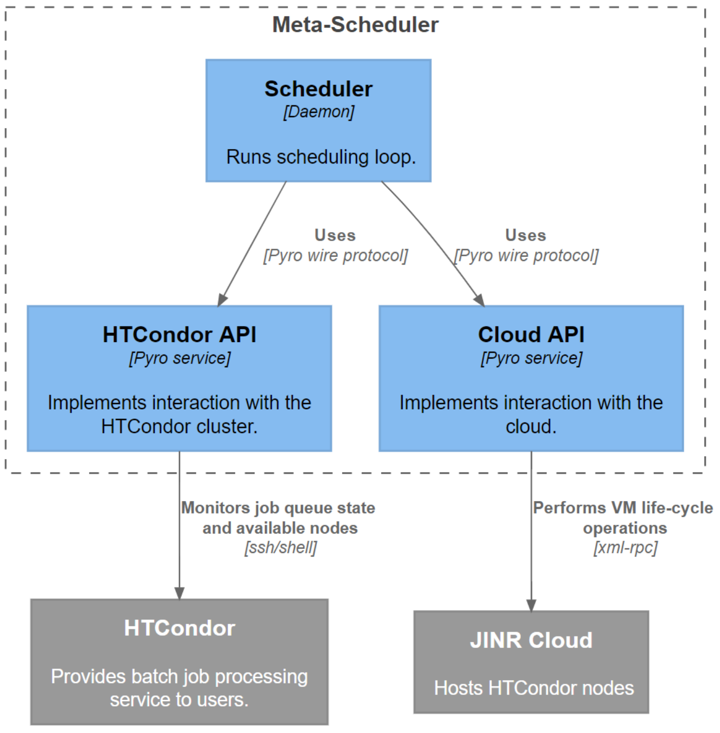
Meta-Scheduler
Scheduler
[Daemon]
Runs scheduling loop.
Uses
[Pyro wire protocol]
Uses
[Pyro wire protocol]
HTCondor API
[Pyro service]
Implements interaction with the HTCondor cluster.
Cloud API
[Pyro service]
Implements interaction with the cloud.
Monitors job queue state and available nodes
[ssh/shell]
Performs VM life-cycle operations
[xml-rpc]
HTCondor
Provides batch job processing service to users.
JINR Cloud
Hosts HTCondor nodes

Figure 5. Meta-scheduler prototype components scheme

The current early version of the prototype implements only the simple automatic scaling of HTCondor worker nodes, depending on the job queue size: when there are idle jobs in the queue, more worker nodes are created (if there are common resources available), and once the jobs are completed, the nodes are deleted.

The further development of the prototype is planned in the following stages:

- Add multi-service support to the scheduler;
- Add multi-role services support;
- Develop a web interface (most likely based on the Django framework [8]) for users and resource managers.

4. Conclusion

The IT industry and data science are rapidly evolving, new technologies are emerging and becoming popular, and the changing IT landscape sets new challenges in computing infrastructures management. To keep up with the evolution of computing models and environments, we need to develop novel tools to help us efficiently handle their growing complexity. In this paper, we have described the idea and development course of one such tool designed for the dynamic load-balancing of multi-service cloud environments to improve computing resources utilization in cloud environments.

References

[1] N. Balashov, A. Baranov, N. Kutovskiy, A, Makhalkin, Ye. Mazhitova, I, Pelevanyuk, R. Semenov Present Status and Main Directions of the JINR Cloud Development //Proc. of 27th International Symposium NEC-2019, Budva, Montenegro. 2019. Vol. 2507. P. 185–189.

[2] M. Armbrust et al. Above the clouds: a Berkeley view of cloud computing //Electrical engineering and computer sciences, Technical Report No. UCB/EECS-2009-28, University of California at Berkeley, February 2009.

[3] Jain N., Raghu B., Khanaa V. Probabilistic Model for Resource Demand Prediction in Cloud //Turkish Journal of Computer and Mathematics Education (TURCOMAT). 2021. – Vol. 12 (6). P. 1766-1771.

[4] Golshani E., Ashtiani M. Proactive auto-scaling for cloud environments using temporal convolutional neural networks //Journal of Parallel and Distributed Computing. 2021. Vol. 154. P. 119-141.

[5] Nwe K. M., Oo M. K., Htay M. M. Efficient resource management for virtual machine allocation in cloud data centers //2018 IEEE 7th Global Conference on Consumer Electronics (GCCE). 2018. P. 419-420.

[6] Larrucea X. et al. Microservices //IEEE Software. 2018. Vol. 35 (3). P. 96-100.

[7] Pyro - Python Remote Objects. Available at: https://pyro5.readthedocs.io (accessed 20.08.2021).

[8] Django: The web framework for perfectionists with deadlines. Available at: https://www.djangoproject.com (accessed 20.08.2021).