EPJ Web of Conferences **214**, 07030 (2019) https://doi.org/10.1051/epjconf/201921407030
CHEP 2018

Managing a heterogeneous scientific computing cluster with cloud-like tools: ideas and experience

Marco Aldinucci[1,4], *Stefano* Bagnasco[2,4,1*], *Matteo* Concas[2,3,4], *Stefano* Lusso[2,4], *Sergio* Rabellino[1,4], *Danilo* Demarchi[2,3], *Sara* Vallero[2,4]

[1]Department of Computer Science, Università di Torino, Italy
[2]Istituto Nazionale di Fisica Nucleare, Torino, Italy
[3]Department of Electronics and Telecommunications, Politecnico di Torino, Italy
[4]Scientific Computing Competence Centre (C3S), Università di Torino, Italy

Abstract. Obtaining CPU cycles on an HPC cluster is nowadays relatively simple and sometimes even cheap for academic institutions. However, in most of the cases providers of HPC services would not allow changes on the configuration, implementation of special features or a lower-level control on the computing infrastructure, for example for testing experimental configurations. The variety of use cases proposed by several departments of the University of Torino, including ones from solid-state chemistry, computational biology, genomics and many others, called for different and sometimes conflicting configurations; furthermore, several R&D activities in the field of scientific computing, with topics ranging from GPU acceleration to Cloud Computing technologies, needed a platform to be carried out on. The Open Computing Cluster for Advanced data Manipulation (OCCAM) is a multi-purpose flexible HPC cluster designed and operated by a collaboration between the University of Torino and the Torino branch of the Istituto Nazionale di Fisica Nucleare. It is aimed at providing a flexible and reconfigurable infrastructure to cater to a wide range of different scientific computing needs, as well as a platform for R&D activities on computational technologies themselves. We describe some of the use cases that prompted the design and construction of the system, its architecture and a first characterisation of its performance by some synthetic benchmark tools and a few realistic use-case tests.

1 Introduction

The Open Computing Cluster for Advanced data Manipulation (OCCAM) is a multi-purpose flexible HPC cluster designed and operated by a collaboration between the University of Torino and the Torino branch of the Istituto Nazionale di Fisica Nucleare through C3S, the University's Scientific Computing Competence Centre [1]. In its current configuration, it includes 32 24-core standard nodes, 4 48-core high-memory 4-way nodes, 4 dual-GPU nodes,

* Corresponding author: stefano.bagnasco@to.infn.it

3 access and management nodes and two storage subsystems for a total of about 1PB of disk space.

At the base of the OCCAM hardware and software architecture was the need to accommodate a wide array of different and sometimes incompatible scientific computing use cases. Some of these issues would have been easily solved by a conventional HPC architecture based on shared software installation and a queue-based batch system, but several others not. For instance, we aimed to support some cases where not only the software to be delivered was not designed to benefit from scalable architectures, but also not intended to be run in a *batch* (non-interactive) way. Interactive sessions and graphical interfaces were sometimes required, nevertheless demanding from a computational perspective. Conventional HPC clusters are queue based and designed for batch-like processing, and the focus is on *maximum computing performance*. Cloud infrastructures are instead aimed at rapid on-demand resource provisioning: leveraging on virtualisation technologies, their strong points are *flexibility* and *elasticity*. To adopt a cloud-like management model on top of an HPC cluster, we focused on running *Scientific Computing Applications* (SCA) rather than jobs.

2 Overall concept and architecture

The core concept of a Scientific Computing Application is defined by its runtime environment (application software, libraries, configurations…), resource requirements (CPU, memory, GPU, low-latency network, storage…) and execution model (batch-like, single image interactive, pipeline…).

At a glance:

- We use Linux containers to package the full application runtime environment and provide a lightweight virtualization layer mostly free from performance penalties, leveraging on the Docker [2] ecosystem to allow users to re-use off-the-shelf images and delegate to their virtual cluster the management of the virtualized resources (e.g. network interfaces), thus decoupling the application support from infrastructure support.
- Resource requests are granted by a reservation-allocation mechanism; in the final architecture, Apache Mesos [3] frameworks will manage resources and manage all inter-application scheduling.
- Relevant management and execution services run as containers on dedicated nodes. Diverse use cases, like batch systems, interactive-session-managed executions, graphical sessions and multi-step analysis pipelines are addressed by deploying compositions of them.

The use of service-oriented Docker instead of more HPC-oriented alternatives like Singularity [4], Shifter [5] or uDocker [6] pose evident security issues, due to the fact that it is not a tool conceived to support multi-user, non-root access use cases directly: enabling users to issue Docker commands is tantamount to give them administrator privileges on hosts. Conversely, however, other tools lack the flexible networking support of Docker, that allows us to build the intercommunicating isolated compositions of containers tailored to a specific SCA.

Security issues are solved by having the system running “vetted” containers in a controlled way instead of having the user run them directly. A set of command line tools on access nodes allow the user to build and submit their containers to a privileged script that runs on worker nodes, to which the user has no direct access. This script, in turn, builds a derived container acting several sanity checks, to cover a number of possible security issues

(for example, allowing the mount of selected filesystems only, vetting /etc/group and /etc/passwd, cleaning up s-bits and so on) and finally runs it on behalf of the user.

In practice, the user workflow is as follows:

- the user can use a Docker-enabled workstation or, alternatively, an OCCAM-specific command to build and test a container image, starting from scratch or adapting any off-the-shelf images available either from Docker Hub, or OCCAM's GitLab Projects shared by other users;
- the image is uploaded to OCCAM's GitLab private image registry (we plan to provide automated build by GitLab CI runners in the future);
- the user can test the image on the architecture on a dedicated test node using the OCCAM command-line tools;
- containers are then run on production nodes for the final computation, again using OCCAM-specific command line tools.

The architecture implements a two-level scheduling mechanism. Bottom-up, we have an in-application scheduling managed by HTCondor [10] as batch system (see next section). The inter-application scheduling is managed by Apache Mesos frameworks. An exception is currently made for single-node applications, i.e. single-node "virtual workstations" provisioned as-a-Service with access to high-memory 4-way nodes or GPUs, and pipeline-like multi-step applications. Indeed, those application are not yet integrated in the Mesos orchestration but enabled using *ad-hoc* tools.

However, the user workflow is defined, and we plan to gradually reach the final configuration without changing any of the user interface semantics.

In the following we will mostly focus, as an example, on the most complete execution model implementation so far: Batch System as-a-Service, a tool for building "traditional" virtual HPC clusters on top of a partition of OCCAM. For a description of the pilot use cases and more examples, as well as for a detailed hardware architecture and benchmarking, please see [7].

3 Batch System as-a-Service

Having users interact with an environment they are familiar with is crucial in the adoption of novel concepts. Batch System as-a-Service allows a user community to use a conventional batch queue interface on top of cloud infrastructures; several tools used in this implementation were developed in the context of the INDIGO-DataCloud [8] and DEEP-HybridDataCloud [9] EU projects.

3.1 A Containerized Batch Farm

HTCondor is a widely used workload management system for compute-intensive jobs providing a job queueing mechanism, scheduling and priority policies, resource management and more. Unlike many other batch systems used in conventional HPC clusters (like for example OpenPBS), being originally designed for opportunistic computing it does not rely on a predefined inventory of resources, being thus well suited for elastic applications in a cloud environment, where computing nodes can be instantiated or terminated depending upon load and available resources.

In our architecture, HTCondor components are separately packaged in Docker containers; in each container, Tini [11] and supervisord [12] are used to start the required services, and

configuration parameters are passed as arguments to the Docker run command and end up as parameters to the container's entrypoint script.

A "master" container runs HTCondor *collector* and *negotiator* daemons, a separate "Submitter" container runs the *scheduler* daemon and any number of "Worker" containers run the *executors* (startd); access to the system is granted through "bastion service" containers running on OCCAM front-end nodes. Each container exposes health status and custom metrics on a dedicated httpd endpoint. The containers are built from ones in which the users package their software, just adding the relevant middleware components and configurations.

"Service containers" such as, in this example, the Master and the Submitter run on a frontend host, whereas Executors run on computational nodes.

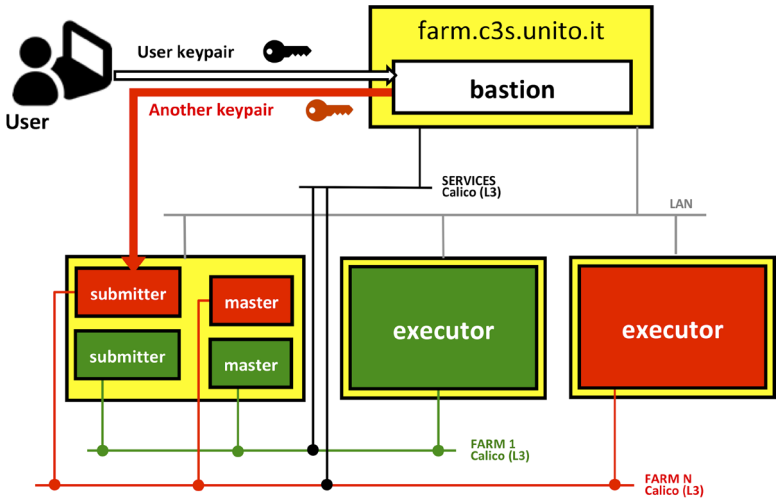

Fig. 1. Architecture of the Batch System as-a-Service. Outer yellow boxes are physical hosts, whereas inner boxes are containers, red and green ones belonging to different virtual clusters.

3.2 Orchestration and Deployment

Starting from such building bricks, we can automatically deploy a scalable virtual batch cluster on the OCCAM infrastructure.

Apache Mesos, an open-source industry-standard resource management system, is used for resource abstraction and management. On top of it, components can either be scheduled as long-running services by Apache Marathon [13] or by HTMFrame [14], a custom Scala implementation of the Mesos Scheduler Driver developed in the context of the INDIGO-DataCloud project.

HTMFrame implements custom policies on Mesos resource offers to instantiate roles in the correct sequence, health-check them and reinstantiate upon failure; it then manages auto-scaling of the cluster according to HTCondor metrics. The *submitter* role publishes the queue and nodes status as reported by HTCondor; in case of waiting jobs, scale-out is implemented by the framework by increasing the number of *executors*. For what concerns scale-in, each *executor* role publishes either a healthy state, in case it is processing jobs, or an unhealthy state, in case it is idle. Idle *executors* are terminated by the framework.

3.3 Networking

Project Calico [15] is used to manage virtual networking using private L3 overlay networks. Each farm is isolated within its own private network and end-users connect to the farm's submitter through a *bastion* service, as depicted in figure 1. ACLs are defined to allow access from the bastion service to the different networks, with any given access container being able to communicate only to the network connecting the services comprising its virtual cluster.
Each farm is mapped to a Linux group through its component's hostnames (i.e. groupname-master, groupname-submitter, groupname-executor), which Mesos DNS [16] resolves in to IPs in the overlay network address range. On the bastion, forwarding rules are in place to re-direct a user belonging to a specific group to the appropriate farm submitter.
In this way we are able to provide an isolated, secure and private infrastructure to each SCA.

4 Performance assessments

We performed a number of tests and measurements based both on synthetic benchmarks and real-world applications, to demonstrate that the additional containerization layer and overlay networking on top of the OCCAM HPC cluster does not affect the performance nor limit the scaling potential of the applications. Moreover, it has been necessary to perform a number of tests also to find the correct combination between the version of the kernel modules for InfiniBand, the Message Passing Interface (MPI) version and the compilers provided, in order to fully exploit the potential of the cluster.

4.1 Networking performance

All OCCAM nodes are connected both by a 10Gb/s Ethernet network and 56Gb/s FDR low-latency InfiniBand network.

Besides the Docker layer, Calico overlay L3 networks introduce some complexity in the Ethernet connection, thus, we expect some performance degradation due to both the virtualization of Ethernet network interfaces from the Docker side and to the non-trivial routing pattern. Indeed, different Docker network stacks are connected across multiple physical nodes using BGP, in order to create a unique L3 network. The InfiniBand network device, on the contrary, is directly exposed to the container.

Tests were made using the Linux utility qperf [17]. Figure 2 shows the results as an average of 10 measurements for each specific test. Errors given by the standard deviation of the measurements are of the order of at most few percent (they are not shown in the figure). Indeed, in the case of Ethernet connections, we observe a bandwidth degradation of about 20% when using Docker containers in the farm set-up and a more dramatic 30% degradation in latency.

On the contrary, in the case of InfiniBand we observe only a negligible deterioration in some latency measurements (see fig. 2) we did not investigate further, possibly due to some slight difference in process running environments.

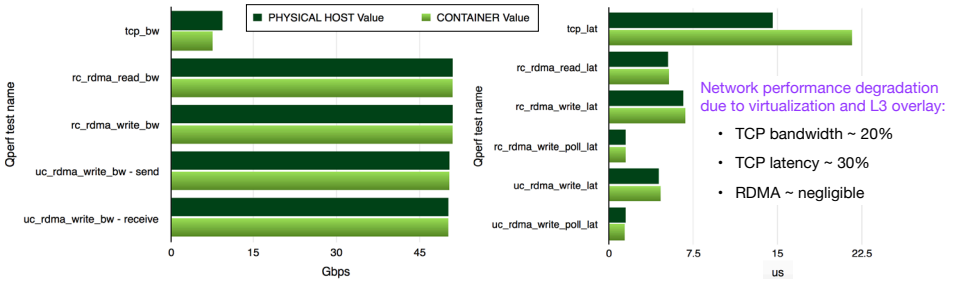

Fig. 2. Results of qperf bandwidth (left) and latency (right) tests.

4.2 Scalability

The overall performance and scalability of the Batch System as-a-Service implementation on OCCAM were tested using CRYSTAL [18], a widely used computational chemistry MPI application. CRYSTAL is currently being used by the largest OCCAM user community and on the largest virtual cluster.

Two different CRYSTAL simulations, compiled with the INTEL-2017 compiler and linked with an OpenMPI version compiled with gcc (a combination recommended by the software authors) were used to compare both the performance of the software in the container and on bare metal and the scalability on OCCAM with that on MARCONI, the CINECA Tier-0 HPC system [19].

The leftmost plot in figure 3 shows the total time taken by a reference CRYSTAL computation performed both on OCCAM physical nodes directly and from within containers running on the same nodes. Each datapoint is the average over 5 measurements, errors are given by the standard deviation and are shown in the plot, but they are not appreciable. The two results are comparable within the errors, the discrepancy between the two being quantified by the results of a power-law fit, also shown in the figure. We conclude that no relevant performance penalty is introduced by using Docker containers.

In the rightmost plot of figure 3, we study the scaling behavior of the reference computation as a function of the number of utilized cores (no containers used). The plot also shows a similar measurement performed on the MARCONI cluster, but in this case a *fast* version of the same simulation was used. Therefore, since we were not interested in comparing the absolute value of the total time taken by the computation on the two clusters, but rather in comparing the scaling behavior, we divided the value of each datapoint of the two sets by the respective measurement at 72 cores. A power-law fit to the MARCONI dataset is also shown as a solid line. We conclude that the scaling behavior is comparable in the two clusters, at least up to the number of cores to which we have datapoints for the OCCAM cluster (the maximum number of cores used in these tests is 96, corresponding to 4 light nodes).

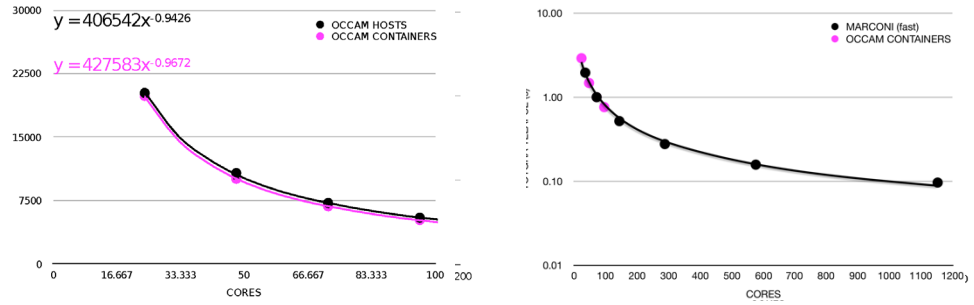

Fig. 3. Comparison of scaling inside and outside the containers (left) and between MARCONI and OCCAM (right), computation elapsed time on the Y axis.

5 Conclusions

The OCCAM cluster has been managed in production for more than one year using an innovative cloud-like management model based on deploying virtual clusters tailored for specific Scientific Computing Applications. Even though not all applications are yet ported to the final design, we already have evidence of the benefits of an architecture based on a two-level scheduling. For all the different use cases, it must be pointed out that a lot of work has been done by the same cluster experts as consultants, in order to fully investigate each use case. The goal is finding the most convenient tradeoff between exploiting the potential the cluster could provide to each workflow and changing as less as possible the way is used to do computations, in order to flatten the learning curve on technical aspects. To overcome the lack of users' confidence with containerization, we successfully organize periodic OCCAM Workshops, where we help researchers in the introduction of this technology in their daily research activities, enabling them to build containers by themselves.
The fundamental concept, inter-application scheduling orchestrated by Apache Mesos, is going to be extended to the entire cluster. This will eventually be true also for those applications that require features not generally allowed on ordinary HPC clusters, such as interactive sessions or graphical interfaces. For use cases that need a batch system, it is available an in-application scheduling via HTCondor, that is able to access dynamically-scaled clusters thanks to the integration with HTMFrame.

The choice of using Docker instead of integrated containerization tools provided by the majority of orchestrators on the market allowed us to quickly address the variety of use cases we had since day zero, while the final design was still being defined. Therefore, we could carry on with the test and development of more general and comprehensive approaches at the same time. Docker integration with Apache Mesos, along with the network management capabilities, was indeed one of the strongest points in the comparison with state of art of other solutions like Singularity and Shifter, at that time. To provide a transparent access to the resources (sessions) to the end user and circumvent potential complications and issues coming from the direct Docker exposition to them, we opted to have a system that runs container in a controlled way, only exposing the access the applications themselves.

Several Computing Applications have been run on the system, with different levels of integration into the architecture, and generally with a reasonable learning curve for the adoption of container and remarkable stability in production mode. In particular, the Batch System as-a-Service virtual clusters were used by several applications and showed that performance and scalability are comparable with those of more conventional clusters,

whereas the Virtual Workstation as-a-Service application proved itself vastly popular with many user communities, whose smaller computing needs and interactive workflow did not justify the effort of running on a traditional batch HPC cluster.

Overall, the first operational experience and performance tests show that the model is viable and, indeed, provides access to resources to communities usually excluded from larger conventional HPC facilities. In the next future, OCCAM computing power will be upgraded and C3S will become a part of HPC4AI [20], a larger distributed scientific High Performance and Cloud computing infrastructure in Torino.

The OCCAM cluster at the Centro di Competenza sul Calcolo Scientifico of the University of Torino was funded through a contribution by the Compagnia di San Paolo, Italy.

Several tools used in this work were developed in the context of the INDIGO-DataCloud project, funded by the EU Horizon 2020 research and innovation programme under grant agreement RIA 653549.

References

1. Competence Centre for Scientific Computing, http://c3s.unito.it/ [accessed 2019-02-27]
2. Docker Company, "Docker" [software], version 17.00-ce, 2018. Available from https://www.docker.com [accessed 2019-02-27]
3. Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz R, … Stoica I (2011). "Mesos: A Platform for Fine-grained Resource Sharing in the Data Center". In Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (pp. 295–308). Berkeley, CA, USA: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=1972457.1972488 [accessed 2019-02-27
4. Kurtzer GM, Sochat V, Bauer MW (2017) Singularity: Scientific containers for mobility of compute. PLoS ONE, 2017 https://doi.org/10.1371/journal.pone.0177459
5. Gerhardt L, Bhimji, W, Canon S, Fasel M, Jacobsen D, Mustafa M, … Tsulaia, V (2017). Shifter: Containers for HPC. Journal of Physics: Conference Series, 898, 082021. https://doi.org/10.1088/1742-6596/898/8/082021
6. Gomes J, Bagnaschi E, Campos I, David M, Alves L, Martins J, Pina J, López-García A, Orviz P, (2018) Enabling rootless Linux Containers in multi-user environments: the udocker tool. Computer Physics Communications, https://doi.org/10.1016/j.cpc.2018.05.021
7. Aldinucci M, Bagnasco S, Lusso S, Pasteris P, Rabellino S, Vallero S (2017) Journal of Physics: Conf. Series **898** 082039
8. D. Salomoni *et al*. (2018) J. Grid Computing **16**, 381
9. DEEP-HybridDataCloud project, "DEEP Genesis" [software], version 1.0.0-1, 2018. Available from https://deep-hybrid-datacloud.eu/ [accessed 2019-02-27]
10. HTCondor project, "HTCondor" [software], Available from https://research.cs.wisc.edu/htcondor/index.html [accessed 2019-02-27]
11. Tini project, "Tini" [software] version 0.18.0, 2018. Available from https://github.com/krallin/tini [accessed 2019-02-27]
12. Supervisord project, "Supervisord" [software] version 3.3.5, 2018. Available from http://supervisord.org/ [accessed 2019-02-27]
13. Mesosphere project, "Marathon" [software], 2018. Available from https://mesosphere.github.io/marathon/ [accessed 2019-02-27]

14. INDIGO-DataCloud project, "HTMFrame" [software], 2018. Available from https://github.com/svallero/HTMframe [accessed 2019-02-27]
15. Project Calico, "Project Calico" [software], version 2.0, 2018. Available from https://www.projectcalico.org/ [accessed 2019-02-27]
16. Mesosphere project, "Mesos-DNS" [software] version 0.1.2, 2018. Available from https://mesosphere.github.io/mesos-dns/ [accessed 2019-02-27]
17. Patchwork project, "qperf" [software] version 0.4.10, 2018. Available from https://github.com/linux-rdma/qperf [accessed 2019-02-27]
18. R. Dovesi *et al*. (2014) Int. J. Quantum Chem. **114**, 1287.
19. MARCONI documents, https://wiki.u-gov.it/confluence/display/SCAIUS/MARCONI+documents [accessed 2019-02-27]
20. Aldinucci, M, Rabellino S, Pironti M, Spiga F, … Galeazzi F (2018) "HPC4AI, an AI-on-demand federated platform endeavour". In ACM Computing Frontiers, Ischia, Italy, 2018. doi:10.1145/3203217.3205340