Seamlessly Managing HPC Workloads Through Kubernetes

Sergio López-Huguet[1(✉)], J. Damià Segrelles[1], Marek Kasztelnik[2], Marian Bubak[2], and Ignacio Blanquer[1]

[1] Instituto de Instrumentación para Imagen Molecular (I3M), Centro mixto CSIC - Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain
serlohu@upv.es, {dquilis,iblanque}@dsic.upv.es

[2] ACC Cyfronet, Sano Centre for Computational Medicine, AGH University of Science and Technology, Nawojki 11, 30-950 Kraków, Poland
ymkaszte@cyfronet.pl, bubak@agh.edu.pl

Abstract. This paper describes an approach to integrate the jobs management of High Performance Computing (HPC) infrastructures in cloud architectures by managing HPC workloads seamlessly from the cloud job scheduler. The paper presents *hpc-connector*, an open source tool that is designed for managing the full life cycle of jobs in the HPC infrastructure from the cloud job scheduler interacting with the workload manager of the HPC system. The key point is that, thanks to running *hpc-connector* in the cloud infrastructure, it is possible to reflect in the cloud infrastructure, the execution of a job running in the HPC infrastructure managed by *hpc-connector*. If the user cancels the cloud-job, as *hpc-connector* catches Operating System (OS) signals (for example, SIGINT), it will cancel the job in the HPC infrastructure too. Furthermore, it can retrieve logs if requested. Therefore, by using *hpc-connector*, the cloud job scheduler can manage the jobs in the HPC infrastructure without requiring any special privilege, as it does not need changes on the Job scheduler. Finally, we perform an experiment training a neural network for automated segmentation of Neuroblastoma tumours in the Prometheus supercomputer using *hpc-connector* as a batch job from a Kubernetes infrastructure.

The work presented in this article has been partially funded by the regional government of the Comunitat Valenciana (Spain), co-funded by the European Union ERDF funds (European Regional Development Fund) of the Comunitat Valenciana 2014–2020, with reference IDIFEDER/2018/032 (High-Performance Algorithms for the Modeling, Simulation and early Detection of diseases in Personalized Medicine). The work is also co-funded by PRIMAGE (PRedictive In-silico Multiscale Analytics to support cancer personalised diaGnosis and prognosis, empowered by imaging biomarkers) a Horizon 2020 RIA project funded under the topic SC1-DTH-07-2018 by the European Commission, with grant agreement no: 826494.

H. Jagode et al. (Eds.): ISC High Performance 2020 Workshops, LNCS 12321, pp. 310–320, 2020.
https://doi.org/10.1007/978-3-030-59851-8_20

Keywords: Integrating cloud and HPC · Kubernetes · Docker and Singularity containers

1 Introduction

Most scientific workloads combine requirements that could be efficiently addressed using a combination of *High-Throughput Computing* (HTC) and *High-Performance Computing* (HPC) workloads [7,8,20]. Focusing on Medical Imaging, HPC is extensively used for artificial intelligence model building and simulation. HTC is widely used in image post-processing and applying trained models to new datasets. HTC workloads can be efficiently tackled on cloud computing infrastructures, which fit to massive, coarsely coupled and embarrassingly parallel jobs. HPC workloads typically require infrastructures composed of a large number of highly-coupled computing nodes. HPC infrastructures are typically provided by singular data centers through specific interfaces.

Cloud computing platforms provide access to a large variety of computing resources on demand and without needing on-premise resources. Therefore, cloud services assist on reducing the cost contention and the ecological impact thanks to the self-adaptive mechanisms that dynamically adjust the cloud infrastructure depending on different aspects. Furthermore, it is possible to build hybrid cloud platforms depending on the institution necessities. Cloud infrastructures are much more flexible than HPC systems. Contrary to Cloud infrastructures, which can be adapted to the applications requirements, in HPC systems applications must be adapted to the execution environment. One important aspect for final users relies on the job management.

On the other hand, an *HPC* cluster delivers a huge amount of specialized and already configured computation resources to the researchers. Clusters can run free and commercial set of toolboxes which are already prepared to be used efficiently in distributed environments. As a result, running an application in this scenario can be easier than in a pure cloud model for the researcher (who wants to perform calculations and does not want to focus on the hardware and software installation and fine tuning).

The institutions can take benefit of employing a hybrid processing platform composed of HPC and cloud infrastructures. The architecture platform can be complex because there are a lot of aspects to consider: authentication, authorization, data storage, software requirements, special hardware, etc. Furthermore, the majority of the institutions that use HPC infrastructures, use infrastructures that are provided by third parties. Therefore, they must adapt their other processing infrastructures (for example, a public or private cloud platform) to use the different HPC infrastructures. This work presents *hpc-connector*[1], an open source tool that allows to seamlessly integrate the cloud architecture with the access to the HPC cluster, without administrator privileges.

[1] https://gitlab.com/primageproject/hpc-connector.

2 Scenario and Related Work

2.1 Architecture

Cloud application architectures typically comprise front-end services and back-end nodes. Front-end services provide external access and manage back-end resources through a job scheduler API or a graphical interface. In some cases, front-end nodes use resource manager tools in order to scale in or out the resources, depending on usage metrics to provide an agreed Quality of Service. The back-end nodes are a set of heterogeneous resources that run the jobs sent by the users through the job scheduler.

There are examples of cloud platforms in the literature that use the previous architecture scheme. In [14], the authors present an architecture to process Internet of Things (IoT) data collected from smart agriculture. In our previous works, we presented a cloud architecture for data analysis [19] and for processing medical imaging [18].

However, there are no examples of hybrid cloud and HPC infrastructures that could provide a seamless interface for both types of workloads. The PRIMAGE project [21] is an ongoing research project that uses artificial intelligence techniques for the processing of medical imaging in paediatric cancer. In this project, the platform architecture combines an HPC infrastructure (the Prometheus supercomputer [10]) and an on-premise cloud platform. The tool presented in this work was designed to solve the problem of combining the execution of some applications in several infrastructures with no administrator privileges.

Job schedulers (or workload managers) manage the remote execution of the applications on the available resources. The most popular job schedulers designed for containers technologies are Docker Swarm, Kubernetes[2], some frameworks of Apache Mesos[3] and Nomad[4]. Regarding workload managers in the HPC environment, there are a lot that are widely used: SLURM [22], Torque[5] or HTCondor[6]. It should be pointed out that *hpc-connector* was designed to integrate any cloud architecture (provided with any job scheduler) with an HPC infrastructure (also provided with any workload manager). In this work, we will use Kubernetes as the cloud job scheduler, and SLURM for the HPC infrastructure.

Application portability and delivery are key issues not only in cloud computing environments but also in HPC. Containers probably are now the most popular technology for application delivery, thanks to the reproducibility, traceability, provenance, isolation, and portability. Docker[7] has reached the maximum popularity as container technology on cloud infrastructures, thanks to its rich ecosystem of tools and great versatility. However, Docker containers run under the root user space and do not provide easily multi-tenancy (it is necessary to

[2] https://kubernetes.io.
[3] http://mesos.apache.org/.
[4] https://www.nomadproject.io/.
[5] https://adaptivecomputing.com/cherry-services/torque-resource-manager/.
[6] https://research.cs.wisc.edu/htcondor/.
[7] https://www.docker.com/.

create previously the users during the container building stage). Singularity containers [17] are widely used in the HPC environments because they run under user space, support multi-tenancy and provide mechanisms to use Message Passing Interface (MPI). There are other container technologies and runtimes, such as Podman[8], Charliecloud[9], Shifter[10], etc., but we will use Docker and Singularity as container technologies for cloud and HPC infrastructures, respectively. It should be noted that the common used HPC workload managers (such as SLURM) can run unprivileged containers without requiring changes or installing a plugin (as containers are processes that are executed in the user space).

2.2 Objectives and Requirements

The goal of this work is to provide a tool that permits the combination of a job scheduler that runs applications embedded in Docker containers (for example, Kubernetes) on the cloud environment, with the HPC workload managers that run applications bare metal or in unprivileged containers. Considering the scenario described in the previous section, the identified requirements and assumptions needed to fulfill are:

(R1) Users must use the same method to submit or cancel jobs to the HPC workload manager, as the job scheduler on the cloud environment does.
(R2) The job scheduler used on the cloud environment must manage the full life cycle of a job in the HPC infrastructure. For example, the job scheduler should be able to submit, cancel, get execution state, get the logs, etc.
(R3) The data required to run the job must be accessible in the HPC infrastructure. There are several options: users upload it in advance, the job downloads it before starting, or there is a shared storage between the HPC infrastructure and any other environments that could use the data.
(R4) Any user authorised to access the cloud and the HPC resources must be able to execute jobs without requiring special privileges in neither the HPC nor the cloud infrastructures.
(R5) The solution should be extensible to deal with different job schedulers and workload managers.

2.3 State of the Art

The combination of the potential of High Performance Computing for simulation and Big Data and cloud computing for massive data processing has become a driving forces for complex disciplines such as Brain science [9]. The relevance of High Performance and Cloud Computing for addressing challenges related to medical imaging has boosted with the take off of the application of Artificial Intelligence [4,5]. A revision study [11] highlights 83 articles applying some kind

[8] https://podman.io/.
[9] https://hpc.github.io/charliecloud/.
[10] https://www.nersc.gov/research-and-development/user-defined-images/.

of HPC techniques in Medical Imaging [2], many of them also suitable of being addressed using cloud computing.

Although there are authors that propose hardware specific configurations based on FPGAs and GPUs [12,16], current tendencies propose the use of cloud computing platforms, especially public offerings [13] with special examples on solutions provided directly by main industry players in cloud [1]. However, in most of the cases, the use of clouds is limited to the storage and access of medical imaging data with low processing capabilities [2,15]. Recently, solutions proposing the combination of container-based platform with computing accelerators have arisen [3].

3 The Proposed Solution: *HPC-Connector*

Institutions that manage HPC and cloud environments can integrate job schedulers by developing the appropriate plug-ins and extending the current job types to make them compatible between both environments. Large HPC consortia and institutions may follow this approach. However, in most cases users face a situation in which they can acquire both types of resources from different providers.

Another approach is to adapt the job scheduler on the cloud platform to be able to communicate with the workload manager, as cloud infrastructures are widely accessible and much more flexible than an HPC infrastructure for a regular user. This option could be complex and cumbersome, as it would require continuous work as new updates to the job scheduler arise. If the job scheduler is released under open-source licenses, the institution must extend it with the desired functionality following the rules from the project developers. It should be pointed out that, if the institution wants to use more than one HPC infrastructure with different configurations (for example, in one case you only can interact using a REST API but, in the other case, you can only interact using ssh commands), the software extension could be even more complex.

Adapting the job scheduler to the workload managers could be complex and, besides, it forces to keep using the adapted job scheduler in the future for making the adaption effort profitable. For this reason, we propose a solution by creating an external tool (*hpc-connector*) that manages the jobs in the HPC infrastructure from the cloud infrastructure, as any other job without special privileges.

The key point of the proposed solution is the following: once a user submits a job to the job scheduler that wants to be executed in the HPC infrastructure (fulfilling R1), a special job is executed in the cloud infrastructure. The special mirror job is a an instance of *hpc-connector*, which will manage the job in the HPC infrastructure (R2). Thus, the mirror job updates the job scheduler as the execution of the HPC job progresses in the HPC infrastructure. As *hpc-connector* does not need any special privilege (like mounting a directory or accessing special kernel directives), the special mirror job can run in the user space (fulfilling R4). After submitting the job, *hpc-connector* will monitor it (R2) until the end or its cancellation by the HPC workload manager. Once the job ends, *hpc-connector* can retrieve, if the HPC infrastructure allows it, the job output (R2). Furthermore, *hpc-connector* is able to catch the SIGINT signal that the job scheduler

can send to the job before killing it. Therefore, if an appropriate cloud job configuration is performed (the cloud job receives a SIGINT signal and it has a grace period before killing it after the signal is received) and the job still running, *hpc-connector* will cancel the job in the HPC infrastructure (R2).

The tool presented is designed to be running in any environment (even in a local machine) because it is implemented in Python, so it can be running embedded in a any type of container or bare metal. Regarding the support of HPC infrastructures, as each HPC backend has its own methods for managing the jobs and the data, it is necessary to implement some specific functionality for each backend in *hpc-connector* to interect with the job submission, information retrieval about the job execution, canceling, or deleting jobs. Furthermore, if the institution wants to retrieve the logs, it is required to implement how to operate with files (like upload, download, or remove) and operations with directories (list, create, or delete). The tool uses the super-class `Backend` so, for each new HPC infrastructure, a new subclass from the `Backend` class must be created with the name of the HPC infrastructure. For example, let's consider two different HPC infrastructures: *cluster1* and *cluster2*. The infrastructure *cluster1* uses SLURM as workload manager with a REST API. On the other hand, *cluster2* also uses SLURM but only provides a REST API to manage the files, so the users must interact with SLURM via ssh. Thus, *cluster2* implementation is different from *cluster1* because, although both use SLURM, the job operations must be performed using ssh for *cluster2*. Therefore, `cluster1` and `cluster2` sub classes from `Backend` class must be implemented. Thus, *hpc-connector* fulfils R5 because it is designed to be running in any environment and it can be extended for new HPC infrastructures.

4 Use Case: Segmentation of Neuroblastoma Tumours

To validate the usefulness of the *hpc-connector*, we performed a test case. The scenario uses a private cloud platform and the Prometheus HPC infrastructure. The cluster deployed on the private cloud infrastructure is composed of 3 virtual nodes with 4 vCPUs, 32 GB of memory RAM, 80 GB of Solid State Disk and 1 NVIDIA Tesla V100 each. The job scheduler used is Kubernetes (version v1.15.9) and the container technology is Docker (version 18.06.2-ce).

The HPC infrastructure is the Prometheus supercomputer [10], which is located in the 289^{th} position of the TOP500 list (June 2020). Prometheus cluster provides REST API to interact with SLURM (version 19.05.5) called Rimrock (Robust Remote Process Controller)[11] and PLGData service[12] to interact with the file system.

The selected use case is a training of a neural network using Tensorflow for performing an automated segmentation of neuroblastoma tumours. Neuroblastoma is the most frequent solid cancer of the early childhood [6]. This use case belongs to the PRIMAGE project [21].

[11] https://submit.plgrid.pl/.
[12] https://data.plgrid.pl/?locale=en.

First, we define two ConfigMap objects, which store non-confidential data in key-value pairs). The Fig. 1 shows the definition (in YAML[13] format) of the ConfigMap that contains the information required by *hpc-connector* to use the backend Prometheus. This ConfigMap will be used for all jobs that want to connect with Prometheus. If we were using another HPC infrastructure, the configuration value would maybe contain other dictionary keys.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: hpc-prometheus
  namespace: serlohu-at-upv-dot-es
data:
  name: Prometheus
  configuration: |
    {
      "ENDPOINT": "https://submit.plgrid.pl",
      "PROXY": "XXXXXX"
    }
```

Fig. 1. ConfigMap definition to specify the Prometheus configuration required by *hpc-connector*.

Once the ConfigMap for accessing properly to the HPC infrastructure is defined, the users must define the job configuration. As we are using Rimrock service from Prometheus cluster, the required parameters are at least, the host and the SLURM script in plain text. In this script, we specify the amount of resources, the special hardware (GPUs), and the batch queue (plgrid-gpu). Then, we implement the tasks: show the hostname, load modules and run the Singularity image (available at the directory `$SCRATCH/singularity/neuroblastoma.sandbox`).

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: hpc-job-gibi230-segmentation
  namespace: serlohu-at-upv-dot-es
data:
  job: |
    {
      "host": "prometheus.cyfronet.pl",
      "script": "#!/bin/bash \n#SBATCH -J gibi230 \n#SBATCH -N 1 \n#SBATCH --ntasks-per-node=24
      \n#SBATCH --time=72:00:00 \n#SBATCH --mem=40gb \n#SBATCH -A primage1gpu \n#SBATCH -p plgrid-gpu
      \n#SBATCH --gres=gpu \n#SBATCH --output=\"gibi230_segmentation.out\" \n#SBATCH
      --error=\"gibi230_segmentation.err\" \ncd $SLURM_SUBMIT_DIR \nsrun /bin/hostname \nmodule load
      plgrid/tools/singularity \nsingularity run --bind $SCRATCH/gibi230_segmentation/Input:/training
      --bind $SCRATCH/gibi230_segmentation/Output:/output --bind $SCRATCH/gibi230_segmentation/
      user_application/:/user_application $SCRATCH/singularity/neuroblastoma.sandbox python /
      user_application/batch.py /training /output"
    }
  monitoring_period: "1800"
```

Fig. 2. Job definition to specify the job configuration required by *hpc-connector* to launch the job using, in this HPC backend, Rimrock.

[13] https://yaml.org/.

Figure 2 shows the ConfigMap definition for the job configuration. This new ConfigMap is specific for each job in Prometheus cluster.

Figure 3 shows the Kubernetes batch job definition. As it is described in Sect. 3, the job executed in the cloud infrastructure consists of running *hpc-connector* to managing the job in Prometheus cluster. It should be noted that his job is configured with a termination grace period and, if the users cancel this Kubernetes job, it has 30 s to execute the command `kill -SIGINT 1`. If this occurs, *hpc-connector* will catch the signal and immediately cancel the job in the HPC infrastructure. Once the job is submitted to Kubernetes, it is possible to check (in real time) the progress of the execution consulting the logs of *hpc-connector*. Figure 4 is a screenshot of the Kubernetes dashboard showing the logs of the created job.

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: hpc-job-gibi230-segmentation
  namespace: serlohu-at-upv-dot-es
spec:
  template:
    spec:
      restartPolicy: "Never"
      terminationGracePeriodSeconds: 30
      containers:
      - name: hpc-job-gibi230-segmentation
        image: registry.gitlab.com/primageproject/hpc-connector:0.0.1
        imagePullPolicy: Always
        lifecycle:
          preStop:
            exec:
              command: ["kill","-SIGINT","1"] # send SIGINT to hpc-connector in order to cancell the job
        args: [ "--backend", "$(BACKEND_NAME)", "--backend-conf", "$(BACKEND_CONF)", "simulate",
        "--job-config", "$(JOB_CONF)", "--monitoring-period", "$(MONITORING_PERIOD)", "--print-logs"]
        env:
          - name: BACKEND_NAME
            valueFrom:
              configMapKeyRef:
                name: hpc-prometheus
                key: name
          - name: BACKEND_CONF
            valueFrom:
              configMapKeyRef:
                name: hpc-prometheus
                key: configuration
          - name: JOB_CONF
            valueFrom:
              configMapKeyRef:
                name: hpc-job-gibi230-segmentation
                key: job
          - name: MONITORING_PERIOD
            valueFrom:
              configMapKeyRef:
                name: hpc-job-gibi230-segmentation
                key: monitoring_period
```

Fig. 3. *hpc-connector* job definition.

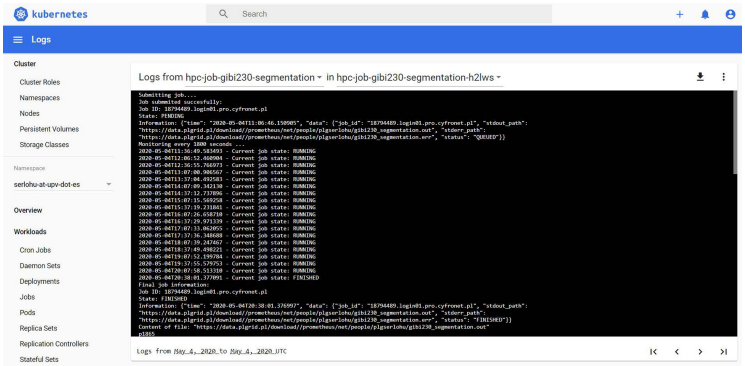

Fig. 4. Consulting the logs of *hpc-connector* using the Kubernetes dashboard.

5 Conclusions

This paper has presented *hpc-connector*, which is an open source tool for seamlessly integrating HPC workloads in cloud infrastructures without requiring administrator privileges or changes on the workload manager, providing the users with the same user interface even across different HPC infrastructures. The tool implemented fulfils the requirements identified in Sect. 2.2: running in the user space, and agnosticism of the workload manager (it is implemented as a Python tool easy extendable to other HPC infrastructures) to manage the jobs in an HPC infrastructure.

The experiment performed in Sect. 4 demonstrated that this approach can address a wider range of complex problems in a convenient way. In this experiment, we successfully trained a neural network using GPUs in an HPC supercomputer (Prometheus) using *hpc-connector* from a Kubernetes job hosted in a private cloud infrastructure.

Future work includes improving the functionality of *hpc-connector*: upload the data (from a repository or from a directory in the Docker container) before submitting the job (if necessary) or consider retrieving the results and storing them in an external repository or in the Docker container itself (for example, if the container has mounted a shared filesystem). Another possible enhancement could be the ability to refresh the HPC credentials when they expire.

References

1. Azure for health. https://azure.microsoft.com/en-us/industries/healthcare/#security. Accessed 07 May 2020
2. Cloud access to mammograms enables earlier breast cancer detection. https://www.itnonline.com/content/cloud-access-mammograms-enables-earlier-breast-cancer-detection. Accessed 07 May 2020

3. Getting to the heart of the HPC and AI the edge in healthcare. https://www.nextplatform.com/2018/03/28/getting-to-the-heart-of-hpc-and-ai-at-the-edge-in-healthcare/. Accessed 07 May 2020
4. High Performance Computing and deep learning in medicine: Enhancing physicians, helping patients. https://ec.europa.eu/digital-single-market/en/news/high-performance-computing-and-deep-learning-medicine-enhancing-physicians-helping-patients. Accessed 07 May 2020
5. Medical Imaging Gets an AI Boost. https://www.hpcwire.com/2019/12/03/medical-imaging-gets-an-ai-boost/. Accessed 07 May 2020
6. Bhatnagar, S.: An audit of malignant solid tumors in infants and neonates. J. Neonatal Surg. **1**, 5 (2012)
7. Cabellos, L., Campos, I., Fernández-Del-Castillo, E., Owsiak, M., Palak, B., Płóciennik, M.: Scientific workflow orchestration interoperating HTC and HPC resources. Comput. Phys. Commun. (2011). https://doi.org/10.1016/j.cpc.2010.12.020
8. Callaghan, S., Maechling, P., Small, P., Milner, K., Juve, G., et al.: Metrics for heterogeneous scientific workflows: a case study of an earthquake science application. Int. J. High Perform. Comput. Appl. (2011). https://doi.org/10.1177/1094342011414743
9. Chen, S., He, Z., Han, X., He, X., et al.: How big data and high-performance computing drive brain science (2019). https://doi.org/10.1016/j.gpb.2019.09.003
10. Cyfronet Krakow, P.: Prometheus supercomputer. www.cyfronet.krakow.pl/computers/15226,artykul,prometheus.html. Accessed 07 May 2020
11. Gulo, C.A.S.J., Sementille, A.C., Tavares, J.M.R.S.: Techniques of medical image processing and analysis accelerated by high-performance computing: a systematic literature review. J. Real-Time Image Process. **16**(6), 1891–1908 (2017). https://doi.org/10.1007/s11554-017-0734-z
12. Hussain, T., Haider, A., Shafique, M., Taleb Ahmed, A.: A high-performance system architecture for medical imaging (2019). https://doi.org/10.5772/intechopen.83581
13. Ivanova, D., Borovska, P., Zahov, S.: Development of PaaS using AWS and Terraform for medical imaging analytics. In: AIP Conference Proceedings (2018). https://doi.org/10.1063/1.5082133
14. Jamalian, S., Rajaei, H.: Data-intensive HPC tasks scheduling with SDN to enable HPC-as-a-service. In: Proceedings - 2015 IEEE 8th International Conference on Cloud Computing, CLOUD 2015, pp. 596–603. Institute of Electrical and Electronics Engineers Inc., August 2015. https://doi.org/10.1109/CLOUD.2015.85
15. Kao, H.Y., et al.: Cloud-based service information system for evaluating quality of life after breast cancer surgery. PLoS ONE (2015). https://doi.org/10.1371/journal.pone.0139252
16. Kovacs, L., Kovacs, R., Hajdu, A.: High performance computing in medical image analysis HuSSaR, June 2018. http://arxiv.org/abs/1806.06171
17. Kurtzer, G.M., Sochat, V., Bauer, M.W.: Singularity: scientific containers for mobility of compute. PLOS ONE **12**(5), 1–20 (2017). https://doi.org/10.1371/journal.pone.0177459
18. López-Huguet, S., García-Castro, F., Alberich-Bayarri, A., Blanquer, I.: A cloud architecture for the execution of medical imaging biomarkers. In: Rodrigues, J., et al. (eds.) ICCS 2019. LNCS, vol. 11538, pp. 130–144. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22744-9_10

19. López-Huguet, S., et al.: A self-managed Mesos cluster for data analytics with QoS guarantees. Future Gener. Comput. Syst., 449–461. https://doi.org/10.1016/j.future.2019.02.047
20. Manuali, C., et al.: Efficient workload distribution bridging HTC and HPC in scientific computing. In: Murgante, B., et al. (eds.) ICCSA 2012. LNCS, vol. 7333, pp. 345–357. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31125-3_27
21. Martí-Bonmatí, L., et al.: PRIMAGE project: predictive *in silico* multiscale analytics to support childhood cancer personalised evaluation empowered by imaging biomarkers. Eur. Radiol. Exp. **4**(1), 1–11 (2020). https://doi.org/10.1186/s41747-020-00150-9
22. Yoo, A.B., Jette, M.A., Grondona, M.: SLURM: simple linux utility for resource management. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS, vol. 2862, pp. 44–60. Springer, Heidelberg (2003). https://doi.org/10.1007/10968987_3