Running HTC and HPC applications opportunistically across private, academic and public clouds

Andrew Lahiff[1,*], *Shaun* de Witt[1], *Miguel* Caballer[2], *Giuseppe* La Rocca[3], *Stanislas* Pamela[1] and *David* Coster[4]

[1]UK Atomic Energy Authority, Culham Science Centre, Oxfordshire, OX14 3DB, United Kingdom
[2]Instituto de Instrumentación para Imagen Molecular (I3M), Centro mixto CSIC – Universitat Politècnica de València, Camino de Vera s/n, 46022, Valencia, Spain
[3]The EGI Foundation, Science Park 140, 1098 XG Amsterdam, The Netherlands
[4]Max-Planck-Institut für Plasmaphysik, Garching bei München, Boltzmannstr. 2, D-85748, Germany

Abstract. The Fusion Science Demonstrator in the European Open Science Cloud for Research Pilot Project aimed to demonstrate that the fusion community can make use of distributed cloud resources. We developed a platform, Prominence, which enables users to transparently exploit idle cloud resources for running scientific workloads. In addition to standard HTC jobs, HPC jobs such as multi-node MPI are supported. All jobs are run in containers to ensure they will reliably run anywhere and are reproduceable. Cloud infrastructure is invisible to users, as all provisioning, including extensive failure handling, is completely automated. On-premises cloud resources can be utilised and at times of peak demand burst onto external clouds. In addition to the traditional "cloud-bursting" onto a single cloud, Prominence allows for bursting across many clouds in a hierarchical manner. Job requirements are taken into account, so jobs with special requirements, e.g. high memory or access to GPUs, are sent only to appropriate clouds. Here we describe Prominence, its architecture, the challenges of using many clouds opportunistically and report on our experiences with several fusion use cases.

1 Introduction

Access to both High Throughput Computing (HTC) and High Performance Computing (HPC) facilities is vitally important to the fusion community, not only for plasma modelling but also for advanced engineering and design, materials research, rendering, uncertainty quantification and advanced data analytics for engineering operations. The computing requirements are expected to increase as the community prepares for ITER, the next generation facility and largest fusion experiment ever built.

[*] Corresponding author: andrew.lahiff@ukaea.uk

The EUROfusion [1] research community today consists of a series of isolated "islands" of computing and storage resources. Users typically access remote resources manually using ssh. Moving to a decentralised computing model is vital for future ITER analysis where no single site will have sufficient resources to run all necessary workflows. In addition, interest in techniques such as uncertainty quantification is gaining momentum and will also contribute to the increasing computing requirements. Getting access to resources is becoming more difficult and it needs to be possible to easily exploit external resources. While there are some facilities, for example MARCONI, providing HPC resources to the fusion community, they are under high demand. Typically access to such resources is competitive and the peer review process means that there can be a long delay between having a need for computing resources and actually being able to have access to them.

The use of clouds for running scientific workloads is well established, in particular for HTC applications. Many platforms have been developed which create either clusters in a cloud or enable a local batch system to burst onto a cloud. However, it has generally been assumed that only one or a few large clouds will be used. In EOSCpilot [2] we had purely opportunistic access to around ten different clouds from EGI FedCloud [3]. We could occasionally get small numbers of cores on some clouds, while on others we could sometimes get larger numbers of cores. There was no way of knowing in advance what resources would be available to us on each cloud. The aim therefore was to aggregate all these resources and make them more useful, benefitting both users who need resources and cloud service providers who would like their resources to be as fully utilised as possible. We developed a new platform, Prominence [4], which, in addition to providing the capability of bursting out from local resources onto external clouds, allows users to transparently make use of potentially many clouds opportunistically.

In this paper we will describe Prominence and look at some fusion use cases and the challenges encountered.

2 Description of Prominence

The basic requirements for a platform enabling users to run batch jobs across many different resources include the ability for jobs to access both software and data. In this section we set out our solutions to these problems in addition to describing the architecture of Prominence and how users interact with the service.

2.1 Application portability and reproducibility

In the fusion energy research community, where computing takes place on traditional batch systems, software is made available via NFS and makes extensive use of environment modules. Each application is typically only ever run on a limited number of clusters and can be difficult to build and run elsewhere. Since we want to make use of distributed computing resources it becomes essential to ensure that applications are portable and reproducible, i.e. they can easily and reliably run anywhere and give the same result.

Due to the potential for a wide variety of applications and dependencies, containers are the most logical way to distribute software. Users or groups can build and test their own container images locally, knowing that their application can be run reliably anywhere. We use Docker Engine to build container images, but we use two unprivileged container runtimes for running jobs, specifically Singularity [5] and udocker [6].

2.2 Interacting with Prominence

User interaction with Prominence is via a RESTful API and JSON, using OpenID Connect (OIDC) token-based authentication and authorisation. This is significantly more flexible than requiring users to connect to a login node and interact with a batch scheduler, as it gives users the freedom to manage jobs from anywhere, and can easily be used programmatically using any language, for example from a Jupyter notebook.

A command line interface (CLI) provides a simple batch system style interface for remotely managing workloads, giving users almost the same experience that they have on a standard batch system but with the advantage that the CLI can be run from anywhere.

Workflows and jobs are described using a JSON schema. While there are many examples of workload descriptions in JSON, YAML and XML, none are commonly used and they don't provide all the features we require. We therefore use our own generic workload descriptions, based on three main components: workflows, jobs and tasks. An example is illustrated in Figure 1.

A task is an atomic computing job, which consists of a container image, command to be executed, any required environment variables and the choice of container runtime. A job contains a list of one or more tasks to be executed sequentially, metadata (arbitrary key-value pairs) and additional information required to run the tasks, in particular the required resources (such as CPUs and memory), any input and/or output data, and optionally policies. Workflows consist of a list of jobs, any dependencies between them, metadata, and optionally policies. Job policies enable users to influence where jobs will run. Both job and workflow policies include retry policies for dealing with failures.

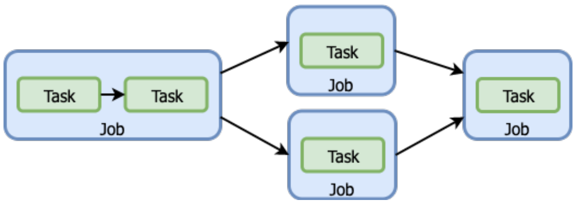

Fig. 1. An example workflow showing how it is built up from jobs, with dependencies between them, where each job consists of one or more tasks.

Since resources are specified at the job level, the resources used by a workflow can be tailored for each individual step. A workflow can contain combinations of both HPC and HTC jobs, where the HPC jobs use MPI and run across multiple nodes while the HTC jobs each run on single nodes, with the jobs potentially being run on different sites dependent on the resources required.

2.3 Architecture

An overview of the Prominence architecture, showing the basic building blocks, is shown in Figure 2. Identity is provided by an OIDC server, such as INDIGO IAM [7] or EGI Check-in [8]. The foundation of Prominence is provided by HTCondor [9], providing essential basic functionality such as a transactional database for jobs, the ability to securely run jobs remotely, and the ability to stream standard output and error while a job is running. HTCondor job router hooks, which allow arbitrary code to be invoked at defined points in the job lifecycle, are used to initiate deployment and deletion of infrastructure on clouds by interaction with a placement policy engine. Infrastructure Manager (IM) [10] is used to manage the infrastructure.

With this model, a unique VM (or group of VMs for multi-node MPI jobs) runs each job and is destroyed immediately after the job finishes. This is of course not very efficient for large numbers of short jobs, but typical fusion jobs run for hours or days, resulting in the time taken to deploy a VM being small compared to the lifetime of a job. The use of a VM per job also helps to ensure that we use resources efficiently since VM flavours can be tailored to each job. The combination of potentially long-running jobs and a wide range of resource requirements would present a challenge in trying to use VMs efficiently if they were all of the same size and each ran multiple jobs.

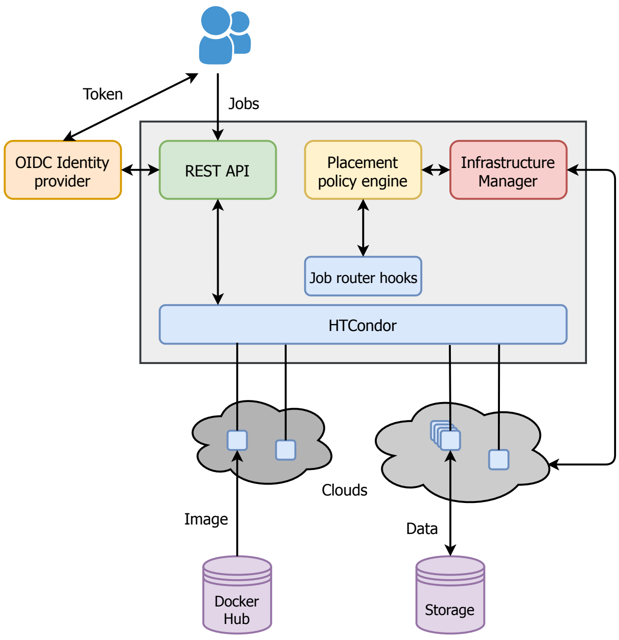

Fig. 2. An overview of the Prominence architecture.

A placement policy engine is used to decide where to deploy infrastructure for each job, based on each job's requirements and preferences. This is implemented using policies in Open Policy Agent (OPA) [11], an open-source general purpose policy engine. OPA is used to store static and dynamic information about each cloud, such as current quota (where available), recent failures, available images and VM flavours. It also maintains policies to determine what clouds meet requirements and to rank clouds based on preferences. Clouds are automatically queried for the required dynamic information.

Extensive failure handling is essential. If deployment on any cloud fails it will be retried up to a specified number of times on the same cloud. After this, if the infrastructure cannot be deployed, another cloud meeting the job's requirements is tried. This process is repeated until either the infrastructure is successfully deployed or all clouds have been tried and failed. There are timeouts for all state changes, so for example, if a VM is stuck in a provisioning state, after a time limit it will be deleted and retried.

The actual deployment of virtual infrastructure is handled by IM, which acts as an abstraction layer allowing us to use a single API for accessing multiple clouds. We have successfully tested Prominence with OpenStack, Microsoft Azure, AWS and Google Cloud Platform, in addition to OpenStack and OpenNebula clouds in EGI FedCloud accessible via the OCCI API.

With this system we can preferentially use a local cloud but burst onto national research clouds when needed, then burst onto EGI resources when needed, and finally burst onto public clouds. Jobs with special requirements, such as high CPU counts or low-latency interconnects, will only be run on appropriate clouds.

Finally, there needs to be some way for jobs to access input data and for users to access output data generated by jobs. In the absence of a standard way of remotely accessing data in the fusion energy research community, we chose to make use of Ceph-based object storage with the S3 API [12]. This enables data to be automatically staged-in and staged-out allowing jobs to access the required data, and output data can be made available to users, all through the use of pre-signed URLs.

The CLI provides a simple way for users to upload any input files or container images, list uploaded files and download output files, using pre-signed URLs supplied by the Prominence RESTful API. In the absence of a private container registry, this also enables jobs to obtain private container images, stored as either a tar-ball or in the Singularity format. This feature is important as many fusion codes cannot be made available on a public registry.

Multi-node MPI jobs generally require access to shared storage. We leverage BeeGFS [13] in order to provide multi-node MPI jobs with converged shared storage, available only for the lifetime of the job, aggregating the performance and capacity of the nodes used to run the job.

3 Example use cases

Here we describe some example fusion use cases which have been migrated to make use of Prominence.

3.1 Breeding blanket design

A breeding blanket covers the interior of a nuclear fusion vessel and has the important role of ensuring tritium self-sufficiency in deuterium-tritium fusion power plants. A blanket design tool was created which automates and parameterises the production of 3D CAD-based neutronics models. Machine learning was used to drive the parametric model creation and optimise the breeder blanket performance in terms of tritium production.

From a computational point of view, this use case was quite straightforward as it just involved running a large number of identical jobs. These jobs were completely self-contained and self-organising, in that each job contacted a remote database to obtain a set of input parameters and wrote the output back into the database once complete. Prominence enabled these jobs to be run across four clouds in EGI FedCloud opportunistically, thus making use of resources previously inaccessible and resulting in a publication [14].

3.2 JET Application Management System

The JET Application Management System (JAMS) [15] is a user interface to a variety of transport and magnetohydrodynamics codes for the integrated simulation of all phases of a Tokamak. It includes the ability to submit and manage jobs on a local batch system. Our aim was to extend this to be able to make use of external resources using Prominence, allowing users to continue running jobs when local resources are full.

Python scripts were written to enable JAMS to be able to submit jobs to Prominence using the RESTful API. The workflow used is fairly standard and applicable to many applications. When a user creates a job in JAMS, a set of input files and data is created in a unique "run directory", typically of the order of MB in size in total. JAMS uploads this run directory to object storage and submits a job to Prominence. Prominence provisions the required resources on a cloud, copies over the run directory and container image from object storage, executes the job then copies the run directory, now containing output files, back to object storage. JAMS is able to query the Prominence RESTful API in order to obtain a pre-signed URL which is used to download the run directory.

A time-consuming aspect of this use case was building the container image. It took around two months of development and testing to build and validate the containerised code, mainly due to having to start completely from scratch in the clean environment of a container rather than relying on all the existing dependencies in the environment of a cluster which has been used for many years.

The new system has been in used in production successfully for around six months.

3.3 HPC applications

The use of true HPC applications, i.e. multi-node MPI applications usually run on batch systems with low-latency interconnects, is very common in the fusion energy research community. Currently none of the cloud resources we have access to provide low-latency interconnects, nevertheless it is worthwhile to check the performance when running these applications on clouds with standard networking as many users prefer their jobs to be running, rather than waiting in a queue, even if they will run for longer. A variety of codes have been tested, including LAMMPS [16], VASP [17], BOUT++ [18], ASCOT [19] and JOREK [20]. Here we will only discuss LAMMPS as it highlights some common issues experienced with the other HPC codes.

The LAMMPS molecular dynamics simulation code is used in many communities, not just for fusion. The first step is to build a container image containing LAMMPS built with the Intel compiler, as it is known that use of the Intel compiler with the USER-INTEL acceleration package provides the best performance. Since the Intel compiler is licensed commercial software, we do this by building LAMMPS on a local cluster then copying the binary into a container image with the free Intel Parallel Studio XE Runtime installed.

Next it is important to consider the effect of containerisation on multi-node MPI jobs. We tried running LAMMPS on an HPC cluster using Singularity and compared this to LAMMPS running on bare metal, with up to eight 32-core machines. The differences in wall times between Singularity and bare metal was found to be negligible.

An interesting issue we encountered was that when the container image was optimised to give the best performance on one system it resulted in a loss of portability. LAMMPS built using the GNU compiler and Open MPI was very portable, and the same container image worked on multiple Intel processors and other systems like AMD EPYC. However, when compiled with the Intel compiler and optimised for AVX512, giving the best performance on recent Intel processors (around four times faster than GNU/Open MPI), it of course didn't work at all on processors not supporting the AVX512 instruction set. Unlike public clouds, none of the academic clouds we have access to provide any means of requesting a particular type of processor, and in some cases have a mixture of machines with different generations of processors supporting different instruction sets, making it difficult to determine the appropriate clouds to run the most optimised codes.

The performance of multi-node MPI jobs on clouds without low-latency interconnects was, as expected, significantly poorer than on HPC systems. Generally runtimes increased as the number of nodes used increased, however with some examples two nodes did give better performance than a single node. The decrease in performance as the number of nodes increases makes sense since the MPI communication increases significantly as the number of nodes increases in the case of nodes with high core counts.

4 Summary and outlook

We have presented a platform enabling users to exploit idle cloud resources for running containerised scientific workloads. All cloud selection and infrastructure provisioning is handled completed automatically and is fully transparent, giving users the same experience as using a local batch system. We are currently working on several additional fusion use cases, including running actual workflows rather than individual jobs.

Finally, it is worth pointing out that Prominence is not fusion specific at all and could be used by other communities.

The European Open Science Cloud for Research pilot project was funded by the European Commission, DG Research & Innovation under contract no. 739563. This work benefited from services and resources provided by the fedcloud.egi.eu Virtual Organization, supported by the national resource providers of the EGI Federation. We acknowledge computing resources supplied by the IRIS (STFC) Consortium.

References

1. F. Romanelli, P. Barabaschi, D. Borba, G. Federici, L. Horton, R. Neu, D. Stork, H. Zohm, "Fusion Electricity: A roadmap to the realization of fusion energy" (2012)
2. EOSCpilot, “The European Open Science Cloud for Research Pilot Project”. Available from https://eoscpilot.eu/ [accessed 2020-06-09]
3. E. Fernández-del-Castillo, D. Scardaci, Á. L. García, Procedia Comput. Sci. **68**, 196 (2015)
4. Prominence project, “Prominence” [software]. Available from https://prominence-eosc.github.io/docs/ [accessed 2020-06-09]
5. G. M. Kurtzer, V. Sochat, M. W. Bauer, PLoS ONE 12(5): e0177459 (2017)
6. J. Gomes, E. Bagnaschi, I. Campos, M. David, L. Alves, J. Martins, J. Pina, A. López-García, P. Orviz, Comput. Phys. Commun. **232**, 84 (2018)
7. A. Ceccanti, M. Hartd, B. Wegh, A.P. Millar, M. Caberletti, E. Vianello, S. Licehammer, J. Phys.: Conf. Ser. **898** 102016 (2017)
8. EGI, “EGI Check-in Service” [software]. Available from https://wiki.egi.eu/wiki/AAI [accessed 2020-06-09]
9. D. Thain, T. Tannenbaum, M. Livny, Concurr. Comp.-Pract. E. **17**, 323 (2005)
10. M. Caballer, I. Blanquer, G. Moltó, C. de Alfonso, J. Grid Computing **13**, 53 (2015)
11. Open Policy Agent project, “Open Policy Agent” [software], version 0.13.0, 2019. Available from https://github.com/open-policy-agent/opa/releases/tag/v0.13.0 [accessed 2020-06-09]
12. Red Hat, “Ceph” [software]. Available from https://ceph.io [accessed 2020-06-09]
13. ThinkParQ, “BeeGFS” [software]. Available from https://www.beegfs.io/content/ [accessed 2020-06-09]
14. J. Shimwell, R. Delaporte-Mathurin, J.-C. Jaboulay, J. Aubert, C. Richardson, C. Bowman, A. Davis, A. Lahiff, J. Bernardi, S. Yasin, X. Tang, Nucl. Fusion **59** 046019 (2019)
15. M. Romanelli, G. Corrigan, V. Parail, S. Wiesen, R. Ambrosino, P. Da Silva Aresta Belo, L. Garzotti, D. Harting. F. Köchl, T. Koskela, L. Lauro-Taroni, C. Marchetto, M. Mattei, E. Militello-Asp, M.F.F. Nave, S. Pamela, A. Salmi, P. Strand, G. Szepesi, Plasma and Fusion Research **9** 3403023 (2014)
16. S. Plimpton, J. Comp. Phys. **117**, 1 (1995)
17. G. Kresse, J. Hafner, Phys. Rev. B **47**, 558 (1993)

18. B.D. Dudson, M.V. Umansky, X.Q. Xu, P.B. Snyder, H.R. Wilson, Comp. Phys. Comm. **180**, 1467 (2009)
19. J.A. Heikkinen, S.K. Sipilä, Phys. Plasmas **2**, 3724 (1995)
20. G.T.A. Huysmans, O. Czarny, Nucl. Fusion **47**, 659 (2007)