HEP Analyses on Dynamically Allocated Opportunistic Computing Resources

M J Schnepf, R F von Cube, C Heidecker, M Fischer, M Giffels, E Kuehn, A Heiss, A Petzold, G Quast and M Sauter

KIT - Karlsruhe Institute of Technology, Germany

E-mail: `matthias.schnepf@kit.edu`

Abstract. The current experiments in high energy physics (HEP) have a huge data rate. To convert the measured data, an enormous number of computing resources is needed and will further increase with upgraded and newer experiments. To fulfill the ever-growing demand the allocation of additional, potentially only temporary available non-HEP dedicated resources is important. These so-called opportunistic resources cannot only be used for analyses in general but are also well-suited to cover the typical unpredictable peak demands for computing resources. For both use cases, the temporary availability of the opportunistic resources requires a dynamic allocation, integration, and management, while their heterogeneity requires optimization to maintain high resource utilization by allocating best matching resources. To find the best matching resources which should be allocated is challenging due to the unpredictable submission behavior as well as an ever-changing mixture of workflows with different requirements.

Instead of predicting the best matching resource, we base our decisions on the utilization of resources. For this reason, we are developing the resource manager `TARDIS` (Transparent Adaptive Resource Dynamic Integration System) which manages and dynamically requests or releases resources. The decision of how many resources `TARDIS` has to request is implemented in `COBalD` (**CO**Bald - The **O**pportunistic **Bal**ancing **D**aemon) to ensure further allocation of well-used resources while reducing the amount of insufficiently used ones. `TARDIS` allocates and manages resources from various resource providers such as HPC centers or commercial and public clouds while ensuring a dynamic allocation and efficient utilization of these heterogeneous opportunistic resources.

Furthermore, `TARDIS` integrates the allocated opportunistic resources into one overlay batch system which provides a single point of entry for all users. In order to provide the dedicated HEP software environment, we use virtualization and container technologies.

In this contribution, we give an overview of the dynamic integration of opportunistic resources via `TARDIS`/`COBalD` in our HEP institute as well as how user analyses benefit from these additional resources.

1. Introduction

High energy physics (HEP) searches for a better understanding of the fundamental forces and building blocks in our universe. The searches include several analyses from detector studies up to searches for a specific particle and its behavior. These analyses are divided into workflows such as event simulation, skimming, and event reconstruction. Each of these workflows has differing requirements in computing resources: event simulations typically need CPU power and memory, whereas event analyses are I/O limited. Typically, users submit a big number of jobs of a single workflow to the batch system. Especially in small batch systems this results in a high demand

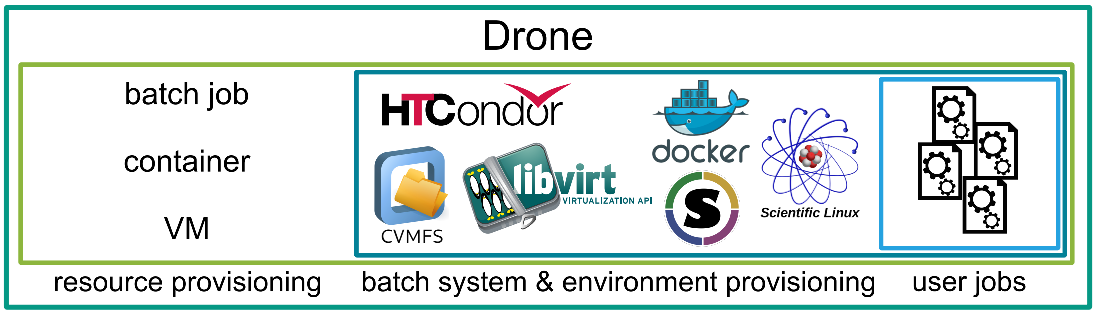

Figure 1. A `drone` provides resources for user jobs with a dedicated software environment. This can be done via container or virtualization technology.

of resources that might not be available. This common situation prevents the submitted jobs and other workflows from starting promptly. Additionally, the demand for computing resources in HEP is increasing due to higher data rates of the experiments [1].

2. Integration of Additional Resources

The most significant part of the computing resources for the LHC collaborations is provided by the Worldwide LHC Computing Grid (WLCG) [2]. The collaborations integrate resources from the WLCG sites into their resource pool via the pilot concept [3]. A pilot is a placeholder job which runs on a Grid site. This pilot job starts a worker node process of the overlay batch system (OBS), which allows running user jobs on the resources of the Grid site allocated to the pilot job. In this concept, users interact only with the OBS. This concept works fine for homogeneous resources where each resource provides the same software environment and similar hardware.

Other resource providers outside the WLCG, e.g. HPC centers, and commercial clouds provide computing resources in various hardware and software configurations and different forms, e.g. as a batch job, or as a virtual machine. To integrate these opportunistic resources into a resource pool of an OBS, we propose the more generalized `drone` concept [4]. The `drones` ensure a dedicated HEP software environment for resources in different forms independent of the hardware configuration. This concept enables us to run HEP jobs on resources that are not designed for HEP.

A `drone` has several parts which are shown in Figure 1. The resource provisioning part of a `drone` depends on how the resources can be allocated. The `drone` can act as a batch job, virtual machine or a container in which the software provisioning part runs. The software provisioning part takes care of running the OBS worker node process and to provide the software environment required by the job. The `drone` provides this environment either natively, via virtualization, or using container technology. The jobs then run inside the provided environment. At our physics institute, we are using an HTCondor [5] instance as the OBS. HTCondor has built-in support for `docker` [6] and `Singularity` [7] containers, which enables us to run each job in a HEP software environment. The default software environment includes `CVMFS` as well as Scientific Linux 6 and is provided transparently. This enables users to use various computing resource without having to adapt their code to each resource.

3. Resource Management

The `drone` concept enables the use of various resources outside of the WLCG. We developed a resource manager `ROCED` [8] to enable on-demand provisioning and management of resources from various providers. We used `ROCED` to manage `drones` at the HPC cluster NEMO [9] at

Freiburg, Germany. Due to the growing demand for computing resources, we need to integrate further resources from various different resource providers. ROCED monitors the job queue of the OBS. Based on the number of jobs that could be run on a particular resource provider, ROCED decides the number of drones to request from that resource provider. Usually, resources from different resource providers are heterogeneous. Therefore, ROCED needs to choose the resources fitting best to the jobs currently in the queue of the OBS. The current version of ROCED does not assess the goodness of fit when calculating the demand for drones. This results in many unused resources e.g. unused CPU cores due to insufficient memory for further jobs.

Further issues with the current version of ROCED include the evolving character of job composition that can result in requirement changes within minutes. Additionally, the availability of resources is often not known in advance. Instead of predicting this complex situation by improving ROCED accordingly, we decided to follow a more flexible and dynamic approach. In that approach, the demand for drones is based on the utilization of the currently running drones. Thereby, we aggregate similar drones to a pool where each pool is individually managed. Our implementation of that approach is the COBalD framework (**CO**BalD - the **O**pportunistic **Bal**ancing **D**aemon) [10].

COBalD handles abstract resources via pools. It increases the demand for resources in a pool if the utilization of these resources in the pool is high. If the utilization of the resources in the pool is low, COBalD will decrease the demand for these resources in that pool. Resources in COBalD can, for example, represent the number of special batch job slots or the number of drones. This abstract handling of resources within COBalD means that it cannot handle drones directly. Therefore, we are developing TARDIS (Transparent Adaptive Resource Dynamic Integration System) [11]. TARDIS requests and manages drones and also translates the utilization of the drones so that COBalD can handle it.

The decision about increasing or decreasing the number of drones in a certain pool depends only on the utilization of that pool. Therefore, a TARDIS instance can be assigned to one pool at a specific provider and act independently from other TARDIS instances. Our implementation enables us to react on site-specific issues, where jobs do not run properly. These issues would result in low utilization of drones and, therefore, to a decreasing number of drones for this provider and pool. Figure 2 shows the resource management workflow of TARDIS.

Each of the TARDIS instances run independently from each other. Therefore, the system is more scalable and reliable compared to a setup with ROCED. It also enables various management policies for different resource providers. At the date of this publication, we use TARDIS in production state to manage all resources from external resource providers.

4. Current Status

Most of the computing resources used at our physics institute are provided by external resource providers. In November 2018, we dynamically integrated resources into our OBS from the HPC cluster NEMO at Freiburg and from the Open Telekom Cloud in the scope of the Helix Nebula Science cloud project [12]. The local institute resources, including desktop PCs and dedicated worker nodes, are statically integrated into the OBS. Figure 3 shows the usage as well as the number of resources, integrated into our OBS.

As can be seen in Figure 3, we did not get as many resources as required to fulfill the demand of all jobs. This results in the difference between the brown and light blue line in Figure 3. This is a result of a fixed quota of resources by Open Telekom Cloud and the fair share at the HPC cluster NEMO. The fair share at an HPC center is one example of why the availability of resources can vary. Also, the number of jobs shows peak loads which are difficult to predict.

The stacked colored areas show the number of jobs running on one of the integrated resources weighted with the number of the requested CPU cores. The blue and yellow areas represent the desktop PCs and dedicated worker nodes located at the physics institute. Some of the CPU

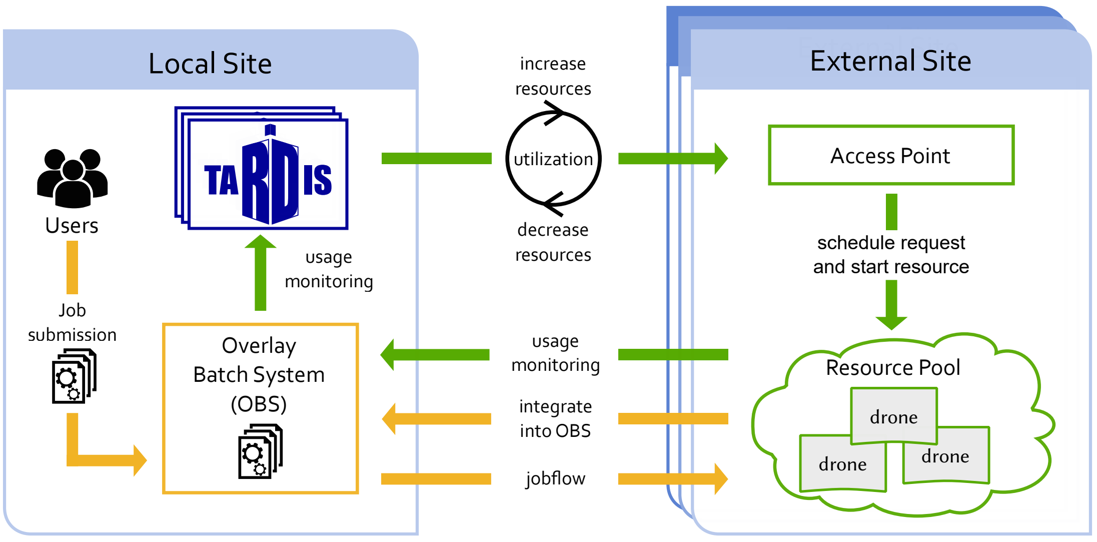

Figure 2. Users interact only with the OBS. The OBS schedules jobs to `drones` which are integrated from external sites. Each `TARDIS` instance monitors the utilization of the resources of its pool via the OBS. Depending on the utilization, `TARDIS` requests additional resources or reduces the number of `drones` from the provider. The resource provider schedules the `drone` on its resource pool. After the `drone` started, the `drone` is integrated into the OBS either by itself or by `TARDIS` depending on the OBS.

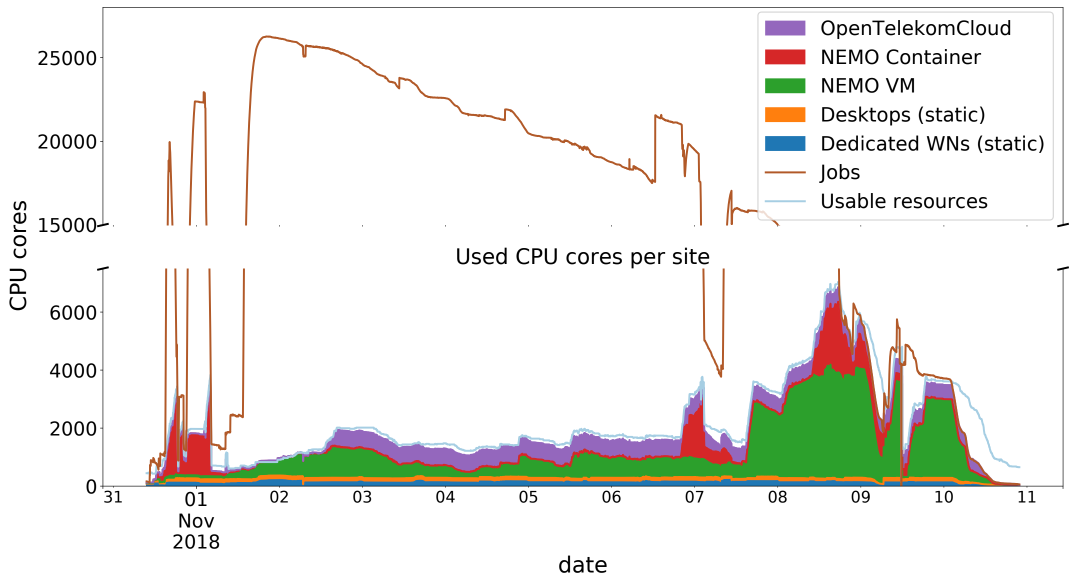

Figure 3. The brown line shows the number of jobs weighted with the number of requested CPU cores. The light blue line shows the number of CPU cores which were integrated into the OBS of the physics institute.

cores could not be matched with a job due to other constraints on the `drone` such as memory or disk space and are represented by the difference between the light blue line and the filled colored areas. However, this is acceptable to provide more resources to the users. The vast amount of additional resources results in a shorter waiting time and enables our users to run not only a higher number of workflows but also more complex workflows to improve their physics analyses.

5. Further Steps

In this contribution, we showed how we integrate opportunistic resources from different resource providers with `TARDIS` and `COBalD` into an OBS and make these transparently available to users. Some of the resource providers have a limited network bandwidth. Therefore, we study the correlation between network utilization and CPU efficiency to include an indirect network scheduling into `TARDIS`. Furthermore, we plan to improve the behavior of `TARDIS` to deal with very low and high utilization to ensure faster and more efficient resource management. We also plan to support further resource providers, which gives a broader range of resources. We assume this will result in a significant increase in the number of resources for our users. This will enable large-scale scalability tests of our approach and setup.

Acknowledgement

The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG as well as the support by the DFG-funded Doctoral School "Karlsruhe School of Elementary and Astroparticle Physics: Science and Technology".

References

[1] HEP Software Foundation 2018 A Roadmap for HEP Software and Computing R&D for the 2020s https://arxiv.org/abs/1712.06982

[2] Eck C et al. 2005 LHC computing Grid : Technical Design Report *Technical Design Report LCG*, https://cds.cern.ch/record/840543

[3] Sfiligoi I, Bradley D C, Holzman B, Mhashilkar P, Padhi S and Wurthwein F 2009 The Pilot Way to Grid Resources Using glideinWMS, *2009 WRI World Congress on Computer Science and Information Engineering* p. 428-432

[4] Schnepf M et al. 2019 Dynamic Integration and Management of Opportunistic Resources for HEP *Proceedings of the 23rd International Conference on Computing in High Energy and Nuclear Physics* to be published

[5] HTCondor Team. (2017, August 1) HTCondor 8.6.5 (Version 8.6.5) Zenodo http://doi.org/10.5281/zenodo.1324566

[6] Docker inc., "docker" [software], version 17.05.0-ce, Available from https://www.docker.com/ [accessed 2019-05-10]

[7] Kurtzer GM, Sochat V, Bauer MW, "singularity" [software], version 2.2.1, 11 May 2017. https://doi.org/10.1371/journal.pone.0177459

[8] ROCED project, "ROCED" [software], version 1.0.0, Available from https://github.com/roced-scheduler/ROCED [accessed 2017-10-12]

[9] Meier K et al. 2016 Dynamic provisioning of a HEP computing infrastructure on a shared hybrid HPC system, *Journal of Physics: Conference Series* Volume 762 012012

[10] Fischer M et al. (2018, December 3) MaineKuehn/cobald: Working version for HNSC and ConcurrencyLimits (Version v0.9.1) Zenodo http://doi.org/10.5281/zenodo.1887873

[11] Giffels M et al. (2018, December 13) MatterMiners/tardis (Version v0.1.0) Zenodo https://zenodo.org/record/2240606

[12] Helix Nebula Science Cloud, http://www.hnscicloud.eu/ [accessed 2019-05-02]