EPJ Web of Conferences **214**, 08009 (2019) https://doi.org/10.1051/epjconf/201921408009
CHEP 2018

Dynamic Integration and Management of Opportunistic Resources for HEP

Matthias J. Schnepf[1,*], *R. Florian* von Cube[2,**], *Max* Fischer[1,***], *Manuel* Giffels[1,****], *Christoph* Heidecker[1,†], *Andreas* Heiss[1,‡], *Eileen* Kuehn[1,§], *Andreas* Petzold[1,¶], *Guenter* Quast[1,‖], and *Martin* Sauter[1,**]

[1]KIT - Karlsruhe Institute of Technology
[2]3rd Institute of Physics A, Rheinisch-Westfälische Technische Hochschule Aachen

Abstract. Demand for computing resources in high energy physics (HEP) shows a highly dynamic behavior, while the provided resources by the Worldwide LHC Computing Grid (WLCG) remains static. It has become evident that opportunistic resources such as High Performance Computing (HPC) centers and commercial clouds are well suited to cover peak loads. However, the utilization of these resources gives rise to new levels of complexity, e.g. resources need to be managed highly dynamically and HEP applications require a very specific software environment usually not provided at opportunistic resources. Furthermore, aspects to consider are limitations in network bandwidth causing I/O-intensive workflows to run inefficiently.

The key component to dynamically run HEP applications on opportunistic resources is the utilization of modern container and virtualization technologies. Based on these technologies, the Karlsruhe Institute of Technology (KIT) has developed ROCED, a resource manager to dynamically integrate and manage a variety of opportunistic resources. In combination with ROCED, HTCondor batch system acts as a powerful single entry point to all available computing resources, leading to a seamless and transparent integration of opportunistic resources into HEP computing.

KIT is currently improving the resource management and job scheduling by focusing on I/O requirements of individual workflows, available network bandwidth as well as scalability. For these reasons, we are currently developing a new resource manager, called TARDIS. In this paper, we give an overview of the utilized technologies, the dynamic management, and integration of resources as well as the status of the I/O-based resource and job scheduling.

[*]e-mail: matthias.schnepf@kit.edu
[**]e-mail: florian.von.cube@rwth-aachen.de
[***]e-mail: max.fischer@kit.edu
[****]e-mail: manuel.giffels@kit.edu
[†]e-mail: christoph.heidecker@kit.edu
[‡]e-mail: andreas.heiss@kit.edu
[§]e-mail: eileen.kuehn@kit.edu
[¶]e-mail: andreas.petzold@kit.edu
[‖]e-mail: guenter.quast@kit.edu
[**]e-mail: martin.sauter@student.kit.edu

1 Current Situation

Most of the institutes involved in HEP computing have a static amount of dedicated worker nodes combined to an institute cluster and managed by a batch system. The institute clusters are usually designed to be constantly utilized. In contrast, the demand for resources at institutes shows peak loads due to working hours, conference deadlines, and machine schedules. This may result in a long waiting time for jobs. Certainly, there are various providers of computing resources such as Grid sites, HPC centers, cloud providers. So it is possible to expand the resources of an institute with additional resources from these resource providers for a given time.

To provide the users simple access to these resources, the idea is to transparently and dynamically integrate resources in an overlay batch system (OBS) as a single point of entry, which leads to new challenges. One challenge is the interaction with different resource providers, that requires customization to request and manages resources to different APIs and policies. Another challenge is the heterogeneity of resources. Often the provided resources do not fit optimal for the jobs which can result in underutilized resources. Furthermore, resources from external resource providers such as Grid sites, HPC centers, and cloud provider take some time to get available. Especially for HPC centers and Grid sites, this time is fluctuating and difficult to predict. All these challenges have to be handled by a resource manager.

An additional challenge is the provisioning of the dedicated software environment on resources which are not designed for HEP, so-called opportunistic resources. To provide the software environment on these opportunistic resources virtualization and container technologies can help. A dedicated software environment allows users to run their jobs on all available resources without customization.

In the following, we describe our concept to integrate opportunistic resources and our experiences with our current resource manager and give an overview of further developments.

2 Resource Allocation and Integration

There are a lot of resource providers which allow allocation of resources for a given time, such as HPC centers, clouds, and Grid sites. In order to transparently make the resources of the different providers available to users, we integrate the resources into an OBS. An OBS can be for example the batch system of an experiment or an institute.

Each of the big HEP collaborations has an OBS instance which is part of a workload management system such as glideinWMS [3] or PanDA [4]. However, the resources for these collaborations are mainly provided by Grid sites of the Worldwide LHC Computing Grid (WLCG) [2]. In order to make the resources of the Grid sites available to their OBSs, the resource manager connected to the OBS requests resources at Grid sites via a so-called pilot job. When this pilot job starts, it allocates resources for the OBS on the Grid site. Furthermore, the pilot job itself starts a worker node process of the OBS whereby jobs of the OBS run on the allocated resources.

However, the pilot concept is not sufficient for opportunistic resources where the resources are provided in different types and do not provide the dedicated software environment for jobs. For this purpose, we designed the DRONE concept which is a more generalized approach to allocate and integrate resources.

2.1 Drone Concept

A DRONE is a process that allocates a certain resource and integrates it into an OBS. This process can either run natively, encapsulated in a container or as a virtual machine. Thereby, it is possible to allocate resources from various providers.

Table 1. DRONE species is defined by two attributes: type of resource allocation and how the software environment is provided. Possible DRONE species and on which resource provider it is used.

provider	kind of DRONE process	provisioning of software env.
WLCG Pilot	natively	natively
NEMO VM (2016-2017)	VM	natively
NEMO VM (2017-2019)	VM	container
ForHLR II & NEMO container	natively	container
OpenTelekomCloud	VM	container

The DRONE starts a worker node process. This worker node process starts jobs which use the allocated resources of the DRONE similar to a pilot job. However, some allocated resources, especially opportunistic resources, do often not provide the dedicated software environment. Therefore, it is necessary for the worker node process to start the jobs natively, in a container or in a virtual machine inside the DRONE. However, how the software environment is provided depends on the possibilities offered by the resource provider, e.g. using container technology on a HPC cluster. Therefore, a DRONE is described via two attributes: type of resource allocation and how the software environment is provided. These two attributes define the species of a DRONE. However, not all combinations are useful. Table 1 shows which species of DRONES we are currently using at KIT and the WLCG pilots which are also described by the DRONE concept.

The common WLCG pilot is a batch job which runs natively on a worker node of a WLCG site. In this case, the OBS worker node process and the jobs run natively on a worker node and the software environment is provided by the Grid site. However, HPC clusters such as ForHLR II [5] do not directly provide the HEP software environment. On these resources, container technology is used to provide a software environment for each job. A special case is the NEMO a HPC cluster in Freiburg [6] which provides the infrastructure to start virtual machines on the NEMO HPC cluster. Commercial clouds provide their resources as virtual machines. In our case, we were allowed to allocate resources at OpenTelekomCloud [7] via Helix Nebula Science Cloud project [8]. As it turned out over the time, one can profit from running containers inside the provisioned virtual machine, in order to provide an isolated environment for each job and allow a job-specific monitoring, e.g., network I/O.

In addition to the operating system provided natively, by container or virtual machines, the HEP community provides fast evolving experiment software via the CERN virtual machine file system (CVMFS) [9]. CVMFS enables to use most recent software without reprovisioning of VM or container images.

3 Resource Management

All the opportunistic resources have to be requested and managed by a resource manager. Depending on the OBS and resource provider, the resource manager has to handle several tasks such as requesting, integrating and releasing resources.

3.1 Resource Manager ROCED

Due to these tasks of a resource manager and the high quantity of APIs to interact with different resource providers, we have started developing a resource manager, which is called ROCED [10] in 2015. ROCED is based on a modular design where adapters enable the interaction between different resource providers and OBSs. This gives high maintainability and easy extensibility. The interaction between ROCED and an OBS is shown in Figure 1.

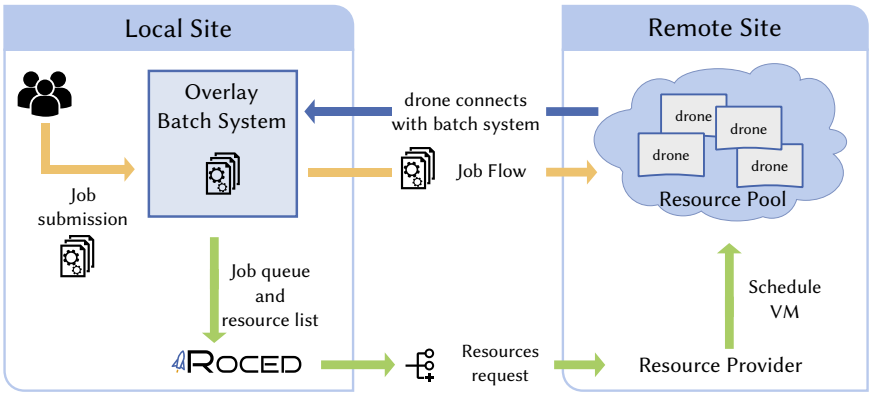

Figure 1. The users interact only with the OBS. ROCED pulls periodically the total amount of CPU cores needed by the jobs which fulfill certain requirements of the external resources such as in terms of number of CPU, memory and disk space. It also pulls periodically the list and status of resources integrated into the OBS. If there are more CPU cores needed than provided, ROCED requests additional resources. Once the resource is available and a DRONE has been started, it connects back to the OBS in order to get workload.

ROCED enabled us to integrate up to 8000 CPU cores of the NEMO HPC cluster dynamically and on-demand into our OBS (HTCondor [11]) at KIT. Figure 2 shows the number of allocated and used resources as well as the demand for resources for a week.

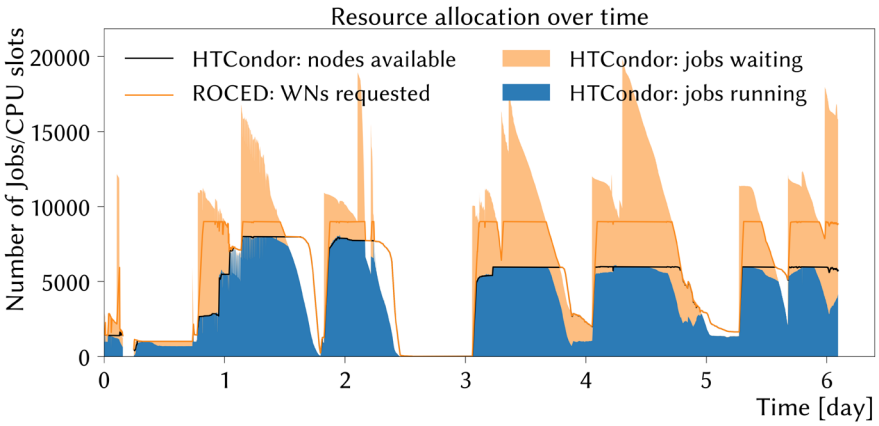

Figure 2. The orange area shows the demand of resources of the OBS at KIT. The amount of resources requested by ROCED is shown by the orange line. The blue area show the number of used resources and the blue line show the number of allocated resources provided by the NEMO HPC cluster in Freiburg. [12]

ROCED monitors the queue for demand and allocates additional resources according to its brokerage feature. This works fine for one type of DRONE or a set of homogeneous resources. However, it becomes more complex for heterogeneous resources. This usually

results in a lot of unused memory and CPUs when we added different kinds of resources which have another ratio of CPU, memory and disk space. One idea to reduce the number of unused resources is to predict the decision of the OBS job scheduler. Depending on that prediction the resource manager should request the corresponding type of DRONES. However, the availability of opportunistic resources is hard to predict precisely, since it depends for example on the utilization of external sites.

3.2 Resource Scheduling via Feedback Loop

Instead of predicting the decision of the OBS job scheduler to find the demand for each kind of DRONE we want to react on the OBS job scheduler via a feedback loop. For that, we define two metrics to describe the status of resources. The first one is the *allocation* of resources which describes the fraction of the provided resources which are assigned for usage. The *allocation* is also an indicator of the demand because not all provided resources are used.

The second metric to describe the status of resources is *utilization*. The *utilization* is the fraction of the provided resources which are actively used. The difference between *allocation* and *utilization* indicates how well the provided resources fit the demand. Based on *allocation* and *utilization* a decision software, so-called controller, decides to adjust the number of that type. This decision making is complemented by two other metrics: *demand* and *supply*, where *demand* represents the volume of resources should be requested and *supply* represents the volume of resources currently presented to the OBS.

In case of a DRONE, the *allocation* is the biggest fraction of assigned and available resource of the DRONE: e.g.

$$allocation = \mathbf{max}\left(\frac{\mathrm{CPU}_{assigned}}{\mathrm{CPU}_{total}}, \frac{\mathrm{memory}_{assigned}}{\mathrm{memory}_{total}}, \frac{\mathrm{disk\,space}_{assigned}}{\mathrm{disk\,space}_{total}}\right) \tag{1}$$

When the allocation is low due to a lot of unused resources inside a DRONE the demand for further DRONES goes to zero. If jobs completely allocate at least one resource type such as CPU cores, the demand for this resource type is high and so is the demand for additional DRONES. Thus more DRONES have to be requested to meet the demand. The *utilization* in case of a DRONE is the smallest fraction of assigned and available resource for the DRONE e.g.:

$$utilization = \mathbf{min}\left(\frac{\mathrm{CPU}_{assigned}}{\mathrm{CPU}_{total}}, \frac{\mathrm{memory}_{assigned}}{\mathrm{memory}_{total}}, \frac{\mathrm{disk\,space}_{assigned}}{\mathrm{disk\,space}_{total}}\right) \tag{2}$$

The difference between the *allocation* and *utilization* of a DRONE indicates how much resources are unused. For example, a DRONE has 8 CPU cores and 32 GB RAM which is a ratio of 4 GB per CPU core. The jobs which run on the DRONE are single core jobs with 3 GB RAM. This results in an *allocation* of one because all CPU cores are allocated. However, the *utilization* is 0.75, because three-quarter of the RAM is allocated. Thus the difference between the *allocation* and *utilization* describes how well the DRONE gets filled with jobs by the OBS and the type of DRONES fits the currently available jobs.

Based on these ideas we developed the COBaLD (COBalD - the Opportunistic Balancing Daemon) framework [13]. COBaLD is successfully used at GridKa, a WLCG Tier-1 center in Karlsruhe, Germany. At GridKa computing resources are shared among the different collaborations. However, the share among collaborations does not correspond with the fluctuating ratio of single core jobs and multicore jobs. This results in unused resources due to drained resources. To avoid the draining, a limit on the number of single core jobs is set. This limit will be adjusted in real time by COBaLD based on the current situation in the GridKa

batch system. Thereby is draining of resources not necessary which results in less unused resources [14].

In the case of the DRONE concept, COBaLD is utilized to adjust the number of DRONES depending on the current situation in the OBS. In order to efficiently manage different resources, the DRONES are organized in pools where each pool is managed by one controller. The controller regulates the number of DRONES by adjusting the *demand* of the pool, depending on its current *allocation* and *utilization*. A simple controller is the LinearController which increases the demand when the *allocation* is above and decreases the demand when the *utilization* is below a configured value. In other words, the controller increases the demand as long as the resources are efficiently used and decreases the demand otherwise.

While COBaLD is taking care of the resource balancing, an interface to request, integrate and manage DRONES at various resource providers is necessary for addition. For this purpose we are currently developing TARDIS (Transparent Adaptive Resource Dynamic Integration System) [15]. TARDIS uses the COBaLD framework to decide the number of DRONES per resource provider as a multi-controller system. This results in high scalability. Additionally, we use the knowledge from the development of ROCED. Thereby, adapters translate information and commands between TARDIS and the OBS as well as the resource provider. This design enables high maintainability and a simple expansion by adding adapters for additional resource providers and OBSs. TARDIS will replace ROCED as our resource manager.

4 Current Status and Further Plans

The DRONE concept allows providing a huge variety and amount of resources to our users in a transparent and dynamic way. However, this also brings new challenges.

All these resources, which we integrate, have no permanent HEP storage. Due to this fact, the jobs have to read their necessary files from remote Grid sites. The network bandwidth between the external computing resources and the HEP storage is often limited and shared. This limited network bandwidth results in a reduced CPU efficiency of jobs, which need data from Grid sites as shown in Figure 3. In some HEP analyses, different jobs or workflows read the same file several times. This avoids the limitation of the network bandwidth between the computing resources and the HEP storage.

Jobs and workflows which recurrently read the same files several times can profit from caching. However, the use of caching in a distributed system brings new complexity [16]. Another method to reduce the inefficient CPU usage is to limit the number of DRONES per provider according to available bandwidth. We are investigating the correlation between the average incoming network throughput and the CPU efficiencies of the jobs. We hope to use this correlation to identify network limitations via CPU efficiency. This information can be evaluated and the allocation of further resources are stopped, in case I/O-intense jobs run inefficiently.

5 Conclusion

The number of resource providers has been increased over the last few years. However, the variety of resources to integrate leads to further challenges. Our DRONE concept enables a dynamic and transparent way to provide dedicated and opportunistic resources to users. Thereby, resources from different providers are integrated into one overlay batch system. This gives users access to a huge number of resources at a single point of entry. Furthermore, the DRONE concept includes container and virtualization technologies to provide a dedicated software environment on all integrated resources. This allows users to run their jobs on all integrated resources without software customizations.

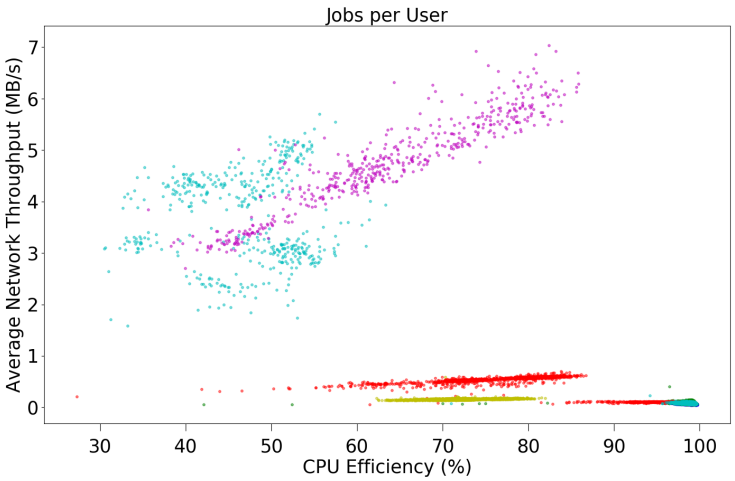

Figure 3. Monitoring data of finished jobs in our OBS which shows the CPU efficiency and the average incoming network throughput per job. Each color represents another user, which typically has one or two workflows. The distribution of the jobs is corresponding to their workflows.

Furthermore, we faced the challenge to manage a heterogeneous set of resource providers and resources in a dynamic way. We described our new approach via a feedback loop which should better react to the current demand and resource situations. This results in more efficient usage of the integrated resources. For this approach, we are currently developing the corresponding software. Furthermore, we are studying the correlation between CPU efficiency and network throughput per job to detect network limitations to further improve resource scheduling.

6 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] Sfiligoi I. et al. *The Pilot Way to Grid Resources Using glideinWMS*, WRI World Congress on Computer Science and Information Engineering (2009) DOI:10.1109/CSIE.2009.950

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

[3] glideinWMS project, [software], http://doi.org/10.5281/zenodo.1309679

[4] Nilsson P et al. *The PanDA System in the ATLAS Experiment*, Proceedings of XII Advanced Computing and Analysis Techniques in Physics Research (2008)

[5] Barthel R. and Raffeiner S. *ForHLR: a New Tier-2 High-Performance Computing System for Research*, Proceedings of the 3rd bwHPC-Symposium, Universitätsbibliothek Heidelberg, Heidelberg, 2017, 73-75

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

[7] OpenTelekomCloud, https://cloud.telekom.de/de/infrastruktur/open-telekom-cloud [accessed 2018-10-02]

[8] Helix Nebula Science Cloud, http://www.hnscicloud.eu/ [accessed 2018-10-02]

[9] Buncic P et al. "CVMFS" [software], version 2.X.Y, http://iopscience.iop.org/1742-6596/219/4/042003

[10] ROCED project, "ROCED" [software],(2018. December 3). Currently used Version at KIT (Version 1.1.0). Zenodo. http://doi.org/10.5281/zenodo.1888310

[11] HTCondor project, "HTCondor" [software],(Version 8.6.12). Zenodo. http://doi.org/10.5281/zenodo.1324566

[12] Heidecker C et al. *Dynamic Resource Extension for Data Intensive Computing with Specialized Software Environments on HPC systems*, Proceedings of the 5rd bwHPC-Symposium, Albert-Ludwigs-Universität Freiburg, Freiburg, to be published

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

[14] Fischer M et al. *Adoption of ARC-CE and HTCondor at GridKa Tier 1*, EPJ Web of Conferences CHEP 2018 proceedings (to be published)

[15] Giffels M et al. [software], in development, 2018, Available on https://github.com/giffels/tardis [accessed 2018-10-02]

[16] Heidecker C et al. *Advancing throughput of HEP analysis work-flows using caching concepts*, EPJ Web of Conferences CHEP 2018 proceedings (to be published)