EPJ Web of Conferences **245**, 07040 (2020) https://doi.org/10.1051/epjconf/202024507040
CHEP 2019

Lightweight dynamic integration of opportunistic resources

Max Fischer[1,*], *Eileen* Kuehn[1], *Manuel* Giffels[1], *Matthias Jochen* Schnepf[1], *Andreas* Petzold[1], and *Andreas* Heiss[1]

[1]Karlsruhe Institute of Technology, Herrmann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany

Abstract. To satisfy future computing demands of the Worldwide LHC Computing Grid (WLCG), opportunistic usage of third-party resources is a promising approach. While the means to make such resources compatible with WLCG requirements are largely satisfied by virtual machines and containers technologies, strategies to acquire and disband many resources from many providers are still a focus of current research. Existing meta-schedulers that manage resources in the WLCG are hitting the limits of their design when tasked to manage heterogeneous resources from many diverse resource providers.
To provide opportunistic resources to the WLCG as part of a regular WLCG site, we propose a new meta-scheduling approach suitable for opportunistic, heterogeneous resource provisioning. Instead of anticipating future resource requirements, our approach observes resource usage and promotes well-used resources. Following this approach, we have developed an inherently robust meta-scheduler, COBalD, for managing diverse, heterogeneous resources given unpredictable resource requirements. This paper explains the key concepts of our approach, and discusses the benefits and limitations of our new approach to dynamic resource provisioning compared to previous approaches.

1 Introduction

Dynamic resource provisioning in the WLCG [1] is commonly based on meta-scheduling and the pilot model [2]: A *meta-scheduler* pre-computes the ideal set of resources for a given set of workflows; so-called *pilot jobs* acquire and integrate these resources into an overlay batch system, which then processes the initial workflows. While this approach offers a high level of control and precision, we have found the strong coupling between components to inherently limit scalability, flexibility and robustness. These shortcomings are more severe when workflows and resources are under limited control – such as an intermediary WLCG site executing externally provided pilot jobs on opportunistic, non-WLCG resources.

To integrate dynamic resources, the GridKa Tier 1 centre has developed a new approach for dynamic provisioning that is suitable for the WLCG and beyond. By design, our approach decouples the distinct responsibilities of workflow scheduling, resource provisioning and meta-scheduling. Instead of seeking an optimal solution for a combined job scheduling and meta-scheduling problem, we divide the task into composable but orthogonal, self-balancing domains. Not only does this naturally provide scalability, flexibility and robustness,

*e-mail: max.fischer@kit.edu

it also allows us to manage a variety of resources and situations in a uniform way. We have successfully used our work for provisiong HPC and Cloud resources to the WLCG, as well as managing abstract resources in the form of multi-core and single-core allocations.

2 Job to Resource to Job Meta-Scheduling: ROCED

Classically, meta-scheduling schemes in the WLCG follow a *job to resource to job* (JRJ) approach. Notable examples are pilot submission frameworks of the major LHC VOs [3, 4]. In addition, the cloud meta-scheduler ROCED [5], previously developed at KIT and deployed for roughly a decade, follows this design.

While details vary, the general design of JRJ meta-schedulers consists of (i) one or several shared queues, to which users submit jobs, (ii) the meta-scheduler itself, which computes and acquires appropriate resources given the submitted jobs, and (iii) the job scheduler, which assigns acquired resources to submitted jobs. Notably, both the meta-scheduler and job scheduler are tasked with selecting processing resources for submitted jobs.

We have previously used ROCED to opportunistically use HPC and Cloud resources, with good results when using one resource provider at a time [6]. However, scaling out to use resources from multiple distinct providers has revealed several fundamental shortcomings, which roughly fall into two categories:

Resource Acquisition means interfacing with a resource provider to acquire resources to run jobs. The pilot usage model of acquire-use-release, as well as virtual machine and container technologies, makes it straightforward to technically support individual resource providers. However, different providers usually offer different resource types, which are not directly comparable – for example, some providers do not support multi-core jobs at all. This makes selecting appropriate resources across multiple providers a hard problem.

Job Scheduling means efficiently assigning as many jobs to as many resources as possible. Due to late-binding of pilots, the job scheduler performs practically the same task as with static resources, which we can optimize sufficiently [7]. In contrast, the meta-scheduler must predict resource usage, since resources are not immediately available. We have found this to be impossible in proper opportunistic use cases, since job demand can only be predicted on the scale of minutes [8] whereas resources take hours to days to acquire.

Since the mentioned shortcomings directly follow from the JRJ approach, they also apply to other meta-schedulers of the same design. Notably, they may prevent not just an optimal resource selection, but any reliable resource selection. As a result, we consider JRJ meta-scheduling unsuited to manage *multiple heterogeneous providers* given *non-static workloads and resources*. It is worth stressing that for a sufficiently static, homogeneous use-case JRJ meta-scheduling provides very good results, though.

3 Feedback Control Loop Meta-Scheduling: COBalD

The goal of our usage of meta-schedulers is not actually accurate job to resource matching, but merely high utilisation of available resources. Furthermore, the precise features of neither individual jobs [9] nor individual resources are known ahead of time. This has prompted us to propose and implement an approach that directly aims at optimising our desired target: The **C**OBalD – the **O**pportunistic **Bal**ancing **D**aemon (COBalD) [10] is a Feedback Control Loop (FCL) meta-scheduler that directly acts on observed resource utilisation.

The core idea of COBalD is to deduce which types of resources are optimal by observing how existing resources are actually used, avoiding the need to inspect, know or predict job requirements. This feedback is then used to control resources, namely to add used and remove

unused resources. Notably, COBalD only monitors, acquires and releases resources; there is no interaction with queued or running jobs, which are handled only by the job scheduler.

As a result of this, COBalD is agnostic with regard to the type of resources and their usage; in fact, it also works for setups not based on jobs. This allows to use the same approach for managing heterogeneous resources, e.g. resources of different CPU count or RAM/CPU ratio. In addition, we have successfully used COBalD to manage entirely virtual resources, namely the partition of multi-core slots at GridKa Tier 1 [7]. For our use-case of opportunistic resource usage in HEP, we provide the COBalD plugin TARDIS [11, 12] to manage virtual machines, containers, and pilots as part of an overlay batch system.

3.1 The COBalD Pool Model

To enable lightweight and efficient resource control, resources are logically abstracted within COBalD to a few features.[1] These features are the *allocation* (fraction of the resource reserved for use), the *utilisation* (fraction of the resource actually used), and the *supply* (volume of the resource). While these are abstract concepts, their definition is usually straightforward for a given resource (see Figure 1). The exact meaning is defined by the resource implementation.

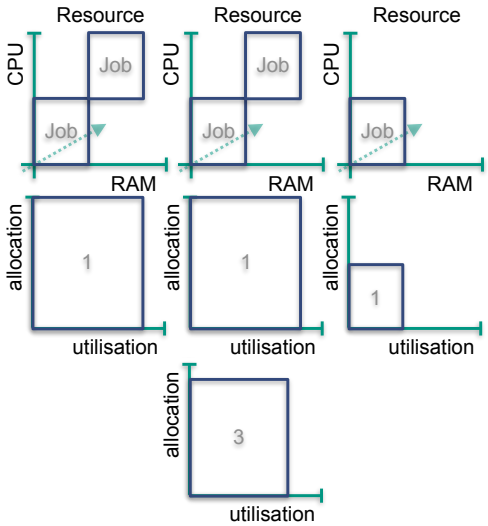

Figure 1. Compute resources and their representation by TARDIS: Compute resources are modelled as set of continuous features, e.g. a total amount of CPU cores, RAM, scratch space. Each job (or any other process) blocks a fraction of these resources for themselves. Allocation is derived from the most-used feature, signifying how much space is available for more jobs. Utilisation is derived from the least-used feature, signifying how much space is wasted by jobs. Thus, allocation and utilisation express how much and how well resources are used.

Figure 2. Aggregated resources in the COBalD Pool Model: A Pool represents several resources by their combined allocation, utilisation and supply. The appearance is yet again that of a resource, but intentionally hides sub-resources.

The resource features have been purposely chosen to allow aggregating multiple resources efficiently into a single *Pool*. Each Pool is itself represented as a single resource, and multiple Pools can be recursively aggregated if needed. Aggregation is a cheap transformation using simple mathematical formulas, for example aggregating the allocation/utilisation and supply of all constituents as their (weighted) average and sum, respectively (see Figure 2). The primary advantage of this representation is that individual resources become indistinguishable: COBalD itself only needs to adjust the desired, total volume of a single Pool, and plugins are free to manage and replace individual resources to match this volume.

Controlling the volume of Pools is done via simple rules, adequate for a Feedback Control Loop. Rules are usually formulated as positive or negative assertions, as thresholds or ratios, configured for the current use-case. Due to the modular design of COBalD, it is simple to

[1] Plugins may of course internally use additional features of resources, e.g. the identifier of a pilot job in a batch system. However, such features are not required or used by other COBalD components.

implement additional kinds of rules if needed. In our experience, it is sufficient to work with basic rules such as "reduce supply if utilisation below 80% otherwise increase supply until 10% unallocated" or similar.

3.2 Orthogonality of Job and Meta-Scheduler

The FCL design means that meta-scheduler and job scheduler are not nested, and that the meta-scheduler does not need to predict any actions of the job scheduler. This allows to operate and configure the job scheduler independently of the meta-scheduler – the only objective is good utilisation, as desired in static scenarios as well. Similarly, the meta-scheduler merely has to observe resources and does not need a model of how the job scheduler operates.

As a result, the FCL design allows to work with arbitrary job schedulers, including those which are again meta-schedulers – a situation that naturally arises when a WLCG site acquires opportunistic resources and runs pilot jobs from VOs. The job scheduler may use non-trivial, stateful scheduling policies, such as preemption, fairshare, or priorities, since the FCL meta-scheduler observes only their net effect of (not) utilising resources. Notably, this means COBalD naturally performs only two actions: provide more resources desirable for the job scheduler, and discard inadequate resources.

Unlike JRJ meta-schedulers, which must be aware of all jobs and all possible resources, an FCL meta-scheduler must only be aware of the subset of resources which it has acquired. This means an FCL meta-scheduler can operate even when the job scheduler has additional resources, even those managed by other FCL meta-schedulers: Multiple independent instances of COBalD can manage resources for the same job scheduler. In this scenario, each instance manages one type of resource – e.g. multi-core or single-core – at one resource provider – e.g. HPC or Cloud (see Figure 3). The job scheduler matches whatever resources are currently needed, and leaves the rest unused. This leads to only suitable resources being used, as unused resources are removed by their COBalD instance.

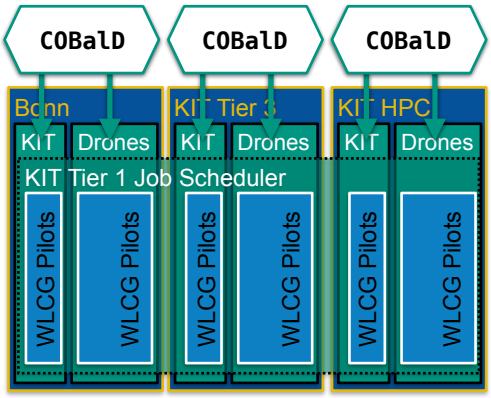

Figure 3. Orthogonal setup of COBalD meta-schedulers and job scheduler at KIT: Independent COBalD instances are deployed for various resource providers, each managing their own set of resources. COBalD uses TARDIS to run drones, the equivalent of pilot jobs, which integrate into a single overlay batch system. The overlay job scheduler, a copy of the regular GridKa Tier 1 scheduler, is the only component aware of all resources. As the job scheduler places WLCG pilots on acquired resources, each COBalD instance observes the usage of its own resource pools.

3.3 Towards Implicit Network Scheduling

Opportunistic resources commonly do not have the network bandwidth as WLCG centres, which offer dedicated connections in the WLCG. As such, network congestion can be a bottleneck for data processing on opportunistic resources. Since the available network bandwidth depends on the destination, may be congested by other connections, and can only be measured by saturation, it cannot generally be assigned a value for use in scheduling. To the

best of our knowledge, there are no means to adequately handle network with a JRJ meta-scheduler.

Instead, we propose to implicitly schedule network bandwidth by its side-effects, namely observing the efficiency of processes using the network. In specific, low CPU efficiency of processes is a clear indicator for lack of network throughput (see Figure 4), and directly measurable on all common operating systems. Thus, using the CPU efficiency as the utilisation of resources automatically causes COBalD to request resources with free network bandwidth and discard resources without sufficient bandwidth.

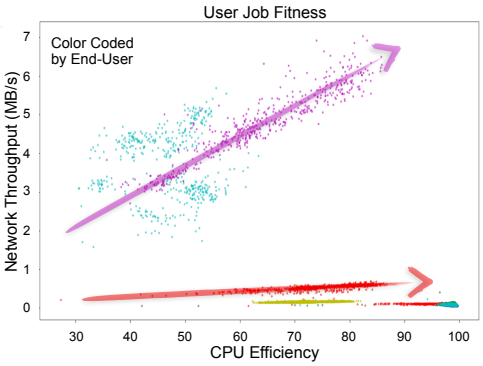

Figure 4. Relation between network throughput and CPU efficiency: Given jobs of the same workflow, too little available network throughput reliably coincides with low CPU efficiency. Notably, there is no general correlation between CPU efficiency and network throughput – other factors may reduce CPU efficiency as well. However, limited network reliably creates a limit for the maximum CPU efficiency possible.

This technique has so far been deployed for testing when backfilling HPC resources. We have observed that this successfully provides a safeguard against network congestion when running data analyses on opportunistic resources. In the future, we want to investigate how to best combine this approach with measuring utilisation by the best-fit. Ideally, we will be able to offer a solution to automatically acquire appropriately sized opportunistic resources, including local features such as CPU, RAM and GPU, as well as shared features such as network.

4 Conclusions

Even though job to resource to job meta-scheduling performs well for homogeneous resources and jobs, we have not been able to apply it to more complex, dynamic cases. JRJ meta-schedulers inherently duplicate responsibilities and introduce a high level of coupling between components. The inherent prediction required by the meta-scheduler has shown to be unfeasible given imprecise job requirements, e.g. from pilots, and unstable resource availability, e.g. opportunistic resources.

Instead, we propose a different approach using meta-schedulers based on a Feedback Control Loop (FCL) design, and have implemented this with COBalD, a lightweight, feedback based meta-scheduler. Instead of acting on predicted resource use, COBalD reacts to observed resource allocation and utilisation. This can be expressed with a generic resource model, capable of covering many use-cases.

We have already successfully used this simpler approach to modelling and managing resources in order to face challenges of providing opportunistic resources for HEP. Our COBalD plugin TARDIS [13] can integrate a variety of resource types into overlay batch systems. Due to the simple architecture, multiple instances of our meta-scheduler can supply the same overlay batch system and reliably provide heterogeneous resources. Finally, our work is the basis for implicitly scheduling network capacities, which may enable data intensive workflows even on opportunistic resources.

References

[1] J. Shiers *The Worldwide LHC Computing Grid (worldwide LCG)"* Computer Physics Communications **177** 219–223 (2007)

[2] M. Turilli, M. Santcroos, S. Jha *A Comprehensive Perspective on Pilot-Job Systems* ACM Comput. Surv. **51** 43 (2018) https://doi.org/10.1145/3177851

[3] I. Sfiligoi, 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

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

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

[6] C. Heidecker, et al. *Dynamic Resource Extension for Data Intensive Computing with Specialized Software Environments on HPC Systems*, Proc. o. t. 5th bwHPC-Sym. Freiburg, 159-161 (2018), DOI:10.15496/publikation-29051

[7] M. Fischer, et al. *Adoption of ARC-CE and HTCondor at GridKa Tier 1*, EPJ Web of Conferences **214**, 03053 (2019)

[8] E. Kuehn, et al. *Predicting resource usage for enhanced job scheduling for opportunistic resources in HEP*, EPJ Web of Conferences CHEP 2019 proceedings (to be published)

[9] E. Kuehn *Online analysis of dynamic streaming data*, 2018 DOI:10.5445/IR/1000083227

[10] COBalD project, "COBalD" [software], DOI:10.5281/zenodo.1887872

[11] COBalD-TARDIS project, "TARDIS" [software], DOI:10.5281/zenodo.2240605

[12] M. Schnepf, et al. *Dynamic Integration and Management of Opportunistic Re- sources for HEP*, EPJ Web of Conferences 214, 08009 (2019), DOI:10.1051/epjconf/201921408009

[13] M. Giffels, et al. *Effective Dynamic Integration and Utilization of Heterogenous Compute Resources*, EPJ Web of Conferences CHEP 2019 proceedings (to be published)