Advancing opportunistic resource management via simulation

Max Fischer and Eileen Kuehn

Karlsruhe Institute of Technology, Hermann-von-Helmholtz-Platz 1,
76344 Eggenstein-Leopoldshafen, Germany

E-mail: `max.fischer@kit.edu, eileen.kuehn@kit.edu`

Abstract. Modern high energy physics experiments and similar compute intensive fields are pushing the limits of dedicated grid and cloud infrastructure. In the past years research into augmenting this dedicated infrastructure by integrating opportunistic resources, i.e. compute resources temporarily acquired from third party resource providers, has yielded various strategies to approach this challenge. However, work on this topic is usually driven by practical needs to use specific resource providers for production workflows; in this context, research is ad hoc and relies on impressions gained during unique situations of resource providers, resource demand and opportunistic resource management. Replicating or even preparing a specific situation to investigate opportunistic resource management is extremely challenging or even impossible. As a result, thorough research in the field of opportunistic resource management is therefore extremely limited.

We propose to tackle this challenge using simulation and to this end present the simulation framework LAPIS, a general purpose scheduling simulator offering programmatic control of resources. We demonstrate this approach by integrating LAPIS with the COBalD/TARDIS resource manager to investigate the behaviour of this resource manager in a simulated environment.

1. Introduction

In order to manage end-user and automated usage of globally distributed compute resources, large collaborations in the High Energy Physics (HEP) domain rely on using *pilot jobs* [1] to form an intermediate scheduling layer. Various *workflow management systems* (WMS), such as GlideinWMS [2] or PanDA [3], act as single-point of entry for jobs to predict the demand for homogeneous grid resources to accurately send out pilot jobs. However, the desire to opportunistically use non-standard resources such as HPC clusters [4] has led to the development of dedicated *meta-schedulers* that solely focus on pilot job scheduling. Strategies for efficient use of heterogeneous and temporary resources is an active research challenge today.

While multiple implementations of meta-schedulers exist [5, 6] and are used productively, they are insufficient for scientific research: due to the many actors, participants and systems involved it is unfeasible to reproduce job loads, resource availability, or generally the environment in which a meta-scheduler operates. While simulation is an established tool to reproduce even complex environments, existing meta-scheduler implementations would still expect to interact with their environment in real-time. In this paper we propose a simulation that also serves as a runtime for a meta-scheduler itself; as a result, both the meta-scheduler and environment act on simulated time only which makes meta-scheduling reproducible on a practical time scale.

We present the considerations, implementation and demonstration based on the meta-scheduler COBalD/TARDIS [7, 8]. While COBalD/TARDIS is somewhat unique in its design of using a reactive feedback loop instead of demand analysis or prediction, this difference is incidental; our simulation does not rely on this design and the approach is in principle applicable to other meta-schedulers.

2. Dissection of a Meta-Scheduler Environment

Though our goal is to simulate meta-scheduling accurately, this does not mean simulating every single part of a production environment (see Figure 1). Instead, we aim to analyse which parts we can emulate, simulate or even remove to simplify the simulation without significantly impacting accuracy. This is vital both for efficiency and to be able to pinpoint which parts trigger which behaviour.

Critically, a significant part of meta-scheduler programs is glue-code to connect various external system. The correctness of such components is subject to testing and verification. So for simulation we assume they work according to specification. As such, whenever two simulated components interact we can omit their connection layer for efficiency.

As the purpose of meta-scheduling is to acquire and release resources in order to execute jobs we must simulate this to some degree. However, a meta-scheduler already highly abstracts both for scalability; in fact COBalD/TARDIS only considers its own pilot jobs but not even end-user jobs. Matching this abstraction we can consider it sufficient to simulate that jobs and resources can be started/acquired for some time as well as their status checked. Complexities such as data transfers, process executions and similar can be omitted. Notably, it is relatively simple to extend job- and resource simulation with a higher level of detail if desired.

The key component of the environment are the concrete schedulers connecting resources and jobs. This applies both for predictive meta-schedulers, which must predict decisions of the schedulers directly, as well as reactive meta-schedulers, which form a coupled system with schedulers. Still, in practice any meta-scheduler must be robust to not fully knowing the schedulers, be this for administrative, technical or feasibility reasons. As such, we consider it most important to simulate classes of schedulers instead of precisely replicating a specific scheduler implementation.

The core piece of a meta-scheduling environment is the meta-scheduler itself and especially its decision engine. In order to study its behaviour, we must use the decision engine implementation with minimal or ideally no modifications. Effectively, this means we must emulate the environment in which the decision engine executes.

3. The LAPIS simulation approach

The basis for our implementation is the LAPIS simulator [9] which is geared towards scheduling and resource management simulations. It builds on the Python `usim` library [10], which provides discrete-event simulation via Python's `async/await` syntax. This makes it possible to extend async programs with resource simulation.

The simulator provides building blocks of a batch system: a job queue, worker nodes and scheduler. Both jobs and workers are modelled as continuous blocks of resources, with each running job blocking its resources on the worker node while running. The scheduler is priority based, matching jobs to resources in-order as long as possible. This corresponds to a simple configuration of the HTCondor [11] batch system commonly used in HEP.

LAPIS is intended to be used as a library to build a simulation programmatically and, as such, is highly extensible. For example, it has been used to study cache locality based scheduling [12] by introducing caches and cache-aware schedulers. We can similarly extend LAPIS to run COBalD/TARDIS itself alongside its simulated batch systems.

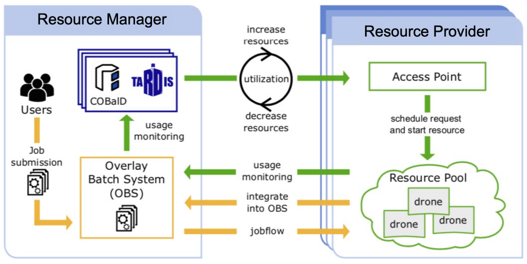

Figure 1. Schematic environment of the COBalD/TARDIS meta-scheduler in production: Each meta-scheduler instance is a self-contained program that only interacts with third-party resource providers and an overlay batch system (OBS). The resource provider is used to spawn pilot jobs – called "drones" in COBalD/TARDIS - that serve as worker nodes to the OBS. By monitoring which resources the OBS scheduler uses, the meta-scheduler gets feedback of which resources it must provide more or less.

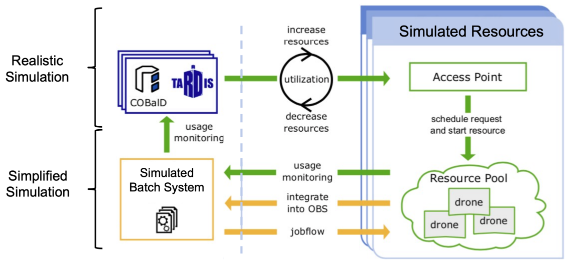

Figure 2. Simulated environment of the COBalD/TARDIS meta-scheduler: Simulation provides a simplified replica of an overlay batch system, including jobs, resources and scheduler but not the execution of actual work. In contrast, the provisioning of resources is closely modeled to match the abstraction level used by the meta-scheduler and includes one or more fully functional instances of COBalD/TARDIS.

To hook the simulated resources to COBalD/TARDIS, we have implemented a simulated resource interface which directly fits the resource abstraction of the meta-scheduler. This interface directly creates or destroys a worker, with the possibility to limit the maximum resource volume available. In addition, we have implemented a minimal emulation of the Python `asyncio` library based on `usim`. As the COBalD/TARDIS decision engine only interacts with the resource interface and `asyncio`, this allows us to run it completely inside LAPIS (see Figure 2).

4. Meta-scheduling example study

To verify and demonstrate our simulation approach, we have chosen a conceptually simple but practically complex use-case. In short, we run multiple meta-scheduler instances to serve the same pilot job infrastructure and job scheduler. This requires the meta-schedulers to implicitly or explicitly cooperate as their decisions affect one another via the shared pilot infrastructure. Notably, COBalD/TARDIS has no mechanism for meta-scheduler instances to communicate and the simulation also ensures this even though all instances run as part of the same simulation program.

For the specific scenario, we have recorded jobs submitted to a Tier 3 batch system. The resource requests are homogeneous but the submission time and runtime are irregular. Three meta-scheduler instances are used, each setup to create resource of only a specific size: The first size matches the memory to CPU ratio of jobs, but is too large for one job yet too small for multiple jobs. The second size matches the CPU request of jobs, but has too much memory. The third size matches the memory to CPU ratio of jobs, but requires multiple jobs to be completely filled. With this scenario, our goal is to study how the group of meta-schedulers react to the job requests that do not fit any single one perfectly.

We directly use the regular monitoring stream of COBalD/TARDIS to track how many resources each instance provides (see Figure 3) during the simulation. The resource volume is sufficient to see the general behaviour, so we purposely exclude more complicated metrics that are not required for this simple scenario. The meta-scheduler instances and simulation still

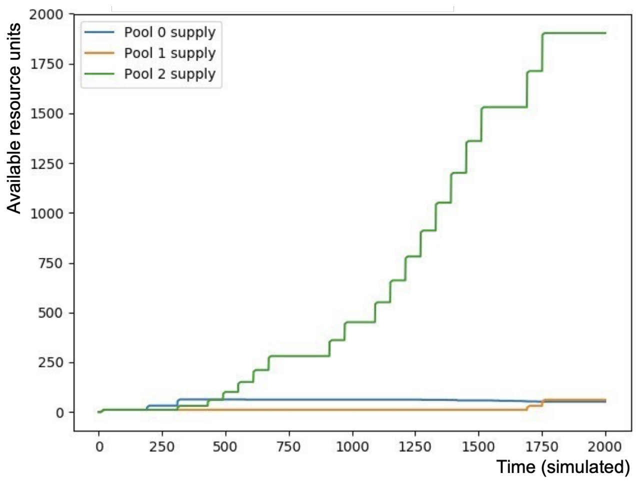

Figure 3. Resources provided by multiple meta-scheduler instances during simulated resource usage: Each resource pool is managed by one instance of COBalD/TARDIS and includes only resources of exactly the same size. The simulated time corresponds to seconds and a resource unit corresponds to the increment of resources in a pool. The data was read from the COBalD/TARDIS monitoring stream during a simulated scenario of gradually inserting jobs.

generate this data and one could use them for an in-depth or follow-up simulation study. In addition, we specifically look only at the initial ramp-up of resources, which should be the most insightful even for people not familiar with different meta-scheduler architectures and challenges.

Right after the simulation starts, we observe that all instances equally increase their provided resources. This is due to the feedback mechanism employed by COBalD/TARDIS, which requires having some resources available to probe whether the scheduler finds them suitable for jobs. While this phase mainly serves as a sanity check here, it is an open area of research how to more efficiently probe a wide array of resource types without having resources idle.

The first interesting phase is up until 5 minutes (300 seconds) during which the number of jobs is still low. There are still too few jobs to significantly fill large resources (Pool 2) and jobs are not a good fit for high-memory resources (Pool 1). Consequently, the small resources (Pool 0) are the best utilised and its meta-scheduler instance provides more resources.

This behaviour gets inverted in the second phase starting at 8 minutes (480 seconds) as the job number increases steadily. Now there are sufficient jobs to fill large resources (Pool 2) as well, which makes these resources more suitable than smaller ones (Pool 0) which have leftover resources after matching one job each. We see the group of meta-schedulers acting as desired here as the most suitable resources for high job pressure (Pool 2) now increase considerably to handle the increasing job load.

The simulation reveals a third phase starting at around 30 minutes (1800 seconds) that we did not anticipate. Here, the configured exponential scale-up of resources means that the most suitable resources (Pool 2) become significantly unused as the meta-scheduler increases provided resources too much at once. As a result, what we expected to be the least suitable resources (Pool 1) become competitive and their respective meta-scheduler increases resource volume as well.

Overall, even this simple scenario already demonstrates the emergent behaviour of running multiple meta-scheduler instances. In addition to the expected behaviour of scaling up resources selected with matching memory to CPU ratio, the scenario shows unexpected effects of how a suboptimal scaling strategy allows other resources to increase. Simulation allows us to observe these effects in a controlled environment, and in the future to repeat them to improve meta-scheduling strategies.

5. Conclusion and Outlook
By its very nature, meta-scheduling is a complex task that is quite removed from everyday experience. In addition to this inherent difficulty, research is further hindered by the unfeasibility

of reproducing and studying specific scenarios in deployed systems. In order to tackle the challenge of reproducibility, in this paper we have proposed and demonstrated a simulation based approach.

By carefully separating functional from incidental parts of a meta-scheduler, creating such a simulation is a feasible task even with limited manpower. This has allowed us to integrate the actual decision engine of a meta-scheduler application into a completely simulated job scheduling and resource acquisition scenario.

Such a simulation setup allows to reproduce observed scenarios or synthetic ones. This can be used to, for example, try different approaches for the same situation or check whether an approach would handle more difficult situations as well. Even a simple scenario can reveal complex emergent behaviour as multiple components work together, as shown in this paper. Simulation allows to study all of these situations safely and precisely.

The simulation and study shown in this paper provide a solid foundation for future work on meta-scheduling strategies. As hinted at in our example simulation, the probing phase when COBalD/TARDIS provisions resources purely to monitor them is an interesting target for optimisations. This phase is currently the limiting factor to using many meta-scheduler instances for handling different resource types; finding a better strategy for this phase would allow scaling up the number of different resources without loss of efficiency. In addition, we want to study and optimise the resource scaling behaviour at large resource volumes at which point our current strategy lacks granularity.

In addition to improving current meta-scheduling strategies, we are also looking into improving the simulation itself. Obvious areas are improving the accuracy of the simulation, as we are for example already looking into more closely replicating the HTCondor scheduler but also other kinds of scheduler. Finally, while our various simplifications to noise factors, such as ignoring latency and unreliability of communication between parts, are based on our own experience we would like to verify these by dedicated high-detail simulations as well.

References

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

[2] I. Sfiligoi, et al 2009 "The Pilot Way to Grid Resources Using glideinWMS" *WRI World Congress on Computer Science and Information Engineering* doi:10.1109/CSIE.2009.950

[3] P. Nilsson, et al. "The PanDA System in the ATLAS Experiment" *Proceedings of XII Advanced Computing and Analysis Techniques in Physics Research* (2008) doi:10.22323/1.070.0027

[4] J. Adelman-McCarthy, et al 2023 "Extending the distributed computing infrastructure of the CMS experiment with HPC resources" *J. Phys.: Conf. Ser.* **2438** 012039 doi:10.1088/1742-6596/2438/1/012039

[5] P. Mhashilkar, et al 2019 "HEPCloud, an Elastic Hybrid HEP Facility using an Intelligent Decision Support System" *EPJ Web Conf.* **214** 03060 doi:10.1051/epjconf/201921403060

[6] P. Marshall, K. Keahey and T. Freeman 2010 "Elastic Site: Using Clouds to Elastically Extend Site Resources" *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing* 43-52 doi:10.1109/CCGRID.2010.80

[7] M. Fischer, et al 2020 "Lightweight dynamic integration of opportunistic resources" *EPJ Web Conf.* **245** 07040 doi:10.1051/epjconf/202024507038

[8] M. Giffels, et al 2020 "Effective Dynamic Integration and Utilization of Heterogenous Compute Resources" *EPJ Web Conf.* **245** 07038 doi:10.1051/epjconf/202024507038

[9] E. Kuehn, T. Fesenbecker, M. Fischer, S. Lange, M. Schnepf 2020 "MatterMiners/lapis: v0.4.1 (v0.4.1). Zenodo." doi:10.5281/zenodo.3822378

[10] M. Fischer, E. Kuehn 2020 "MaineKuehn/usim: μSim - Simply Simulate (v0.4.3). Zenodo." doi:10.5281/zenodo.3813587

[11] D. Thain, T. Tannenbaum, M. Livny 2005 "Distributed Computing in Practice: the Condor Experience" *Concurrency and Computation: Practice and Experience* 17 323–356 doi:10.1002/cpe.938

[12] T. Feßenbecker 2020 "Modeling of distributed coordinated caching for LHC data analyses" MSc Thesis