Fair sharing of resources between clusters with AUDITOR*

Benjamin Rottler
Institute of Physics
Freiburg University

Michael Böhler
Institute of Physics
Freiburg University

Anton J. Gamel
Institute of Physics
Freiburg University

Dirk Sammel
Institute of Physics
Freiburg University

Markus Schumacher
Institute of Physics
Freiburg University

Raghuvar Vijayakumar
Institute of Physics
Freiburg University

Abstract

For several years, we have been dynamically and opportunistically integrating the computing resources of the HPC cluster NEMO into the HTC cluster ATLAS-BFG using the COBalD/TARDIS software. To increase usage efficiency, we allow the integrated resources to be shared between the various High Energy Physics (HEP) research groups in Freiburg. However, resource sharing also requires accounting. This is done with AUDITOR (AccoUnting DatahandlIng Toolbox for Opportunistic Resources), a flexible and extensible accounting ecosystem that can cover a wide range of use cases and infrastructures. Accounting data is recorded via so-called collectors and stored in a database. So-called plugins can access the data and take measures based on the accounted data. In this work, we present how NEMO resources can be fairly shared among contributing working groups when integrated into ATLAS-BFG using AUDITOR.

1 Introduction

High Energy Physics (HEP) studies fundamental particles and their interactions with the primary goal of discovering new elementary particles and precisely measuring the properties of existing ones. To achieve these goals, particles are accelerated to high energies and are brought to collision at the Large Hadron Collider (LHC) [1], the largest and most powerful particle accelerator in the world, located at CERN. It can collide protons up to 40 million times per second. The ATLAS detector [2], a general-purpose detector, records particle collisions at a rate of up to 3000 collisions per second, generating approximately $10\,\mathrm{PB}$ of data annually. This data is then analyzed by physicists around the world to gain more insight into the fundamental laws of nature.

After data acquisition, the recorded collision events undergo several phases of analysis. First, events are reconstructed to identify particles and to measure their properties. Subsequent steps filter out irrelevant events and aggregate low-level detector measurements into high-level physics observables. This analysis process starts on the Worldwide LHC Computing Grid (WLCG) [3], a global network of computing resources, and ends on local university clusters for final evaluation. To understand the complex processes that occur during particle collisions, the data analysis is supported by extensive simulations performed on the WLCG. In Freiburg, two compute clusters are available for HEP data analysis: ATLAS-BFG and NEMO.

ATLAS-BFG, a High Throughput Computing (HTC) cluster, is equipped with 3400 CPU cores and offers $4\,\mathrm{PB}$ of object storage. It is integrated into the WLCG and provides a specialized environment tailored for ATLAS data analysis. Job scheduling is facilitated by the Slurm [4] scheduler. The cluster is used to execute ATLAS simulation and event reconstruction tasks, as well as analysis jobs originating from local researchers.

NEMO[1], a High Performance Computing (HPC) cluster, is part of the bwHPC initiative serving the broader scientific community in Baden-Württemberg. Designed to support multiple scientific fields, NEMO is equipped with $18\,000$ CPU cores and $800\,\mathrm{TB}$ of parallel storage. Computing resources on NEMO are allocated via a batch system managed by the Moab [5] scheduler, adhering to a single-user node policy, where each node is only accessible to one user at a time. This cluster can be used by local researchers for their user analysis jobs.

*Proceedings of the 9th bwHPC-Symposium 2023. Mannheim Conference Series (MaConf). 2025.

[1] Neuroscience, Elementary Particle Physics, Microsystems Engineering and Material Science

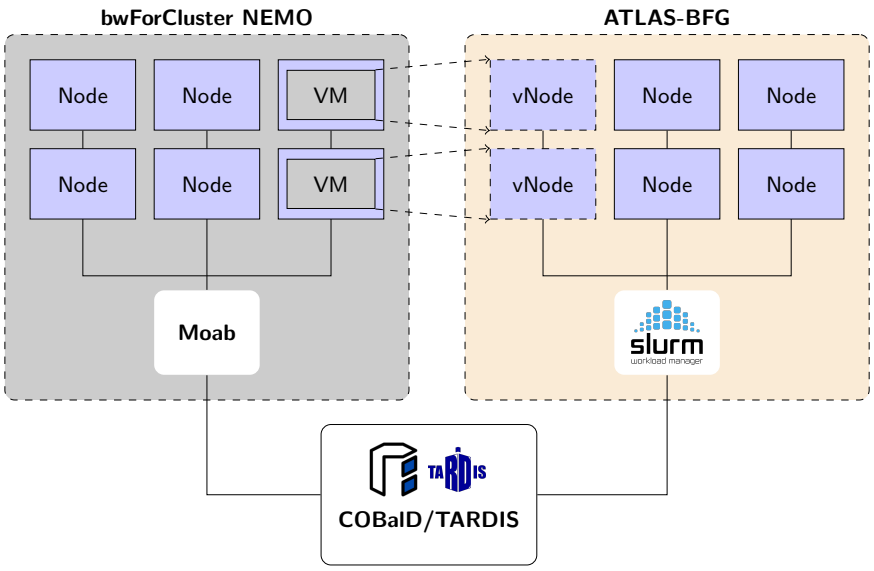

Figure 1: Integrating resources from the NEMO cluster (based on the Moab scheduler) into the ATLAS-BFG cluster (using the Slurm scheduler). NEMO acts as the underlying batch system (UBS) that provides resources for the overlay batch system (OBS), ATLAS-BFG. The resources are provided in the form of virtual machines (VMs) that appear as virtual nodes (vNodes) in the OBS. COBalD/TARDIS monitors the utilization of the vNodes and dynamically increases/decreases the number of VMs.

Since 2019, we have been dynamically and opportunistically integrating the computing resources of NEMO into ATLAS-BFG using the software tools COBalD [6] and TARDIS [7]. The COBalD/TARDIS system allows the orchestration of virtual machines (VMs) in one batch system, allowing them to be utilized as opportunistic resources in another batch system. The first batch system serves as an underlying batch system (UBS), while the second is classified as an overlay batch system (OBS). In our case, as shown in Figure 1, NEMO acts as the UBS and ATLAS-BFG acts as the OBS. When a new VM starts on NEMO, it connects as a virtual node (vNode) to the batch system managed by Slurm on ATLAS-BFG. COBalD/TARDIS monitors the CPU and memory usage of these vNodes and queues new VMs on NEMO when a predefined upper threshold for resource usage is reached. Furthermore, when no new VMs are required, idle VMs can be shut down, freeing up the previously allocated resources.

In Freiburg, four different HEP groups have a share in NEMO and are therefore allowed to use the cluster. Each group is allocated its own group fair share, which is shared among the group members. The fair share is a resource allocation policy that is utilised in computing environments, particularly in job schedulers such as Slurm, PBS and HTCondor, with the objective of ensuring the equitable distribution of computing resources. We operate a COBalD/TARDIS instance for each HEP group, which uses the corresponding group fair share in NEMO. On ATLAS-BFG, there is a Slurm queue for each group, to which users of that group can submit compute jobs. To efficiently utilize the opportunistic resources, we allow jobs from one group to be executed on a VM that was started by another group. This effectively unifies and distributes each group's fair share among the four HEP groups. Figure 2 shows an example of this setup with two groups.

Due to several factors, the amount of resources provided by each group may differ. For instance, users can still directly connect to NEMO and submit jobs there, which reduces the corresponding group fair share on NEMO. This can result in unfair situations, where one group provides fewer resources on NEMO but uses a larger portion of the integrated resources on ATLAS-BFG than the other groups. To restore fairness, the job priority for the groups on ATLAS-BFG should match the amount of resources provided on NEMO. This means that jobs from a group that provides more resources will be scheduled with a higher priority than jobs from the other groups. The AUDITOR [8] ecosystem, which will be introduced in the next section, can be used to achieve this.

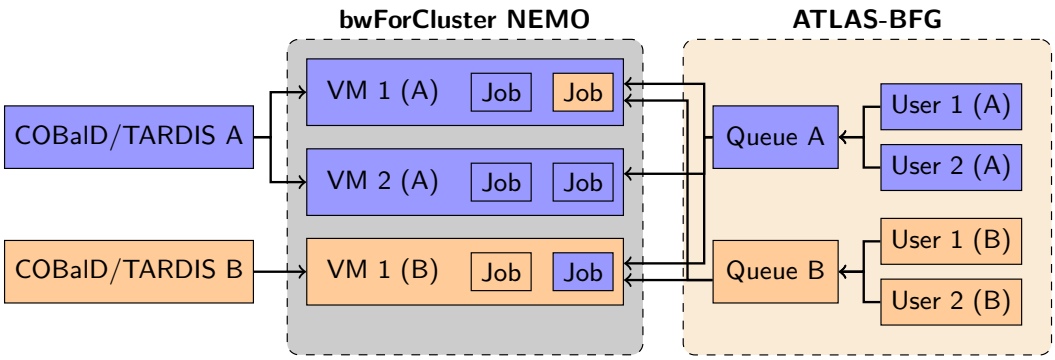

Figure 2: Example of two HEP groups A (blue) and B (orange) sharing resources on NEMO. Both groups have a share on NEMO and use COBalD/TARDIS to start VMs on NEMO. On the ATLAS-BFG cluster, users from both groups submit their jobs to separate queues. Because the VMs are set up to be shared between groups, jobs from users in group A can be assigned to a VM started by group B, and vice versa.

2 The AUDITOR accounting ecosystem

The AUDITOR ecosystem comprises three types of components, as shown in Figure 3. The *core component* stores *records* in a PostgreSQL [9] database. *Collectors* gather data and send it as records to the core component, while *plugins* perform specific actions based on the stored information. Both collectors and plugins interact with the core component through a REST API. To enable the extension of the AUDITOR ecosystem, client libraries that abstract the REST API are available for both the Rust [10] and Python [11] programming languages. The Python client library is a lightweight Python layer built on top of the Rust client library.

2.1 The AUDITOR core component

The core component of AUDITOR is written in the Rust programming language and stores all data in a PostgreSQL database. It is designed to be stateless, making it more resilient against data loss and suitable for high-availability setups with multiple instances distributed behind a load balancer. The core component is available as an RPM package and a Docker container for easy installation.

2.2 Collectors

The collectors are responsible for gathering relevant accounting data from various sources and transmitting it to the core component in the form of records. Four different collectors are available: the TARDIS Collector, the Slurm Collector, the Slurm Epilog Collector, and the HTCondor Collector.

As stated in the introduction, COBalD/TARDIS manages VMs in an UBS. These VMs are referred to as *drones* in the context of COBalD/TARDIS. Each drone can be in one of several states, such as *booting*, *running*, or *stopped*. TARDIS tracks transitions between these states and forwards each transition event to a plugin interface. The TARDIS Collector connects to this plugin interface. Because only state changes are transmitted, a complete record can only be created after the drone has been terminated. To avoid the collector having to track each drone from creation to termination, the TARDIS Collector first sends an incomplete record of a drone to the core component and updates the record later with the stop time of the drone once the drone is terminated. This simplifies the collector but increases the number of interactions with AUDITOR, as each record requires creation and updating. The collector is distributed with the TARDIS [7] software.

The Slurm Collector and Slurm Epilog Collector can be used to collect accounting information from a Slurm batch system. The Slurm Epilog Collector utilizes the epilog functionality of the Slurm scheduler to execute scripts at the end of each batch job. It collects job information using the Slurm command line interface (CLI) and sends it to the core component. However, its usage may affect job runtime due to processing delays. Additionally, the information available from Slurm at the time of the epilog may be limited and may not cover

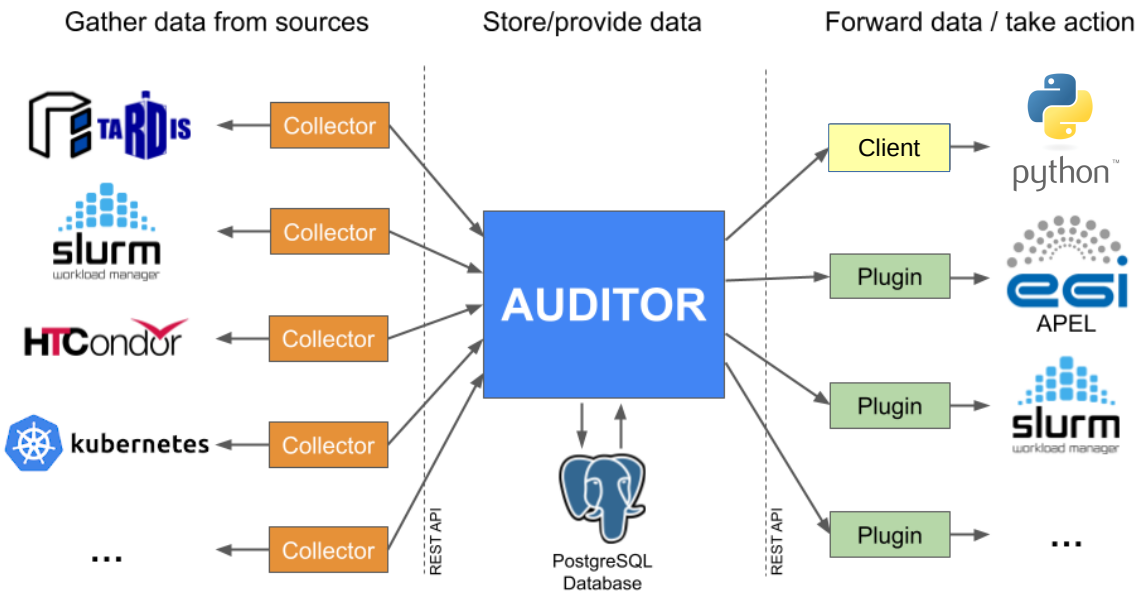

Figure 3: Overview of the AUDITOR ecosystem. AUDITOR accepts records from collectors and stores them in a PostgreSQL database. It grants access to these records to the plugins, which perform actions based on the stored information.

all use cases. To address these issues, the Slurm Collector operates independently and periodically queries the Slurm accounting database using the Slurm CLI. The records are then transmitted to the core component at regular, configurable time intervals. Both collectors are compiled into portable, statically linked binary files with no system dependencies. They are highly configurable and require only the installation of the Slurm client software.

The HTCondor Collector provides functionality similar to that of the Slurm Collector but for the HTCondor [12] scheduler. It queries accountable data from HTCondor using the HTCondor CLI. The key difference between the HTCondor Collector and the Slurm Collector is that the former is written in Python and therefore requires a compatible Python distribution on the host where it is running.

The Kubernetes collector gets its info from two places: the Kubernetes API and a Prometheus instance. This is a necessary process since Kubernetes does not provide its resource metrics, such as CPU time, via its API. Consequently, the collector must possess the capability to access both the API and Prometheus.

2.3 Plugins

Plugins can perform various tasks based on the information stored as records in the AUDITOR database. When two clusters are connected via COBalD/TARDIS, the Priority Plugin can adjust the group priority in the OBS based on the resources provided in the UBS. The APEL Accounting Plugin forwards the accounting information gathered in AUDITOR to the EGI APEL accounting platform [13]. Another potential use case is a utilization reporting tool that analyzes requested versus the consumed resources per user job and provides a weekly overview of potential savings and corresponding CO_2 footprint to raise user awareness. Please note that the last plugin is not yet available at the time of writing. The following section presents the priority plugin in detail.

3 Fair sharing of resources between clusters

As described in the introduction, resources from the NEMO cluster are opportunistically integrated into the ATLAS-BFG cluster using COBalD/TARDIS and shared among the four HEP groups. To ensure fair allocation

of resources among the these groups, the job priority for the groups on ATLAS-BFG should correspond to the amount of resources provided on NEMO. This can be achieved using a combination of the TARDIS Collector, AUDITOR, and the priority plugin. The TARDIS Collector gathers accounting data from COBalD/TARDIS and stores it in the AUDITOR database. The Priority Plugin calculates the priority for each group based on the resources allocated on NEMO by COBalD/TARDIS and adjusts the priority in the Slurm scheduler on ATLAS-BFG accordingly.

To calculate the priority, the plugin first computes the resources provided by each group $i \in [A, \ldots, D]$ in vCore hours during the previous 14 days, denoted as $c_i(t)$:

$$c_i = \int_{t_{\text{now}}-14\,\text{d}}^{t_{\text{now}}} N_i(t)\,\text{d}t\,. \tag{1}$$

Here, $N_i(t)$ represents the number of vCores that group i provided at time t. The priority p_i for group i is then defined as the ratio of resources provided by a single group c_i to the total amount of vCore hours by all HEP groups. The fraction is then scaled to the range between the minimum and maximum priority values, p_{min} and p_{max}:

$$p_i = \frac{c_i}{\sum_j c_j} \cdot (p_{\text{max}} - p_{\text{min}}) + p_{\text{min}}\,. \tag{2}$$

The group priorities p_i are adjusted hourly on the OBS, which in this case is the Slurm scheduler. The minimum and maximum priority values for Slurm have been chosen as $p_{\text{min}} = 1$ and $p_{\text{max}} = 2^{16} - 1 = 65535$, respectively.

Figures 4 and 5 show the integrated vCore hours calculated according to Eq. 1 and the updated group priorities determined using Eq. 2, respectively. It is evident that Group A provides fewer resources than the other groups, which is reflected in their lower priority. As a result, members of Group A experience significantly longer waiting times for job submissions.

The Priority Plugin makes the data from Figures 4 and 5 available through a Prometheus [14] exporter. When connected to a Grafana [15] instance, the data can be visualized in real time.

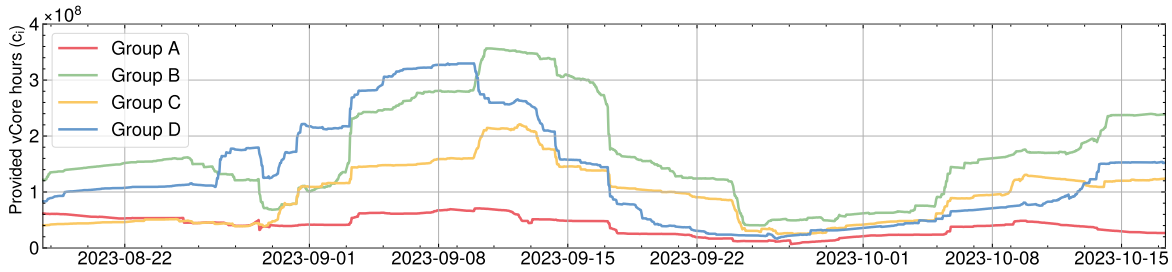

Figure 4: Integral over the vCore hours $c_i(t)$ of the previous 14 days per group i according to Eq. 1 recorded on the NEMO cluster in the period August - Oktober 2023.

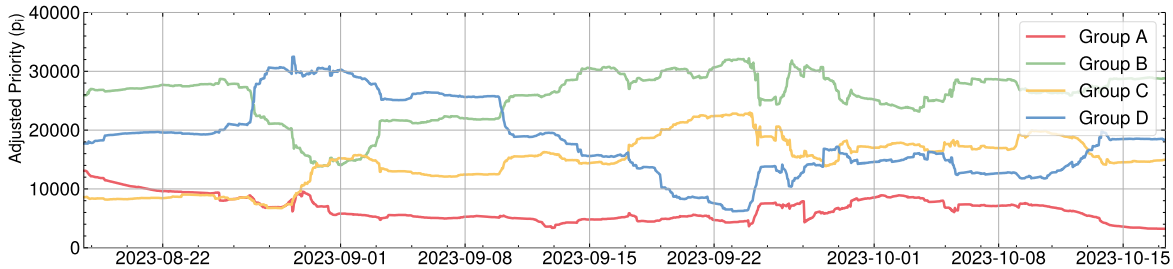

Figure 5: Calculated priority $p_i(t)$ per group i using Eq. 2 based on resources used on NEMO.

4 Conclusions

AUDITOR is a modular accounting ecosystem designed with flexibility in mind. Its REST interface and client libraries in Rust and Python enable a broad community of users to contribute to the expansion of the ecosystem and to implement their own collectors and plugins quickly and easily. The existing collectors and plugins can be combined in various ways to implement different use cases.

Resources of the NEMO cluster have been integrated into the ATLAS-BFG cluster using COBaID/TARDIS in Freiburg for several years. To ensure fair resource sharing among the four HEP groups in Freiburg, AUDITOR, together with the TARDIS Collector and the Priority Plugin, is utilized to adjust the job priority on ATLAS-BFG based on the resources provided on NEMO.

5 Acknowledgements

This work was supported by the Federal Ministry of Education and Research (BMBF) within the project 05H21VFRC2 "Entwicklung, Integration und Optimierung von digitalen Infrastrukturen für ErUM" in the context of the collaborative research centre "Föderierte Digitale Infrastrukturen für die Erforschung von Universum und Materie (FIDIUM)".

The HPC-cluster NEMO in Freiburg is supported by the Ministry of Science, Research and the Arts Baden-Württemberg through the bwHPC grant and by the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG.

References

[1] L. Evans and P. Bryant, *LHC Machine*, JINST 3 (2008) S08001

[2] ATLAS Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, JINST 3 (2008) S08003

[3] I. Bird, K. Bos, N. Brook, et al., *LHC computing Grid: Technical design report*, CERN-LHCC-2005-024

[4] A. Yoo, M. Jette and M. Grondona, *SLURM: Simple Linux Utility for Resource Management*, Job Scheduling Strategies For Parallel Processing, 44-60 (2003)

[5] Adaptive Computing, Inc., *Introduction to Cloud for HPC.*, White Paper (2011)

[6] M. Fischer, E. Kuehn, M. Giffels, et al., *Lightweight dynamic integration of opportunistic resources*, EPJ Web of Conferences **245**, 07040 (2020)

[7] M. Fischer, M. Giffels, A. Haas, et al., *Effective Dynamic Integration and Utilization of Heterogenous Compute Resources*, EPJ Web of Conferences **245**, 07038 (2020)

[8] Boehler, M., von Cube, R., Fischer, M., et al., AUDITOR: Accounting data handling toolbox for opportunistic resources. *The European Physical Journal C*. **85** (2025,3)

[9] PostgreSQL Global Development Group, *PostgreSQL*, https://www.postgresql.org, accessed 8th February 2024

[10] N. Matsakis and F. Klock, *The Rust language*, ACM SIGAda Ada Letters **34**, 103-104 (2014)

[11] G. Van Rossum and F. Drake, *Python 3 Reference Manual*, CreateSpace (2009)

[12] D. Thain, T. Tannenbaum and M. Livny, *Distributed computing in practice: the Condor experience.*, Concurrency and Computation: Practice and Experience **17**, 323-356 (2005)

[13] M. Jiang, C. Del Cano Novales, G. Mathieu, J. Casson, et al., *An APEL Tool Based CPU Usage Accounting Infrastructure for Large Scale Computing Grids*, Data Driven e-Science, 175-186, Springer (2011)

[14] B. Rabenstein and J. Volz, *Prometheus: A Next-Generation Monitoring System (Talk)*, USENIX Association, 2015

[15] Grafana Labs, *Grafana*, https://grafana.com, accessed 9th February 2024