# AUDITOR: Accounting for opportunistic resources

*Michael* Boehler[1,*], *Anton J.* Gamel[1], *Stefan* Kroboth[1], *Benjamin* Rottler[1], *Dirk* Sammel[1], and *Markus* Schumacher[1]

[1]Albert-Ludwigs-Universität Freiburg, Physikalisches Institut, Hermann-Herder-Str. 3, 79104 Freiburg, Germany

**Abstract.** The increasing computational demand in High Energy Physics (HEP) as well as increasing concerns about energy efficiency in high-performance/high-throughput computing are driving forces in the search for more efficient ways to utilise available resources. Since avoiding idle resources is key in achieving high efficiency, an appropriate measure is sharing of idle resources of underutilised sites with fully occupied sites. The software COBalD/TARDIS can automatically, transparently, and dynamically (dis)integrate such resources in an opportunistic manner. Sharing resources however also requires accounting.

In this work we introduce AUDITOR (AccoUnting DatahandlIng Toolbox for Opportunistic Resources), a flexible and extensible accounting system that is able to cover a wide range of use cases and infrastructures. AUDITOR gathers accounting data via so-called collectors which are designed to monitor batch systems, COBalD/TARDIS, cloud schedulers, or other sources of information. The data is stored in a database and provided to so-called plugins, which act based on accounting records. An action could for instance be creating a bill of utilised resources, computing the $CO_2$ footprint, adjusting parameters of a service, or forwarding accounting information to other accounting systems. Depending on the use case, a suitable collector and plugin can be chosen from a growing ecosystem of collectors and plugins. Libraries for interacting with AUDITOR are provided to facilitate the development of collectors and plugins by the community.

## 1 Introduction

In the current discussion about more sustainability and the demand for maximum energy efficiency, approaches for opportunistic resource integration of underutilised computing clusters into computing clusters with continuous computing tasks are becoming increasingly attractive. A combination of the COBalD [1] and TARDIS [2] software tools has proven extremely useful for dynamically integrating and returning opportunistic resources. It is capable of spawning so-called drones in a batch system that provides opportunistic resources. This batch system serves as an underlying batch system (UBS). The resources of the drones are then made available on another cluster, the so-called overlay batch system (OBS). Based on the use of these resources, the number of drones is dynamically adjusted. If more resources are available and the drones are utilised beyond a certain threshold, more drones are launched.

*e-mail: michael.boehler@physik.uni-freiburg.de

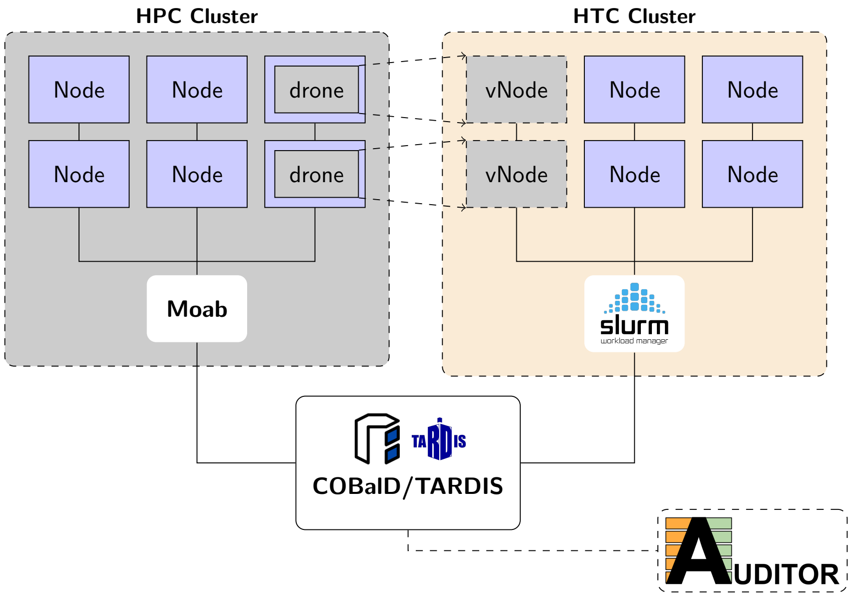

**Figure 1.** Example of two clusters connected via COBalD/TARDIS. The HPC cluster servers as underlying batch system (UBS) that provides resources for the overlay batch system (OBS), the HTC cluster as virtual machines (vNodes). COBalD/TARDIS monitors drone utilisation and dynamically increases/decreases the number of drones. AUDITOR takes care of the proper accounting of the resources used.

Otherwise, no new drones are launched and the resources are returned to the UBS. Figure 1 shows an example of two clusters connected via COBalD/TARDIS [3]. Here, the UBS is an HPC cluster that provides additional resources for the OBS, the HTC cluster. The HPC cluster is operated with the MOAB [4] scheduler and the HTC cluster with Slurm [5]. Since Slurm is a traditional batch system and can only handle resources with IP addresses, the drones contain virtual machines (vNodes) that connect to Slurm as soon as they are operational. They then appear as additional vNodes in the OBS. If several workgroups provide resources from the HPC cluster to the HTC cluster and use resources from both clusters, there could be unequal sharing if one group uses more resources than it provides. To prevent this, the resources provided by each group could be measured using an independent accounting instance and the group priority in the OBS could be adjusted accordingly. Another use case would be HTC clusters that contribute to the worldwide LHC computing grid (WLCG). These must fulfil certain requirements: A dedicated compute element must be deployed and a specific queue must be registered in the WLCG workflow management system. The method of dynamically integrating available resources into an existing WLCG computing cluster avoids the deployment of the additional middleware components. However, the setup described above does not adequately consider the accounting of the opportunistic resources. An appropriate accounting system is needed that allows for separate accounting based on the resources provided. The approach described in this paper, an open source software AUDITOR (**A**ccoUnting **D**ata HandlIng **T**oolbox for **O**pportunistic **R**esources), will accommodate very different infrastructures and many potential use cases. It is therefore designed and implemented in a highly modular, flexible and easily extensible architecture.

## 2 The AUDITOR Accounting Ecosystem

The AUDITOR ecosystem consists of three different types of components: the core component, which stores so-called records in a PostgreSQL [7] database, the collectors, which gather data from data sources and store it as a record, and the plugins, which perform certain actions based on the stored information (see Fig. 2). Both the collectors and the plugins can store and retrieve data from AUDITOR via a REST API. To facilitate the extension of the AUDITOR ecosystem, client libraries are provided in two programming languages, both Rust [8] and Python [9]. The Python client library is a lightweight Python layer on top of the Rust client library, implemented with the PyO3 library [10].

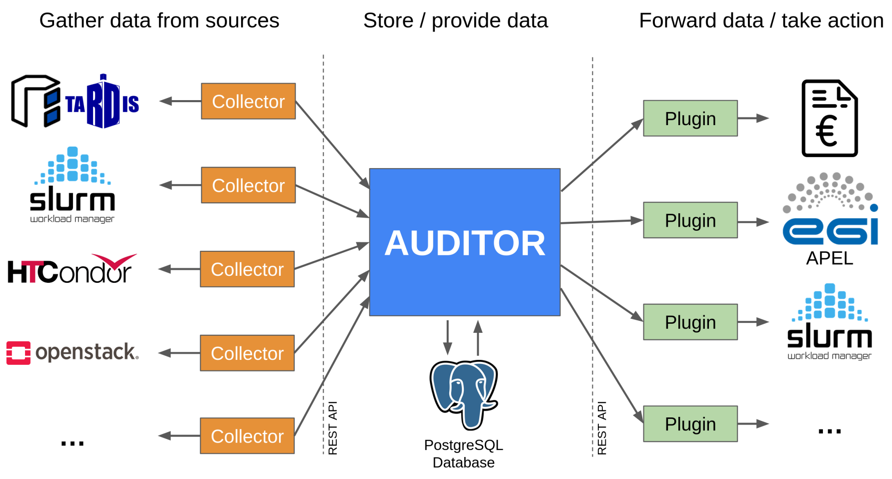

**Figure 2.** Overview of the AUDITOR ecosystem. AUDITOR accepts records from collectors and stores them in a PostgreSQL database. It grants access to these records to the plugins, which can perform actions based on the stored information.

### 2.1 The AUDITOR Core Component

The core component of AUDITOR is written in the programming language Rust. Rust is an open source, multi-paradigm system programming language designed to be secure, concurrent, and practical. Security refers in particular to the avoidance of programming errors that lead to memory access errors or buffer overflows and thus potentially to security vulnerabilities, especially in concurrent processes. AUDITOR stores all its data in a PostgreSQL database. The connection to the PostgreSQL database is established via the sqlx library [11], which enables both compile-time SQL query verification and SQL table migration. Data is accessed via a REST API implemented using the ACTIX WEB library [12]. Because the core component itself is stateless, it is quite robust against data loss and is ideal for use in a high-availability setup with multiple components behind a load balancer. To ensure a quick and easy installation, AUDITOR is provided via a CI pipeline both in RPM package manager format and as a Docker container [13].

A record represents a single accountable unit in AUDITOR. It consists of information such as: the *record_id*, which is the unique identifier.; the *meta* field, which can be used to provide the context of a record. This is a HashMap where a key (string) is mapped to an array of strings. Figure 3 shows an example record: there the group, site, and user ids are stored in the *meta* field. Any number of properties, which can be considered in the accounting, can be stored in the *components* field. It consists of at least a *name* and an *amount* (what kind of resources and how much of them should be considered for accounting). Since a typical computing job uses CPU and memory resources, both *components* can be considered for accounting. Multiple *scores* can be assigned to a single *component*. For example the record in Fig. 3 supports both the HEPSPEC06 [14] and the new hepscore23 [15], metrics. The *start_time* and *stop_time* fields store the times from when to when the resources were used. The data received from AUDITOR also contains a *runtime* field. This is the difference between *stop_time* and *start_time*. All dates and times in the database are in UTC.

```
{
    "record_id": "hpc-4126142",
    "meta": {
      "group_id": [ "atlpr"],
      "site_id": [ "hpc" ],
      "user_id": [ "atlpr001"  ]
    },
    "components": [
      {
        "name": "Cores",
        "amount": 8,
        "scores": [
          {
            "name": "HEPSPEC06",
            "value": 10.0
          },
          {
            "name": "HEPScore23",
            "value": 10.0
          }
        ]
      },
      {
        "name": "Memory",
        "amount": 16000,
        "scores": []
      }
    ],
    "start_time": "2023-02-24T00:27:58Z",
    "stop_time": "2023-02-24T03:41:35Z",
    "runtime": 11617
},
```

**Figure 3.** JSON representation of an example record. The meta field can contain the context of the record. The components indicate what kind of resources and how much of them should be considered for accounting.

## 2.2 Collectors

The task of the collectors is to gather the relevant accounting data from various data sources and to transmit them to the AUDITOR REST endpoint in the form of records. Two Slurm collectors and a TARDIS collector are presented here. An HTCondor [16] collector is currently being developed.

To collect the accounting data from a slurm batch system one can use either the *Slurm Epilog Collector* or the *Slurm Collector*. The *Slurm Epilog Collector* is based on the Epilog functionality of Slurm. Any script can be executed automatically at the end of each batch job (either on the head node or on the client node). In the case of the *Slurm Epilog collector*, the Slurm command ‘scontrol show job <jobid>’ is executed for each slurm job and the relevant information is parsed, converted into a record and sent to AUDITOR. The collector is compiled into a portable, statically linked binary with no system dependencies and is extensively configurable, making it well suited to different use cases and environments. The Slurm Epilog collector is a simple but efficient implementation for collecting essential accounting information of Slurm jobs. The fact that the Epilog collector runs in the Epilog as part of the job itself can mean that if there are delays in parsing or transferring the data to AUDITOR, the job runtime and therefore CPU efficiency will be affected. In addition, not all the accounting information of a Slurm job is available during the execution of the Epilog. The *Slurm Collector* has been implemented specifically for use cases for which these limitations of the *Slurm Epilog Collector* are not acceptable.

The *Slurm Collector* runs as a stand-alone tool that regularly queries the Slurm accounting database for new jobs using the command line tool ‘sacct’. A new record is created for each job and placed in a queue, which then transmits the records to AUDITOR at regular, configurable intervals. If the transmission to AUDITOR fails, the record is put back into the queue and the transmission process is repeated at the next interval. To avoid data loss, the contents of the queue are also stored on disk in the form of a SQLite3[17] database. The collector is also compiled into a portable, statically linked binary file that has no system dependencies. The Slurm Collector can also be configured extensively. The only requirement is that the Slurm client software must be installed.

As described in section 1, COBalD/TARDIS manages a fleet of drones that are the opportunistic resources. Each drone can be in one of a finite number of states (e.g. starting, running, stopped, ...). TARDIS only tracks the transition between these states. Each state transition event is also forwarded to a plugin interface of TARDIS, to which the TARDIS collector connects. Since 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 uses the */add* endpoint of AUDITOR to create an incomplete record of a drone, and then the */update* endpoint to add the *stop_time* to the record once the drone has terminated. This reduces the complexity of the collector, but increases the number of interactions between the collector and AUDITOR, as each record must first be created and then updated. This collector is delivered with the TARDIS[18] software.

## 2.3 Plugins

Plugins can perform a wide variety of tasks based on the information stored as records in the AUDITOR database. The first available plugin is the priority plugin, which is able to manipulate the group priority in the scheduler of the overlay batch system based on the provided resources in the underlying batch system when two clusters are connected via COBalD/TARDIS. The AUDITOR-APEL-plugin is able to forward the accounting information gathered in AUDITOR to the EGI APEL accounting platform. Another possible use case is a utilisation reporting tool that analyses requested versus the consumed resources per user job and provides a weekly overview with potential savings and corresponsing $CO_2$ footprint to raise user awareness. The last two plugins are not yet ready at the time of writing. In the following, the priority plugin is presented in detail.

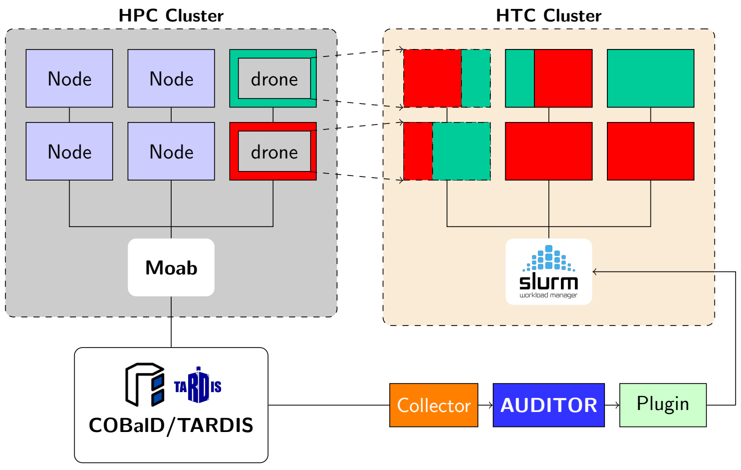

**Figure 4.** The two groups Red and Green provide resources of the HPC cluster to the HTC cluster. The priority plug-in ensures that the job priority in the HTC cluster is adjusted according to the amount of resources provided by each group.

## 3 Example Use Case - Adapting the Priority based on provided Resources

The opportunistic use of HPC resources has been implemented at the University of Freiburg[19] as follows: Four working groups of High Energy Physics (HEP) have access to the local HPC cluster NEMO. Service accounts of each of these work groups can request resources (as drones) on the HPC cluster as needed and make them available to all users of the four HEP groups as shared resource in the Overlay Batch System (OBS). These shared resources are the provided resources.

Since the individual work groups can calculate directly on the HPC cluster or opportunistically via the OBS, it must be ensured that the priority of the respective work group in the OBS is adapted according to the resources they have made available. The more resources they contribute to the OBS, the greater their priority becomes. This was implemented with a suitable collector, a dedicated AUDITOR instance, and the specially developed priority plugin.

The TARDIS Collector gathers the information about the provided resources per work group from the drones and stores the records in the AUDITOR instance. The priority plug-in fetches the necessary data from AUDITOR and calculates the provided resources $c_i$ in vCore hours of the last 14 days per working group [A,..,D]:

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

where $N_i(t)$ is the number of cores provided for all records associated with group $i \in A, B, C, D$ at a time instance t. Priority ($p_i$) is defined as the ratio of resources provided by a single group ($c_i$) to the total amount of vCore hours provided by all HEP groups ($\sum_j c_j$) multiplied by the difference between $p_{max}$ and $p_{min}$, the minimum and maximum priority values set:

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

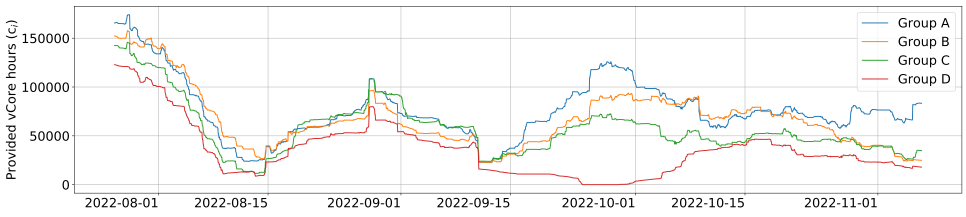

**Figure 5.** Integral over the vcores hours ($c_i(t)$) of the previous 14 days per group (i) according to Eq. 1 recorded on the NEMO cluster in the period from August - November 2022.

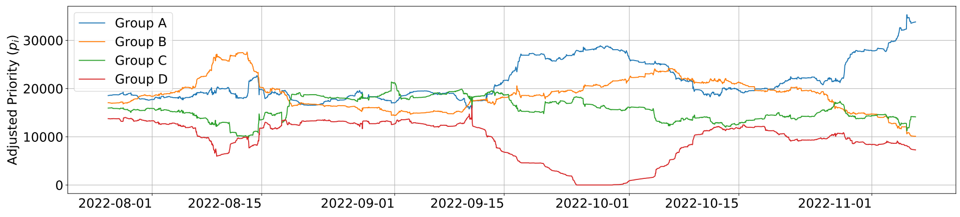

**Figure 6.** Calculated priority ($p_i(t)$) per group ($i$) using equation 2 based on NEMO jobs recorded in the period from August to November 2022.

where $i \in A, B, C, D$. The group priorities $p_i$ are adjusted daily on the OBS, in this use case on the Slurm scheduler. The minimum and maximum priority are set to $p_{min} = 1$ and $p_{max} = 65335$, respectively Figure 4 shows the jobs of two of the four working groups in green and red respectively. Both groups have spawned a drone in the HPC cluster. Group red uses two complete worker nodes in the HTC cluster and the share of another 3 worker nodes with group green Although the HPC cluster has a *single user per node* policy, the virtual nodes in the OBS can be used as shared resources. This increases the CPU efficiency, as the tasks of individual users would often not fill an entire node in the HPC cluster. Figure 5 and figure 6 show the resources allocated on the HPC cluster in vCore hours and the new group priorities calculated according to Eq. 2. It can be seen that at the beginning of October 2022, group D did not provide any resources on the HPC cluster, so its group priority was minimised. Since the Slurm scheduler also takes waiting time and other criteria into account, group D was not completely banned from the OBS, but the drastically reduced group priority resulted in a significantly longer waiting time for jobs submitted by members of group D.

## 4 Conclusion

AUDITOR is a highly modular and flexible accounting ecosystem. Its REST interface and client libraries in Rust and Python allow 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 any way to implement various use cases. If a particular use case cannot be implemented with the existing components, it can easily be extended to cover any other use case in the context of job accounting in distributed computing systems.

## 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] Fischer, M., Kuehn, E., Giffels, M., et al. MatterMiners/cobald: v0.13.0. (Zenodo,2022,8), https://zenodo.org/record/7032186

[2] Giffels, M., Fischer, M., Haas, A., et al. MatterMiners/tardis: The Escape. (Zenodo,2023,2), https://zenodo.org/record/7032186

[3] Böhler, M., Caspart, R., Fischer, M., et al. Transparent Integration of Opportunistic Resources into the WLCG Compute Infrastructure. *EPJ Web Conf.*. **251** pp. 02039 (2021)

[4] Adaptive Computing Moab workload manager. (2023), https://adaptivecomputing.com/moab-hpc-suite. Accessed 08 Aug 2023

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

[6] Bos, K., Brook, N., Duellmann, et al. LHC computing Grid: Technical Design Report. Version 1.06 (20 Jun 2005). (CERN,2005)

[7] PostgreSQL Global Development Group PostgreSQL. (2021), https://www.postgresql.org. Accessed 08 Aug 2023

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

[9] Van Rossum, G. & Drake, F. Python 3 Reference Manual. (CreateSpace,2009)

[10] PyO3 project and contributors PyO3. (2023), https://pyo3.rs. Accessed 08 aug 2023

[11] The LaunchBadge team sqlx: The Rust SQL Toolkit. (2023), https://github.com/launchbadge/sqlx. Accessed 08 aug 2023

[12] The Actix team actix-web. (2023), https://actix.rs. Accessed 08 Aug 2023

[13] Merkel, D. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*. **2014**, 2 (2014)

[14] Michelotto, M., Alef, M., Iribarren, et al. A comparison of HEP code with SPEC benchmarks on multi-core worker nodes. *Journal Of Physics: Conference Series*. **219**, 052009 (2010,4)

[15] Giordano, D., Alef, M., Atzori, L., et al. HEPiX Benchmarking Solution for WLCG Computing Resources. *Computing And Software For Big Science*. **5** (2021,12),

[16] The HTCondor developers HTCondor software suite. (2023), https://htcondor.org. Accessed 08 Aug 2023

[17] SQLite Consortium SQLite. (2023), https://sqlite.org. Accessed 08 Aug 2023

[18] Giffels, M., Fischer, M., Haas, A., et al. Auditor accounting Plugin in TARDIS. (2023), https://cobald-tardis.readthedocs.io/en/latest/plugins/plugins.html. Accessed 08 aug 2023

[19] Kroboth, S., Böhler, M., Gamel, A., Rottler, B. & Schumacher, M. Opportunistic extension of a local compute cluster with NEMO resources for HEP workflows. *Proceedings Of The 7th BwHPC Symposium*. **7** pp. 43-48 (2022)