Accounting of Computing Resources with AUDITOR

Michael Boehler[1,*], *Ralf Florian* von Cube[2], *Max* Fischer[2], *Oliver* Freyermuth[3], *Manuel* Giffels[2], *Michael* Huebner[3], *Raphael* Kleinemuehle[4], *Benjamin* Rottler[1], *Dirk* Sammel[1], *Matthias* Schnepf[2], *Markus* Schumacher[1], and *Raghuvar* Vijayakumar[1]

[1]Albert-Ludwigs-Universität Freiburg, Freiburg, Germany
[2]Karlsruher Institut für Technologie, Karlsruhe, Germany
[3]Rheinische Friedrich-Wilhelms-Universität Bonn, Bonn, Germany
[4]Bergische Universität Wuppertal, Wuppertal, Germany

Abstract. New strategies for the provisioning of compute resources, e.g. in the form of dynamically integrated resources enabled by the COBalD/TARDIS software toolkit, require a new approach of collecting accounting data. AUDITOR, a flexible and expandable accounting ecosystem that can cover a wide range of use cases and infrastructures, has been developed specifically for this purpose. Accounting data are collected via so-called collectors and stored in a database. So-called plugins can access the data and act based on the accounting information. Access to the data is handled by the core component of AUDITOR, which provides a REST API together with a Rust and a Python client library.
An HTCondor collector, a Slurm collector and a TARDIS collector are currently available, and a Kubernetes collector is already in the works. The APEL plugin enables, for example, the creation of APEL accounting summaries and their transmission to the APEL accounting server. Although the original aim for the development of AUDITOR was to enable the accounting of opportunistic resources managed by COBalD/TARDIS, it can also be used for standard accounting of a WLCG computing resource. As AUDITOR uses a highly flexible data structure to store accounting data, extensions such as GPU resource accounting can be added with minimal effort.
This contribution provides insights into the design of AUDITOR and shows how it can be used to enable a number of different use cases.

1 Introduction

The increasing complexity of modern compute resource provisioning, especially through dynamically integrated systems like those enabled by the COBalD/TARDIS [1, 2] software toolkit, calls for innovative approaches to provide accurate accounting data. To address this need, the **A**ccou**n**ting **D**atahandl**I**ng **T**oolbox for **O**pportunistic **R**esources [3] (AUDITOR) ecosystem was developed. AUDITOR offers a flexible, expandable framework that can handle a broad spectrum of use cases and infrastructures. AUDITOR has proven to be well-suited for established WLCG [4] computing resource accounting. Its highly adaptable data structure allows for easy extension of stored accounting metrics, such as the inclusion of GPU resource

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

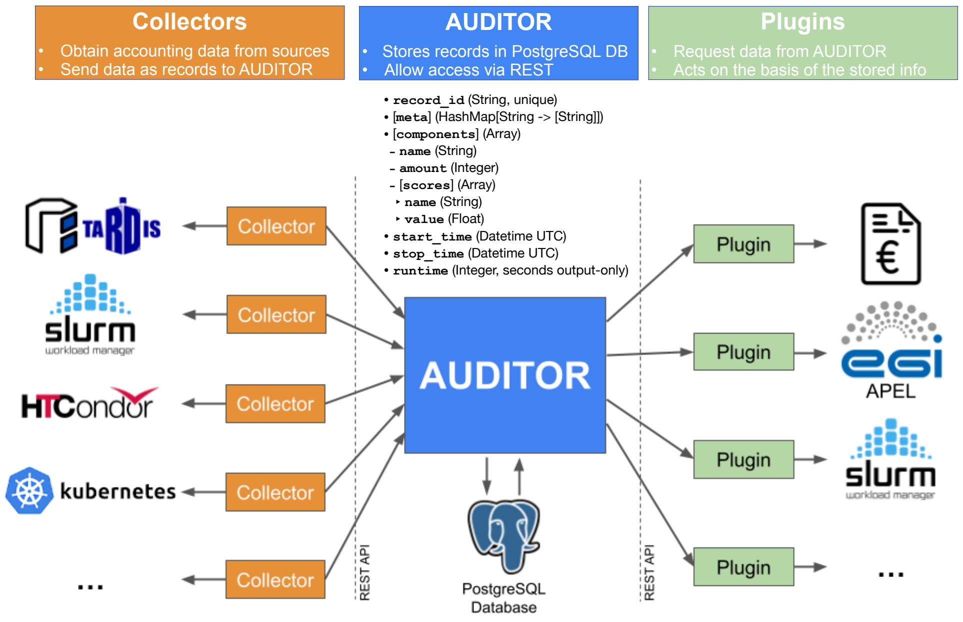


Figure 1. Overview of the AUDITOR accounting ecosystem. AUDITOR accepts records from collectors (left column) and stores them in a PostgreSQL database. It grants access to these records to the plugins (right column), which can perform actions based on the stored information. The middle column also lists the fields of an accounting records.

accounting, making it a versatile tool for a range of scenarios. The first productive AUDITOR use case was to control the scheduling priority of a group based on the proportion of compute resources that this group opportunistically provided [5]. This article highlights key aspects of AUDITOR and demonstrates its wide range of applications using three different use cases, emphasising its flexibility and ease of integration.

2 The AUDITOR Accounting Ecosystem

The AUDITOR accounting ecosystem is designed to manage and track the accounting data of heterogeneous computing resources. It consists of three main components: collectors, core component, and plugins. These components work together to gather, store, and process accounting data, facilitating resource usage tracking and reporting in distributed computing environments.

2.1 Core component

The core component of AUDITOR is built using the Rust [6] programming language, known for its security, concurrency, and reliability. It interacts with a PostgreSQL database to store data and uses the ACTIX WEB library to provide a REST API for data access. The core system is designed to be stateless, ensuring robustness and ease of scalability, making it suitable for high-availability setups. Data is stored as records, which represent individual accounting units containing metadata and resource usage details. Each record has fields such as *record_id, meta*, *components*, and *time* (start and stop times). A record can have several components. Each component has a *name* and an *amount* which specifies the type of resource and how much has been allocated by a particular job, e.g. core and 8, and an attached list of *scores*. The scores contain a *name* and a *value*, e.g. HEPScore23 [7] and 10.0, which gives the name of a particular performance metric and the corresponding value. This value is then used to calculate normalised performance metrics. The structure of a record is shown in the centre of Fig. 1.

AUDITOR is available as both an RPM and a Docker container for easy installation.

In order to cover a wide range of use cases and to involve as many experts as possible, a Python interface (python-auditor) is also provided, which allows to develop collectors and plug-ins directly in the widely used programming language Python. Python-auditor also makes it possible to import auditor records directly into Python or Jupyter notebooks and process them interactively into presentable plots.

A Prometheus [8] exporter is also available to enable real-time monitoring of accounting data, making it easier to track system performance and resource utilisation.

2.2 Collectors

Collectors are responsible for gathering accounting data from various sources and sending it to AUDITOR. These include:

Slurm Epilog Collector: This collector works with the Slurm [9] job scheduler by automatically executing a script at the end of a job (via the Epilog functionality). It parses job information, converts it to a record, and sends it to AUDITOR. This implementation is very lean and yet configurable. However, depending on the accounting information required, the information may not yet be available at the time the epilog runs. In this case, the Slurm Collector must be used.

Slurm Collector: While the Slurm Epilog Collector is triggered as a script by the batch job itself, the Slurm Collector is an independent service that regularly queries the Slurm accounting database with the `sacct` command. If transmission fails, the record is reattempted in the next cycle, ensuring reliability. Data is also stored in an SQLite3 database to prevent loss.

HTCondor Collector: This collector efficiently tails the job history in an HTCondor cluster. It can be used both for an HTCondor on a local as well as many remote machines. Due to the same architecture, it also supports the HTCondor-CE [10] for grid operations.

TARDIS Collector: For tracking drones managed by COBalD/TARDIS, the TARDIS Collector records state transitions of drones (e.g. starting, running, stopped). It interacts with AUDITOR's REST API by first creating an incomplete record and updating it when the drone terminates. The TARDIS collector is provided in the software repository of the TARDIS project.

Kubernetes Collector: This collector retrieves information from two sources: the Kubernetes API and a Prometheus instance. This is necessary because Kubernetes does not make its resource metrics, like CPU time, available via its API. This means that the collector must be able to access both the API and Prometheus.

2.3 Plugins

Plugins in AUDITOR perform various actions based on the accounting data stored in the system. The current and planned plugins are the following:

Priority Plugin: This plugin adjusts the group priority in the scheduler of an overlay batch system based on the resource usage data from connected clusters via COBalD/TARDIS.

https://doi.org/10.1051/epjconf/202533701320

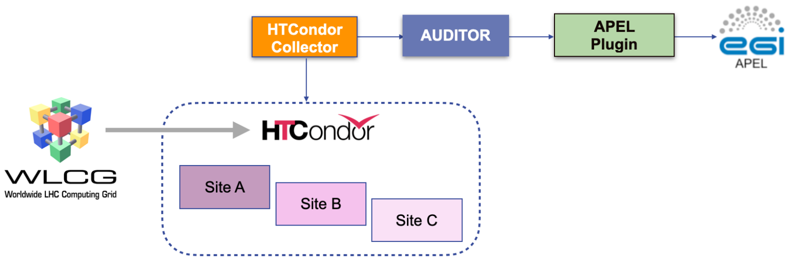


Figure 2. AUDITOR accounting pipeline for use cases that connect computing resources to an HTCondor batch system. Site A,B, and C provide HTCondor resources as drones via COBalD/TARDIS. Jobs are submitted to the batch system via an HTCondor CE. AUDITOR uses the HTCondor collector to retrieve the accounting data from the HTCondor overlay batch system and stores it as records in the AUDITOR database. The APEL plug-in creates reports and sends them to the APEL accounting server.

AUDITOR-APEL Plugin: This plugin retrieves the records from AUDITOR, formats the accounting data accordingly, and forwards it to the Accounting Processor for Event Logs [11] (APEL) platform operated by the European Grid Infrastructure (EGI).

Utilization Reporting Plugin: A future plugin that will analyse resource consumption of local users and will report on discrepancies between requested and actual usage, helping raise awareness of potential savings and corresponding CO_2 footprints.

3 Selected Accounting Use Cases

As described in section 2, AUDITOR was originally developed to account for (opportunistic) resources that are dynamically and transparently integrated into another cluster. In addition, AUDITOR was designed to cover as many use cases as possible and to be easily expandable, so that use cases not yet covered can be implemented with AUDITOR with little effort. This section provides an overview of some selected use cases.

3.1 Account resources behind one CE for different providers

The first important use case to be described here is the original idea for the development of AUDITOR. Several computing sites A,B, and C (see Fig. 2) have available computing resources and offer them to an overlay cluster that contributes to the WLCG. Sites A to C want to be properly accounted for the provided computing time. Before AUDITOR was developed, there was no accounting system that could show the individual share of sub-clusters A, B and C.

In this particular use case, an HTCondor Compute Element [12] (CE) is able to receive compute jobs from the WLCG workflow management system. Jobs that are submitted to the CE are executed in containers that have been started as so-called drones by COBalD/TARDIS on the batch systems at sites A, B, and C. It is completely transparent to the user where the jobs are ultimately processed. AUDITOR obtains the accounting data from the higher-level batch system, in this case HTCondor [13], via the so-called HTCondor collector. The configuration of the HTCondor collector contains a section in which node names or IP addresses can be mapped to a specific sub-cluster (site A,B, and C). The accounting data for each job is saved as a record in the AUDITOR database. The APEL plugin must be used to report the individual resources of the sub-clusters within the WLCG. The APEL plugin creates reports

Table 1. Accounting data from the EGI accounting portal, selected for the resource centre UNI-Bonn in HEPScore23 hours. Since the overlay batchsystem HTCondor and the corresponding CEs (cloud-htcondor-ce-1 and cloud-htcondor-ce-2) are operated at KIT in Karlsruhe, the submit host attached to the UNI-BONN resources have a domain name from GridKa at KIT.

SubmitHost	2023	2024	Total
cloud-htcondor-ce-1-kit.gridka.de	1106259	0	1106259
cloud-htcondor-ce-2-kit.gridka.de	61173	176411	237584
Total	1167432	176411	1343843

and sends them to the APEL server. The plugin can either create summarised reports for the current month or individual job reports. In both cases, sync messages are created for the current month. In the configuration of the APEL plugin, one can specify which sub-clusters site A,B, and C should be reported to the APEL accounting server.

The first use case where this setup has been deployed and used in production are the opportunistic resources, whose CEs are located at KIT, and one of the serving sub-clusters are resources provided by the University of Bonn. The setup with the overlay batch system at KIT and the resources provided by Bonn allows the University of Bonn to contribute to the WLCG without hosting all the infrastructure required for a properly registered WLCG site.

Table 1 shows that the University Bonn has provided almost 1.2 million HEPScore23hours to WLCG in 2023. The corresponding CEs `cloud-htcondor-ce-1-kit.gridka.de` and `cloud-htcondor-ce-2-kit.gridka.de` are operated at GridKa at KIT. In 2024 fewer resources could be provided by the University of Bonn, due to a reorganisation of the local compute center.

3.2 Replace EGI accounting client by AUDITOR pipeline

As described in section 3.1, AUDITOR possesses the capacity to account for combined compute resources in an overlay batch system and to report individual accounting data to the APEL accounting server. Given that all the components required in the AUDITOR accounting pipeline for the aforementioned use case can be utilised in the standard setup of an HTC compute cluster based on an HTCondor CE and an HTCondor batch system, AUDITOR can also be employed for this purpose. The identical pipeline as in section 3.1 must be used here. The minimal AUDITOR accounting pipeline is shown Fig. 3.

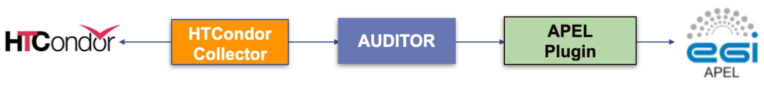


Figure 3. Minimal AUDITOR accounting pipeline to collect accounting data from a HTCondor batch system, store it as records in an AUDITOR instance and report the accounting metrics via the APEL plugin to the APEL server of EGI.

The GridKa site administrators at KIT have installed an HTCondor collector, an AUDITOR instance with a dedicated PostgreSQL database and an APEL plugin on each CE. This means that each CE can be used with an identical node configuration. On each CE, the relevant HTCondor collector retrieves the accounting data from HTCondor for each computing job and stores the corresponding accounting records in AUDITOR. Each APEL plug-in then reports only the accounting data from the CE on which it is installed.

Table 2. Accounting data from the EGI accounting portal, showing the number of processed compute jobs at the WLCG Tier-1 Site of GridKa centre FZK-LCG2 at KIT in Karlsruhe Germany. GridKa started to migrate the entire accounting from APEL client to AUDITOR in May 2024. Since July 2024 accounting is solely reported by AUDITOR. The accounting of the upper 5 submit hosts was transmitted with the APEL client software, the lower 5 hosts with AUDITOR.
https://accounting.egi.eu/egi/site/FZK-LCG2/

SubmitHost	2024 Apr	2024 May	2024 Jun	2024 July
pps-htcondor-ce.gridka.de-condor	1776	0	0	0
htcondor-ce-1-kit.gridka.de-condor	428245	574879	58557	0
htcondor-ce-2-kit.gridka.de-condor	407580	356821	0	0
htcondor-ce-4-kit.gridka.de-condor	375687	38381	0	0
htcondor-ce-3-kit.gridka.de-condor	373856	237105	0	0
pps-htcondor-ce.gridka.de	0	970	1411	1330
htcondor-ce-1-kit.gridka.de	0	0	137814	395526
htcondor-ce-2-kit.gridka.de	0	107959	472298	459624
htcondor-ce-3-kit.gridka.de	0	118096	467672	457001
htcondor-ce-4-kit.gridka.de	0	255491	320958	460155
Total	1607491	1708451	1458710	1773636

Since GridKa is one of the largest WLCG Tier 1 sites and processes almost 2 million compute jobs per month, it can be concluded that AUDITOR scales quite efficiently and can be deployed on any other WLCG site. It is worth mentioning that the database schema has been completely revised and optimised in AUDITOR v0.6.3, improving read performance by a factor of 100. In the time reported here, GridKa has been running the accounting pipeline with AUDITOR v0.5.0 and no latency issues have been observed.

3.3 Collect accounting data from several sources

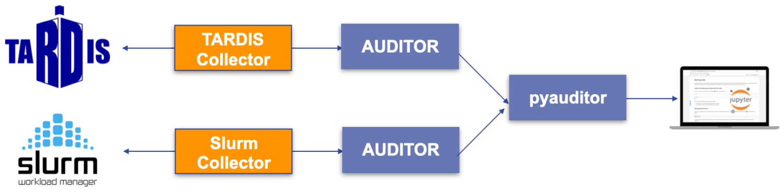


Figure 4. Possible setup with two different sources of accounting data, the TARDIS meta scheduler and the Slurm batch system. Accounting data of both systems are collected with individual collectors, the TARDIS collector and the Slurm collector. Both collectors send the accounting data to individual AUDITOR instances. Another possible scenario would be sending accounting data from both collectors to the same AUDITOR instance. The python-auditor client connects to both AUDITOR instances and allows to combine the accounting data.

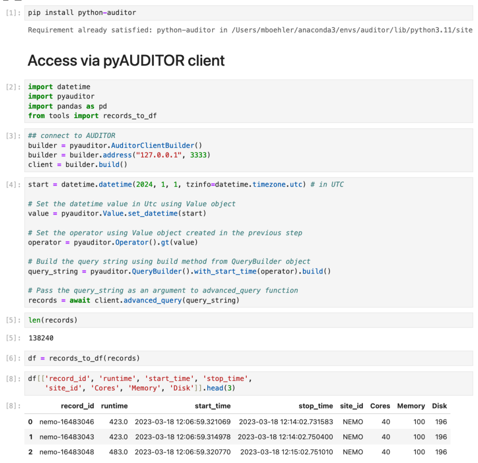
Large compute sites not only operate one compute cluster, but often host a variety of such clusters. The operators or, in particular, the cost bearers are often interested in the utilisation of all systems, for which the accounting data of the various systems must then be evaluated at the end of the year or at the end of the funding period. Since AUDITOR, with its modular structure, is designed precisely to collect accounting data from a wide variety of sources and, with the python client python-auditor, offers easy access to all the data, such an evaluation can then be carried out very easily in python or, for example, interactively in a Jupyter notebook.

Figure 4 shows the setup at the University of Freiburg, where resources of an HPC cluster were integrated into an HTC cluster using COBalD/TARDIS and made available to the local users in a dedicated queue of a Slurm batch system.

In this case, one AUDITOR instance is run to capture the HPC resource accounting data using the TARDIS collector, and another instance is run to capture the resource utilisation by local user jobs managed by the Slurm overlay batch system separately using a Slurm collector. It would have been possible to have both collectors report to a single AUDITOR instance, but this was deliberately not set up here as the entire AUDITOR system was still being trialled at the time of installation.

A short code snippet in Fig. 5 (A) shows how to install the python-auditor client, setup a client connection to an AUDITOR instance, retrieve the records with an advanced query, and finally evaluate and plot the data with widely used python tools from the Pandas and Matplotlib libraries. The resulting plots in Fig. 5 (B) have been created with the snippet shown in (A) with some additional lines of code.

A



```
[1]: pip install python-auditor

     Requirement already satisfied: python-auditor in /Users/mboehler/anaconda3/envs/auditor/lib/python3.11/site
```

Access via pyAUDITOR client

```
[2]: import datetime
     import pyauditor
     import pandas as pd
     from tools import records_to_df

[3]: ## connect to AUDITOR
     builder = pyauditor.AuditorClientBuilder()
     builder = builder.address("127.0.0.1", 3333)
     client = builder.build()

[4]: start = datetime.datetime(2024, 1, 1, tzinfo=datetime.timezone.utc) # in UTC

     # Set the datetime value in Utc using Value object
     value = pyauditor.Value.set_datetime(start)

     # Set the operator using Value object created in the previous step
     operator = pyauditor.Operator().gt(value)

     # Build the query string using build method from QueryBuilder object
     query_string = pyauditor.QueryBuilder().with_start_time(operator).build()

     # Pass the query_string as an argument to advanced_query function
     records = await client.advanced_query(query_string)

[5]: len(records)

[5]: 138240

[6]: df = records_to_df(records)

[8]: df[['record_id', 'runtime', 'start_time', 'stop_time',
        'site_id', 'Cores', 'Memory', 'Disk']].head(3)
```

[8]:

	record_id	runtime	start_time	stop_time	site_id	Cores	Memory	Disk
0	nemo-16483046	423.0	2023-03-18 12:06:59.321069	2023-03-18 12:14:02.731583	NEMO	40	100	196
1	nemo-16483043	423.0	2023-03-18 12:06:59.314978	2023-03-18 12:14:02.750400	NEMO	40	100	196
2	nemo-16483048	483.0	2023-03-18 12:06:59.320770	2023-03-18 12:15:02.751010	NEMO	40	100	196

B

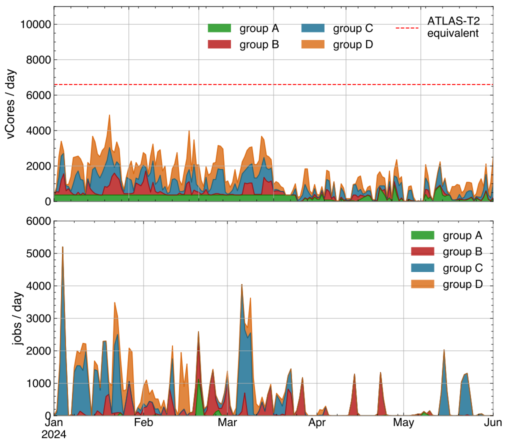


Figure 5. A: Short python code snippet showing how to install, connect and retrieve accounting records from the AUDITOR data base via the python client python-auditor. B: Combined accounting data from two different sources: upper plot shows the integrated vCores from the HPC cluster into the HTC cluster at the University of Freiburg per day, the lower plot shows the number of compute jobs executed in these resources for the same days in 2024.

4 Conclsions and Outlook

The AUDITOR ecosystem is a flexible and extensible framework that provides detailed resource usage tracking and integrates with various data sources, allowing for enhanced accountability and reporting in distributed systems. The core system, collectors, and plugins interact seamlessly via a REST API, with the ecosystem designed for high scalability, reliability, and ease of extension.

The three use cases have shown that AUDITOR allows resources that have been dynamically aggregated into an overlay batch system to be accounted individually, that AUDITOR scales to be used as a accounting system for one of the largest WLCG Tier 1 centres GridKa with almost 2 million compute jobs per month, and that it can be used to analyse accounting details from different compute clusters together.

Acknowledgments

This work was supported by the Federal Ministry of Education and Research (BMBF) within the joint project “Federated Digital Infrastructures for Research on Universe and Matter” (FIDIUM) and its subprojects (05H21VFRC2, 05H21VKRC2 and 05H21PXRC2). 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] M. Fischer et al., MatterMiners/cobald: v0.14.0 (2023), https://zenodo.org/record/1887872
[2] M. Giffels et al., MatterMiners/tardis: 0.8.2 (2024), https://zenodo.org/doi/10.5281/zenodo.2240605
[3] M. Boehler et al., The accounting ecosystem AUDITOR (2024), https://zenodo.org/doi/10.5281/zenodo.12653483
[4] K. Bos et al., LHC computing Grid: Technical Design Report. Version 1.06 (20 Jun 2005) (2005), http://cds.cern.ch/record/840543
[5] M. Boehler et al., Auditor: Accounting for opportunistic resources, EPJ Web of Conferences **295**, 04008 (2024). 10.1051/epjconf/202429504008
[6] N.D. Matsakis, F.S. Klock, The rust language, ACM SIGAda Ada Letters **34**, 103 (2014). 10.1145/2692956.2663188
[7] D. Giordano et al., HEPScore: A new CPU benchmark for the WLCG (2024), http://dx.doi.org/10.1051/epjconf/202429507024
[8] B. Rabenstein, J. Volz, Prometheus: A Next-Generation monitoring system (talk) (2015)
[9] A.B. Yoo et al., SLURM: Simple Linux Utility for Resource Management (Springer Berlin Heidelberg, 2003), p. 44–60, ISBN 9783540397274, http://dx.doi.org/10.1007/10968987_3
[10] B. Bockelman et al., Principles, technologies, and time: The translational journey of the HTCondor-CE, Journal of Computational Science **52**, 101213 (2021). 10.1016/j.jocs.2020.101213
[11] M. Jiang et al., An APEL Tool Based CPU Usage Accounting Infrastructure for Large Scale Computing Grids (2011), http://dx.doi.org/10.1007/978-1-4419-8014-4_14
[12] B. Lin et al., htcondor/htcondor-ce: HTCondor-CE 24.1.2 (2024), https://zenodo.org/doi/10.5281/zenodo.3856680
[13] D. Thain et al., Distributed computing in practice: the condor experience., Concurrency - Practice and Experience **17**, 323 (2005).