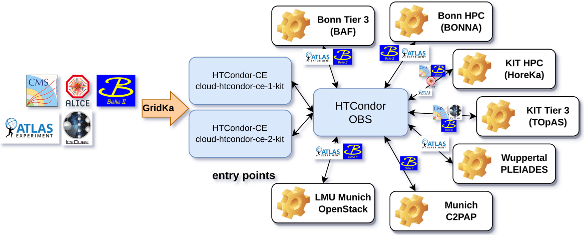
# AUDITOR:
# An Accounting tool for Grid Sites and Opportunistic Resources

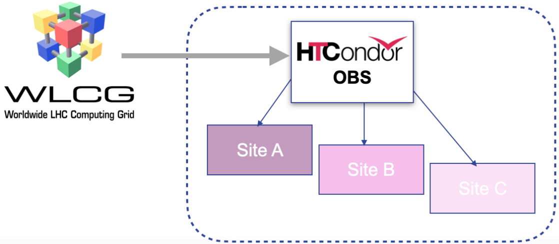
Matthias J. Schnepf on behalf of the AUDITOR Group | 4. November 2024

# Grid Subsites



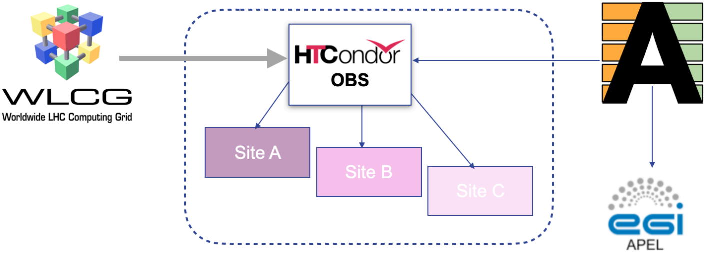
- Grid Site provide CE and storage
- several small/non-Grid like resource provider
- CE provides Grid-like access to several computing resources
- all resources are in one resource pool/**O**verlay **B**atch **S**ystem (OBS)

# Accounting opportunistic resources



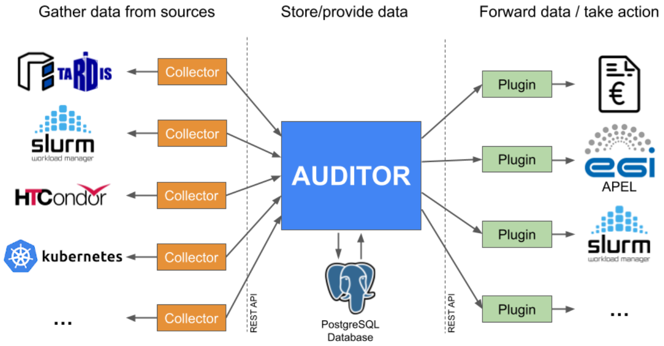
- COBaID/TARDIS allows multiple resources to be clustered in an **O**verlay **B**atch **S**ystem
  - Sub clusters **cannot be accounted individually** with existing tools
  - Requires a dedicated mechanism for accounting
- Challenges
  - Vastly different infrastructures
  - Many potential use cases

# Accounting opportunistic resources



- COBalD/TARDIS allows multiple resources to be clustered in an **O**verlay **B**atch **S**ystem
  - Sub clusters **cannot be accounted individually** with existing tools
  - Requires a dedicated mechanism for accounting
- Challenges
  - Vastly different infrastructures
  - Many potential use cases
- **AUDITOR** provides multi-purpose accounting ecosystem

# AUDITOR: Modular Accounting Ecosystem



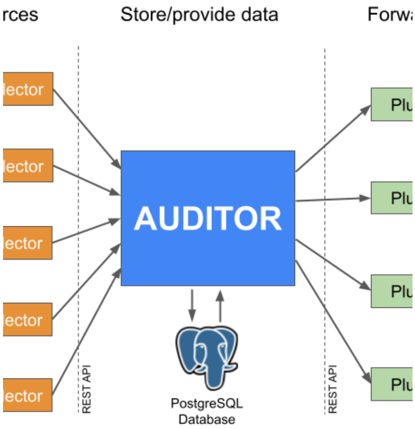
- **Collectors**
  - Accumulate data
- **Core component**
  - Accept data
  - Store data
  - Provide data
- **Plugins**
  - Take action based on stored data

## Documentation and code

→ https://github.com/ALU-Schumacher/AUDITOR

**AUDITOR: A**cco**U**nting **D**ata handl**I**ng **T**oolbox for **O**pportunistic **R**esources

# AUDITOR: Core component



- Implemented in **Rust**
  - Access via REST interface
- Unit of accountable resources: **Record**
- Data stored in PostgreSQL
- Completely stateless
  - No dataloss
  - Suitable for high availability setups
- Provided as **RPM** or **Docker container**
- Client libraries in Rust and Python

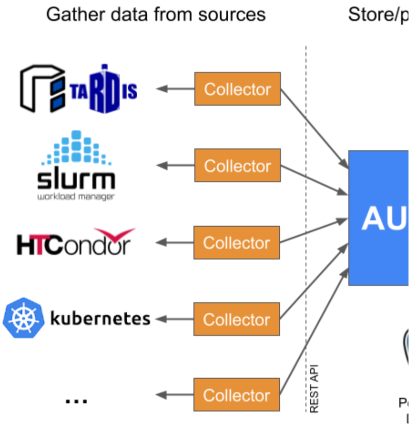
# Record in AUDITOR

- `record_id`: uniquely identifies the record
- `meta`: multiple key value pairs of the form `String -> [String]`
- `components`: arbitrary number of resources that are to be accounted for (CPU, RAM, Disk, GPU, . . . )
  - `scores`: (multiple) accounting scores supported
- `start_time`, `end_time`: datetime in UTC
- `runtime`: calculated as `end_time - start_time`

→ `meta` & `component` fields allow for maximal flexibility

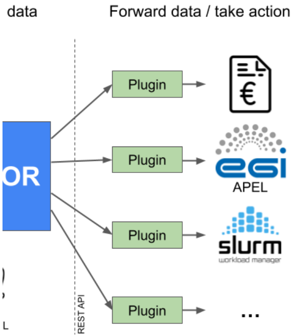
```
{
    "record_id": "hpc-4126142",
    "meta": {
      "group_id": [ "atlpr" ],
      "site_id": [ "hpc" ],
      "user_id": [ "atlpr001" ]
    },
    "components": [
      {
        "name": "Cores",
        "amount": 8,
        "scores": [
          {
            "name": "HEPSPEC06",
            "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
},
```

# Collectors: Accumulate data



- collect metadata about used resources
  - walltime
  - CPU usage
  - weight/score (e.g. HEPScore23 or $CO_2$ equivalent)
  - . . .
- provided collectors
  - **TARDIS Collector** (developed @ Freiburg)
  - **SLURM Collectors** (developed @ Freiburg)
  - **HTCondor Collector** (developed @ KIT)
  - **Kubernetes Collector** (developed @ Uni Wuppertal)
- You miss a collector? Feel free to join

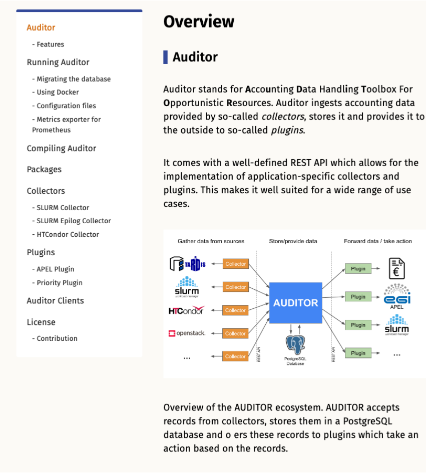
# Plugins



- **Priority plugin**
  - Compute priorities from a list of records
  - Update priorities on a batch cluster
- **APEL accounting plugin**
  - Properly accounts individual sites or subsites (e.g. behind COBalD/TARDIS)
  - Reports accounting data to the APEL accounting platform
- **Utilization report** (future project)
  - Analyse requested vs. consumed resources of a user
  - Send a weekly report with possible savings and $CO_2$ footprint

# AUDITOR

- Extensive documentation



Auditor
- Features

Running Auditor
- Migrating the database
- Using Docker
- Configuration files
- Metrics exporter for Prometheus

Compiling Auditor

Packages

Collectors
- SLURM Collector
- SLURM Epilog Collector
- HTCondor Collector

Plugins
- APEL Plugin
- Priority Plugin

Auditor Clients

License
- Contribution

## Overview

### Auditor

Auditor stands for **A**ccounting **D**ata Handling **T**oolbox For **O**pportunistic **R**esources. Auditor ingests accounting data provided by so-called *collectors*, stores it and provides it to the outside to so-called *plugins*.

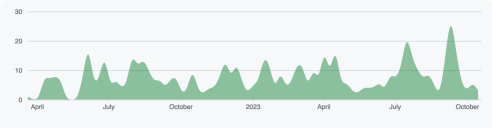
It comes with a well-defined REST API which allows for the implementation of application-specific collectors and plugins. This makes it well suited for a wide range of use cases.

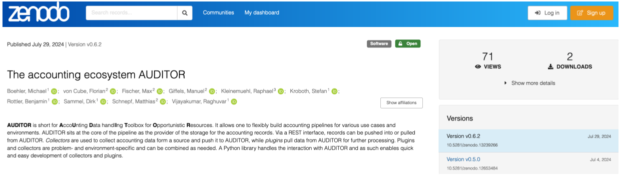
Overview of the AUDITOR ecosystem. AUDITOR accepts records from collectors, stores them in a PostgreSQL database and o ers these records to plugins which take an action based on the records.

https://doi.org/10.21203/rs.3.rs-4741479/v1

Some numbers:

- 8 contributors
- from 3 universities
  - Freiburg (main developement), KIT, Uni Wuppertal
- 16 releases - latest v0.6.3
- Continuous improvements: Commits





Published July 29, 2024 | Version v0.6.2

The accounting ecosystem AUDITOR

Boehler, Michael[1]; von Cube, Florian[2]; Fischer, Max[2]; Giffels, Manuel[2]; Kleinemuehl, Raphael[3]; Kroboth, Stefan[1]; Rottler, Benjamin[1]; Sammel, Dirk[1]; Schnepf, Matthias[2]; Vijayakumar, Raghuvar[1]

**AUDITOR** is short for **A**ccoU**n**ting **D**ata handl**I**ng **T**oolbox for **O**pportunistic **R**esources. It allows one to flexibly build accounting pipelines for various use cases and environments. AUDITOR sits at the core of the pipeline as the provider of the storage for the accounting records. Via a REST interface, records can be pushed into or pulled from AUDITOR. *Collectors* are used to collect accounting data form a source and push it to AUDITOR, while *plugins* pull data from AUDITOR for further processing. Plugins and collectors are problem- and environment-specific and can be combined as needed. A Python library handles the interaction with AUDITOR and as such enables quick and easy development of collectors and plugins.

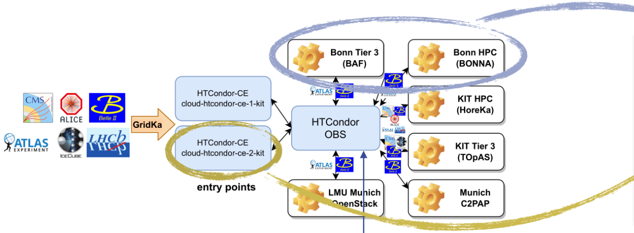
71 VIEWS

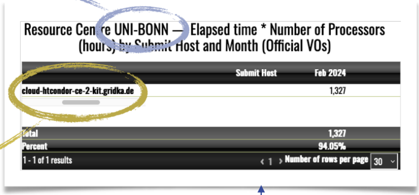
2 DOWNLOADS

Versions

| | |
|---|---|
| Version v0.6.2<br>10.5281/zenodo.13239266 | Jul 29, 2024 |
| Version v0.5.0<br>10.5281/zenodo.12653484 | Jul 4, 2024 |

# WLCG Subsite Accounting

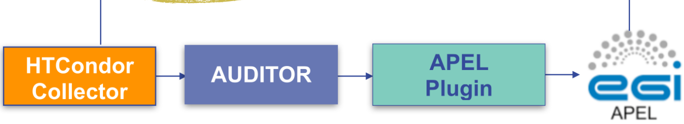




Resource Centre UNI-BONN — Elapsed time * Number of Processors (hours) by Submit Host and Month (Official VOs)

| | Submit Host | Feb 2024 |
|---|---|---|
| cloud-htcondor-ce-2-kit.gridka.de | | 1,327 |
| Total | | 1,327 |
| Percent | | 94.05% |

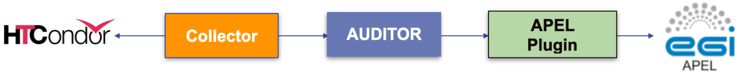
1 - 1 of 1 results ‹ 1 › Number of rows per page 30

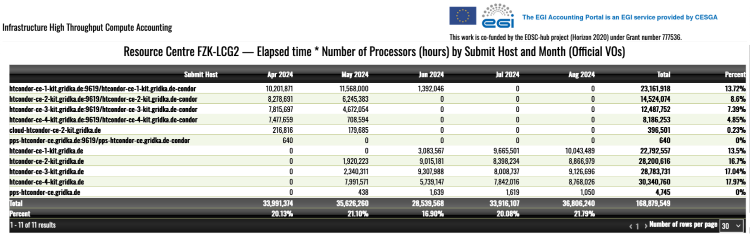


- Grid infrastructure hosted and maintained in Karlsruhe, resources provided by Bonn
- **AUDITOR** accounting pipeline allows to account for sub-clusters individually

# WLCG Grid Site Accounting

- KIT replaced accounting of the APEL client by AUDITOR pipeline in May 2024





Infrastructure High Throughput Compute Accounting

The EGI Accounting Portal is an EGI service provided by CESGA

This work is co-funded by the EOSC-hub project (Horizon 2020) under Grant number 777536.

Resource Centre FZK-LCG2 — Elapsed time * Number of Processors (hours) by Submit Host and Month (Official VOs)

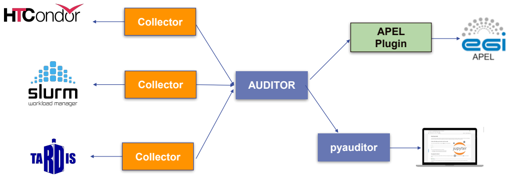
| Submit Host | Apr 2024 | May 2024 | Jun 2024 | Jul 2024 | Aug 2024 | Total | Percent |
|---|---|---|---|---|---|---|---|
| htcondor-ce-1-kit.gridka.de:9619/htcondor-ce-1-kit.gridka.de-condor | 10,201,871 | 11,568,000 | 1,392,046 | 0 | 0 | 23,161,918 | 13.72% |
| htcondor-ce-2-kit.gridka.de:9619/htcondor-ce-2-kit.gridka.de-condor | 8,278,691 | 6,245,383 | 0 | 0 | 0 | 14,524,074 | 8.6% |
| htcondor-ce-3-kit.gridka.de:9619/htcondor-ce-3-kit.gridka.de-condor | 7,815,697 | 4,672,054 | 0 | 0 | 0 | 12,487,752 | 7.39% |
| htcondor-ce-4-kit.gridka.de:9619/htcondor-ce-4-kit.gridka.de-condor | 7,477,659 | 708,594 | 0 | 0 | 0 | 8,186,253 | 4.85% |
| cloud-htcondor-ce-2-kit.gridka.de | 216,816 | 179,685 | 0 | 0 | 0 | 396,501 | 0.23% |
| pps-htcondor-ce.gridka.de:9619/pps-htcondor-ce.gridka.de-condor | 640 | 0 | 0 | 0 | 0 | 640 | 0% |
| htcondor-ce-1-kit.gridka.de | 0 | 0 | 3,083,567 | 9,665,501 | 10,043,489 | 22,792,557 | 13.5% |
| htcondor-ce-2-kit.gridka.de | 0 | 1,920,223 | 9,015,181 | 8,398,234 | 8,866,979 | 28,200,616 | 16.7% |
| htcondor-ce-3-kit.gridka.de | 0 | 2,340,311 | 9,307,988 | 8,008,737 | 9,126,696 | 28,783,731 | 17.04% |
| htcondor-ce-4-kit.gridka.de | 0 | 7,991,571 | 5,739,147 | 7,842,016 | 8,768,026 | 30,340,760 | 17.97% |
| pps-htcondor-ce.gridka.de | 0 | 438 | 1,639 | 1,619 | 1,050 | 4,745 | 0% |
| Total | 33,991,374 | 35,626,260 | 28,539,568 | 33,916,107 | 36,806,240 | 168,879,549 | |
| Percent | 20.13% | 21.10% | 16.90% | 20.08% | 21.79% | | |

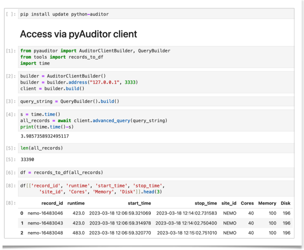
1 - 11 of 11 results

‹ 1 › Number of rows per page 30

- AUDITOR is able to provide the accounting of the DE-Tier 1
  - largest WLCG Tier-1 that supports allfour WLCG experiments

# Collecting Accounting Info with AUDITOR





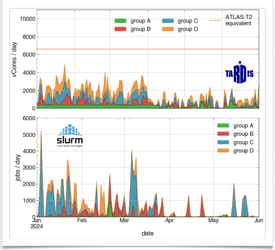
```
[ ]: pip install update python-auditor
```

### Access via pyAuditor client

```
[1]: from pyauditor import AuditorClientBuilder, QueryBuilder
     from tools import records_to_df
     import time

[2]: builder = AuditorClientBuilder()
     builder = builder.address("127.0.0.1", 3333)
     client = builder.build()

[3]: query_string = QueryBuilder().build()

[4]: s = time.time()
     all_records = await client.advanced_query(query_string)
     print(time.time()-s)

     3.9857358932495117

[5]: len(all_records)

[5]: 33390

[6]: df = records_to_df(all_records)

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

[8]:

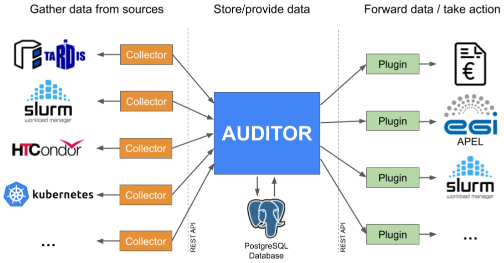
| | record_id | runtime | start_time | stop_time | site_id | Cores | Memory | Disk |
|---|---|---|---|---|---|---|---|---|
| 0 | nemo-16483046 | 423.0 | 2023-03-18 12:06:59.321069 | 2023-03-18 12:14:02.731583 | NEMO | 40 | 100 | 196 |
| 1 | nemo-16483043 | 423.0 | 2023-03-18 12:06:59.314978 | 2023-03-18 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 |



- accounting data can be collected in one or more AUDITOR instances from multiple sources
- APEL plugin can report for one or more queues
- pyauditor allows to integrate AUDITOR client into python env

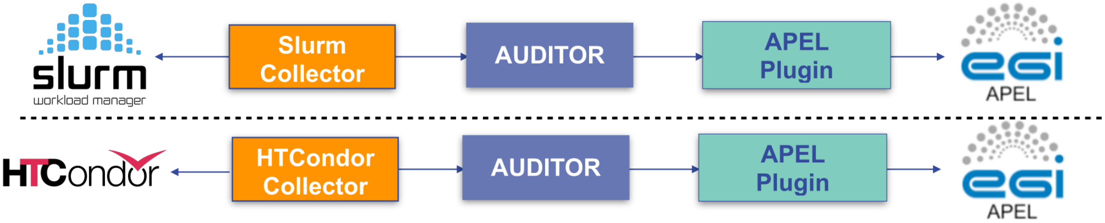
# Conclusion

- Provides an accounting ecosystem for various use cases
- Allows to account for different resources shared by one overlay batch system
- Allows to collect accounting data from multiple sources
- provision via containers independent of the OS
- Flexible structure of records and ecosystem allows to quickly adapt to future use cases
- Website: https://alu-schumacher.github.io/AUDITOR/
- GitHub: https://github.com/ALU-Schumacher/AUDITOR/
- FIDIUM: https://fidium.erumdatahub.de
- Email: auditor@physik.uni-freiburg.de



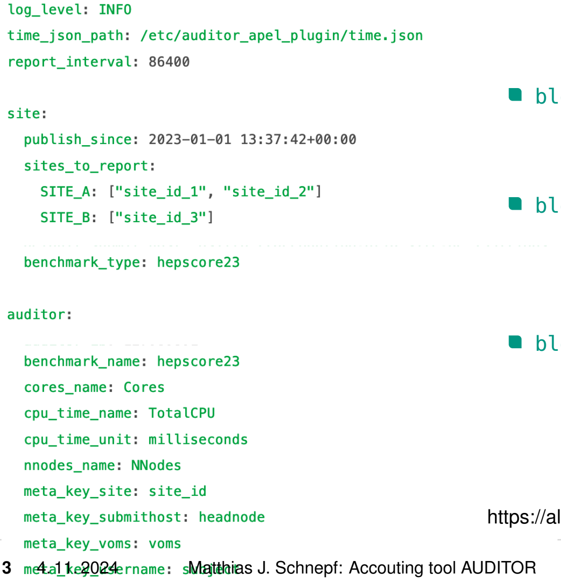
# Backup

# WLCG Accounting Use Case



- Collect accounting data from SLURM or HTCondor
- Store data as records in AUDITOR DB
- APEL plugin retrieves records from AUDITOR
  - creates APEL job summary from records
  - sends summary to defined APEL server
- Sites planing to use AUDITOR for accounting:
  - DESY-HH, Uni Wuppertal, ... ← ATLAS DE T1 (GridKa) moved reporting to AUDITOR

# APEL Plugin

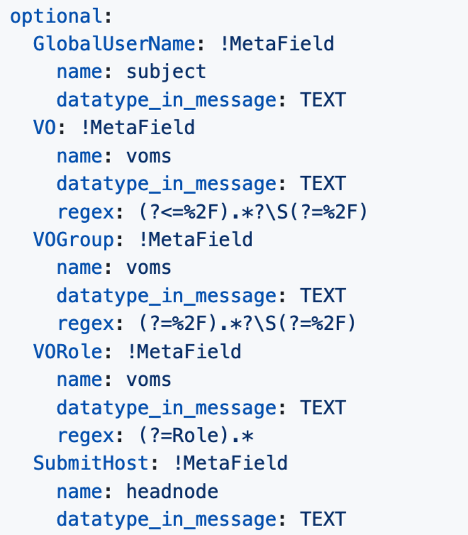


```
log_level: INFO
time_json_path: /etc/auditor_apel_plugin/time.json
report_interval: 86400

site:
  publish_since: 2023-01-01 13:37:42+00:00
  sites_to_report:
    SITE_A: ["site_id_1", "site_id_2"]
    SITE_B: ["site_id_3"]

  benchmark_type: hepscore23

auditor:

  benchmark_name: hepscore23
  cores_name: Cores
  cpu_time_name: TotalCPU
  cpu_time_unit: milliseconds
  nnodes_name: NNodes
  meta_key_site: site_id
  meta_key_submithost: headnode
  meta_key_voms: voms
  meta_key_username: subject
```

- block 1: configure serivce
  - file to store current state
  - time in seconds between reports
- block 2: configure site(s) to be reported
  - sites_to_report:
    keys: names of the sites in the GOCDB,
    values: corresponding site names in AUDITOR records
- block 3: configure metrics to be reported
  - meta_key_voms:
    key in meta field to be used as voms

https://alu-schumacher.github.io/AUDITOR//v0.5.0/#apel-plugin

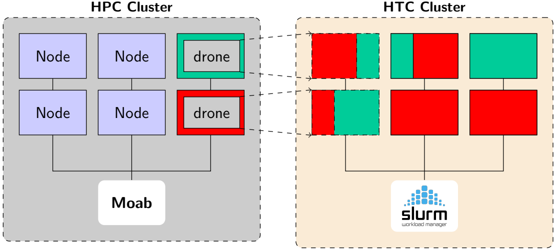
# APEL Plugin



```
optional:
  GlobalUserName: !MetaField
    name: subject
    datatype_in_message: TEXT
  VO: !MetaField
    name: voms
    datatype_in_message: TEXT
    regex: (?<=%2F).*?\S(?=%2F)
  VOGroup: !MetaField
    name: voms
    datatype_in_message: TEXT
    regex: (?=%2F).*?\S(?=%2F)
  VORole: !MetaField
    name: voms
    datatype_in_message: TEXT
    regex: (?=Role).*
  SubmitHost: !MetaField
    name: headnode
    datatype_in_message: TEXT
```

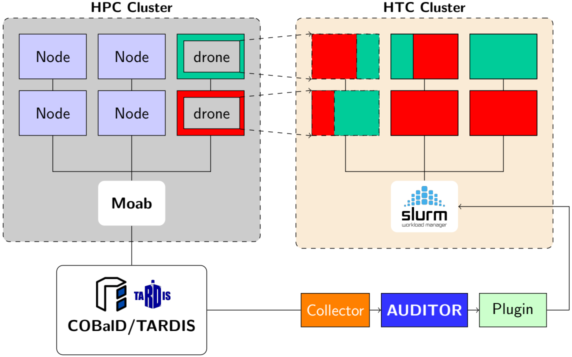
- dynamic mapping of any MetaField via regex
  - this allows to report accounting data for different VOs submitted with **tokens**
- plugin configuration a bit more complicated, but much more flexible

# Priority Use Case



- HPC resources integrated with COBalD/TARDIS
- Several HEP groups provide HPC resources
- Resources shared among HEP groups
- How to guarantee fair share on HTC cluster?

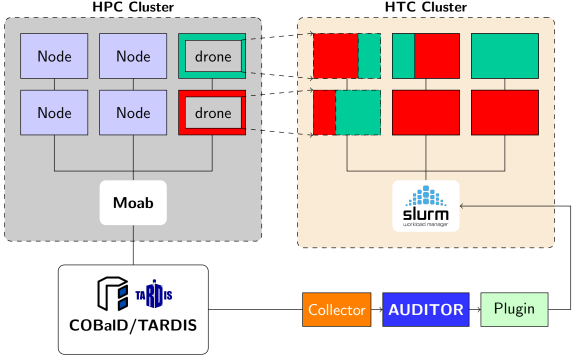
# Priority Use Case



- HPC resources integrated with COBalD/TARDIS
- Several HEP groups provide HPC resources
- Resources shared among HEP groups
- How to guarantee fair share on HTC cluster?

- **TARDIS collector** retrieves info of provided resources on the NEMO cluster
- **AUDITOR** accounts for provided resources of individual groups [**A** and **B**]
- **Priority plugin** adjusts priorities on HTC cluster

# Priority Use Case



HPC Cluster

Node

Node

drone

Node

Node

drone

Moab

HTC Cluster

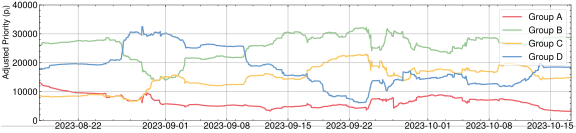
slurm workload manager

COBalD/TARDIS

Collector

AUDITOR

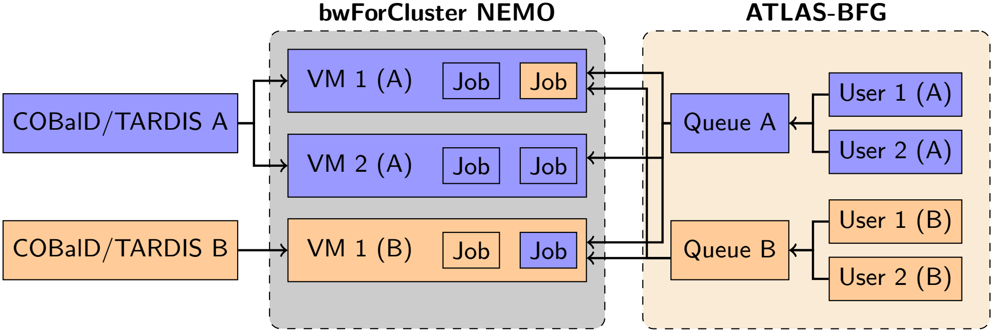
Plugin



- HPC resources integrated with COBalD/TARDIS
- Several HEP groups provide HPC resources
- Resources shared among HEP groups
- How to guarantee fair share on HTC cluster?

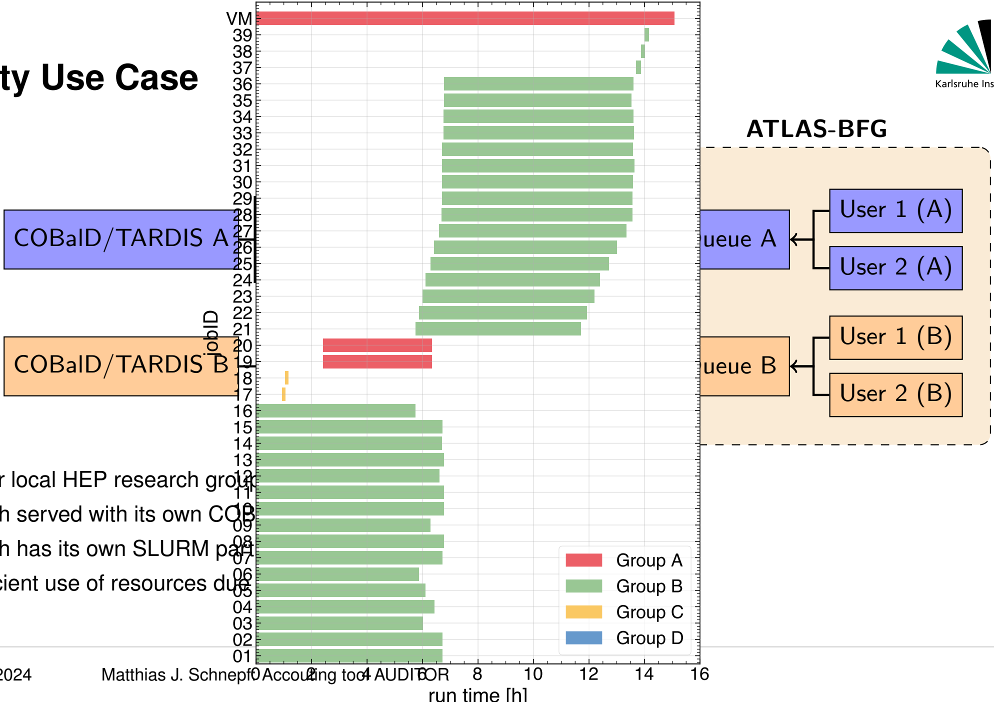
- **TARDIS collector** retrieves info of provided resources on the NEMO cluster
- **AUDITOR** accounts for provided resources of individual groups [**A** and **B**]
- **Priority plugin** adjusts priorities on HTC cluster

# Priority Use Case



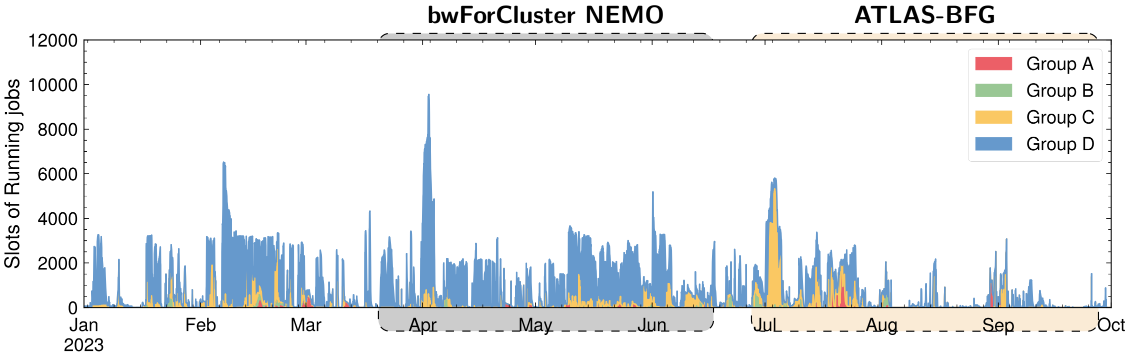
- Four local HEP research groups (A to D) with a share in NEMO
- Each served with its own COBalD/TARDIS instance
- Each has its own SLURM partition (job queue)
- Efficient use of resources due to sharing VMs across HEP groups

# Priority Use Case



- Four local HEP research grou
- Each served with its own COB
- Each has its own SLURM part
- Efficient use of resources due

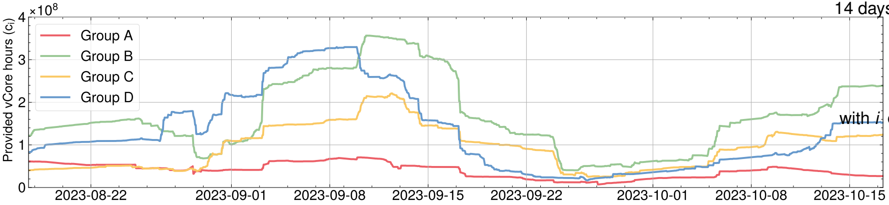
# Priority Use Case



- Four local HEP research groups (A to D) with a share in NEMO
- Each served with its own COBalD/TARDIS instance
- Each has its own SLURM partition (job queue)
- Efficient use of resources due to sharing VMs across HEP groups

# Priority Use Case

- Provided resources of the four local HEP groups

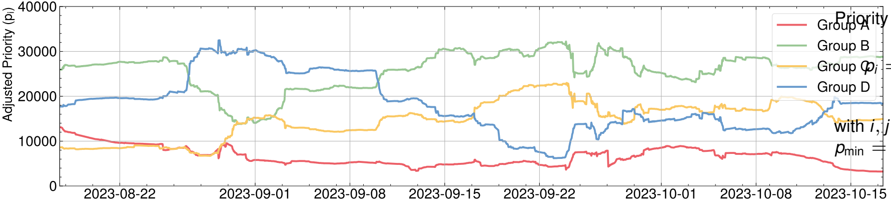


Integral of provided vCore hours over last 14 days for each group:

$$c_i = \int_{t_{\mathrm{now}} - 14\,\mathrm{d}}^{t_{\mathrm{now}}} N_i(t)\mathrm{d}t$$

with $i \in A, B, C, D$

- Priority is adjusted according to the provided resources



Priority $p_i$ is defined as:

$$p_i = \frac{c_i}{\sum_j c_j}\left(p_{\mathrm{max}} - p_{\mathrm{min}}\right) + p_{\mathrm{min}}$$

with $i, j \in A, B, C, D$;
$p_{\mathrm{min}} = 1$; $p_{\mathrm{max}} = 65535$