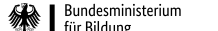
universität freiburg

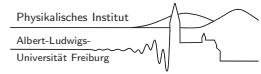
Accounting with AUDITOR

Michael Böhler

WLCG/HSF Workshop at DESY Hamburg, May 13-17 2024

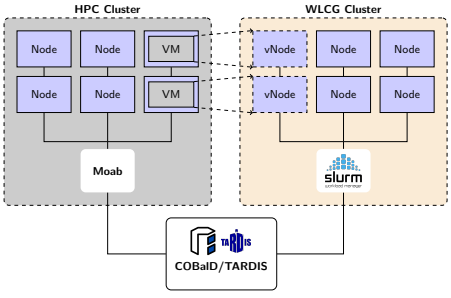
May 14th 2024





Motivation

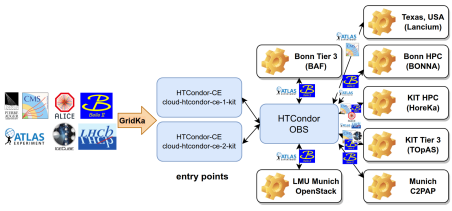
Accounting opportunistic resources



- Opportunistic resources can be integrated dynamically and transparently into clusters with COBalD/TARDIS
- One or multiple resources behind an OBS
  - **Cannot be accounted properly** with existing tools
  - Requires a dedicated mechanism for accounting
- Challenges
  - Vastly different infrastructures
  - Many potential use cases
- **AUDITOR** provides multi-purpose accounting ecosystem

Motivation

Accounting opportunistic resources

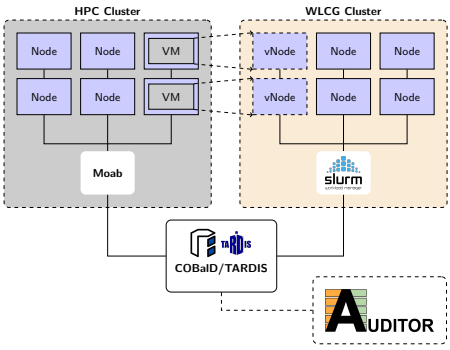


Transformation of German University-Tier-2
(talk by Sebastian Wozniewski Monday 16:00)

- Opportunistic resources can be integrated dynamically and transparently into clusters with COBaID/TARDIS
- One or multiple resources behind an OBS
  - **Cannot be accounted properly** with existing tools
  - Requires a dedicated mechanism for accounting
- Challenges
  - Vastly different infrastructures
  - Many potential use cases
- **AUDITOR** provides multi-purpose accounting ecosystem

Motivation

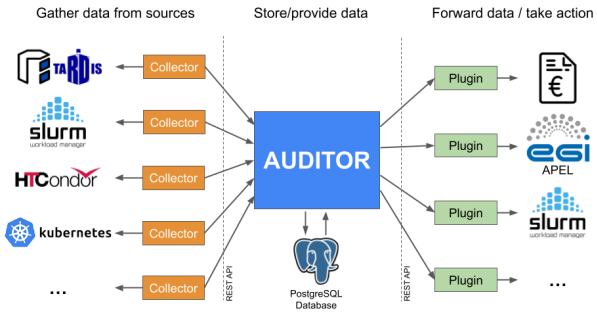
Accounting opportunistic resources



- Opportunistic resources can be integrated dynamically and transparently into clusters with COBalD/TARDIS
- One or multiple resources behind an OBS
  - **Cannot be accounted properly** with existing tools
  - Requires a dedicated mechanism for accounting
- Challenges
  - Vastly different infrastructures
  - Many potential use cases
- **AUDITOR** provides multi-purpose accounting ecosystem

Auditor

Accounting Ecosystem



AUDITOR: AccoUnting Data handlIng Toolbox for Opportunistic Resources

Modular accounting ecosystem

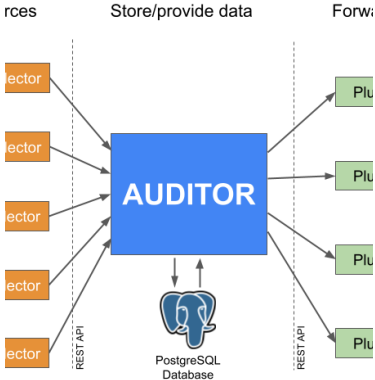
- **Collectors**
  - Accumulate data
- **Core component**
  - Accept data
  - Store data
  - Provide data
- **Plugins**
  - Take action based on stored data

Documentation and code

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

Auditor

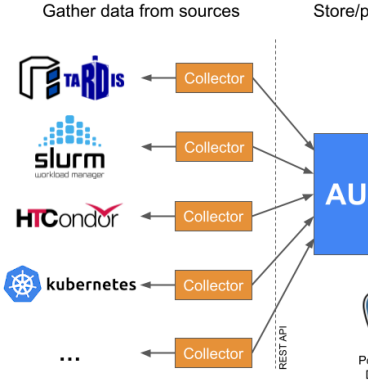
Core component



- 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

Collectors

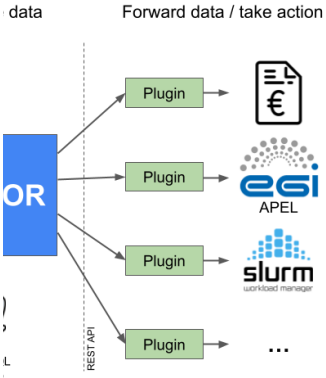
Accumulate data



- **TARDIS Collector**
  - Collect drone information
- **SLURM Collectors** (2 types)
  - Collect information about SLURM jobs via SLURM CLI commands
- **HTCondor Collector** (developed @ KIT)
  - Equivalent of SLURM collector for HTCondor
- **Kubernetes Collector** (WIP @ Uni Wuppertal)
  - Collects information from Kubernetes pods

Plugins

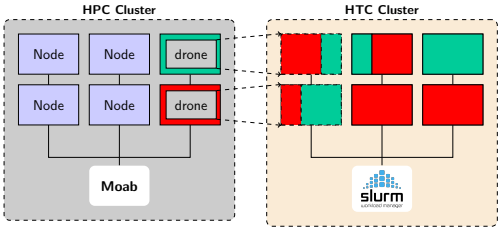
Take action based on stored data



- **Priority plugin**
  - Compute priorities from a list of records
  - Update priorities on a batch cluster
- **APEL accounting plugin**
  - Properly accounts individual sites behind COBaID/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

Priority Use Case

Adapting priorities on HTC cluster based on provided HPC resources

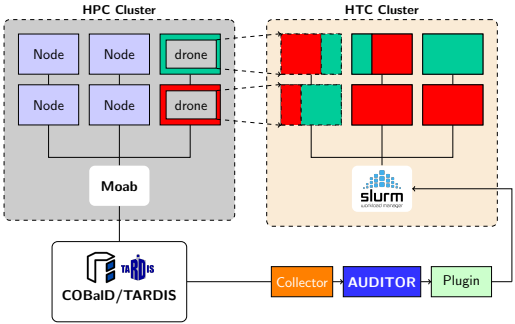


- HPC resources integrated with COBalD/TARDIS
- Several HEP groups provide HPC resources
- Resources shared among HEP groups
- How to guarantee fair share on HTC cluster?

- **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

Adapting priorities on HTC cluster based on provided HPC resources

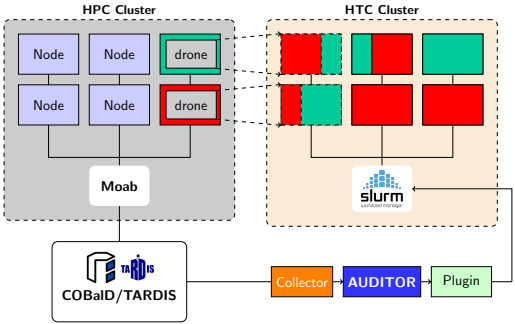


- HPC resources integrated with COBalD/TARDIS
- Several HEP groups provide HPC resources
- Resources shared among HEP groups
- How to guarantee fair share on HTC cluster?

- **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

Adapting priorities on HTC cluster based on provided HPC resources

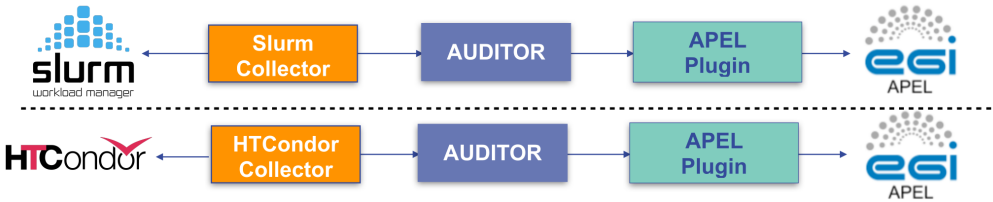


- HPC resources integrated with COBalD/TARDIS
- Several HEP groups provide HPC resources
- Resources shared among HEP groups
- How to guarantee fair share on HTC cluster?

- **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

WLCG Accounting Use Case

APEL Accounting with AUDITOR

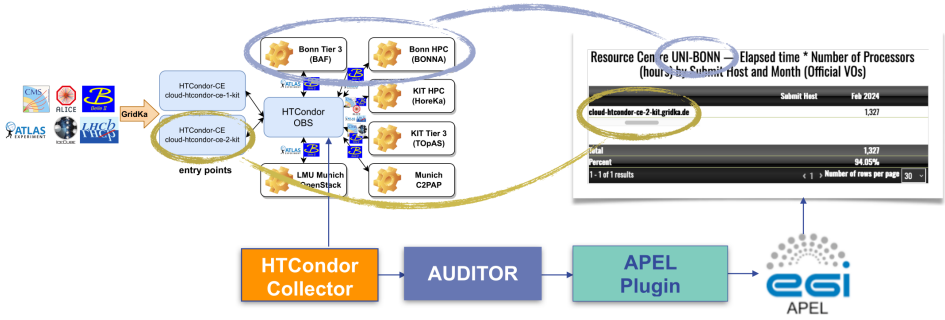


- Collect accounting data from SLURM or HTCondor
- Store data as records in AUDITOR DB
- APEL plugin retrieves records from AUDITOR
  - creates APEL job summary from records
  - sends summary to defined APEL server
- Sites planing to use AUDITOR for accounting:
  - DESY-HH, ~~GridKa~~, Uni Wuppertal,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

BRAND NEW

WLCG Accounting Use Case

First working prototype



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

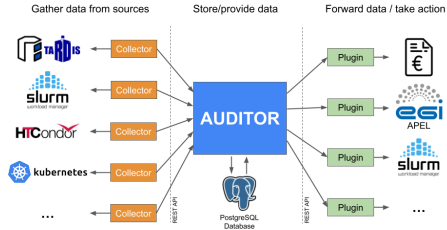
- Reporting is currently done with incremental summary reports
- Support for individual job reports will be available in next release

Conclusion

AUDITOR

- provides an accounting ecosystem for various use cases
- allows to account for different resources shared by one overlay batch system
- provision via containers independent of the OS

- Flexible structure of records and ecosystem allows to quickly adapt to future use cases
  - e.g. GPU resources
  - v0.6.0 APEL plugin supports configurable meta fields for reporting (token $\rightarrow$ VO mapping)
  - dynamically changing corepower values in case of CPU throttling



References

Website: https://alu-schumacher.github.io/AUDITOR/

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

FIDIUM: https://fidium.erumdatahub.de

Michael Boehler

Albert-Ludwigs-Universität Freiburg

michael.boehler@physik.uni-freiburg.de

Back-Up...

Record

Unit of accountable resources

- `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
},
```

APEL Plugin

Configuration

```
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

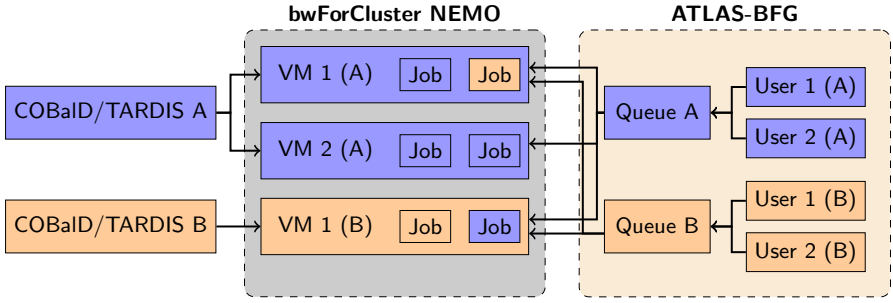
Configuration in Release v0.6.0

```
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

Motivation



- 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

Motivation

COBaID/TARDIS

COBaID/TARDIS

AS-BFG

User 1 (A)

User 2 (A)

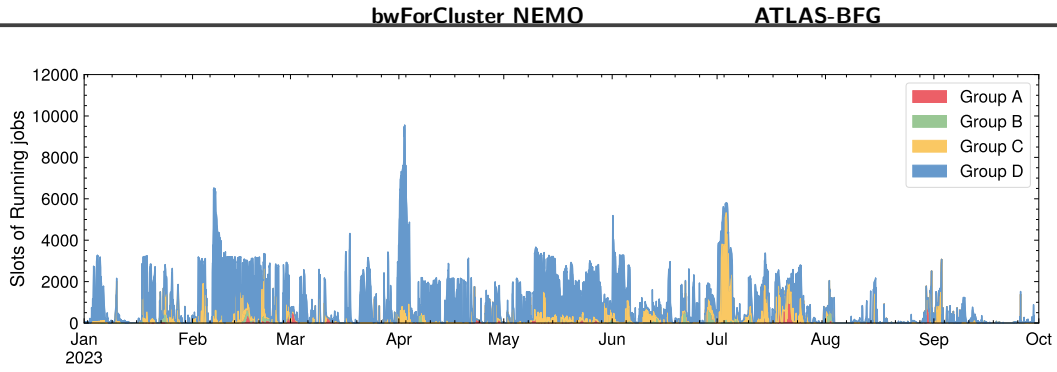
User 1 (B)

User 2 (B)

- Four local HEP rese
- Each served with its
- Each has its own SL
- Efficient use of resou

Priority Use Case

Motivation

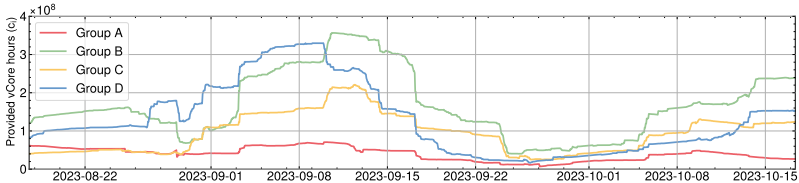


- Each has its own SLURM partition (job queue)
- Efficient use of resources due to sharing VMs across HEP groups

Priority Use Case

Results from HEP groups @ University of Freiburg

- Provided resources of the four local HEP groups

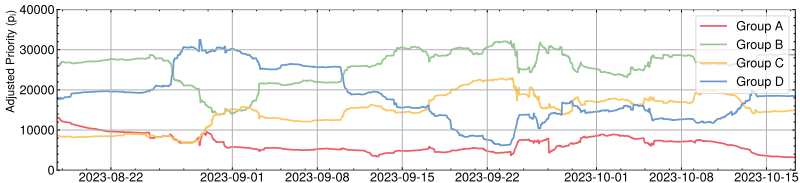


Integral of provided vCore hours over last 14 days for each group:

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

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

- Priority is adjusted according to the provided resources



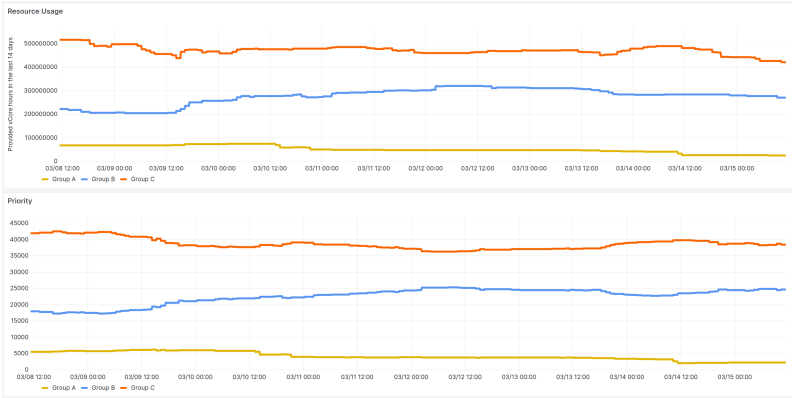
Priority p_i is defined as:

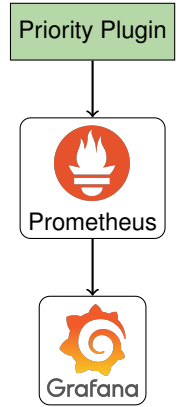
$$p_i = \frac{c_i}{\sum_j c_j}\,(p_{\text{max}} - p_{\text{min}}) + p_{\text{min}}$$

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

Priority Use Case

Real-time monitoring

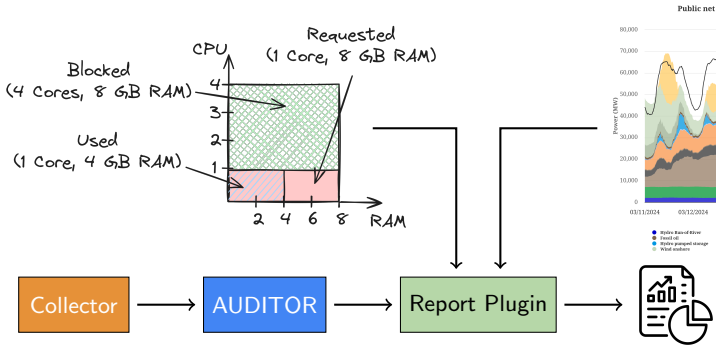




- Resource usage and priority can be made available for Prometheus
- Real-time monitoring of priority adjustments (with e.g. Grafana)

Future plans

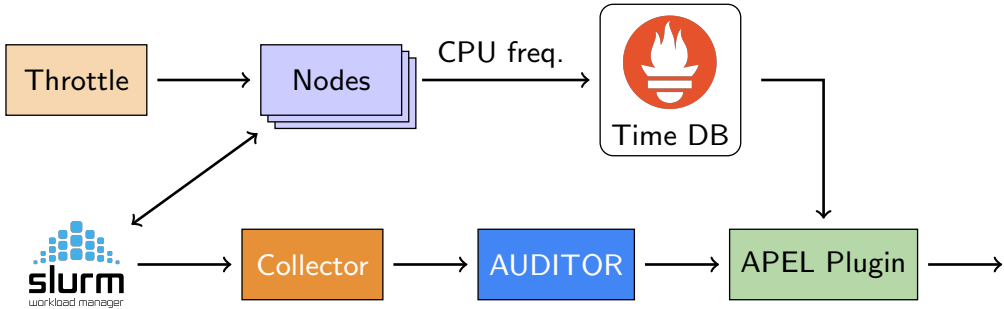
Utilization Report Use Case



- Compare used vs. requested vs. blocked resources
- Optionally: Calculate CO_2 usage of jobs based on energy mix
- Send weekly report to users or site admins

Future plans

CPU throttling - impact on accounting



- Recent idea: CPU throttling according to available energy mix
- Store CPU frequency per node and time interval
  - Records are not the right place to store this information
  - Use external time DB (Prometheus, InfluxDB, . . .)
- Create APEL reports with adjusted HEPScore values based on info from time DB