Accounting of opportunistic resources

Stefan Kroboth

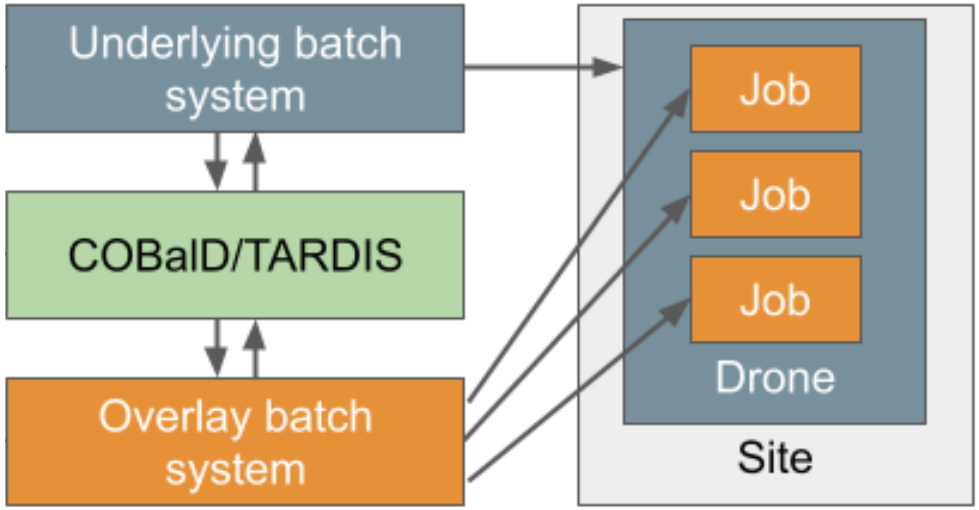
May 11th, 2021

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

ErUM-Data IDT

Opportunistic resources are hidden behind other resources



Introduction

Various potential scenarios

- Accounting with an experiment such as ATLAS/CMS/...
- Inter-site billing between sites sharing resources
- Matching fairshares/priority to provided resources

Tracking the usage of resources is therefore vital

- Collecting data
- Storing data
- Providing this data to other services

Planning

Which information do we need?

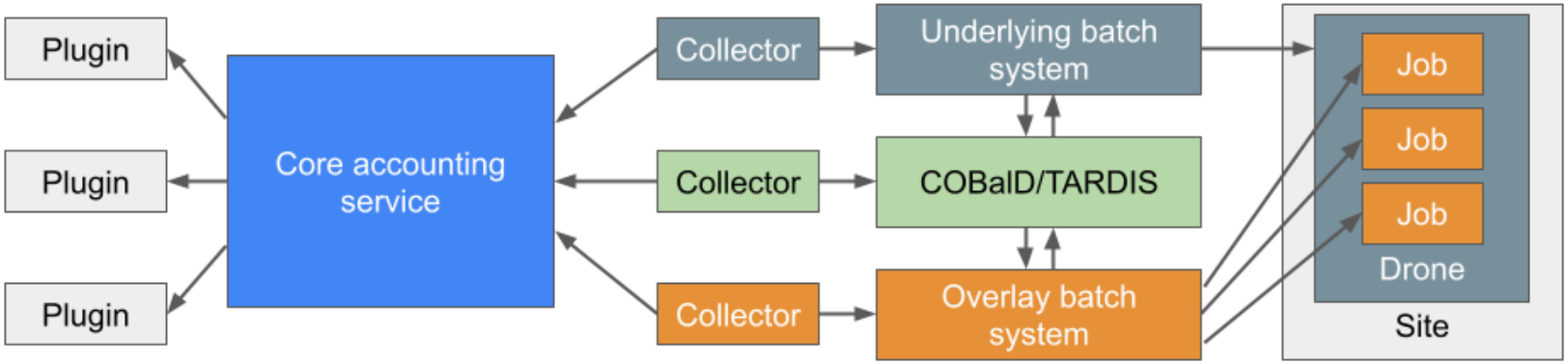
- How to define resources?
- How to compute the usage of resources?
- Which data to collect?

What should the interfaces look like?

- How to collect the data?
- How to store the data?
- How to transfer the data to other entities / use it for different purposes

Accounting architecture is designed around these questions

Architecture / Interfaces



- Core Service: Accepts information about consumed resources and stores it in a database
- Collector: Obtains information from a source (Batch system schedulers, COBalD/TARDIS, . . .) and forwards it to the Core Service
- Plugin: Fetches data from the Core Service and takes an action, such as forwarding it to external endpoints (WLCG) or adapting fairshares, creating a bill,. . .

Collected data

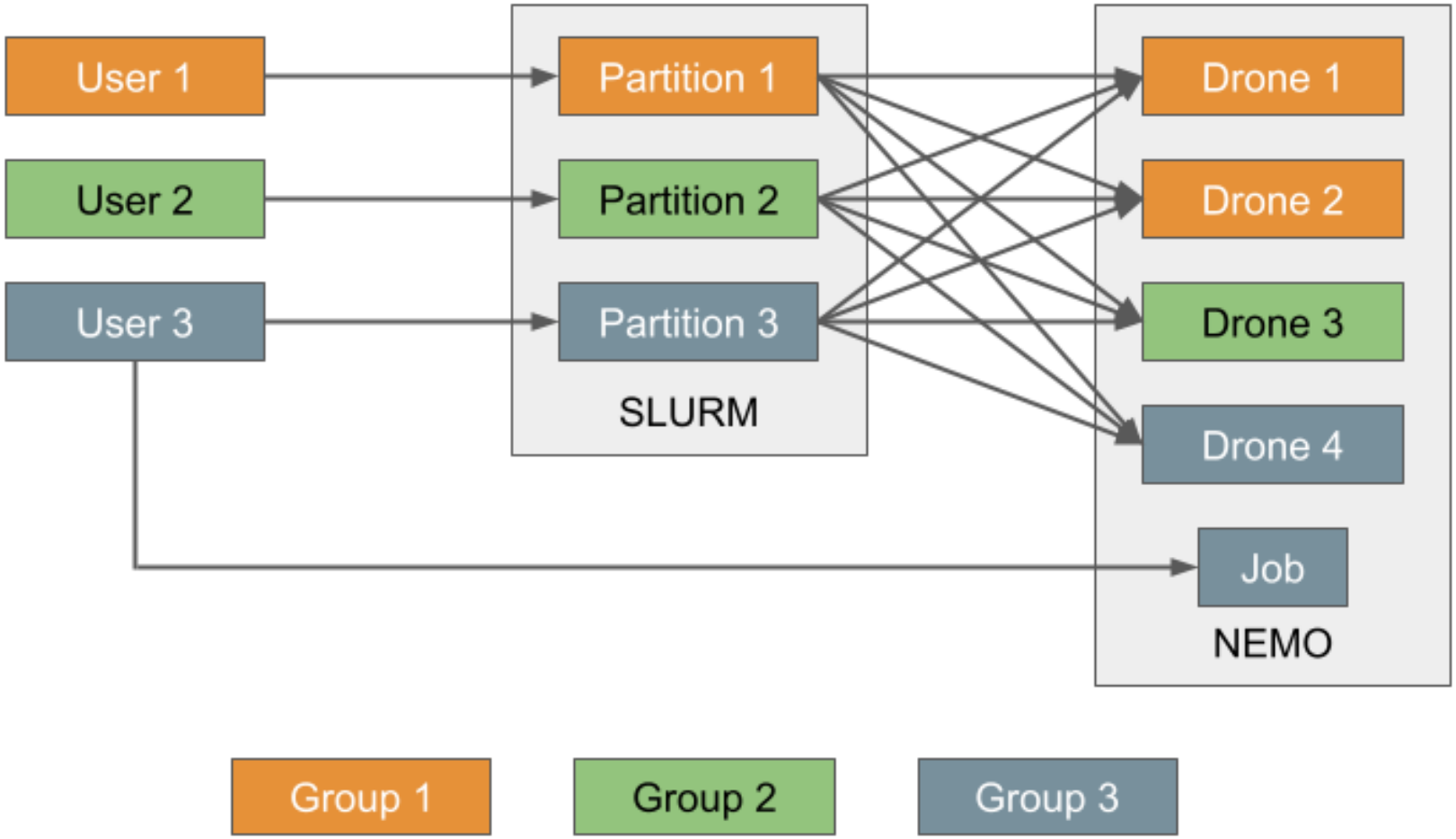
A single resource to be accounted for is defined as:

- Job ID (primary key)
- Site ID (primary key)
- User ID
- Group ID
- Components[ ]
  - Type
  - Amount
  - Factor
- Start time (optional)
- Stop time (optional)
- Runtime
- Update time

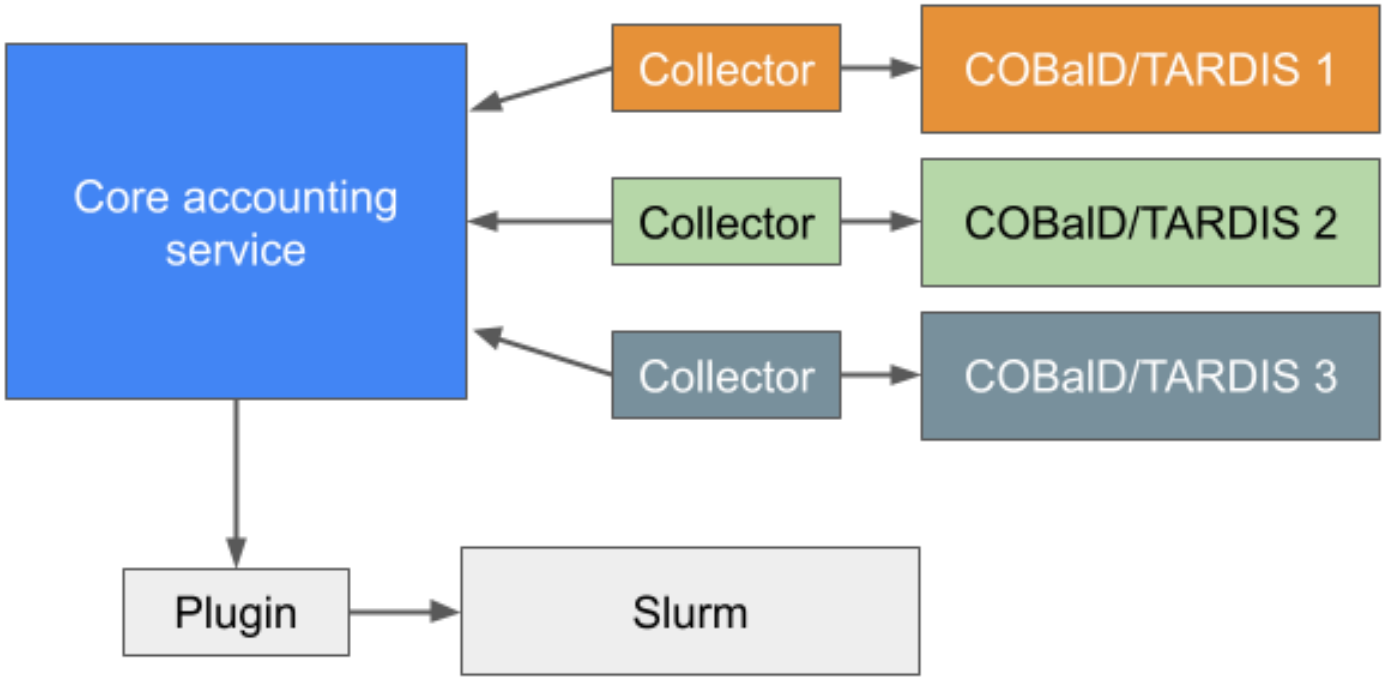
Prototype

- Main functionality of core accounting service implemented (accepting, storing and providing data)
- Implementation
  - Implemented in Rust
  - Actix-web actor framework
  - REST interface
  - PostgreSQL database (other data backends possible)
  - Completely stateless
- Open points
  - Collectors and plugins for various applications (written in Python)
  - Python library for connecting plugins and collectors to the core accounting service
  - Authentication

Testbed usecase in Freiburg



Testbed usecase in Freiburg



Summary and outlook

- Implemented a prototype of the core accounting service
  - Interfaces for plugins and collectors
  - Data storage
- Next steps
  - Test implementation @ Freiburg
  - Building a library to facilitate the implementation of plugins and collectors
  - Implement collectors
    - COBalD/TARDIS
    - Various batch systems
  - Implement plugins
    - Various experiments