MASARYK UNIVERSITY

FACULTY OF INFORMATICS

PBSMon2, web service for displaying MetaCenter status

Master's Thesis

BC. MARCEL LUKČO

Brno, Fall 2025

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

PBSMon2, web service for displaying MetaCenter status

Master's Thesis

BC. MARCEL LUKČO

Advisor: Mgr. Miroslav Ruda

Department of design and development of software systems

Brno, Fall 2025

FACULTAS ARTIS INFORMATICÆ • UNIVERSITAS MASARYKIANA •

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Marcel Lukčo

Advisor: Mgr. Miroslav Ruda

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Mgr. Miroslav Ruda, for his valuable guidance, insightful comments, and continuous support throughout the development of this thesis.

I would also like to thank RNDr. Martin Kuba, Ph.D., for his technical assistance and expertise, which significantly contributed to the practical part of this work.

My thanks also go to Mgr. Václav Chlumský, Ing. František Řezníček, and Mgr. Ivana Křenková for their collaboration, helpful advice, and support during the integration with related systems.

Abstract

TBD

Keywords

cloud, cloud computing, distributed computing, MetaCentrum

Contents

List of Tables

List of Figures

Introduction

Currently, high-performance computing plays a key role in the implementation of research projects across a wide range of scientific disciplines. In the academic sector, it is therefore essential to ensure access to sufficient computing power to enable the implementation of demanding simulations, the processing of large data sets, and the effective analysis of results. National computing infrastructures, including CESNET, have been created for this purpose. This academic institution offers computing resources dedicated exclusively to research and academic purposes.

The PBSmon web service was developed to monitor these resources and provide an overview of their status. This application ensures the regular collection of metadata from computing nodes, queues, servers, and running jobs using native calls to PBS servers. It then transforms this data into a visual form accessible to users. Thanks to PBSmon, users can monitor the current system load, the status of individual jobs, and the availability of computing resources in real time.

However, the original PBSmon application is monolithic and uses outdated technologies, such as Java library Stripes, which is no longer maintained. Growing demands for security, sustainability, and support for containerized deployment require a fundamental refactoring of the system.

This thesis deals with the design and implementation of a new version of the PBSmon system, built on modern technologies: backend within the NestJS framework and frontend within the React framework. The goal is to create a modular, maintainable, and cloud-native architecture that preserves the functionality of the original solution while being ready for further development and adaptation to current technological standards.

1 Analysis of the Existing Solution – PBS MON

The Pbsmon application serves as a central tool for collecting and visualizing data from various systems related to computing resource management. It automatically collects information from the PBS (Portable Batch System) [1], the Perun identity management system, and virtual machines (OpenStack). It then processes this data and provides users with a comprehensive view of the current status of the infrastructure, jobs, and user activities within the entire MetaCentrum infrastructure.

The web interface is used to visualize data from the PBS, Perun, and OpenStack systems. The information displayed includes a personal view of jobs, available computing resources, an overview of computing machines and their utilization, a list of users, and other statistics. Among other things, the following data is collected:

- PBS nodes (status, load, available resources),
- information about virtual machines
- physical machines (availability, utilization, technical parameters),
- users (identity, activity, jobs),
- queues (status, priority, configuration),
- jobs (status, start time, resources used, user).

1.1 System Architecture

The existing solution is built as a monolithic Java application that collects data from various systems (PBS, Perun, OpenStack) and displays it in a web interface. The architecture is tightly coupled and uses multiple technologies, including C libraries and shell scripts.

1.1.1 Data collection environments

Pbsmon collects following data from the following environments:

- PBS environment
  - Compute clusters managed by PBS systems
  - Collection of Fairshare metrics from pbscache, which determine the user's priority when launching a job
  - Collection of the /etc/group file from PBS, essential for assigning users to groups used in access control lists (ACLs)
- Perun environment - an identity and group management system. Returns information about users and the list of physical machines.
- OpenStack environment - Information about virtual machines running on physical hosts

Each of these environments has its own method of communication and data representation. Within the system architecture, these differences are abstracted through interfaces and adapters that ensure unified data processing. All of these enviroments and their data structure are described in the following sections.

1.1.2 Data Processing and Unification

The data obtained from various sources — such as compute clusters (PBS), cloud platforms (e.g., OpenStack), and the identity management system (Perun) — differ in their formats and representations. After collection, the data are first stored in an in-memory cache and subsequently mapped and unified to establish logical relationships between entities from different systems. Among other things, the following relationships are established:

- Hierarchical structure: user(Perun) → jobs (PBS) → queues (PBS)
- Mapping: virtual machines (OpenStack) and physical machines (Perun)

1.1.3 Web Presentation

The presentation of data is performed through server-side rendering of HTML pages. Only logged in users, that have approved access to Pbsmon can access the web interface. The authorization is performed by third party system using OICD token, that will be described in detail in section 1.6.

These pages display detailed views of jobs, nodes, system states, and other information relevant to both users and administrators. As mentioned earlier, the web interface is strictly read-only and does not provide any functionality for data input or modification. The frontend is not implemented as a standalone application; it is an integral part of the monolithic Java system.

One of the key features is so called **QSUB assembler**, which allows user to add required parameters for the job submission and pbsmon will notify the user about the nodes that fullfils the requirements. Among other, it produces a shell script that can be executed to submit the job to the PBS server with the required parameters

All of this will be described in more detail in section 1.7.

1.2 Data Collection from PBS Environment

The Portable Batch System (PBS) is a distributed workload management system designed to schedule and monitor computational jobs across multiple compute nodes in a cluster environment. [1]

In a typical configuration, the PBS environment consists of a central management server and a set of compute nodes, which are individual physical or virtual hosts providing computational resources such as CPUs, memory, GPUs, and local storage. [1]

A compute node (also referred to as a host) represents the fundamental execution unit in the cluster — it is the machine where user jobs are actually executed. Each node communicates with the central PBS server, which manages job submission, scheduling, and resource allocation. [1]

1.2.1 Entities in PBS Environment

PBS provides structured data that enables continuous monitoring of the computing environment operation, scheduler behavior, and resource utilization. From the PBS perspective, the following main entities are collected: [1]

Server

The PBS server is the authority for the entire cluster: it receives and registers jobs, maintains queues and nodes, tracks their states, and publishes global statistics (e.g., counts of jobs by state). It provides the following data: [1]

- **Policies and limits** – defines and enforces scheduling and resource utilization rules (limits on job and CPU counts, job array rules, rescheduling on node failure, scheduling enable/disable).
- **Resources and default settings** – manages available/assigned resources and defaults (e.g., resources_available, resources_default, default_chunk, default_queue) and provides summary utilization (assigned memory/CPU/nodes).
- **Integration with scheduler and reservations** – provides parameters for the scheduler (iteration, backfill, sorting/fairshare

formula) and supports time-based resource reservations (advance/standing/maintenance).

- **Security and access** – manages access policies through ACLs (hosts, users, managers/operators), supports Kerberos/realm policies, and handles credential management/renewal.

- **Operations and management** – configures logging, email notifications, license quotas/counters, and system version; allows management of hooks and other server-level objects.

[1]

Jobs

A job is the basic computational unit submitted by a user to the PBS server—it contains command(s) to execute and resource requirements (CPU, memory, time, GPU). The server queues it, schedules it to nodes, monitors its progress, and upon completion evaluates its result (including outputs and return code).[1]

A job can be a standalone task or a member of a job array (with multiple subjobs) sharing the same resource template. PBS provides the following data about jobs:[1]

- **Identity and ownership** – job ID and name, owner (Job_Owner), project/VO (project), target queue (queue), server, and submit host.

- **Lifecycle and scheduling** – state and substate (job_state, substate), priority, run count, holds (Hold_Types), rerunnability (Rerunable), credentials and validity (credential_id, credential_validity).

- **Requested resources and placement** – Resource_List.* (e.g., select, ncpus, mem, walltime, place, scratch_*, mpiprocs, ompthreads, nodect) and runtime identifiers (session_id).

- **Timing metrics** – ctime, qtime, stime, mtime, obittime, etime + derived indicators (e.g., eligible_time).

- **I/O and environment** – working directory (jobdir), stdout/stderr paths (Output_Path, Error_Path), submission arguments (Submit_arguments), and environment variables (Variable_List).
- **Result and diagnostics** – return code (Exit_status) and auxiliary fields for auditing and progress tracking.

[1]

Queues

A queue is a logical structure for accepting and processing jobs—it defines its type (Execution vs. Route), resource defaults and limits, access rules, and how jobs are either executed on nodes or redirected to target queues. PBS provides the following data about queues: [1]

- **Identity and type** – queue name, queue_type (Execution/Route), Priority (weight in scheduling), operational state (enabled/started), and optionally hasnodes.
- **State and utilization** – aggregates such as total_jobs and state_count, and currently assigned resources resources_assigned.* (e.g., memory, CPUs, nodes, MPI processes).
- **Policies and limits** – resource boundaries resources_max.* and minimums resources_min.* (including GPU and walltime), extra rules like kill_delay, backfill_depth, or from_route_only (accepts jobs only via routing).
- **Defaults and placement** – resources_default. (e.g., CPUs, walltime, placement, GPUs) and default_chunk.* (e.g., implicit chunk size, queue_list for targeting).
- **Routing (Route queues)** – route_destinations defines target execution queues to which jobs are automatically redirected.
- **Access and security** – ACL toggles and lists: acl_user_enable/acl_users, acl_group_enable/acl_groups, or acl_host_enable/acl_hosts.
- **Organizational tags** – optional attributes such as fairshare_tree (fairshare hierarchy) or partition (infrastructure label/partition) for logical segmentation and policy purposes.

(1)

Nodes

A node represents a physical machine in the PBS environment that provides computational resources for job execution. Each node is managed by the PBS server and can be in various operational states depending on its availability and current workload. PBS provides the following data about nodes: [1]

- **Identity and location** – node name (vnode), hostname (host), cluster identifier (resources_available.cluster), and the name of the execution daemon (Mom) that manages the node.

- **State and availability** – primary state (state) indicating the operational status (e.g., free, job-busy, down, offline) and auxiliary state (state_aux) providing additional context.

- **Available resources** – total capacity of the node defined by `resources_available.*` attributes, including:

  - CPU resources: ncpus (number of CPUs), pcpus (physical CPUs), cpu_vendor, cpu_flag (CPU feature flags)
  - Memory: mem (total memory), vmem (virtual memory), hpmem (high-performance memory)
  - Accelerators: ngpus (number of GPUs), gpu_mem, gpu_cap (compute capability), cuda_version
  - Scratch storage: scratch_local, scratch_shared, scratch_ssd, scratch_shm (shared memory)
  - Network: ethernet_speed, infiniband
  - Software: singularity (container support), os, osfamily, arch
  - Organizational: queue_list (queues accessible from this node), cluster-specific tags

- **Assigned resources** – currently allocated resources tracked by `resources_assigned.*` attributes, including:

  - ncpus, ngpus, naccelerators (number of assigned CPUs, GPUs, accelerators)
  - mem, vmem, hbmem, accelerator_memory (assigned memory)
  - scratch_local, scratch_ssd (assigned storage)

- **Active jobs** – list of job identifiers (jobs) currently running on the node, with each job identified by its ID and task index (e.g., 14964063.pbs-m1.metacentrum.cz/0).
- **Sharing and placement** – sharing mode (`sharing`) that determines resource allocation (e.g., `default_shared`, `force_exclusive`), and reservation support (`resv_enable`).
- **Timestamps** – last_state_change_time (timestamp of the last state transition) and last_used_time (timestamp when the node was last utilized).

[1]

The distinction between available and assigned resources enables monitoring of node utilization, while the state information provides insight into node availability for job scheduling. The jobs list allows tracking which specific jobs are consuming resources on each node, which is essential for resource accounting and troubleshooting.

Reservations

Reservations are a mechanism in PBS that allows nodes to be reserved for exclusive use during a specific time period. When a reservation is created, the specified nodes become unavailable for regular job scheduling. For each reservation, PBS automatically creates a dedicated queue that provides access to the reserved resources. [1]

PBS provides the following data about reservations:

- **Identity** – reservation name (Reserve_Name), unique identifier (name), and the associated queue name (queue) that is created for the reservation. [1]
- **Ownership and access** – reservation owner (Reserve_Owner) and list of authorized users (Authorized_Users) [1]

- **State information** – reservation state (reserve_state) and substate (reserve_substate) indicating the current status of the reservation (e.g., active, confirmed, or in transition). [1]
- **Time constraints** – reservation start time (reserve_start), end time (reserve_end), and duration (reserve_duration) specified as Unix timestamps. [1]
- **Resource requirements** – the resources reserved by the reservation, including:
  - Memory: Resource_List.mem (total memory reserved)
  - CPUs: Resource_List.ncpus (number of CPUs)
  - GPUs: Resource_List.ngpus (number of GPUs, if applicable)
  - Nodes: Resource_List.nodect (number of nodes) and Resource_List.select (detailed node selection specification)
  - Placement: Resource_List.place (placement policy, e.g., `free`, `exclhost`)
  - Walltime: Resource_List.walltime (maximum execution time for jobs in the reservation)

  [1]
- **Reserved nodes** – list of actual nodes allocated to the reservation (resv_nodes) with their specific resource allocations. [1]
- **Metadata** – creation time (ctime), modification time (mtime), submission host (Submit_Host), and partition information (partition, if applicable). [1]
- **Retry information** – reservation count (reserve_count) and retry attempts (reserve_retry) for tracking reservation lifecycle. [1]

Resources

Resources in PBS represent units of computational capacity that can be requested by jobs, allocated to nodes, and managed by queues and the server. Each resource is defined with a unique name and attributes that specify its data type and where it can be used within the PBS system. [1]

PBS provides a comprehensive list of all available resources in the system, where each resource definition includes: [1]

- **Resource name** – the identifier used to reference the resource in job submissions, node configurations, and queue settings (e.g., `cput`, `mem`, `ncpus`, `ngpus`, `walltime`).
- **Type** – the data type of the resource value, which determines how the resource is interpreted and validated. Common types include:
  - `long` – integer values (e.g., CPU count, GPU count)
  - `size` – memory or storage values with units (e.g., bytes, KB, MB, GB)
  - `string` – text values
  - `float` – floating-point numeric values
  - `time` – time duration values
- **Flag** – a string of characters indicating where the resource can be used or referenced:
  - `h` – can be used at the host/node level
  - `q` – can be used at the queue level
  - `n` – can be used at the node level
  - `m` – can be used at the server level

  [1]

Scheduler status

The PBS scheduler is responsible for making decisions about which jobs to run, when to run them, and on which nodes to execute them. A PBS system can have multiple schedulers, each managing a specific partition or set of resources. The scheduler continuously evaluates queued jobs against available resources and applies scheduling policies to optimize resource utilization and meet job requirements. [1]

PBS provides status information about each scheduler instance, including: [1]

- **Identity and location** – scheduler name, the host where the scheduler daemon runs (sched_host), and the partition it manages (partition).
- **Operational state** – current state of the scheduler (e.g., `scheduling`, `idle`) and whether scheduling is enabled (scheduling).
- **Scheduling cycle** – scheduler cycle length (sched_cycle_length) defining how frequently the scheduler evaluates and schedules jobs, and the current iteration count (scheduler_iteration).
- **Processor set configuration** – settings for processor set (pset) handling: do_not_span_psets (prevents jobs from spanning multiple processor sets) and only_explicit_psets (restricts scheduling to explicitly defined processor sets).
- **Scheduling mode** – throughput_mode indicates whether the scheduler prioritizes throughput optimization, and opt_backfill_fuzzy (if present) specifies the backfill optimization level.
- **Preemption settings** – configuration for job preemption: preempt_queue_prio (priority threshold for preemption), preempt_prio (queues or job types that can be preempted), preempt_order (preemption order strategy), and preempt_sort (sorting method for preemption selection, e.g., `min_time_since_start`).
- **Integration and hooks** – job_run_wait specifies the hook that controls when jobs can start execution (e.g., `runjob_hook`).
- **File system paths** – sched_priv (path to scheduler private directory) and sched_log (path to scheduler log files).
- **Logging and monitoring** – log_events (bitmask specifying which events to log) and server_dyn_res_alarm (alarm threshold for dynamic resource changes).

[1]

Fairshare metrics

A list of users, their fairshare values, and the timestamp when the record was last modified. fairshare values are used per scheduler.[1] This data is retrieved for both the QSUB assembler and user monitoring purposes.

1.2.2 Data Collection Mechanism

Batch Interface Library (IFL)

PBS provides a C library, which represents the programming interface (API) of the PBS system, also known as the Batch Interface Library (IFL). [1]

This library allows external applications and tools to communicate with the PBS server. It provides functions for remote management of batch jobs, querying the system state, and managing computational resources through TCP/IP communication. Using the library, it is possible to implement a client application that: [1]

- establishes a connection to the server (pbs_connect),
- authenticates the user,
- creates and submits jobs (pbs_submit),
- queries their status (pbs_statjob, pbs_selstat),
- modifies or deletes jobs (pbs_alterjob, pbs_deljob),
- works with information about the server, queues, nodes, or scheduler.[1]

Thus, library represents a key component for implementing a tool that enables data collection and monitoring of jobs managed by the PBS server.

Subsequently, current Pbsmon implementation contains C code that, when invoked, retrieves information from the PBS server using the library functions and then stores this data into a file, which the Java application later processes and saves into the in-memory store (see Listing 1.1).

This collection process is triggered whenever the user wants to display any Pbsmon page and when the data in the memory cache is older than 60 seconds.

Another important note is that collected data represents the current state of the PBS server. It does not include any historical data.

Listing 1.1: Partial code snippet for data collection from PBS server

```c
#include <pbs_error.h>
#include <pbs_ifl.h>

int main(int argc, char **argv) {
    // ...
    con = pbs_connect(server);
    if(con<0) {
        return 1;
    }
    /* get server info */
    bs = pbs_statserver(con, NULL, NULL);
    process_data(bs,"servers");
    /* get queues info */
    bs = pbs_statque(con, "", NULL, NULL);
    process_data(bs,"queues");
    /* get nodes info */
    bs = pbs_statnode(con, "", NULL, NULL);
    process_data(bs,"nodes");
    /* get jobs info: t - job arrays, x -
       finished jobs*/
    bs = pbs_statjob(con, "", NULL, "tx");
    process_data(bs,"jobs");
    /* get reservations info */
    bs = pbs_statresv(con, NULL, NULL, NULL);
    process_data(bs,"reservations");
    /* get resources info */
    bs = pbs_statrsc(con, NULL, NULL, NULL);
    process_data(bs,"resources");
    /* get scheduler info */
    bs = pbs_statsched(con, NULL, NULL);
    process_data(bs,"schedulers");
    /* end connection */
    pbs_disconnect(con);
    return 0;
}
```

Fairshare metrics collection

Fairshare metrics are collected using the bash command.

Listing 1.2: Metrics collection script

```
list_cache <pbsServer> fairshare{.elixir}
```

This shell command returns a CSV file with columns:
user, last_modified, fairshare.

Acl list collection

In addition to entities and caches, the shell script located on PBS server automatically pushes its /etc/group file every time it has changed and writes it to the filesystem of the Pbsmon server. It is located in the following path:

/etc/pbsmon/acl/<pbsServer> - for each PBS server, there is a separate file with the list of ACLs and their members.

1.3 Data Collection from Perun

Perun is an open-source system developed in Java that serves for comprehensive management of identities, groups, attributes, and access to various resources and services. It is a modular solution designed for efficient management of users, organizations, and projects. The system is built with an emphasis on operation in distributed environments and on integration with existing identity systems in the fields of research and education.[2]

1.3.1 Entities in Perun

Entities retrieved from Perun are two independent data domains — users and machines. Each domain is exported by Perun as a separate JSON file. In addition to that, perun collects and aftewards send etc/-group file from each PBS server. The detailed description of these entities:

Users

The user dataset contains information about all registered users in the MetaCentrum infrastructure, including their identifiers, names, organizational affiliations, and assigned virtual organizations. These data are used to enrich job statistics and to provide a link between computational activity and user identity.

Machines

In addition to management of users, Perun also collect information about physical computing resources of the MetaCentrum infrastructure. These data are then passed to Pbsmon from Perun. Data are hierachically structured and grouped by Organization -> Cluster -> Computing Node.

Each record is an institution that has at least one cluster. Cluster is a group of computing nodes that are owned by the same institution. Computing node is a physical machine that is part of the cluster.

Each computing node is described by the following metadata:

- CPU configuration

- Memory
- Storage
- Owner institution
- The list of individual node hostnames

These data are neccessary to get additional information about the complete information about the computing node for the running PBS jobs. Additionally, these data independetly from PBS are important for knowing the complete information about the computing nodes for whole MetaCentrum infrastructure.

etc/group Files

In addition to user and machine metadata, Perun is also responsible for collecting **/etc/group** files from individual PBS servers. These files are included in the export and provide information about local UNIX groups and permissions. These files are then propagated to Pbsmon.

1.3.2 Data Collection Mechanism

The integration of the Pbsmon system with Perun is designed using a PUSH model. Instead of direct access to the Perun database or invoking its API, Perun periodically generates JSON files containing all relevant information about users and computing resources. These files are then transferred via SSH directly to the Pbsmon server, where they are stored in the filesystem, and subsequently loaded and processed by Pbsmon. These files are stored in the filesystem as following:

- **/etc/pbsmon/pbsmon_users.json**
- **/etc/pbsmon/pbsmon_machines.json**

Change Detection and Synchronization

Currently, whenever user opens any page in Pbsmon, the system checks if the files were modified since last load. If they were, the files are loaded and processed by Pbsmon.

1.4 Data Collection from Virtual Machines (OpenStack)

OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed and provisioned through APIs with common authentication mechanisms. [3]

Beyond standard infrastructure-as-a-service functionality, additional components provide orchestration, fault management and service management amongst other services to provide operators flexibility to customize their infrastructure and ensure high availability of user applications. [3]

Within Metacentrum infrastructure, OpenStack is used to provide virtual machines to the users with specific needs that are not covered by the PBS environment. There are some computing nodes that are part of MetaCentrum infrastructure, but are reserved ad hoc for OpenStack. Within nodes returned by Perun mentioned in 1.3.1 are also nodes that are used for OpenStack.

Current Pbsmon implementation gets very limited information about OpenStack virtual machines for each cluster. For each cluster, there is a list of virtual machines with their reserved CPU, name and user ID.

1.4.1 Data Collection Mechanism

The integration of the Pbsmon system with OpenStack is designed using a PUSH model. Instead of direct access to the API, JSON files are transferred via SSH directly to the Pbsmon server, where they are stored in the filesystem, and subsequently loaded and processed by Pbsmon. For each cluster, there is a separate file with the list of virtual machines with their reserved CPU, name and user ID.

An example of the response structure for the OpenStack cluster named "glados" is provided in Appendix A.

1.5 Retrieval of historical data

As mentioned in section 1.2.2, the data collected by PBSmon represents only the current state of the PBS infrastructure. To access historical information about completed jobs, their resource usage, and long-term statistics, a separate application is responsible for collecting and storing historical PBS data in a PostgreSQL database.

This historical data collection system continuously records information about finished jobs, including their execution times, resource consumption, and user associations.

1.6 Authentication

The current Pbsmon solution uses third-party authentication provided by Perun's e-INFRA system. Authentication is implemented using the OpenID Connect (OIDC) protocol, which allows users to authenticate through the centralized identity provider managed by Perun. [2]

This approach ensures that only authorized users with valid credentials from the Perun identity management system can access the Pbsmon web interface. [2]

The authentication flow follows the standard OIDC protocol, where users are redirected to the Proxy IdP for login, and upon successful authentication, an OIDC token is issued and used to grant access to the Pbsmon application. [2]

1.7 Web Layer and User Interface

TBD - This is available only to the logged in users. Layout description, showing some Figure, Describing the views:

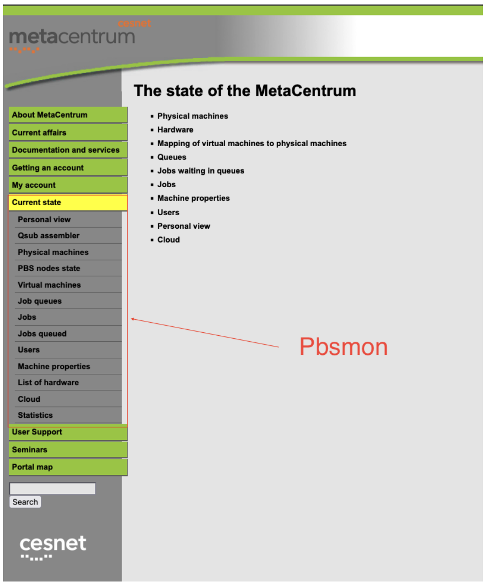

Figure 1.1: Pbsmon layout

1.7.1 Personal view

The Personal view serves as the main dashboard for users, representing one of the most important and frequently used features of Pbsmon. This view provides users with an overview of their job statistics and quick access to other relevant pages.

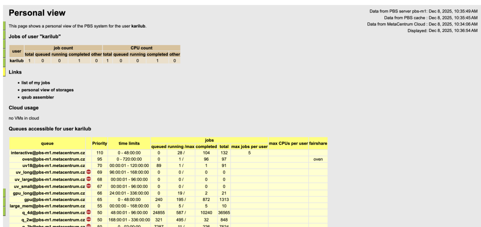

Figure 1.2: Personal view dashboard in the old Pbsmon

The dashboard displays total counts of jobs associated with the user. However, since PBS always returns the current state of the system, the Personal view focuses on showing counts of *relevant* jobs—those that are considered active. Specifically, this includes jobs that are currently in the queue, currently running, or have finished within the previous few days. This filtering ensures that users see meaningful information about their recent and ongoing computational work, rather than being overwhelmed by historical data from all past jobs.

The Personal view also contains links to other pages within the system. One such link leads to the "My jobs" page, which displays detailed information about the user's jobs. This functionality is highly relevant and could be directly incorporated into the new dashboard within the new solution, as it provides essential job management capabilities. Another link directs users to the "Queue" page, which shows information about job queues across the system. While this information is useful, it is considered less relevant for individual users and therefore does not need to be included in the new dashboard.

One limitation of the current implementation is that the dashboard only displays CPU usage and does not include GPU resources. This limitation stems from the fact that GPU usage was not a significant concern at the start of the old Pbsmon development, but has since become increasingly important as GPU computing has gained prominence in the MetaCentrum infrastructure.

1.7.2 QSUB assembler

TBD - important -

1.7.3 Rest pages

TBD - other pages

1.8 Summary of the Analysis

TBD - Strength and weaknesses

2 Design of the New Solution

2.1 Functional requirements

TBD

2.2 GUI design

TBD - using figma see appendix

2.3 Architecture design

TBD

2.4 Selected technologies

TBD

3 Implementation

3.1 System Architecture

TBD

3.2 API

TBD

3.2.1 Documentation

TBD - realized using OpenAPI, automatically generated from code, see appendix

3.3 Page views

TBD

4 Testing and Evaluation

4.1 Unit tests

TBD

4.2 Integration tests

TBD

4.3 Performance tests

TBD

5 Deployment

TBD

6 Future improvements

Conclusion

A OpenStack Response Example

This appendix contains an example of the JSON response structure for the OpenStack cluster named "glados" that is collected by the PBSmon system.

Listing A.1: Example response for OpenStack cluster "glados"


```
[
    {
        "CPUs": "40",
        "Hypervisor": "ics-gladosag-007-ostack.
           priv.cloud.muni.cz",
        "VMs": [
            {
                "CPUs": "38",
                "created": "2020-10-09T09:32:42Z
                   ",
                "instance_state": "ACTIVE",
                "name": "RationAI-node-2",
                "user_id": "1633180
                   b677608f61e96784ee5cbc608c0f4b62d@einfra
                   .cesnet.cz"
            }
        ]
    },
    {
        "CPUs": "40",
        "Hypervisor": "ics-gladosag-006-ostack.
           priv.cloud.muni.cz",
        "VMs": []
    },
    {
        "CPUs": "40",
        "Hypervisor": "ics-gladosag-003-ostack.
           priv.cloud.muni.cz",
        "VMs": [
            {
```

```
                "CPUs": "38",
                "created": "2020-01-17T10:29:22Z
                   ",
                "instance_state": "ACTIVE",
                "name": "RationAI-node-1",
                "user_id": "1633180
                   b677608f61e96784ee5cbc608c0f4b62d@einfra
                   .cesnet.cz"
            }
        ]
    },
    {
        "CPUs": "40",
        "Hypervisor": "ics-gladosag-004-ostack.
           priv.cloud.muni.cz",
        "VMs": []
    },
    {
        "CPUs": "40",
        "Hypervisor": "ics-gladosag-001-ostack.
           priv.cloud.muni.cz",
        "VMs": [
            {
                "CPUs": "16",
                "created": "2022-08-17T13:07:15Z
                   ",
                "instance_state": "ACTIVE",
                "name": "front-82e95edc-1e2d-11
                   ed-9687-0ee20d64cb6e",
                "user_id": "9
                   cd61d48508661633e261f711634b749fdc5d9fcc20769
                   .eu"
            }
        ]
    }
]
```

Bibliography

1. *Altair PBS Professional 2022.1 - Big Book* [online]. Troy (MI): Altair Engineering, Inc., 2022-07-16 [visited on 2022-07-16]. Available from: https://help.altair.com/2022.1.0/PBS%20Professional/PBS2022.1.pdf.
2. *Perun AAI: The solution for identity and access management* [online]. Perun AAI, 2025 [visited on 2025-01-27]. Available from: https://perun-aai.org/.
3. *OpenStack: The Most Widely Deployed Open Source Cloud Software in the World* [online]. OpenInfra Foundation, 2025 [visited on 2025-01-27]. Available from: https://www.openstack.org/.