Department of Physics and Astronomy
University of Heidelberg

Bachelor Thesis in Physics
submitted by

Patrick Häussermann

born in Backnang (Germany)

2018

Integration of the Slurm workload manager into the BrainScaleS monitoring platform

This Bachelor Thesis has been carried out by Patrick Häussermann at the
Electronic Visions Group in Heidelberg
under the supervision of

Prof. Dr. Christian Enss
and
Dr. Johannes Schemmel

Abstract

The neuromorphic system BrainScaleS implements physical models of neural networks. At its core are twenty wafers, comprised of so-called HICANNs that physically emulate adaptively spiking neurons and plastic synapses. The BrainScaleS computing platform can be requested by hundreds of experiments per day, making it necessary to schedule and manage submitted workloads automatically. This is done using the *Slurm workload manager* software that is widely used in HPC *(High Performance Computing)*. Slurm provides automatic scheduling and execution of submitted jobs but also implements a framework to manage available computing resources.

While most of the components of BrainScaleS are already part of an existing monitoring platform, Slurm is not. This thesis aims to remedy that and integrate Slurm into the existing monitoring platform. To this end, existing configurations were revised and periodic queries to Slurm and the cluster were implemented. These provide a wealth of information which is analyzed and stored. The existing visualization solution has been extended with the new data, providing insight into the status of Slurm and the cluster. Further development of the changes already implemented during the internship enables the display of status and usage history of licenses representing user-allocatable hardware.

These changes enable the generation of comprehensive usage reports and statistics and allow for a more proactive approach by identifying possible optimizations of the cluster and software.

Zusammenfassung

Das neuromorphe System BrainScaleS implementiert physikalische Modelle von neuronalen Netzwerken. Der Kern des Systems besteht aus zwanzig Wafer, auf welchen sogennante HICANNs ein physisches Model von adaptiv feuernden Neuronen und plastischen Synapsen implementieren. Dieses System wird öfters von hunderten Experimenten täglich genutzt, was es nötig macht diese automatisch verwalten zu lassen. Dazu wird der *Slurm workload manager* verwendet, eine Software welche im High Performance Computing weite Verbreitung findet. Slurm ermöglicht das automatische Starten und Verwalten von Experimenten, bietet aber auch Funktionen um vorhandene Computerressourcen zu verwalten.
Slurm war bisher noch nicht Teil der bereits bestehenden Überwachungsinfrastruktur. Diese Arbeit ändert dies. Es wird eine Lösung umgesetzt, welche es erlaubt Slurm, sowie auch den Cluster als ganzes zu Überwachen. Dazu mussten viele der bestehenden Konfigurationen angepasst werden. Zusätzlich werden nun regelmäßig Daten aus Slurm abgefragt, analysiert und gespeichert. Diese Daten werden ebenfalls aufbereitet und grafisch dargstellt. Dies erlaubt tiefe Einblicke in den Status von Slurm und den Cluster selbst. Änderungen die bereits während des vorhergegangenen Praktikums gemacht wurden, konnten weiter ausgebaut werden und erlauben nun auch die Darstellung von Status und Nutzungsverlauf der verwendeten Hardwarelizenzen. Die gemachten Änderungen ermöglichen es nun ausführliche Statistiken und Berichte zu erstellen.

Contents

Contents **i**

1 Introduction **1**
1.1 Motivation 1
1.2 Cluster configuration 2
1.3 Slurm workload manager 3
1.3.1 Resource management 5
1.3.2 Automated HICANN init 6
1.4 Underlying technologies and formats 7
1.4.1 JSON data format 7
1.4.2 Elasticsearch 8
1.4.3 Logstash 9
1.4.4 Filebeat 9
1.4.5 Kibana 9
1.4.6 Grafana 9

2 Implemented changes and improvements **11**
2.1 Metric aggregation 11
2.1.1 Slurm exporter scripts 13
2.1.2 Epilog script 17
2.1.3 Runtime measurements and scalability considerations . 17
2.2 Data processing 19
2.3 Grafana Dashboards 21
2.3.1 Slurm - Overview 21
2.3.2 Slurm - Node Details 22
2.3.3 Slurm - Job info 22
2.3.4 Slurm - License status 22

3 Summary and outlook **31**
3.1 Summary 31
3.2 Outlook 32

Bibliography **33**

Glossary **35**

A Configuration Files **36**
A.1 filebeat configuration (excerpt) 36

B Additional tables **37**
B.1 Script performance measurements 37
B.2 Gerrit change sets 37

Chapter 1

Introduction

1.1 Motivation

BrainScaleS, or NM-PM-1, is part the Human Brain Project and implements neuromorphic computing in a non von Neumann architecture, using custom neuromorphic hardware. As such, it takes inspiration from many fields of science, such as physics, biology, mathematics and computer science. Like any other large-scale experiment, it is dependent on long-term stability, relying on the optimal functionality of the components used.

Whether in the laboratory or in simulations, without careful monitoring of experiment conditions and parameters, problems can go unnoticed, having serious impact on results or operations.

Since May 2016, more than 340,000 experiments[1] were run on the BrainScaleS computing platform and, like any other large-scale experiment, this requires a way to manage the large amount of experiments and their associated users. To achieve this and to ensure the fair and efficient distribution of resources among users, BrainScaleS relies heavily on the *Slurm workload manager* to manage and schedule all submitted jobs. These jobs range from simple computing operations to complex simulations and experiments on neuromorphic hardware. Slurm is well-established software that is widely used in high-performance computing, running on many of the world's most powerful supercomputers.

The number of jobs submitted is increasing rapidly, with more than 41,000 jobs in September 2018 alone. Without Slurm to manage resources and schedule jobs, experimentation would come to a halt.

The aim of this bachelor thesis is to integrate Slurm into the existing monitoring solution to provide a way to reliably monitor the status of Slurm and the cluster itself. This can help to identify current problems or resource

[1] Internal usage report for September 2018, generated by E. Müller.

bottlenecks and predict and avoid future problems, keeping any unnecessary Slurm downtime as low as possible.

1.2 Cluster configuration

At the core of BrainScaleS, is a custom-made wafer consisting of 384 *HICANN*s *(High-Input Count Analog Neuronal Network Chip)*. These HICANNs can be considered silicon-based microelectronic analogs to the cells and connections found in the human brain. Implementing the biological *AdEx (Adaptive exponential integrate-and-fire)* model [1], each HICANN is able to implement up to 512 neurons and ∼115.000 synapses.

For assembly reasons, wafers are divided into 48 so-called "reticles", each containing 8 adjacent HICANNs. Every reticle is assigned its own *FPGA (Field Programmable Gate Array)* to handle communication with the rest of the setup. It is also connected to an *Analog Breakout PCB*, which allows up to 12 *Analog Readout Modules* to read out membrane voltage.

Wafers and support hardware like FPGAs, interface modules and power supplies are housed in an assembly called a wafer module. In total there are twenty such wafer modules distributed over five industry-standard 19" racks. A schematic overview of such a wafer module is provided in figure 1.

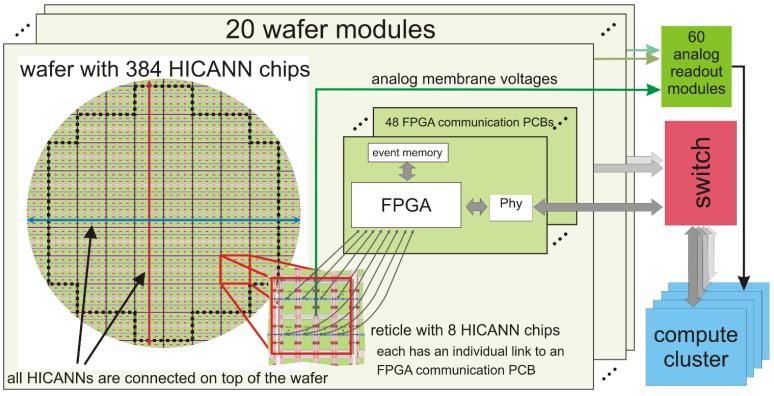

Figure 1: Schematic wafer overview. Each reticle (red rectangle) contains 8 HICANNs which implement the physical model of neurons and synapses and is assigned a dedicated FPGA for communication with the rest of the setup [2].

All wafer modules are supported by a number of additional systems, such as networking devices and compute nodes. There are twenty such compute nodes, each of which can process the bundled communication of up to 48 FPGAs. Three storage nodes provide highly-available disk storage for user and experiment data. Figure 2 shows a picture of the current setup.

Figure 2: Photo of current NM-PM 1 setup. Wafer modules are located near the top and bottom of the lateral racks. Networking devices can be seen above each wafer module. Compute and storage nodes are situated in the center of the structure [2].

The current setup also utilizes all of the compute nodes for Slurm. This way, otherwise unused or underutilized nodes can be allocated for experiments and simulations. Using already existing hardware not only helps to increase efficiency, it also allows to operate at a lower cost.

Including other systems assigned to Slurm, there are currently 31 such nodes providing a total of 244 CPU cores and more than 750Gb of memory for Slurm to manage.

1.3 Slurm workload manager

In order to run multiple experiments simultaneously and to ensure the fair and efficient distribution of resources among users and experiments, a so-called *job scheduler* is used. This job scheduler manages all submitted jobs, regardless of whether they are started directly from the command line or submitted through the web interface of the HBP Neuromorphic Computing Platform. This online platform allows registered users to submit experiments which are then executed on BrainScaleS or the ”SpiNNaker” multicore system, located at the University of Manchester.

BrainScaleS uses a customized version of the free and open-source *Slurm workload manager*. Slurm is focused primarily on Linux clusters and highly scalable, running on many of the worlds most powerful supercomputers[2]. The core of the software, the control daemon, is installed on the server `hel`. It is responsible for scheduling and managing jobs and also handles the communication with connected nodes. These nodes are then assigned experiments and other workloads for execution. Many systems of the Electronic Visions network have a compute daemon installed and are thus part of the Slurm cluster.

Jobs that are submitted by command line, i.e. started directly on `hel`, do not necessarily have to be experiments, but may contain other workloads that benefit from distributed computing, such as software compilation.

The architecture of a typical Slurm installation can be seen in figure 3. The current configuration at the BrainScaleS project does not use a backup daemon, but it does make use of the optional database backend.

This way, information about past and present jobs is saved to a MySQL database (`slurmdbd`), also running on `hel`. Slurm provides a variety of commands to access and query this database. This allows for the retrieval of job information and the generation of reports and statistics. The optional backup control daemon offers the possibility of running a second instance of `slurmctld`, the core of the software. When installed on a different server, this provides another instance to which jobs can be submitted and automatic fail-over in case one of the main control daemon goes down.

Slurm also allows for the creation of partitions as a way to group and organize different hardware or computational needs. Partitions are possibly overlapping sets of nodes and can be understood as individual job queues, each of which can have different constrains such as job length, resource usage or users permitted. All jobs submitted must be assigned to at least one partition. They are then allocated nodes from the corresponding set until available resources (CPU cores, global licenses, etc.) within that partition are exhausted. Jobs of low priority or those waiting for resources to become available are queued.

There are currently 10 available partitions. The most important are listed below:

experiment For running experiments on the BrainScaleS hardware.

simulation Used for running simulations. Can be used for "BrainScaleS-ESS", the software simulator of the BrainScaleS system.

compile Special partition to offload software compile operations onto the Slurm cluster.

[2] `https://www.prweb.com/releases/2012/11/prweb10149109.htm`, retrieved 22.10.2018

short Job runtime is limited to a maximum of 60 minutes.

dls/spikey/cube Partitions for accessing specialized hardware queues such as "Spikey", the Heidelberg single-chip system.

Due to its open-source nature, Slurm provides additional plugins to extend or provide certain functionalities. A plugin currently in use is the `job submit` plugin, which is the first thing to run after a job is submitted or modified. The purpose of this plugin is to automatically convert user-supplied hardware parameters into the underlying licenses.

These licenses represent user-allocatable hardware needed for some experiment runs, such as FPGAs for communication with the setup or *ADCs (Analog-to-Digital Converter)* to read out and digitize membrane voltages. By assigning a license to each hardware available, it is possible to include these into the resources managed by Slurm.

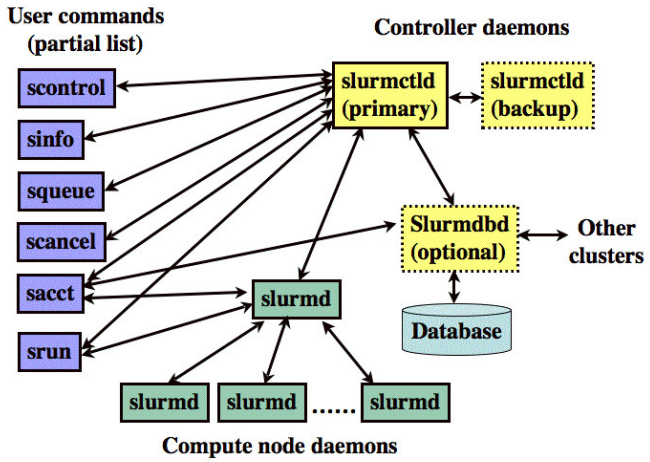

Figure 3: Architecture of a typical Slurm installation as used for BrainScaleS. The possibility for a backup control daemon is currently unused, but the optional database daemon is active [6].

1.3.1 Resource management

The resource managed by Slurm is primarily the available computing power of the cluster, but can also be available memory or the licenses mentioned above. A node usually has enough computing power or memory to run several experiments concurrently, though the exact number depends strongly on the requirements of the experiment itself and the partition used. Different partitions allocate a different amount of computing power and/or memory by

default. Licenses cannot be shared and are uniquely assigned to a single experiment or user.

Scheduling and starting of jobs is done by customizable scheduling and backfill algorithms. A quick scheduling attempt is done whenever a job is submitted or finished. More comprehensive scheduling attempts, in which a larger amount of jobs are considered, is done periodically [5].

Such a scheduling cycle attempts to get a lock on queued jobs and then tries to get the resources needed. Scheduling is done top-down, starting with the job of highest priority and going in descendant order. Once a job can not get the resources needed, the loop keeps going but just for jobs requesting other partitions [5].

Backfills are complex operations in which Slurm might allow jobs of lower priority to start early, but only if doing so won't affect the expected start time of any job with higher priority. For that, the start time of all queued jobs is periodically estimated and possible openings are filled with jobs whose parameters or partition defaults would allow them to be slotted in. Allowing jobs to be backfilled usually results in a significantly higher utilization as these would otherwise be started in strict priority order.

In order to help manage hardware-licenses, several existing provisions like the `job submit` plugin or `prolog` and `epilog` scripts are already in place. These have been extended or reworked during the preceding internship to provide features used for license management and automated init. A short summary of the automatic initialization is provided in the next section.

1.3.2 Automated HICANN init

The automated initialization of HICANNs was the main goal of the preceding internship and enables Slurm to automatically initialize HICANNs directly adjacent to those used by an experiment. Some of the changes made are now the basis for further work and improvements.

HICANNs or more precisely the connecting L1 bus lines have to be initialized in software, since there are no provisions for that on the hardware side. This leads to a random voltage on the L1 bus lines after power-up, which can influence adjacent HICANNs used by other experiments. Initialization is done using the same licenses used to exclusively allocate the FPGAs needed for communication with the HICANNs. Each license represents one FPGA, i.e. 8 HICANNs. At the time of the internship only whole licenses and therefore all 8 associated HICANNs could be initialized. By now, individual HICANNs can also be initialized.

Changes made to the `job submit` plugin allow Slurm to determine HICANNs adjacent to those used by an experiment and make the corresponding licenses available as environment variable. Several scripts then use these env vars to manage and initialize the licenses used. Management is done with help of the same MySQL database used by Slurm. There, license name, status and a timestamp are saved. The timestamp is updated whenever a license status was changed. Licenses used by experiments are automatically set "dirty", while initialized neighbours are set "clean".

Since the automated initialization is intended to save time, only licenses that are marked "dirty" or have a timestamp longer than 24 hours in the past are initialized by default. The value of 24 hours was determined empirically and represents a cutoff after which outside influences or voltage fluctuations make it prudent to consider a license to be "dirty".

Users can overwrite the default behavior with the help of two new command line parameters: These allow to either skip the init completely (`--skip-hicann-init`), or force all neighboring HICANNs to be initialized, regardless of their status (`--force-hicann-init`).

Revisions made while implementing the new management features resulted in a much faster execution speed of the important `prolog`-script which is called prior to every job or experiment. In a typical usage scenario, the new script is about three times as fast as the old one. A table with detailed measurements can be found in the appendix (Table B.1).

1.4 Underlying technologies and formats

This section provides a brief overview of the technologies and formats used for the work done in this thesis. The basis for some of the implemented changes and additions is a monitoring solution set up by Daniel Kutny as part of his internship and bachelor thesis [3].

In order to expand the existing solution and include Slurm in the monitoring, numerous changes to the existing programs and configurations have been made. Some of the programs and data formats used are briefly explained below, see [3] for a more detailed description of the installation.

1.4.1 JSON data format

JSON or *JavaScript Object Notation* is a human readable data format derived from JavaScript and is often used for serializing and transmitting structured data. This format is popular because it is light-weight and easy for people to read and write.

A JSON object consists of a pair of curly brackets (braces) "`{`" and "`}`", which contain one or more unordered name-value pairs. Name and value are separated by a colon character "`:`". Names and string values have to be double-quoted, e.g. `{"firstName": "John"}`. Additional name-value pairs are separated by a comma "`,`". Ordered lists (arrays) are supported by surrounding comma-separated values or objects within square brackets. Listing 1.1 shows a valid JSON object [4].

```
{
  "firstName": "John",
  "lastName": "Doe",
  "Address": {
    "Street": "253 Example Drive",
    "City": "Metro City",
    "State": "Nevada"
    },
  "active": true,
  "membershipnumber": 357325,
  "tags": [ "admin", "editor", 99 ]
}
```

Listing 1.1: A valid JSON object showcasing the easy readability of this format. Spaces and newlines between objects aren't necessary but help readability.

All software explained in the following sections already supports the JSON data format, either on the input/output side, or internally. This simplifies many steps in data processing making it the most obvious choice.

1.4.2 Elasticsearch

Elasticsearch is the underlying software used for what is commonly called an "Elastic Stack". This is a popular logging and analytic platform and also the basis for the current monitoring solution. It allows to reliably input and store data from any source and any format. This data can be searched, analyzed and visualized in real time. Although elasticsearch is only one of three programs typically used in Elastic Stack, it is so popular that it has become synonymous with the name of the stack itself.

Elasticsearch stores data in an unstructured way ("NoSQL"), meaning that it is stored in a huge number of documents, rather than individual tables and rules linking those tables together. This enables full-text search in where every single document stored is searched for a given search phrase, similar to a keyword search within a PDF or text document.

Input data and documents that have similar characteristics can be indexed under a common name, allowing for a more narrowly constrained search. This function is currently used to distinguish between different types of logs

that are entered into the database, e.g. `wafer` containing information about wafer allocations, or the newly added `slurmstats`, `slurmtop` and `jobinfo` pertaining to Slurm.

Although elasticsearch is not ideal for storing a continuous stream of name-value pairs ("time series data"), it is extremely fast, even on such massive scales seen at *CERN* [8]. Its speed and ease of use make it one of the most widely used logging solution in IT environments.

1.4.3 Logstash

Logstash is another component of a typical "Elastic stack" and can be thought of as data processing pipeline. A configurable input listens for incoming data to which a series of filters and conditions is applied. It is then output as JSON document, typically to elasticsearch for indexing. Examples of such a pipeline can be found in listing 2.4.

Logstash allows for arbitrarily complex processing pipelines which can handle almost all data formats. However, such complex pipelines should generally be avoided, if only for reasons of speed.

1.4.4 Filebeat

Filebeat is a daemon that can be configured to watch for file or directory changes. New files or file entries are automatically forwarded to either logstash for more advanced processing or directly into elasticsearch for storage. Filebeat keeps track of files and lines already parsed and slows down or stops if it detects a problem on the ingest side. This allows indexing to continue automatically after a network failure or problem with the logstash pipeline.

1.4.5 Kibana

Kibana is the last of the three programs that make up a "Elastic Stack". It is a browser-based tool for visualizing and real-time analysis of data in elasticsearch. However, it is currently only used to manage elasticsearch. Visualization is done by a more powerful tool, Grafana.

1.4.6 Grafana

Grafana is the main tool used to visualize data at the BrainScaleS. It is web-based and structured as a series of dashboards that can be configured to visualize data using a variety of plugins and graphs. These range from simple tables and graphs to complex heat maps and carpet plots. A feature that distinguishes Grafana from similar programs is the large amount of data sources it supports. With Grafana it is even possible to directly query the database used by Slurm, an option that was made use of in this bachelor thesis.

Chapter 2

Implemented changes and improvements

This chapter describes the changes and additions made that allow for the inclusion of Slurm into the existing monitoring solution. A flowchart showing the interactions between the software used to monitor Slurm can be found in figure 4.

Most configurations or scripts are only shown as excerpts as some of these span several hundred lines of code. Files can either be found in the appendix or are available on the internal *gerrit* code collaboration tool. A table containing all Change-IDs can be found in the appendix, section B.2.

To collect and process the information gathered by Slurm, several new scripts were written. Data aggregation is described in the following section, while the scripts themselves are covered in section 2.1.1. In order to visualize the collected information, it has to be processed first. For that purpose it is forwarded by filebeat to the server `monviz` where it is processed by logstash. This is explained in section 2.2. After processing it is automatically indexed into elasticsearch. Finally, Grafana accesses elasticsearch and MySQL to visualize this data. The new dashboards created for this purpose, as well as the information displayed is explained in detail in section 2.3.

2.1 Metric aggregation

While the existing solution uses the well-established *syslog* format for input into logstash, this is not sufficient here. The amount of information generated by the newly added scripts, as well as the type of data gathered would make a complex logstash configuration necessary. To avoid writing such a highly specialized processing pipeline, especially in consideration of robustness and easy expandability, all data gathered by the new scripts is directly formatted as JSON documents. Since logstash already works internally with JSON, a

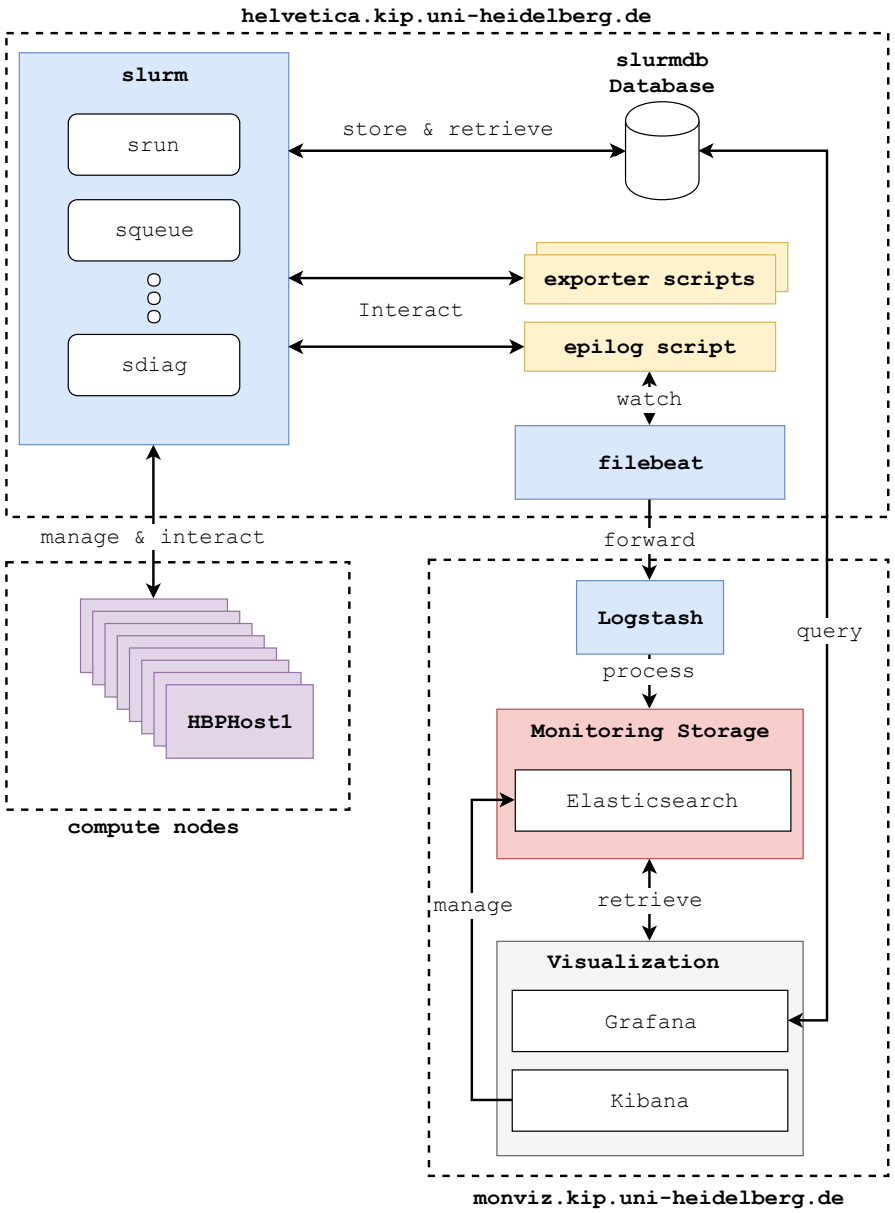

Figure 4: Flowchart showing the programs and their interactions that allow the monitoring of Slurm. Dotted rectangles represent different systems.

few basic filter operations are sufficient to process the new documents. This additional input format required changes to all configuration files as well as a new logstash pipeline. However, the necessary restructuring of the logstash pipeline has the advantage that additional data sources can be configured by simply adding a new `if-then` block for data processing.

Monitoring of Slurm and its connected nodes is done with the help of two exporter scripts, `slurmstats.sh` and `slurmtop.sh`. Both periodically query the local Slurm installation using Slurm-specific commands.
A third script already in use has been extended. This `epilog`-script is automatically executed by Slurm at the end of a job run. It now logs additional details about the job from within it was called.

All scripts write to files in `/var/log/elasticsearch`, with the filename depending on the type of script:

`slurmstats.json` Output from the main script. Contains information about the health of the cluster, current activity and details about the job queue.

`slurmtop.json` Primarily statistical information such as job numbers and user information.

`jobinfo.json` Output from `epilog.sh`, containing selected information about the job it was started from.

The directory is monitored by filebeat and all logs are forwarded to logstash for processing before they are indexed into elasticsearch. To distinguish the new JSON-formatted data from already occurring inputs into logstash, filebeat now tags forwarded data from these three files with additional fields (see Listing A.1). Logstash then decides on the presence of these fields which filter and processing routine is applied to the incoming data (section 2.2).

2.1.1 Slurm exporter scripts

Both exporter scripts are designed to be run via *cron*. While the first script is designed to run every minute, the second should only run hourly, as it contains some fairly time-consuming database queries. This might otherwise needlessly slow down the Slurm installation in times of high load.
The scripts can be run under any user, as long as it is permitted to execute Slurm commands like `sdiag` and has sufficient rights to write the output data to directories watched by filebeat.

The scripts are structured to make a series of queries to different Slurm functions, the output of which gets parsed, processed and formatted before it is written to disk. This is the simplest and most robust way to collect information, as it uses Slurm's own functions to extract data. This is helped by the fact, that some of the commands used allow output in a format that is either already similar to JSON, or easily parsable.

All information is collected in separate JSON objects (see 1.4.1) but output as a single document. This separation helps readability and makes it easier

to expand functionality if necessary. Additionally, all calls to Slurm are made in such a way, that erroneous or empty replies do not impact the execution of the rest of the script, or invalidate the JSON output. Listing 2.1 shows an excerpt from the first script in which a very simple operation queries and formats aggregated node information.

To minimize overhead, most of the numerical data (e.g. scheduler statistics or load information) is combined into JSON arrays, which will then be split into individual fields by logstash (see section 2.2). Similarly, only non-zero values are output for node and partition information. This can shrink document size considerably during nighttime or times of low activity.

All scripts have a modular structure and are commented in great detail. This makes them easy to understand and expand in the future.

```
# output aggregated cpu and node states as array to
# minimize overhead.
# Array has to be split up and assigned to fields by logstash
__get_cluster_info() {
    local temp
    # get CPU and Node states which are in the form
    # "idle/allocated/down/total".
    # Use another "/" to separate the CPU and Node part.
    # This way...
    temp="$(sinfo -h -o%C/%F)"
        if [[ "${temp}" ]]; then
    # ... we can simply replace all "/" with comma to build the
    # whole JSON array
    loadinfo="[${temp//"/"/,}]"
fi
}

[...]

__get_node_info
# cluster load
if [[ ${loadinfo} ]]; then
    payload+="\"loadinfo\": ${loadinfo}"
fi
```

Listing 2.1: An excerpt from the first script showing a very basic example to get aggregated information.

Slurmstats script

The first script collects most of the data displayed in Grafana. It is responsible for retrieving and outputting information about the Slurm installation itself as well as all connected nodes and the job queue.

Calls to `squeue` are used to get a list of all current jobs and their status (running, pending, etc.). This output gets parsed and the total of each state is calculated. After that, `squeue` is called two more times, but with different parameters each time. This provides important information about currently running jobs and the size and shape of the job queue.

Queries to `sinfo` allow for the retrieval of the current status, CPU load and memory usage of all connected nodes. Another call is made to collect the same information in aggregated form, providing an overview of the whole cluster.

The last metrics are collected by calling `sdiag` which outputs general diagnostic information for Slurm. The output mainly consists of internal statistics such as current scheduler length or cycle duration, but also contains information about the total number of jobs submitted, started, completed or canceled so far. These values are important for assessing the health of the Slurm cluster as they directly reflect the performance of Slurm and its scheduler. This can also help with troubleshooting and optimizing performance. All of these stats reset daily at midnight UTC.

The following listing shows output from the first script. For reasons of brevity only a few of the 31 hosts are shown. The large arrays at the end of the file contain all of the aforementioned diagnostics and load information in compacted form to minimize overhead. These arrays have to be split up and assigned to individual fields by logstash.

```
{"timestamp": "2018-11-02T15:57:01+0100", "AMTHost1": {"state":
    "idle"}, "AMTHost3": {"state": "idle"}, "AMTHost11":
    {"state": "down*"}, "AMTHost12": {"state": "idle", "load":
    2}, "AMTHost13": {"state": "idle"}, "AMTHost14": {"state":
    "mixed", "mem": 3740}, [...]
"partition": {"dls": 1}, "jobqueue": {"completed": 2, "running":
    1}, "pendingjobs": {"simulation": 1}, "schedinfo":
[3,0,161,161,128,31,0,28,1806,1905,52,0,2,0,1,1,918,918,918,1,1,1],
"loadinfo": [2,210,32,244,1,26,4,31]}
```

Listing 2.2: Sample output (abridged) from the first script. This script gathers node and partition information as well as scheduler and load statistics.

Slurmtop script

This script collects accounting information and statistical data. It is meant to be run only once per hour as it contains some fairly time consuming queries. This is sufficient, since all statistics created only report daily or monthly trends.

The script starts off by using `sreport` to display the top 5 users (by CPU hours used) of the day. It then uses this information to query the job count, i.e. the amount of jobs started by each of these users. Similar statistics are generated for month-to-date.

Additionally, for each of the top 5 users the distribution of allocated CPU cores is determined. Once on a per-job basis, but also on basis of total CPU hours spent. This is done since these two stats can differ – running the most jobs does not necessarily mean that the most CPU hours are consumed. A similar report showing the distribution of CPU cores among all jobs is also generated. These statistics allow to determine the average amount of CPU cores requested by jobs and the resulting CPU hours used. This can help to adjust partition defaults to more reflect real requirements. See figure 7 and 11 for examples of these statistics.

Finally, the total number of jobs started for the current month is determined. Depending on the date, this can be an extremely time-consuming query and is therefore further restricted to run only once at 4 am.

Data gathered by this script is shown in listing 2.3. It is mainly statistical in nature and is meant for the generation of detailed usage reports or to get an overview of cluster usage and utilization. However, this data, together with the data collected by the first script, can also help to optimize certain parameters concerning scheduling or partition defaults.


```
{"timestamp": "2018-10-31T04:10:01+0100", "cpuusage": {"0-1": [0,
    0], "2-3": [41, 144], "4-7": [1, 0], "8-15": [27, 6], "16+":
    [0, 0]}, "today1": {"name": "jgoeltz", "hours": 144, "jobs":
    18}, "today2": {"name": "vis_jenkins", "hours": 7, "jobs":
    51}, "month1": {"name": "jgoeltz", "hours": 6492, "jobs":
    1002}, "month2": {"name": "vis_jenkins", "hours": 1961,
    "jobs": 5028}, "month3": {"name": "bcramer", "hours": 1486,
    "jobs": 3373}, "month4": {"name": "fkungl", "hours": 742,
    "jobs": 314}, "month5": {"name": "bq402", "hours": 622,
    "jobs": 8199}, "cpuutil": [223,128,0,625,0,976], "jobsmtd":
    52070}}
```


Listing 2.3: The second script outputs daily statistics, such as CPU allocations of jobs, load information, user and job statistics as well as accounting information.

2.1.2 Epilog script

The `epilog`-script is already in use by Slurm and had to be extended to collect data allowing the display of license usage history. Previously it was only used for logging and network configuration.

When called at the end of a job run, it now executes `squeue` to query details about its own Job-ID (`$SLURM_JOB_ID`). Information currently gathered includes runtime, user, the command or script executed and licenses used. This can be easily extended to include further parameters if required.

Similar to the exporter scripts the output of `squeue` gets parsed, formatted as JSON document and written to a file indexed into elasticsearch. This enables display of licenses usage history and other details about past jobs. By connecting Grafana to the existing MySQL installation, it is now also possible to query and display the license status in real time.

It is worth noting, that unlike the other two scripts, the `epilog`-script is not meant to be run periodically. It is part of a series of scripts that are automatically called by Slurm at set points during job execution. This `epilog`-script is called just before a job terminates [7].

2.1.3 Runtime measurements and scalability considerations

In order to estimate the scalability of the current solution, the runtime of all three scripts was measured. These measurements are provided in table 1 and were made using the Linux `time` command. The first two scripts `slurmstats.sh` and `slurmtop.sh`, are the exporter scripts used to periodically query and format data from Slurm. The third script, `epilog.sh` is automatically called by Slurm at the end of every job to output additional data about the job it was started from.

Averages were calculated over a total of 20 executions of each script. Minimum and maximum values emphasize the impact of load fluctuations caused by other programs. The median is provided to give a better idea of values that can be expected without these disturbances.

In light of future expansions with potentially hundreds of wafers, considerations about the scalability of the current metric aggregation should be made as well. Although the two exporter scripts were designed with speed in mind and the epilog script revised accordingly, script runtime can possibly improved further. However, runtime is generally load-dependent and while `hel` the server running Slurm is quite powerful, there are a large number of other processes and users which can influence script execution. This can overshadow any possible speed improvements.

Script execution is generally very fast. The script started every minute, `slurmstats.sh`, takes on average only 336 ms to finish. Additional fine-tuning of the data parsing and formatting done might help to further speed up runtime. The second script, `slurmtop.sh` contains some time-consuming database queries and as such, expected runtime is much longer than the first script. Since this script is only run every minute, the average runtime of ∼17 seconds is within reason. This cannot be improved much further, as the limiting factor are the database queries. The `epilog.sh` script has an average runtime of only 17 ms. This is good, because it is the only script that is not scheduled to run, but is executed on demand by Slurm. Depending on the workload, this script is potentially called several times a second.

In terms of data generated, not all three scripts are affected by scaling up the amount of wafers. In general, the data output of only two scripts scales (linearly) with the amount of wafers or compute nodes available, namely the `slurmstats.sh` and `epilog.sh` script. The third script, `slurmtop.sh`, outputs only statistical information and is thus unaffected by any kind of scaling. To be a little more precise, only the amount of compute nodes that are managed by Slurm (see figure 3 for a typical Slurm architecture) influences the data generated by the new scripts directly.

This is because the first script (`slurmstats.sh`) outputs information like status and CPU load of every connected node. As such, the script generates marginally more data the more nodes are connected.

The `epilog.sh` only indirectly generates more data. More wafers generally means more experiments or more wafers used per experiment and, since this script outputs information at the end of every experiment, it generates more data the more experiments are carried out.

In case of the first script affected by scaling, `slurmstats.sh`, more nodes managed by Slurm means more status information of connected nodes is output. Such an entry consists of the node name, its state, CPU load and memory usage but is generally only 60 bytes in size.

Calculating the additional data generated for a hypothetical cluster with **ten times** the amount of compute nodes currently used (310 vs 31), yields: 1440 script executions per day $\times$ 310 nodes $\times$ 60 Byte per node $\times$ 30 days $\approx$ 800 Mb of data per month. For a cluster utilizing over 300 computing nodes, this is negligible in terms of both storage and network bandwidth.

The `epilog.sh` script outputs roughly 350 byte of information per job. Scaling up the current amount of jobs similar to the example above, this gives roughly 15,000 jobs per day. As such, 15000 jobs per day $\times$ 350 Byte per job $\times$ 30 days $\approx$ 160 Mb of data are generated each month. About one-fifth the amount of `slurmstats.sh`.

Therefore, neither the amount of data generated nor the script runtime should pose a problem for the scalability of the current solution.

Script	Runtime (ms)			
	Average	Min	Max	Median
`slurmstats.sh`	336±15	261	470	374
`slurmtop.sh`	17183±828	15169	26705	16272
`epilog.sh`	17±1	12	29	15

Table 1: Runtime measurements for $n = 20$ script executions as measured by the Linux `time` command.

2.2 Data processing

Log files from all three scripts are forwarded by filebeat to logstash for processing. During processing, the timestamp of the log entry is adopted as the internal timestamp used by elasticsearch. This ensures that all indexed documents use the correct timestamp, even in case of intermittent network outages or problems preventing files from being indexed temporarily.

Other filter operations are dependent on the source of the data. From which script the data originated is determined by the presences of optional fields tagged on by filebeat (`{"slurmstats": true}`, `{"slurmtop": true}` or `{"jobinfo": true}`). These field names correspond to the filename the data was read from.

Since all scripts already output and write JSON-formatted data, processing can be kept relatively simple. It mostly consists of assigning elements of arrays to individual fields. This is done because most scheduler and load statistics are consolidated into a few big arrays to reduce overhead, see listing 2.2 for an example. Every field created this way then also has to be assigned a type, e.g. `integer` or `string`. Therefore, assigning an element to a field requires two operations. This is no problem however, as these kind of operations are extremely fast and the number of elements is manageable (~ 60).

Output, or more precisely, the name under which the document is indexed into elasticsearch also depends on the source of the data. It follows the scheme "*fieldname-YYYY.MM.dd*", where *fieldname* is either one of the three field names mentioned above, or *wafer* in case none of the fields are present (as is the case for input from some already existing data sources). *YYYY.MM.dd* represents the four-digit year, two-digit month and two-digit day notation.

This naming convention follows elasticsearch's best practices and allows to restrict search queries to indices representing certain days or time periods.

Listing 2.4 shows a sample logstash pipeline using operations that are used in the actual configuration file.:
Logstash listens for input from filebeat and if the input document contains a field *slurmstats*, a filter is applied. In this example, the date from the field *timestamp* is read and used as the internal indexing timestamp of elasticsearch. The field is then removed and the whole document is output to elasticsearch. The index (here: *slurmstats* or *wafer*) under which it is saved depends on the type of log, as determined by the presence of the *slurmstats* field.

```
# a sample logstash configuration

# listen for input from filebeat
input { beats { port => "5044" } }

# apply filters
filter {
    # apply only to data containing a field named "slurmstats"
    if [slurmstats] {
        # set elastic date to timestamp the entry was written
        date {
            "match" => ["timestamp", "yyyy-MM-dd'T'HH:mm:ss+ZZ",
            "yyyy-MM-dd'T'HH:mm:ss+Z", "ISO8601"]
        }
        # remove field "timestamp"
        mutate { remove_field => ["timestamp"]
        }
    }
}

# output to elasticsearch. Index depends on the type of log
output {
    if [slurmstats] {
        # field present, save as slurmstats
        elasticsearch { index => "slurmstats-%{+YYYY.MM.dd}" }
    } else {
        # field missing, different type of log, default to wafer
        elasticsearch { index => "wafer-%{+YYYY.MM.dd}" }
    }
}
```

Listing 2.4: A simple logstash configuration showing a pipeline to process incoming log files depending on the presence of a field (here: *Slurmstats*).

2.3 Grafana Dashboards

Several new dashboards have been added to visualize the wealth of newly available data and tying into work done during the preceding internship, additional dashboards also show license status and job information. Dashboards in Grafana are customizable web pages to which panels can be added. These panels are used to visualize data from different sources and range from simple tables and graphs to complex heat maps and carpet plots.

Most of the panels that were added have additional information about the associated metric or parameter, which can be viewed by hovering over the little ”i” in the upper left corner of the graph. The following sections provides an overview over all added dashboards.

2.3.1 Slurm - Overview

The first dashboard (figures 5 through 9) displays details about the status of Slurm. It is designed to make important data and metrics visible at first glance.

The top row shows information regarding the health of Slurm and the cluster. Gauge-style graphs show the total resource utilization (CPU cores and memory available) of the cluster. Other panels show node states, the current job queue length as well as several parameters regarding the status of Slurm itself.

The second row (figure 6) displays more detailed information about the job queue, including graphs showing the partition usage over the last 3 hours and details about the partitions used by currently running jobs or requested by jobs queued.

Activity and usage statistics are displayed on the third row. Activity (measured by jobs submitted) is displayed using a third-party plugin ”carpet plot”. Each bin represents a 15-minute average of jobs submitted. Several tables display the top 3 users of the day and top 5 month-to-date with the amount jobs submitted and CPU hours consumed. Another table shows information about the total of jobs submitted, completed or canceled.

The last row (figure 9) displays several graphs of specific diagnostics information regarding the scheduling and backfill operations done by Slurm. These provide insight into the state of the installation as they contain statistics that directly reflect the performance of Slurm and its scheduling algorithm. They can also help to adjust configuration parameters or queue policies for optimal performance.

2.3.2 Slurm - Node Details

The second dashboard (Figure 10) displays detailed information about all connected compute nodes. A simple ”traffic light” provides information about the status of each node (green – *idle*, yellow – *allocated*, red – *down*).

Utilization history for the last 3 hours and gauge-style graphs showing current cpu load and memory utilization provide good insight into the current workload of each node. This could be used in conjunction with data from the first and third dashboards to identify partitioning bottlenecks where there are not enough nodes associated with frequently used partitions.

2.3.3 Slurm - Job info

This dashboard contains information and statistics pertaining to recently completed jobs (Figure 11). Graphs at the top show the distribution of allocated CPU cores of past jobs. Once on a per-job basis and once on basis of total CPU hours used. This allows, for example, to gauge the CPU usage of typical experiments and adjust default values accordingly.

The table below displays information about the last 15 jobs completed. It shows the most important parameters such as JobID, username, runtime and licenses by directly displaying the retrieved JSON object. This can easily be expanded to show other information available.

2.3.4 Slurm - License status

The last dashboard implemented (Figure 12) shows the init-state of all licenses. Usage history of licenses is shown in tabular form as well as a graph. This dashboard is complete but is still waiting for a live data feed that will become available with an already planned upgrade of Slurm.

Display of license status follows the same rules as the query for ”dirty” licenses done by the automatic HICANN init: Licenses are considered ”dirty” i.e. in need of initialization, if they are either marked as such or their initialization date lies more than a day in the past (see subsection 1.3.2).

Figure 5: The first row is designed so that the health of the Slurm cluster can be assessed immediately.

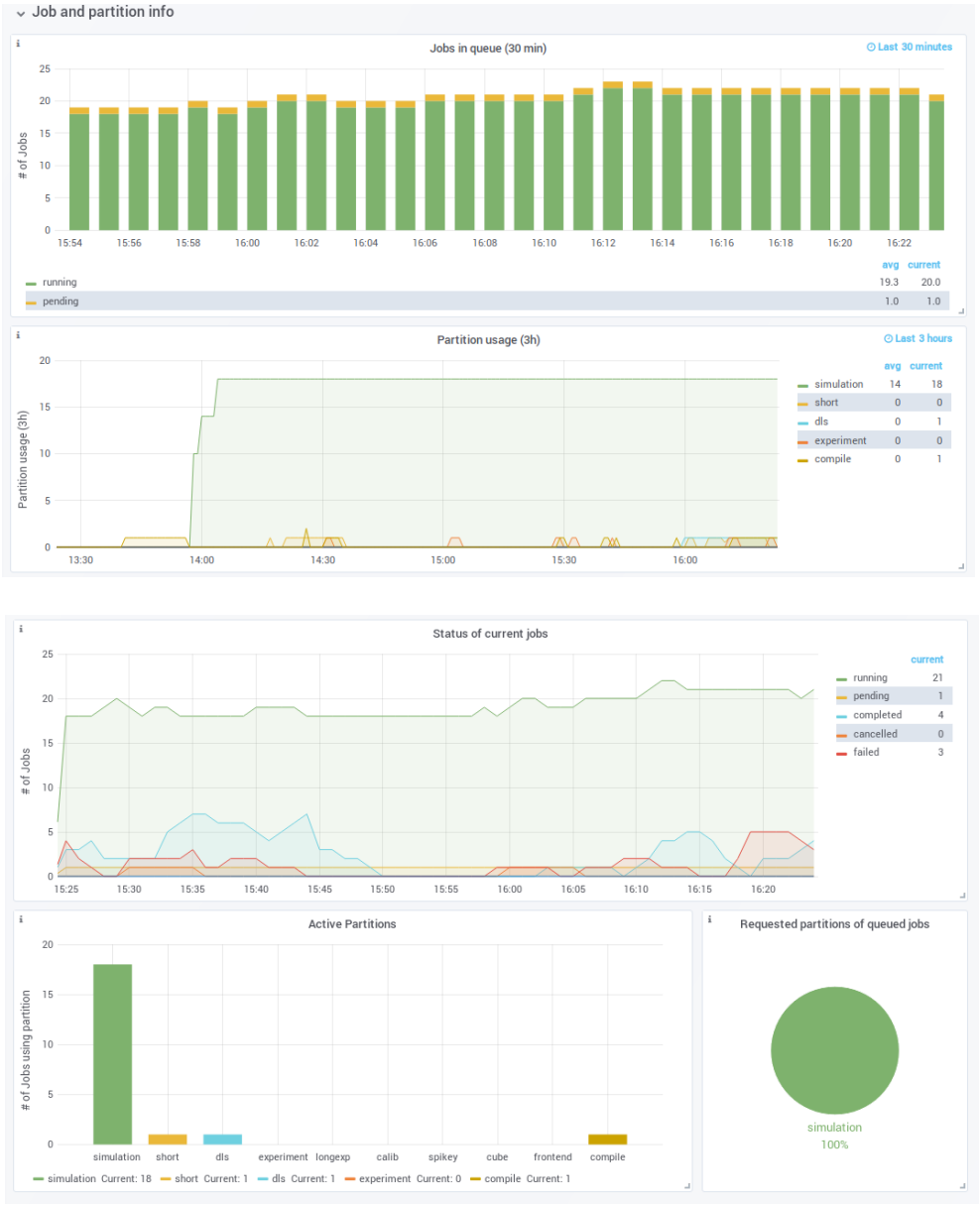

Figure 6: The second row shows detailed information about the current job queue and partition usage.

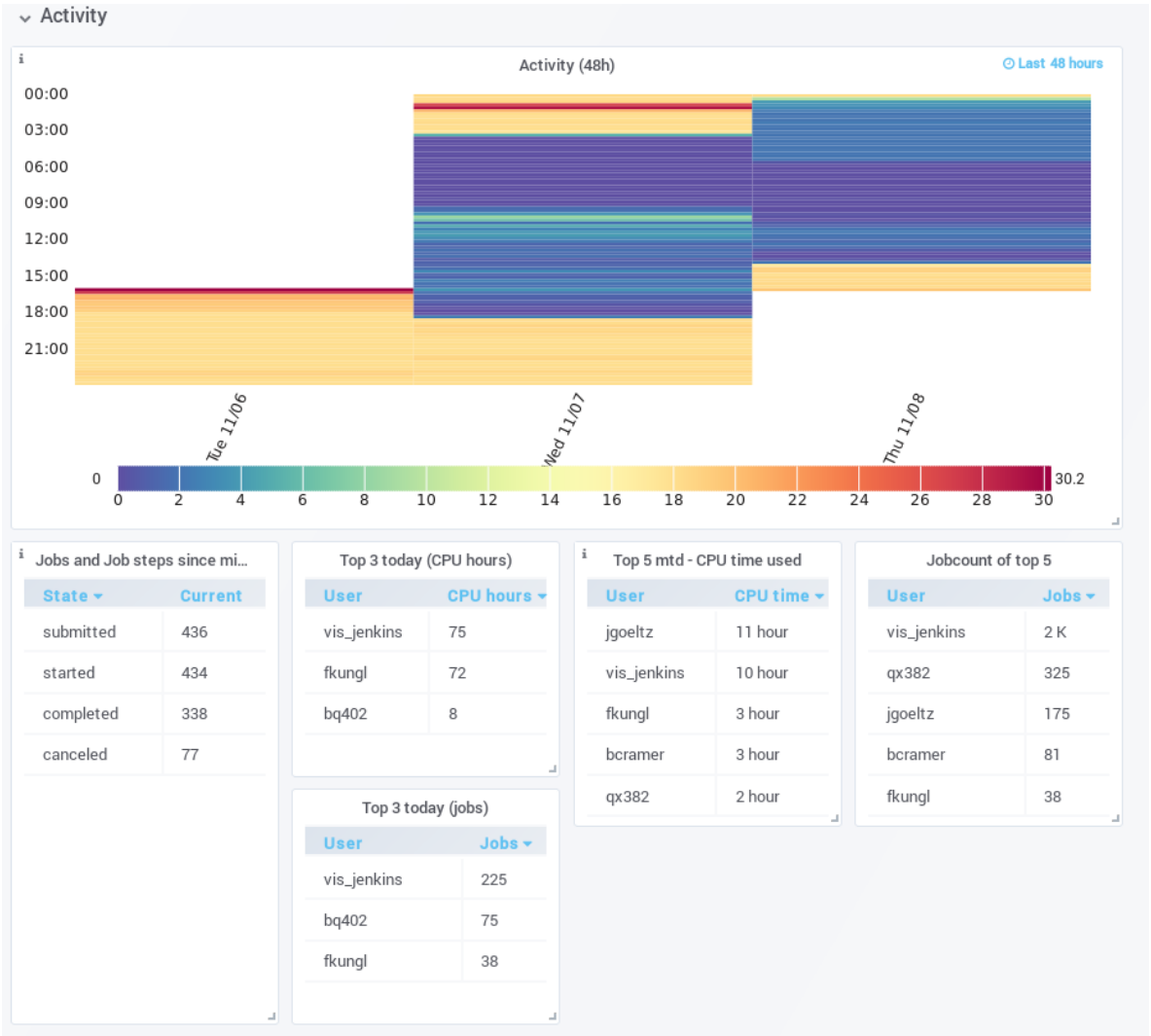

Figure 7: Activity display and "top-style" user statistics. Activity is measured by jobs submitted, averaged in 15 minute bins. Displayed user statistics are for the day so far as well as month-to-date.

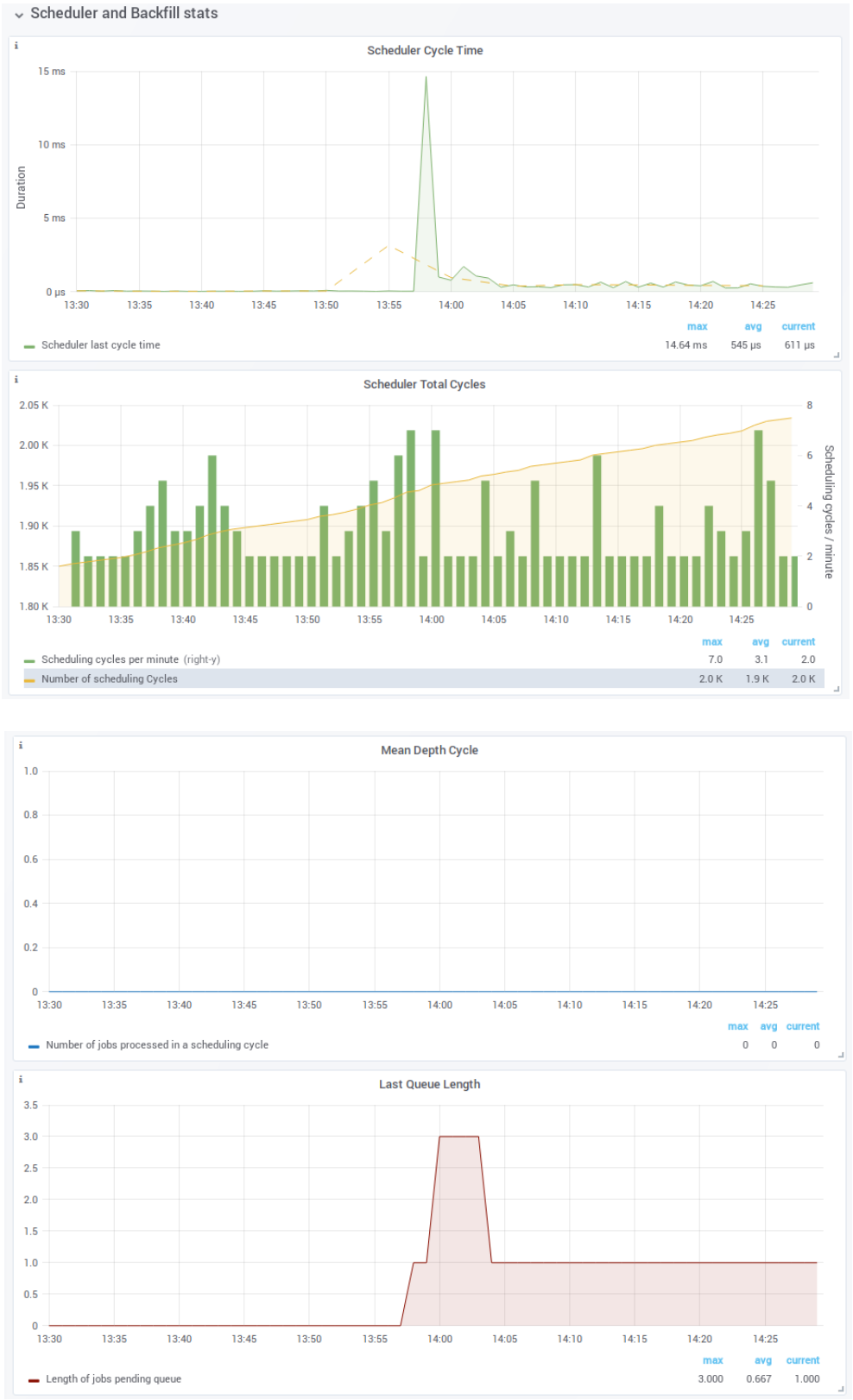

Figure 8: Internal statistics of Slurm, such as scheduler, backfill and job queue information.

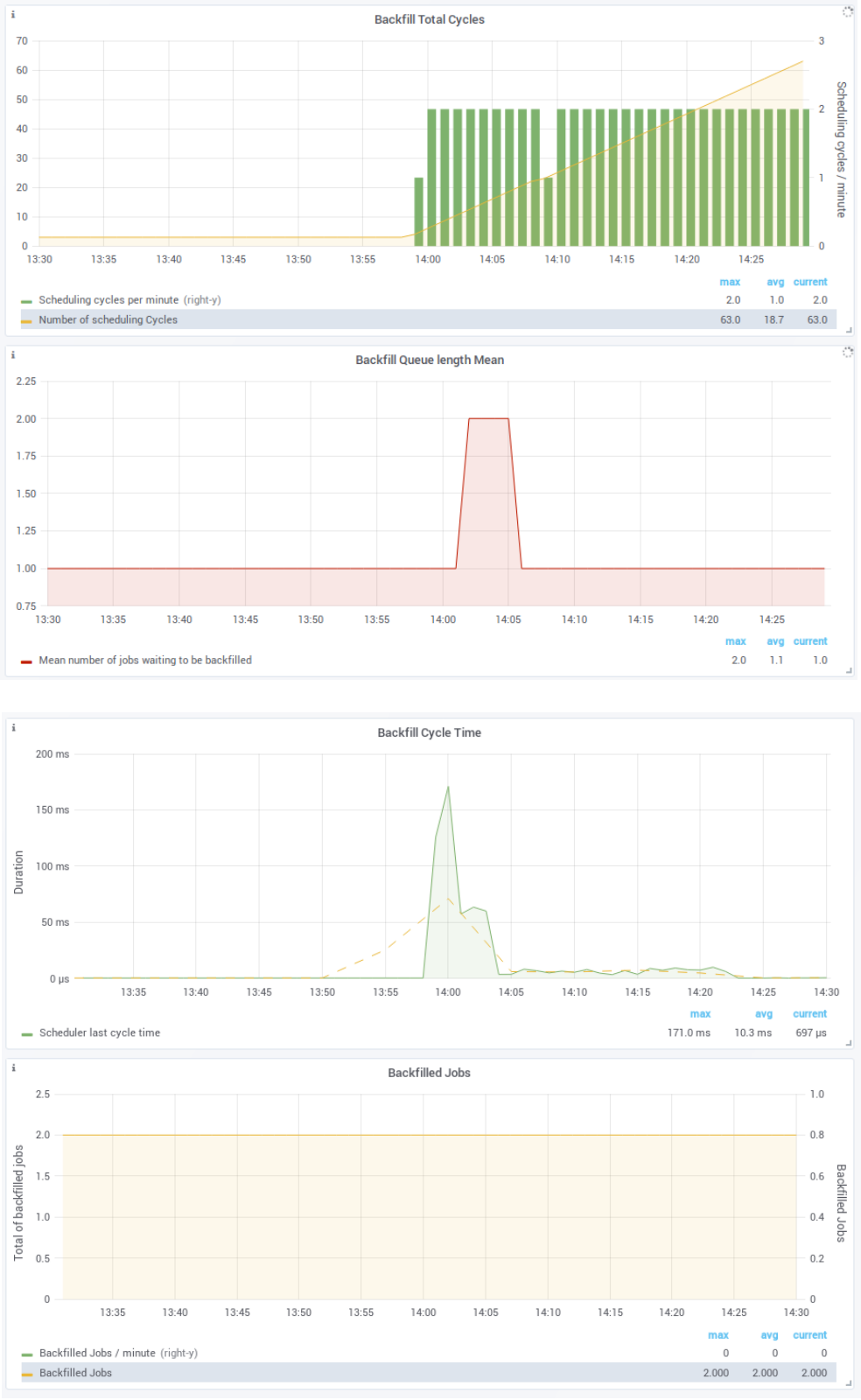

Figure 9: Internal statistics of Slurm, such as scheduler, backfill and job queue information. Continued.

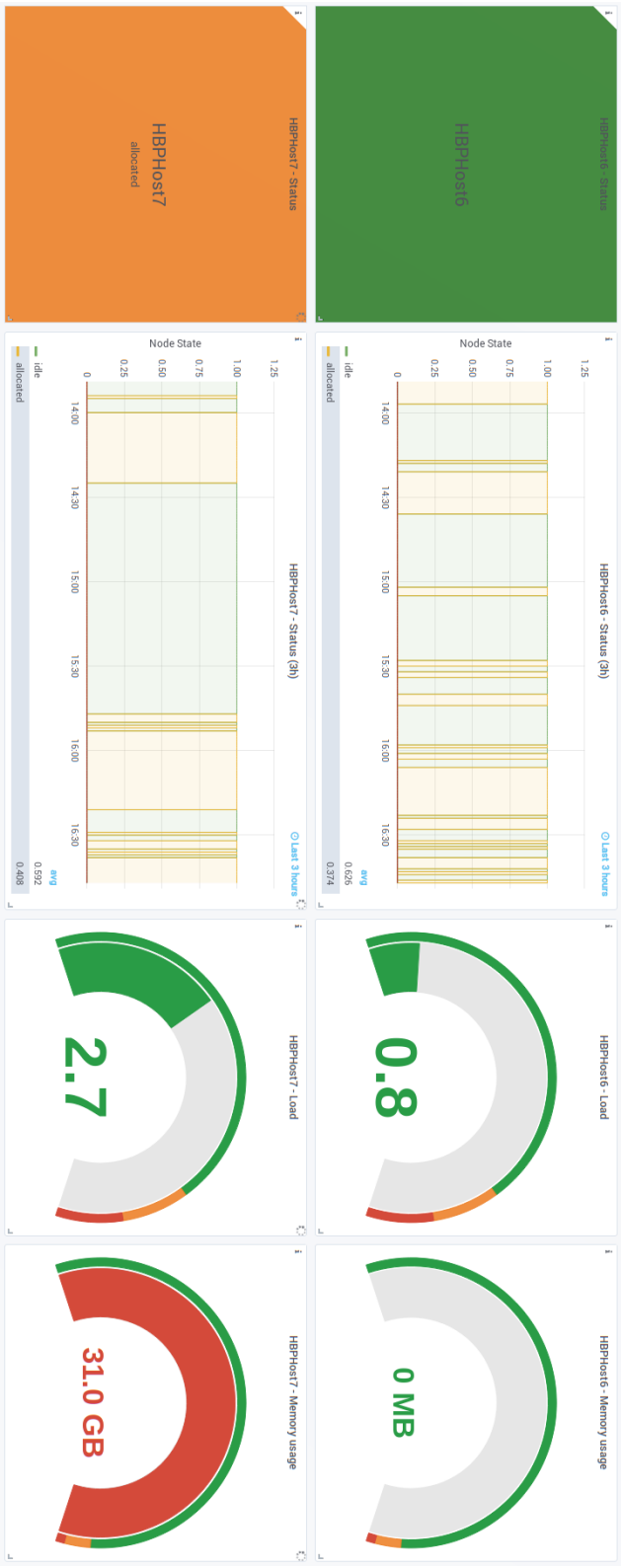

Figure 10: Sample view from the second dashboard displaying node information. For reasons of clarity only 2 of 31 nodes are shown here. Displayed are the status, utilization history as well as current CPU load and memory usage.

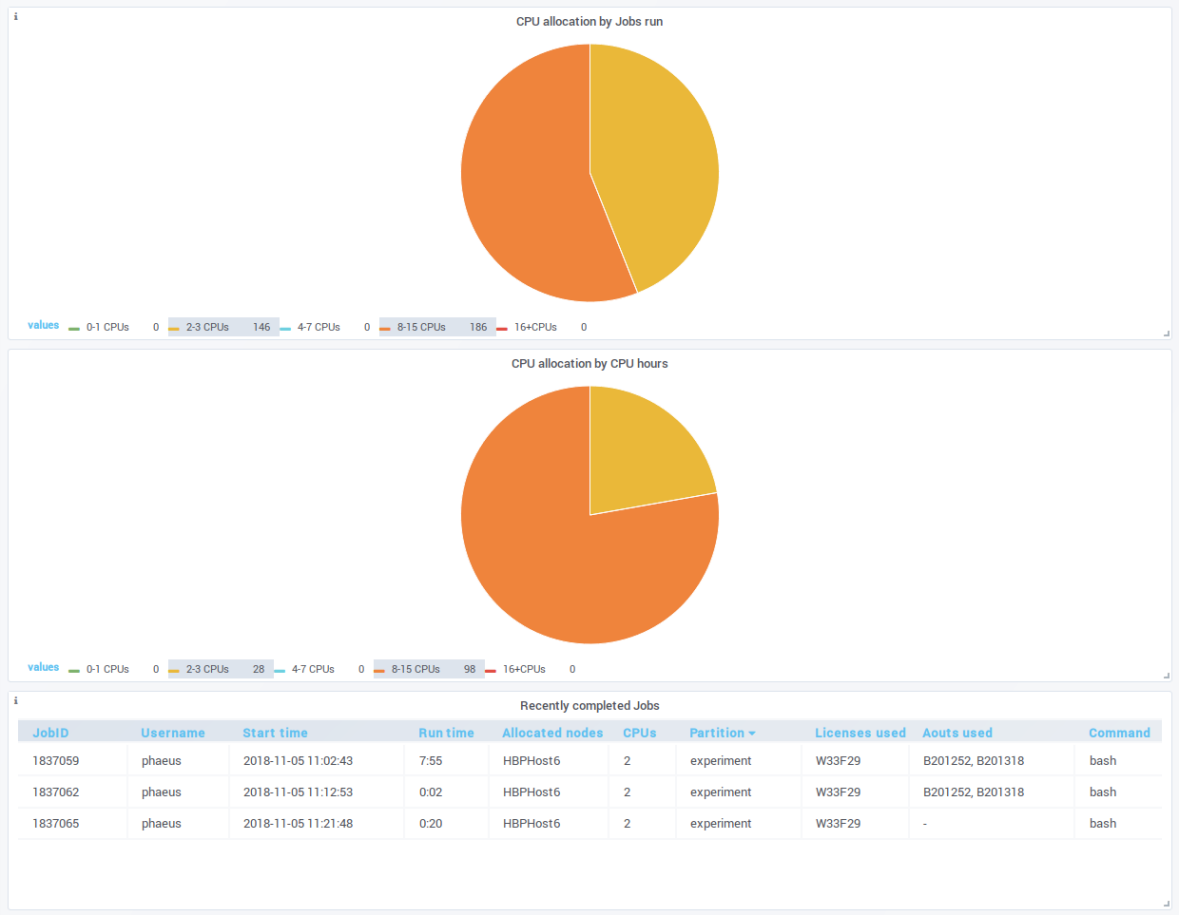

Figure 11: Sample view from the fourth dashboard, showing the distribution of CPU cores for past jobs. The table below shows basic information about recently completed jobs. This dashboard is still somewhat of a work-in-progress.

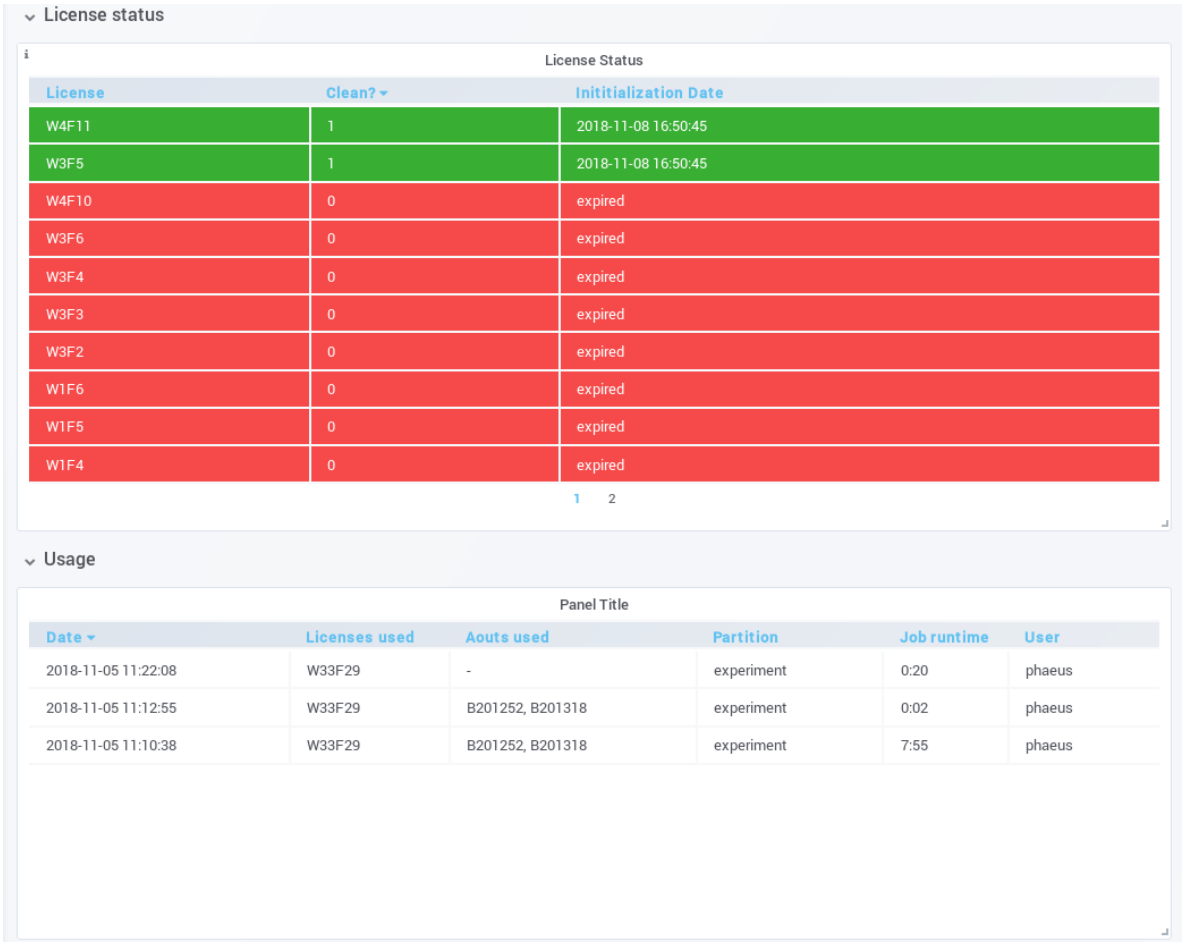

Figure 12: View of the last dashboard. It shows usage history and the current initialization-status of licenses. A graphical representation of usage history is still in the works and not shown here.

Chapter 3

Summary and outlook

3.1 Summary

In this thesis, *Slurm*, the workload manager responsible for scheduling all experiments carried out on the BrainScaleS system, was integrated into the existing monitoring solution. This makes it possible to directly monitor and determine the health of the cluster and can help to detect any problems that might interfere with or prevent experiments to run. This data can be used to optimize Slurm parameters.

The integration of Slurm was realized by adding and extending several scripts and revising most existing configuration files. Two new modular scripts periodically query the installed Slurm instance for information. The output of these scripts is then parsed, processed and formatted before it is written to disk. A third script which is already in use by Slurm was extended to query and log details about the job from where it was called.

The switch to a new data format allows for easier post-processing of the collected information and makes these scripts easy to understand and extend. All data obtained is processed and indexed into elasticsearch, making it available for search and visualization and permitting the generation of reports and statistics. Several new Grafana dashboards have been developed to convey this wealth of information. Data ranges from simple user statistics and utilization graphs to detailed diagnostic output reflecting the health of the cluster. This can provide valuable insight for optimization or troubleshooting.

Runtime measurements of the newly added and revised scripts show, that these pose no problem regarding the scalability of the implemented solutions. Additional considerations and calculations regarding the scaling of wafers and data generated by these scripts show, that even a ten-fold increase in wafers and computing nodes generates less than 1 Gb of additional data each month. A negligible amount in terms of both storage and network bandwidth required.

3.2 Outlook

Integration of Slurm into the existing monitoring system should only be seen as the first step towards an optimal and fail-safe configuration. With possible future expansions and an ever increasing number of users and experiments, it is becoming increasingly important to configure Slurm optimally. This work has created a solid basis for this. The data collected provides valuable insight into the health of Slurm and the cluster. By making all this data available in elasticsearch, much more detailed usage statistics and reports can be generated.

It also enables a more pro-active approach by fine-tuning Slurm and system parameters. Usage history and job information also provide insight into the requirements of experiments typically carried out, allowing to adjust partition defaults or scheduling parameters to better reflect these requirements. This can help to increase utilization of the cluster.

Although it is a robust implementation, there are a few things that can be improved further. This is particularly true for the `epilog`-script used to gather job information.

For future data collection, the optional `elasticsearch` plugin provided by Slurm could be used [9]. This would be a better solution to the `epilog.sh`-script, outputting much more data. By integrating directly into Slurm, this plugin would also provide a faster and more efficient way to collect job data. However, due to additional security measures of the current monitoring platform, this plugin would have to be modified first.

To expand available information to include jobs that have already been completed, future work could parse and process the complete database used by Slurm. This would make any information since the introduction of Slurm available.

The implemented solution offers a high degree of scalability, certainly beyond the point at which the current cluster and Slurm architecture has to be reconsidered as well. In such a case, the possibility for a backup control daemon should be utilized. It might then also become prudent to host the database on a separate system. Scripts could then be moved and executed on the server hosting the backup Slurm control daemon (and possibly the database) and would therefore be largely free of interference from other programs. In addition, both exporter scripts mostly request and output aggregated information, as such, scaling up the amount of wafers has no significant impact on either runtime or the amount of data generated by these scripts.

Bibliography

[1] Naud R, Marcille N, Clopath C, Gerstner W. *Firing patterns in the adaptive exponential integrate-and-fire model,* Biological cybernetics, 99(4-5), 335-47.
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2798047/

[2] Davison, A. P., Müller, E., Schmitt, S., Vogginger, B., Lester, D., & Pfeil, T. (2016). *HBP Neuromorphic Computing Platform Guidebook,* Release 2018-10-24 (2012d53), 2018.
https://electronicvisions.github.io/hbp-sp9-guidebook/, retrieved 28.10.2018.

[3] Daniel Kutny. *Development of a Modern Monitoring Platform for the BrainScaleS System,* 2017

[4] Internet Engineering Task Force (IETF). *The JavaScript Object Notation (JSON) Data Interchange Format,*
https://tools.ietf.org/html/rfc8259, retrieved 22.10.2018

[5] Morris Jette, *Tuning Slurm Scheduling for Optimal Responsiveness and Utilization,*
https://Slurm.schedmd.com/SUG14/sched_tutorial.pdf, retrieved 22.10.2018

[6] The Slurm development team. *Slurm workload manager - Official documentation,* https://Slurm.schedmd.com/

[7] The Slurm development team. *Slurm workload manager - Prolog and Epilog Guide,* https://Slurm.schedmd.com/prolog_epilog.html

[8] F.Stagni, A. Casajus Ramo, L. Tomassetti, Z. Mathe. *Evaluation of NoSQL databases for DIRAC monitoring and beyond,*
http://cds.cern.ch/record/2011172/files/LHCb-TALK-2015-060.pdf, retrieved 23.10.2018

[9] Sánchez Graells, Alejandro, *Integration of a job completion accounting system in a HPC environment by extending a workload manager,*
https://upcommons.upc.edu/handle/2117/79252

Glossary

cron Cron is a time-based job scheduler found in most Linux distributions. It is suitable for scheduling repetitive task such as daily backup operations or repeated script runs. Entries in the configuration files used by cron are referred to as *cron jobs*. 13

daemon A daemon is a program or process that runs in the background and is dormant when not required. 4, 9, 32

environment variable An environment variable (or *env var*) is a named object set for the current user-environment on linux. It consists of a name and value and can be used by one or more programs and scripts. 7

FPGA A FPGA (Field-programmable Gate Array) is an integrated circuit which can be reprogrammed to desired application or functionality requirements after manufacturing. 2, 6

MySQL MySQL is the most popular open-source database and the basis for many modern applications and websites. 4, 7, 11, 17

Appendix A

Configuration Files

A.1 filebeat configuration (excerpt)

```
- type: log
enabled: true
paths:
  - /var/log/elasticsearch/slurmstats.json
fields:
  slurmstats: true
fields_under_root: true
json.keys_under_root: true
json.add_error_key: true

- type: log
enabled: true
paths:
  - /var/log/elasticsearch/slurmtop.json
fields:
  slurmtop: true
fields_under_root: true
json.keys_under_root: true
json.add_error_key: true

- type: log
enabled: true
paths:
  - /var/log/elasticsearch/jobinfo.json
fields:
  jobinfo: true
fields_under_root: true
json.keys_under_root: true
json.add_error_key: true
```

Listing A.1: Excerpt of the filebeat configuration file (filebeat.yml). These changes allow tagging and forwarding of the new data gathered.

The remaining configuration files and scripts are too long to be shown here. They can either be found on the internal *gerrit* code collaboration (see section B.2 for a list of changes)tool or directly in the relevant directories on `monviz` and `hel`.

Appendix B

Additional tables

B.1 Script performance measurements

	Average runtime (ms)		
# of licenses	New	Old	Speedup (%)
1	1,02±0,05	2,21±0,13	216
5	1,51±0,09	7,14±0,46	472
48	1,41±0,07	13,43±0,63	952
144	1,47±0,05	18,69±0,88	>1200

Table 2: Runtime comparison between unoptimized (old) and optimized (new) `prolog.sh`-script. Averages are calculated for $n = 10$ script executions.

B.2 Gerrit change sets

Change-ID	Commit
`I01b3c43b7dba7764677b27a67add729d835d990c`	Added python script to automatically generate licenses from hwdb by parsing db.yaml
`I406a00989d55079fd49bed7859b6253bb1db5b11`	Substantial Speed improvements to prolog/epilog scripts.
`I2250e154f0760ebf699ba8a11da5fa66cf762a1f`	Added functionality to track license status via Slurmdbd.
`Ife9de24cd7844cbb3add13bdb4cd50163c0b1348`	First support for automated HICANN init
`I7e5b1d63f0f5b2b673549c71aedb1d7d5246be1b`	Added wrappers for cardinal direction checks.
`Iffe01b8f7b4cbebffb36d0b3b6d99931cd4af631`	Slurm Exporter Scripts

Statement of Originality

I certify that this thesis, and the research to which it refers, are the product of my own work. Any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline.

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen Hilfsmittel als die angegebenen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Diese Versicherung bezieht sich auch auf die bildlichen Darstellungen.

Heidelberg, den 14. November 2018. _____

(Signature)