Leveraging Jupyter on Maxwell HPC: joyful, visual and green computing

Yves Kemp, Arlena Mills-Marzoli, Neele Rahmlow, Sven Sternberger, Axel Wichmann, Frank Schlünzen

Abstract: Jupyter notebooks are great tools to mitigate the complexities of (heterogeneous) HPC systems, like the Maxwell cluster at DESY which serves the computational needs of all user facilities on campus, as well as a wide variety of applications ranging from plasma accelerators to quantum chemistry. We aim to expand the Jupyter ecosystem using frameworks like streamlit to provide application environments tailored to the needs of less experienced users, including real-time visualization capabilities. On this basis we are implementing for example Jupyter-driven remote desktops, user-friendly dashboards to compose or monitor batch-jobs, and visual frontends for data catalogues like SciCat. The implementations are accompanied by visual tools for resource utilization and CO2 footprints suitable both for users as well as admins.

Glossary:

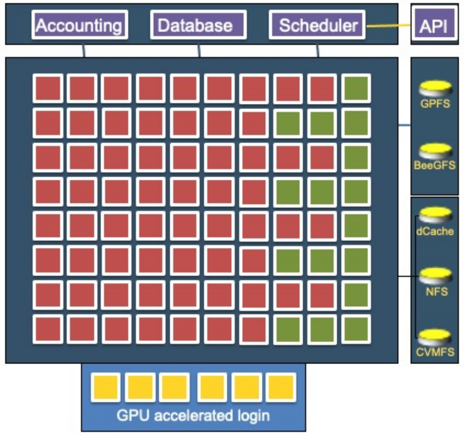
IDAF: **I**nterdisciplinary **D**ata and **A**nalysis **F**acility
an LK-II facility in MT (Matter and Technologies) at DESY

Maxwell HPC: The HPC compute part of the IDAF

Maxwell HPC Cluster

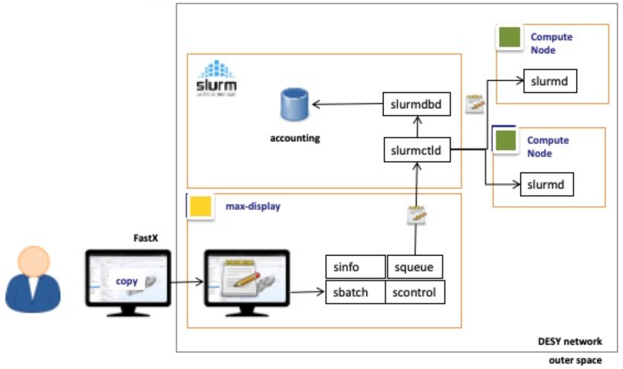
Characteristics

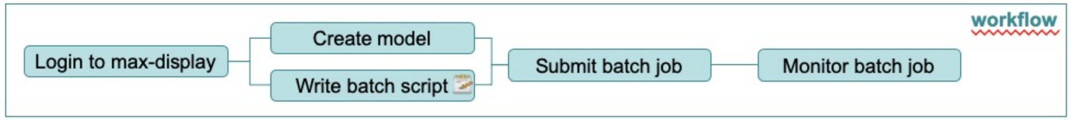
Total Number of compute nodes (CPU + GPU)	942	
Total number of cores with hyperthreading	84440	
Total number of physical cores	42348	
Theoretical CPU peak performance	1399	TFlops
Total RAM	536	TB
Number of GPU nodes	193	
Total number of GPUs	376	
Theoretical GPU peak performance	2615	TFlops
Total peak performance	**4014**	**TFlops**



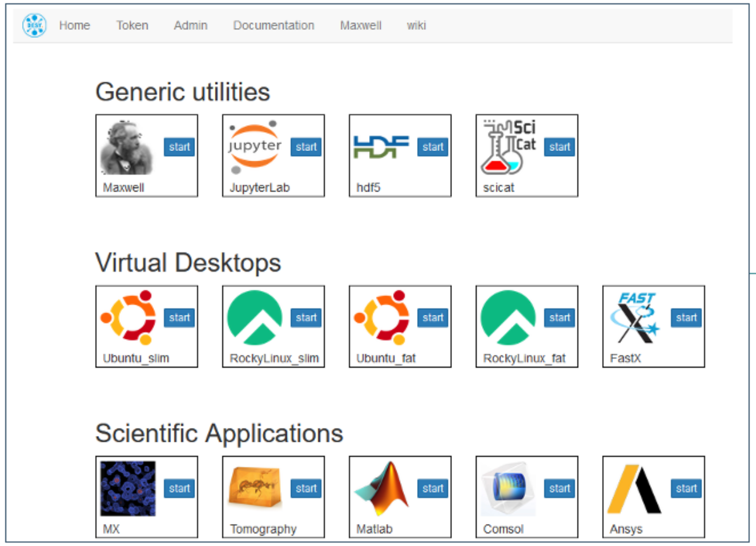
The Maxwell High Performance Computing cluster has all the ingredients of a typical HPC platform like low latency, fast network (Infiniband), cluster file-systems and a scheduler (SLURM) to guarantee a rapid and fair distribution of workload on the compute resources.The workloads and requirements are however very diverse, requiring an *atomic* partitioning and very heterogeneous hardware, which makes it impossible harvesting the 4 Petaflops in a single batch job.

The regular Workflow





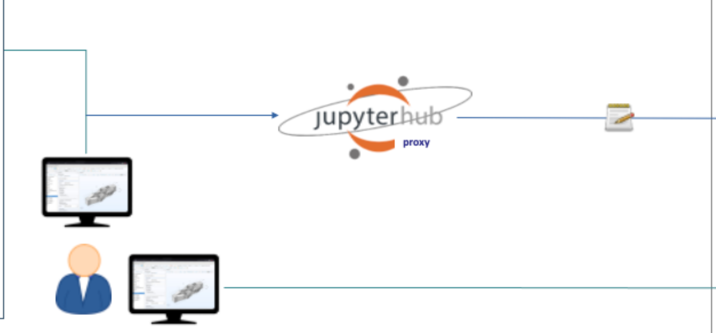
Most users will use the graphical login nodes (max-display) for graphical work and submission of batch-jobs. The login nodes are clustered and well equipped with GPGPUs and memory allowing for example CAD modeling. The nodes can be reached via ssh, a FastX client or a web-browser from anywhere in the internet, but are completely isolated from the DESY internal network for security reasons, and even root privileges are not sufficient to modify any files on the cluster filesystem (root squashing). The impact of compromised account will be minimal.

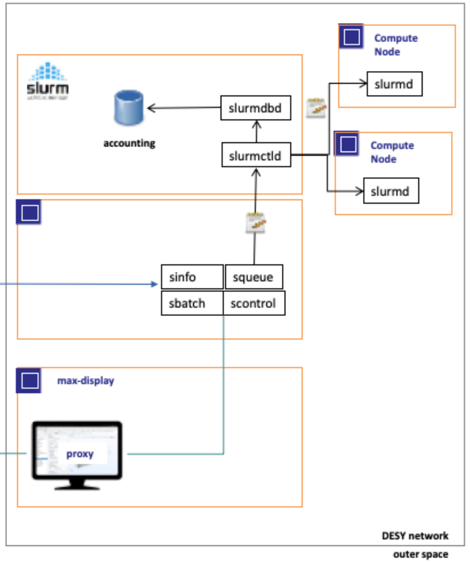


Jupyterhub modified as a dashboard service with a starting collection of pre-configured applications. The Hub uses a custom batch-spawner to launch (most of) the applications as jupyter-proxied batch jobs.

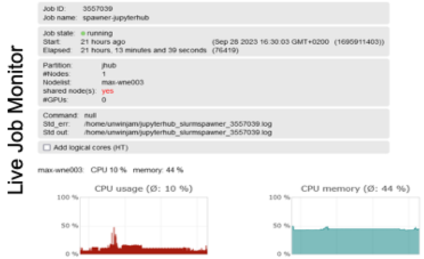
JupyterHub as a proxy

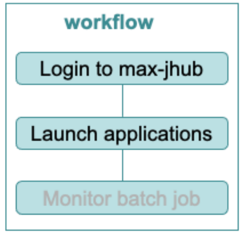
JupyterHub can proxy almost every kind of application. Our batch-spawner implementation (maxapp-spawner) creates and submits the batch scripts, defining the proxied ports and application specific compute requirements. The proxied application runs as a batch job entirely in user space, which makes it very simple to allow access for example to storage, avoiding the nightmare of implementing GPFS extended ACLs in a generic webservice. User need to remember only a single access point – the jupyterhub – which controls and keeps track of applications.





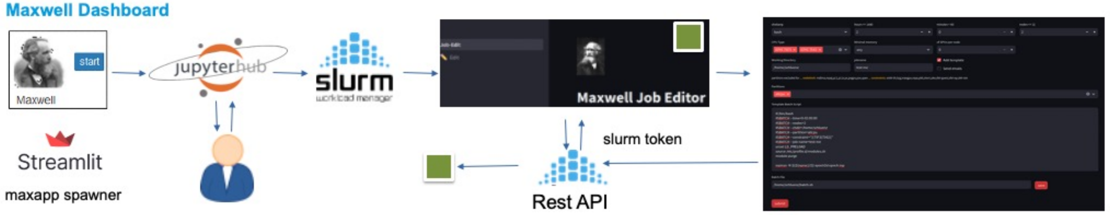
Applications like comsol, matlab need a graphical frontend. This can also be a local desktop using max-display as a proxy.





Proxied applications

Maxwell Dashboard



There are a few python frameworks out in the wild making dashboarding quite easy. Streamlit is one of those frameworks coming with strong support for data frames and visualization.

Based on streamlit we are working on a generic dashboard which provides a quick overview on batch-jobs, resource utilization and power consumption. It allows to create batch-scripts with zero SLURM knowledge, use batch templates for commonly used applications and submit the job through SLURMs REST API. An API token is generated automatically in the background, hiding all of the clusters complexity from (inexperienced) users.

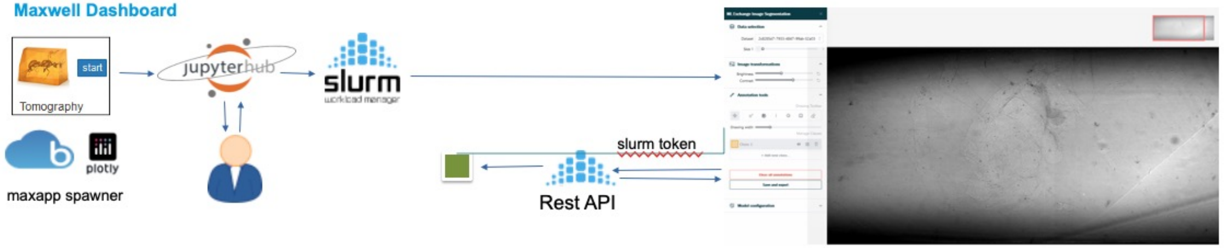
Data Catalog Frontend

Data generated from experiments at Petra III, FLASH, Eu.XFEL are stored in GPFS. SciCat is a meta-data catalog implementation giving access to meta-data and snapshots uploaded to the data catalog. SciCat is designed to run on cloud infrastructure being deployed on kubernetes pods. The services naturally run in an unprivileged context, and have consequently no access to any experimental data (and k8s pods are not intended to locally embedded storage). A streamlit application – running in user context – can be used to provide full access to data, and allow simple media tools to view and manipulate for example images or HDF5 container; based on selections customized jupyter notebooks can be launched for data processing. Access to jupyter notebooks and SciCat are achieved through REST API tokens without any need for user actions.

Virtual Desktop

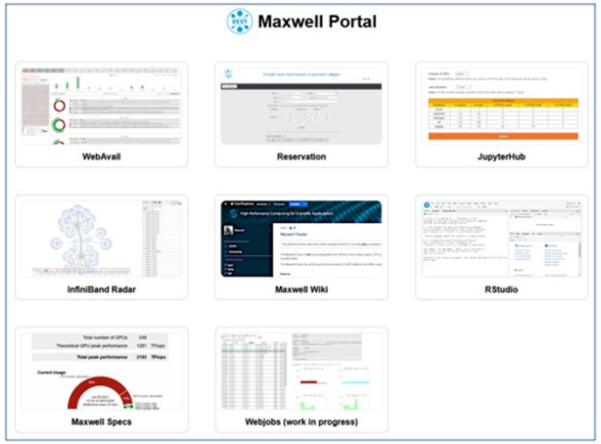
Some application do not run on the operating system of the Maxwell cluster, and some are much easier deployed as docker or singularity images. To facilitate the process we provide a custom VNC setup for operating systems like Ubuntu 22.04 or RedHat EL9. The batch-spawner launches the corresponding image as a regular batch-job, pulling the image from a Harbor container registry. The user can access the desktop through the jupyterhub in the web-browser, and the session can be secured with the users password. The setup lacks GPU acceleration, and websocket proxying is not yet implemented, but for most users in need of alternative operating systems the setup should be quite sufficient.

Maxwell Dashboard

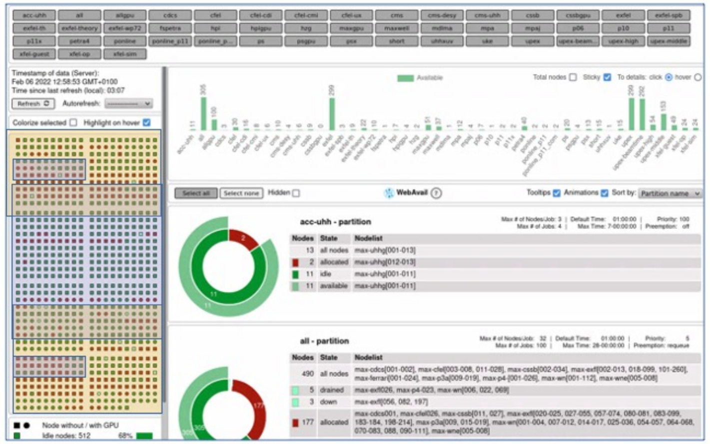


Artificial intelligence is heavily used on the cluster, and one of the applications in high demand is automated image segmentation and registration. A prototype has been implemented based on mlexchange. mlexchange is a machine learning pipeline using pre-trained models and interactive web-annotation tools for iterative improvement of the ML model. The setup uses a bluesky/tinder file catalog and plotly dashboards serving both files and application through a jupyterhub proxy'd application. The file catalog is created on the fly as part of the batch job. While the prototype works smoothly, it still needs some components to be implemented, and the need for a file catalog is not exactly matching our storage configuration.

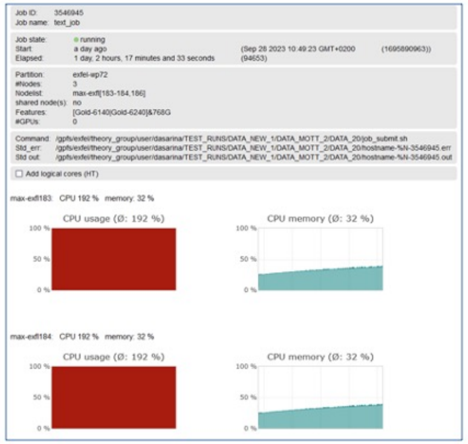
Services



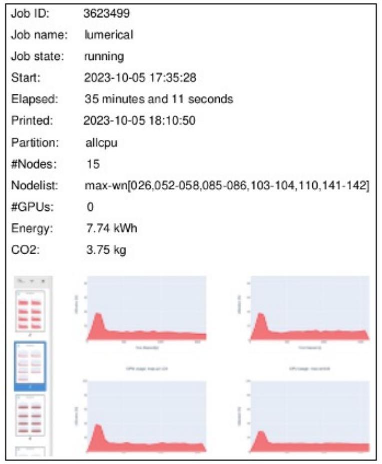
The maxwell portal is the entry-point to the most prominent services on the cluster. This includes access to the jupyterhub, R-Studio, cluster reservation tools, and the often demanded *one-click* visualization of characteristics and utilization of the cluster.



SLURM tools are fairly limited for cluster visualization, and don't report the availability of resources which can be freed through preemption. The web-availability service is an in-house development giving a very detailed and highly flexible view of the state of the cluster, while minimizing the load on the SLURM scheduler.

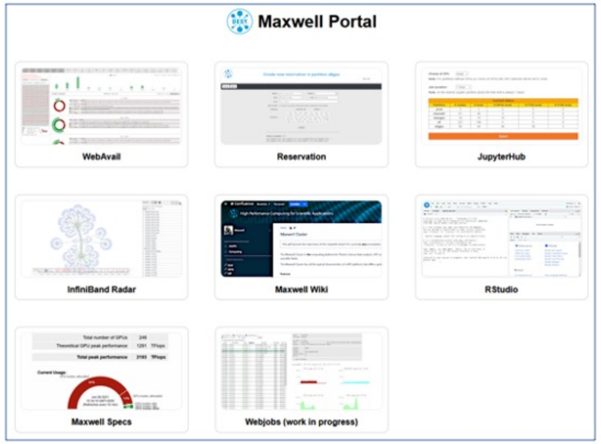


Cluster users are not always aware of its intricate nature, which quite frequently leads to poor utilization of resources, and a very uneven distribution of the workload on the compute nodes. The webjobs service – also an in-house development – visualizes the distribution at a glance, and can alert users of poor or uneven resource utilization, which helps to maximize throughput and hence to minimize energy consumption

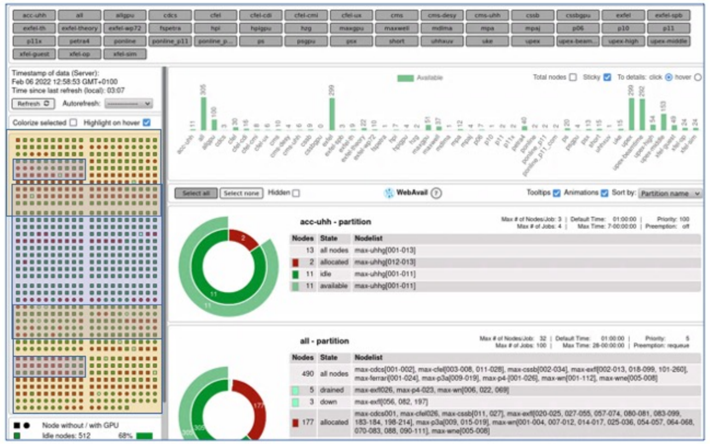


Job reports allow storing the job profile incl. energy consumption persistently together with the batch script, for later reproducibility and comparison. It raises awareness of the costs involved and the impact on the climate (even if it was small).

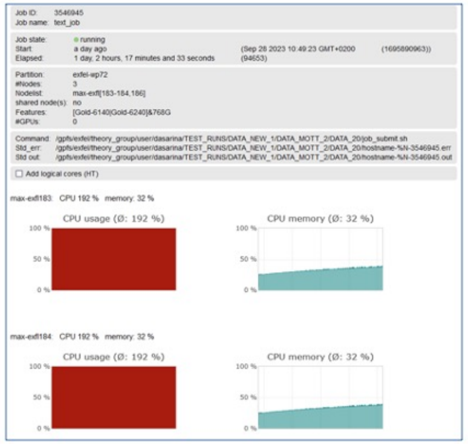
Services



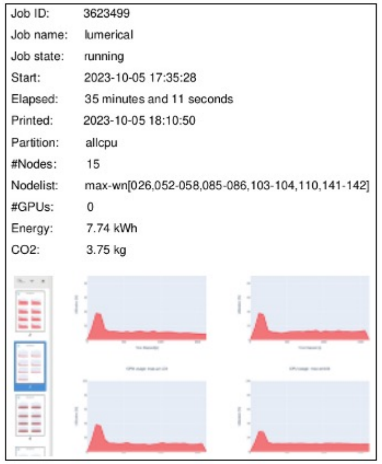
The maxwell portal is the entry-point to the most prominent services on the cluster. This includes access to the jupyterhub, R-Studio, cluster reservation tools, and the often demanded *one-click* visualization of characteristics and utilization of the cluster.



SLURM tools are fairly limited for cluster visualization, and don't report the availability of resources which can be freed through preemption. The web-availability service is an in-house development giving a very detailed and highly flexible view of the state of the cluster, while minimizing the load on the SLURM scheduler.



Cluster users are not always aware of its intricate nature, which quite frequently leads to poor utilization of resources, and a very uneven distribution of the workload on the compute nodes. The webjobs service – also an in-house development – visualizes the distribution at a glance, and can alert users of poor or uneven resource utilization, which helps to maximize throughput and hence to minimize energy consumption



Job reports allow storing the job profile incl. energy consumption persistently together with the batch script, for later reproducibility and comparison. It raises awareness of the costs involved and the impact on the climate (even if it was small).

Sustainability is an important aspect in cluster management. The cluster consumes roughly 7GWh per year, which accounts for roughly 3.3% of the energy consumed on campus. A very high cluster utilization – which is desirable – leaves not much room for energy savings. However, with simple measures in the configuration of the cluster automatically throttling CPUs at the end of a job, weekends average energy usage reduced from ~690kWh to ~590kWh leading to energy savings of at least 250MWh annually.