Interactive HPC and the LUNARC Desktop Environment

Jonas Lindemann
LUNARC, Center for Scientific and Technical Computing
Lund university
Lund, Skane
jonas.lindemann@lunarc.lu.se

Anders Follin
LUNARC, Center for Scientific and Technical Computing
Lund unviversity
Lund, Skane
anders.follin@lunarc.lu.se

***Abstract*—Since 2011, LUNARC has aimed to provide an interactive HPC environment for its resource users. Several different architectures have been used, but since 2013, we have been using a remote desktop environment based on Cendio's ThinLinc [1] combined with a custom backend framework, GfxLauncher [2], supporting hardware-accelerated graphics applications and Jupyter Notebooks [3] submitted to the backend cluster.**

***Index Terms*—remote desktop, HPC, interactive visualization, hardware accelerated graphics, OpenGL, Vulkan, VNC**

I. Introduction

LUNARC [4] is the Center for Scientific and Technical Computing at Lund University. The center provides HPC resources such as computing, storage, and visualization for researchers at LU. For a long time, the only access methods for HPC resources were SSH and a terminal. For experienced users, this is a very efficient and quick way to use HPC resources. However, it has a steep learning curve. Many new users struggle with using a terminal to perform their tasks. It is possible to run graphical applications through SSH, but this usually requires a complicated setup and installation of special client-side software, such as X servers. This configuration is also significantly different on Linux, macOS, and Windows, making it hard to provide well-maintained documentation.

A. Teradici based remote desktop solution

Around 2010/11, we saw the need for a more user-friendly and rich environment for HPC resources. The first step in this direction was to use a Teradici-based PCoIP [6] solution. A dedicated front-end node was set up with a GPU to enable a hardware-accelerated graphics environment. This environment was accessed using a dedicated PCoIP receiver or a special LCD screen with a built-in PCoIP client. Users could buy their client or borrow a client from LUNARC to access this environment.

The Teradici environment was a perfect solution for providing access to a desktop environment with hardware acceleration. The desktop environment was exported directly from the graphic card's DVI/HDMI connector, which made it very compatible with hardware-accelerated environments. Using the same technology, a Windows-based environment could also be made available.

The drawback of the Teradici solution was making it widely available to all users of LUNARC resources. Most groups didn't buy the special clients; only a few did. We needed a different approach.

B. Software-based remote desktop solutions

In 2012/13, we started experimenting with remote desktop solutions such as NoMachine [7] and VNC [8]. These solutions provided an easy way of accessing a Linux-based desktop. However, at the time, we thought they had too many security issues for us to implement them in production.

The recommendation at the time was to use SSH tunneling to provide access to remote desktop services. However, this again presents additional obstacles for the user as it requires special SSH client configuration. A further obstacle is the two-factor-based authentication system used at LUNARC, which NoMachine and VNC didn't support.

C. Cendio Thinlinc remote desktop service

During our research, we came across the Thinlinc services from Cendio. Thinlinc is based on TigerVNC [9] and uses SSH as a transport layer for VNC. It also provides precompiled clients for all platforms, making it easy for users to install and connect to the Thinlinc service. This seemed like a solution we could offer. We set up a test front-end and evaluated the product. One problem encountered was that the client connection process is a two-step process. First, the client connects to the master service (vsmserver) and receives information on which agent service (vsmagent - the actual VNC server) to connect. This was problematic with our two-factor authentication system as it didn't allow token reuse. We worked together with our vendor on this and were able to solve this issue.

At this point, we decided to focus on Thinlinc as the access layer to our underlying desktop environment at LUNARC.

II. Related work

There are several existing solutions for implementing interactive HPC solutions. For us, the definition of interactive HPC service is a solution that can provision custom interactive sessions where users can specify custom resources, such as a number of CPUs, memory, and graphics. A session can be a

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00251

terminal, desktop or a web page. However, the user has to be able to work interactively with the computing resource. The backend primarily consists of Linux-based HPC resources.

Some commercial providers provide solutions for interactive HPC usage. One of the larger ones is NICE DCV [10], which was acquired by AWS a couple of years ago. NICE DCV primarily provides high-performance remote desktops and 3D graphics for Linux and Windows. There is no specific integration with HPC resources, but it could act as a front-end system to an HPC cluster.

NoMachine [11] provides an enterprise terminal server where you can request desktops depending on your requirements. The solution does not directly support integration with HPC services but could be a front-end service to an HPC cluster.

There are more established solutions for interactive HPC in the open-source space. One offering that can be considered an interactive desktop service is JupyterHub [12]. This service is usually deployed on a cloud service. Users can request interactive notebooks with different computing and storage properties.

Another platform that has grown significantly during the last years is Open OnDemand [13]. It is an open platform that provides a web-based portal where users can request access to applications. Applications can be web-based or interactive. The platform has an extensive plugin infrastructure for adding applications. Desktop applications are provided by launching desktop environments through the batch system. Desktop environments offered are Gnome 2, Mate, or Xfce. Access to the desktop uses the noVNC client.

UCloud [14] is a relatively new portal-based service providing access to an interactive digital research environment that can handle data throughout the data lifecycle. In addition to interactive services, UCloud has a similar model as Open OnDemand, providing access to batch, interactive, desktop, and virtual machines. UCloud also has an architecture that provides additional functionality using plugins implemented in Kotlin.

III. A Remote Desktop Prototype

We developed a prototype to evaluate how a remote desktop environment works (2013-14). This was based on a standard Linux-based desktop service using ThinLinc on our Alarik resource. At the same time, we also evaluated how to provide hardware-accelerated graphics for specific applications. By default, VNC only supports software rendering of graphics. To solve this issue, you need to use a solution such as VirtualGL [15], which offloads the rendering of OpenGL to a GPU and transfers the rendered images back to the remote client. The architecture is shown in figure 1.

In this first iteration of the LUNARC Desktop environment, applications requiring hardware-accelerated graphics were started on special application servers equipped with installed NVIDIA GPUs and the VirtualGL software. To hide the complexity, scripts were created using the `vglrun` and

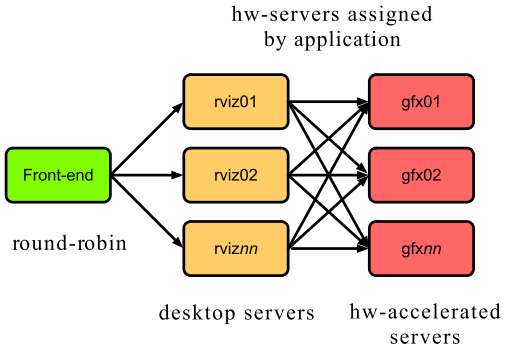

Fig. 1. Alarik prototype architecture.

`vglconnect` commands to start the server application and to display their output on the main desktop.

This first iteration proved very popular with our users, and we saw an increasing number of users using the environment. An interesting observation was that even users used to terminal-based usage started using the desktop service, as it provided an excellent way of keeping multiple terminal sessions alive and allowing them to reconnect to check job status.

Many lessons were learned about implementing a production environment. Scalability can be achieved by adding servers, but it comes with limitations. Each user session is tied to a specific back-end server, sharing resources with other users on the same node. This arrangement presents challenges in effectively enforcing resource limits. Users with demanding visualization needs can potentially over-allocate memory, leading to a degraded experience for others sharing the same node.

Using this approach, providing users exclusive access to dedicated servers for running graphical applications is also challenging, as you need to assign a user dedicated access to the application servers manually. Applications must also be configured to a specific server, which leads to load-balancing issues if a particular application is widely used.

Supporting a remote desktop environment differs significantly from managing a traditional HPC resource. It requires developing specialized tools for monitoring desktop sessions to ensure a smooth user experience. Moreover, integrating the desktop infrastructure with existing authentication solutions can be complex.

One conclusion from this prototype was that we should use the batch scheduler to allocate resources when running desktop applications on backend nodes. Using SLURM also enables strict control over resource use concerning memory, CPU, GPU, and wall time. A user starting an application can't over-allocate the node, and the application will be automatically killed when the wall time has been exceeded. The GfxLauncher [2] framework was developed to provide a user-friendly user interface to start graphical applications through SLURM with options for the user to give requirements; see

figures 2 and 3.

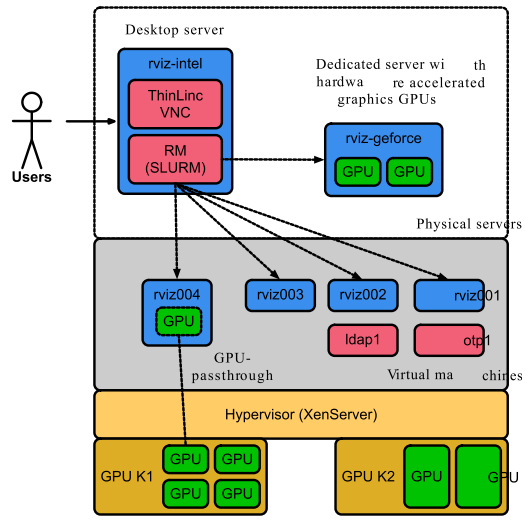

Fig. 2. Proposed architecture from prototype work.

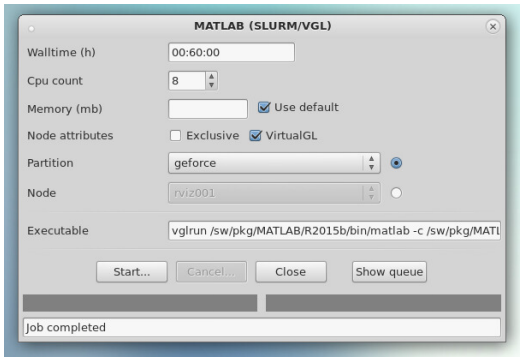

Fig. 3. First version of the GfxLauncher framework.

The knowledge gained from this prototype was integrated into the first production version of the LUNARC HPC desktop, which was implemented in the Aurora resource in 2016.

IV. LUNARC HPC Desktop Architecture

The current iteration of the LUNARC HPC Desktop was expanded and further developed in our latest resource, COSMOS [16], in 2023. This added several dedicated nodes with NVIDIA A40 GPUs dedicated to graphics and visualization. Additional load-balanced frontends are now also used to serve desktop sessions. All interactive sessions now go through SLURM, and we don't provide any shared application servers for our users. The main architecture is shown in figure 4.

We have two partitions for the graphical nodes, one with Intel-based CPUs and one with AMD. As graphics nodes can also run regular jobs requiring GPU support, we have reserved one graphics server in each partition restricted to interactive graphical applications. If you are running regular graphical

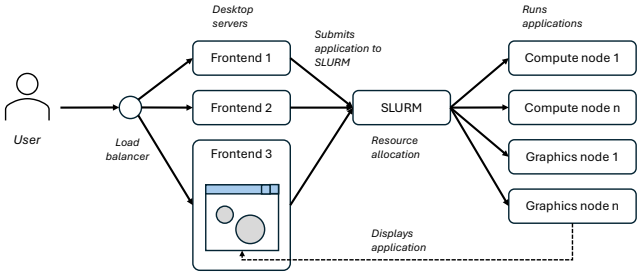

Fig. 4. Current LUNARC HPC Desktop architecture.

applications or notebooks, running these on the regular nodes is also possible.

Using SLURM as the resource scheduler for interactive applications has many benefits. As mentioned before, resource limits are controlled. Also, maintaining the desktop environment requires no special setups or cloud resources. Installed graphical applications are handled like any other installed HPC application through the module system. One design goal for the GfxLauncher framework was that a research engineer could quickly create a launch script for an interactive application. A simple batch script can be created for the application that specifies which modules must be loaded and how to launch the application on a node. The same script also has special tags that tell the menu generator how to configure the launch parameters for the user interface. An existing script can be copied and modified to add an application. The menu generator automatically updates the menu when a user logs in to a front end. A typical launch script with tags and launch instructions is shown below:

```
#!/bin/sh

##LDT category = "Visualisation"
##LDT title = "ParaView 5.11.1"
##LDT group = "ondemand"
##LDT vgl = "yes"
##LDT feature_disable = "yes"

PARAVIEW_DIR=/sw/pkg/paraview/5.11.1/bin

vglrun $PARAVIEW_DIR/paraview
```

V. SLURM configuration

We have 3 main use cases for running interactively:

1) 2D graphical applications through SLURM.
2) 3D hardware accelerated applications through SLURM
3) Jupyter Notebooks/Labs with and without GPUs.

To make interactive HPC feasible, we must have enough resources so that users don't have to wait too long for resources. When we procured our latest resource, we procured several NVIDIA A40 nodes for interactive use. 2D graphical applications are configured to run on standard CPU nodes and Jupyter Notebooks. All standard CPU nodes have a max time limit of 7 days. Applications requiring 3D acceleration

Partition	Description	Count	Time limit
lu48	AMD 48 core nodes	185	7 days
lu32	Intel 32 core nodes	22	7 days
gpua40	AMD/NVIDIA A40 48c	6	24 h
gpua40i	Intel/NVIDIA A40 48c	6	48 h
gpua100	AMD/NVIDIA A100 48c	6	7 days
gpua100i	AMD/NVIDIA A100 32c	4	7 days

TABLE I
SLURM PARTITION AND TIME LIMITS

(OpenGL) run on dedicated NVIDIA A40 nodes with a 24-hour max time limit. Some older Intel/NVIDIA A40 nodes have also been given a 48-hour max time limit to run some standard GPU loads. The available partitions and their properties in our current production resource are shown in table I.

VI. GFXLAUNCHER FRAMEWORK

One of the critical components of the LUNARC HPC Desktop is the GfxLauncher framework. This tool makes launching applications through SLURM easy for users. The framework provides a configurable user interface where the user can specify resource requirements and launch the desired application. GfxLauncher also tracks the application and shows information on how much allocated time is left. Users can also stop the running application, restart it, and reconnect to any web services running on an allocated node.

The framework currently supports launching the following applications through SLURM:

- OpenGL applications using VirtualGL and the vglconnect mechanism.
- Normal applications using the SSH mechanism.
- Jupyter Notebooks/Lab through SSH.
- Windows desktop sessions through SSH and Xrdp.

The GfxLauncher framework is a Python-based [17] tool that provides several tools for implementing the user interface for launching applications through SLURM and generating a menu system from the special tagged launch scripts. The user interface uses Qt for Python [18] and should work on most Linux-based desktops. The GfxLauncher framework also provides additional tools, gfxusage, to display the job queue and gfxnodes to display node status.

The following sections describe the different launch methods in GfxLauncher in more technical detail.

A. SSH Based Application Launch

To launch Xorg-based [19] applications securely through SLURM requires them to be launched through SSH with options for Xorg tunneling enabled. This is often not directly available through SLURM-based launch methods. GfxLauncher uses a placeholder job to allocate the node. This launch method requires SSH access to the nodes. The process is described in figure 5.

B. VGLConnect based Application Launch

Launching VirtualGL applications through SLURM has the same requirements as launching through SSH. In addition, the front-end and nodes must have working VirtualGL installations. GfxLauncher uses vglconnect instead of SSH to connect to the allocated node. vglconnect uses SSH to set up and tunnel the VirtualGL connection. Figure 6.

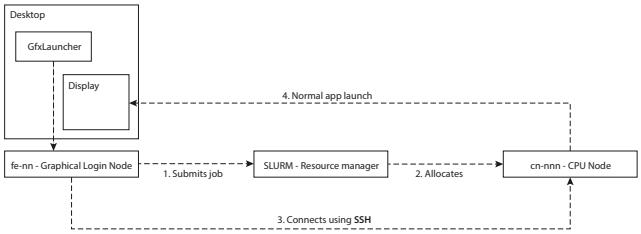

Fig. 5. SSH-based launch method.

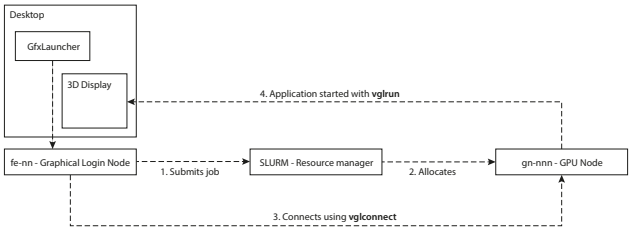

Fig. 6. VGLConnect launch method.

C. Notebook Job Launch

Starting a job running a Jupyter Notebook or Jupyter Lab session resembles conventional job submission. A job is submitted to SLURM to start up the notebook web server. GfxLauncher then waits for the job to start and monitors the job output for the URL to the Jupyter web server. It then starts a browser session to this URL. If the user closes the browser window by mistake, there is a special button in the user interface for reconnecting to the Jupyter server. Figure 7 shows this process.

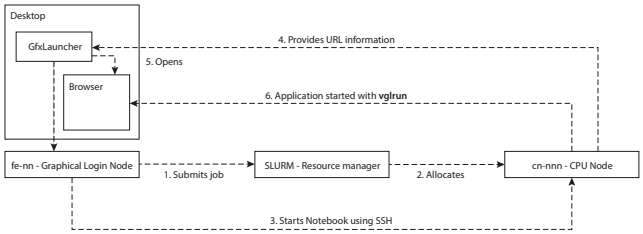

Fig. 7. Notebook launch method.

A use case at another national center required that the connection to the notebook not go over regular HTTP traffic from the front end to the backend nodes. Generating an HTTPS certificate for this purpose is quite a complex process, so instead, GfxLauncher starts the Notebook server listening at the node localhost and then sets up a remote port forward (SSH) to connect to the locally running notebook.

VII. User Interface and Experience

The user experience of the LUNARC HPC Desktop is designed to be as unobtrusive as possible for the user. When logging in using the Thinlinc client, the user is presented with a slightly customized Linux desktop. The desktop menu provides standard shortcuts for browsing files, opening terminals, and internet browsers. All applications provided through the GfxLauncher framework are in the desktop menu in different categories prefixed by ”Applications - [Category]”. The LUNARC HPC Desktop is shown in figure 8.

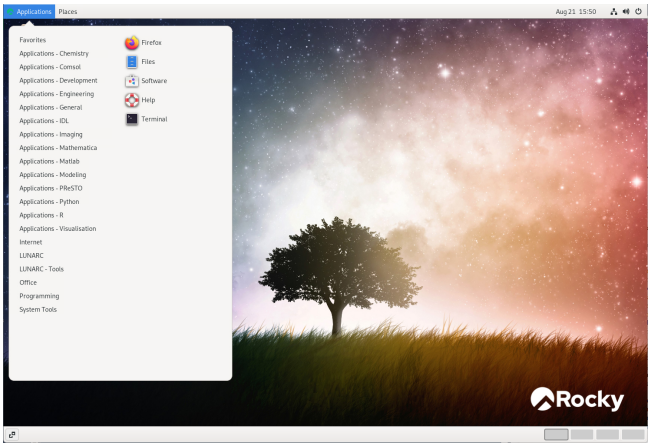

Fig. 8. The LUNARC HPC Desktop.

The menu prefix is a configuration option in GfxLauncher. The GfxLauncher generates all of the shortcuts in the ”Applications” categories. Selecting any application brings up the launcher user interface with options for resource selection, wall-time, and job options. The information in the user interface is configured by the menu generator, which will generate the necessary command line options for the launcher. For example, choosing MATLAB [20] from the menu will display a user interface as shown in figure 9.

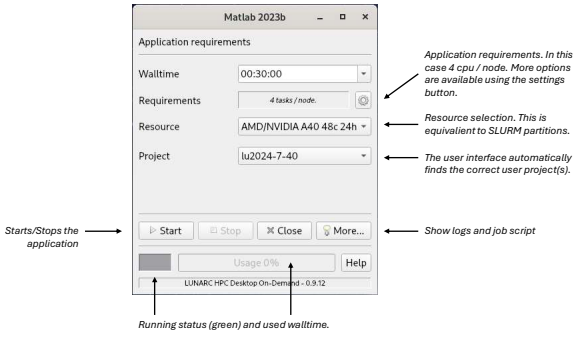

Fig. 9. The GfxLauncher user interface.

The user can select the requested wall time in the top combo box. The ”Requirements” shows the currently selected requirements for running the application. In this case, 4 CPUs are allocated for the application. The settings button next to the label can be chosen if the default is insufficient. This brings up the dialog shown in figure 10.

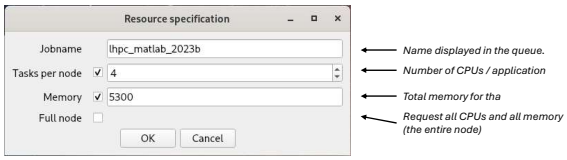

Fig. 10. The GfxLauncher user interface.

Clicking the ”Start” button starts the application and submits it to SLURM. If a node is available, the application is shown on the desktop. The indicator in the lower left corner turns green, and the progress bar starts and displays how much wall time has been used. Figure 11

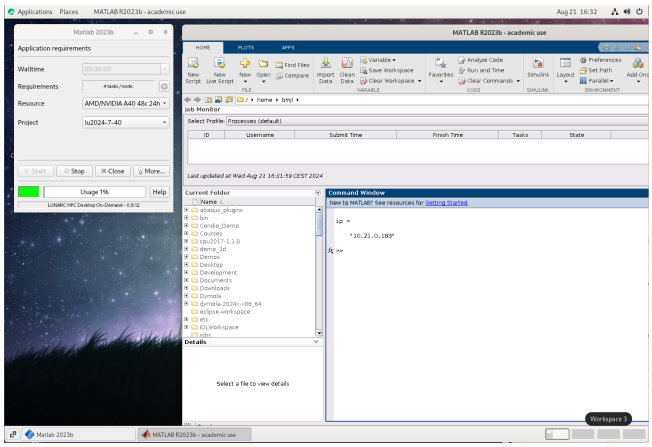

Fig. 11. MATLAB running using GfxLauncher.

Running graphics applications requiring hardware-accelerated OpenGL is accomplished in the same way. Figure 12 shows ParaView [21] running on a graphics node.

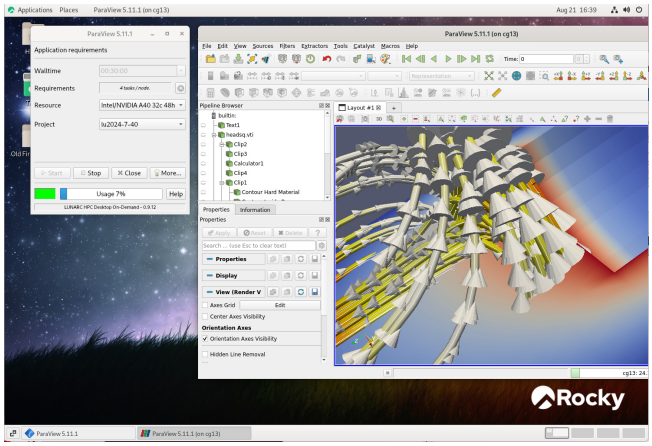

Fig. 12. ParaView running using GfxLauncher.

Notice that the user never has to know the application requirements. This is configured using the launch scripts. Also, the configuration of GfxLauncher prevents the user from selecting the wrong resources for the selected application.

Like the other application types, a Jupyter Notebook or a Jupyter Lab can also be started using the launcher. The user interface presented is a bit different in this case. An additional button is presented where you can select the Python environment used to run the notebook. Figure 13 shows this interface.

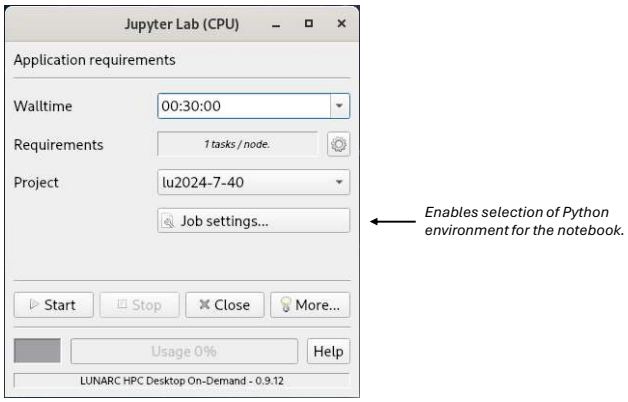

Fig. 13. User interface for launching a notebook.

The LMOD [22] module to run the notebook can be selected in this user interface. It is also possible to choose your custom environments in this interface. Figure 14 shows this interface.

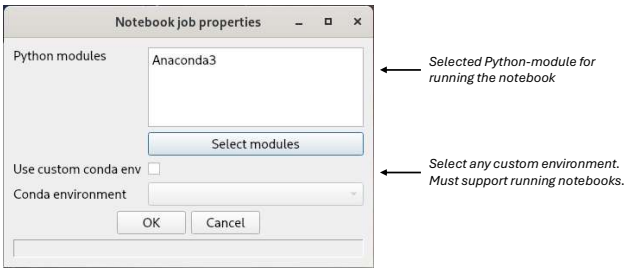

Fig. 14. Selecting a Python environment for the notebook.

A Jupyter Notebook is a web server running on the allocated node and presented through a web server on the desktop. If the user closes the browser, there is no obvious way of reconnecting to the running notebook unless you know the exact URL. To solve this, the user interface for running notebooks adds a button when the notebook has started so that you can reconnect to the notebook again, as shown in figure 15.

VIII. Performance and Scalability

During the prototype phase, we did a lot of performance tests using VirtualGL. In this study, the most limiting factors for performance were the bandwidth between the front-end and

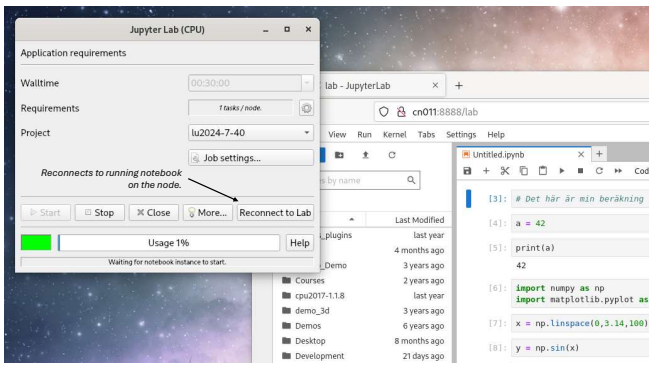

Fig. 15. Running a Notebook with user interface for reconnecting to the instance.

the backend server. Also, the size of the running application window significantly influences performance. Applications that constantly update the window can push as much as 30 MB/s between the front and back end. However, most scientific applications do not continuously update the frame buffer, so the perceived performance is still good. For more information, please see the online report on the prototype evaluation [23].

Running the interactive desktop environment in an HPC environment also provides an infrastructure that can handle a lot of bandwidth. We currently use the 25 Gbit ethernet infrastructure to run the graphics, but the Infiniband network could also increase the bandwidth for graphical applications. This would probably also lower the latency of mouse interactions and frame updates.

As of this writing, we haven't seen any congestion or performance problems with applications running on the backend nodes. However, there have been some issues with performance on the front-end servers when users run local applications directly on the server. Since then, we have implemented limits on how many resources users can allocate to these servers.

IX. Usage and Statistics

When talking to our users, one of the primary use cases of the desktop service is performing data analysis and visualization on existing data or data generated from completed or running simulations. The main applications for this are MATLAB, R-Studio [23], and Spyder. Some researchers also use ABAQUS/CAE [25], the graphical FEA environment from Dassault, to do modeling and initial computations that will later run as batch jobs. The environment also visualizes large models that can't be loaded on their regular workstations. Our CFD users also perform visualization on their completed and running jobs using ParaView.

It is easy to extract statistics from jobs run through the GfxLauncher as the default job name contains lhpc or lhpc with the application name as the suffix. In the statistics below, we removed failed jobs and jobs shorter than 5 minutes. Since May 23 2023, 4630 application launches have been registered in SLURM. We also have had 132 unique users submitting

jobs through GfxLauncher. Fig. 16 shows the most started interactive applications through SLURM. MATLAB and R-Studio dominate the usage, but interestingly, ParaView and the engineering application ABAQUS/CAE are also at the top. Another observation is that running Spyder [26] through the desktop is higher than using Notebooks.

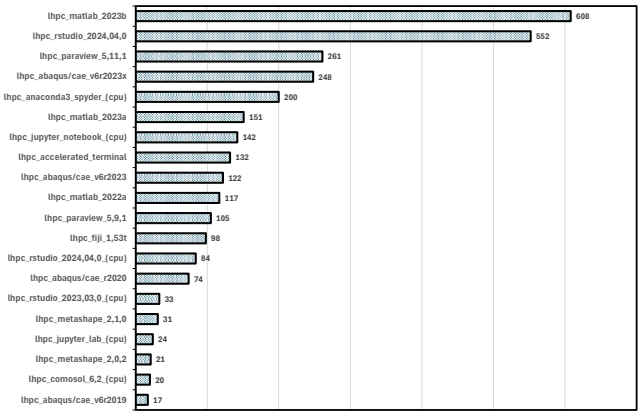

Fig. 16. Most used applications.

The job length plot in Fig. 17 shows many users run long jobs up to 24h (The job limits on the interactive nodes are 24h). This could be a way of circumventing the queue, but there is also a lot of variation. There are also many shorter jobs, probably due to the default job length being set to 30 minutes, and when a job has started, the job length can't be changed. Another reason for shorter jobs is if you want to open and view or analyze data on a running job. ABAQUS/CAE seems to have long usage times, but also Notebooks and R-studio. To further investigate this, we will need to perform a user survey. Desktop usage tends to be a lot of idling, keeping the started application running on the desktop. This is probably the case for MATLAB, Notebooks, and R-Studio.

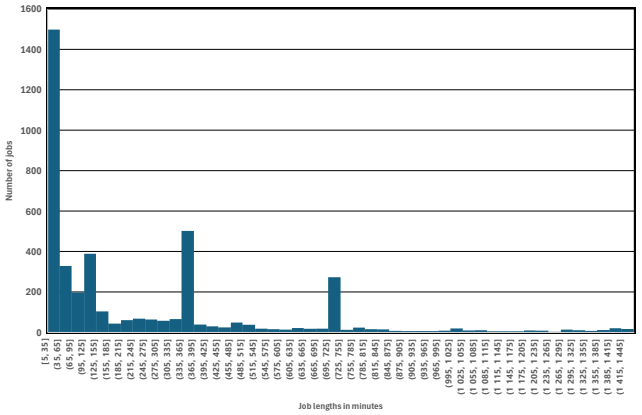

Fig. 17. Typical job lengths.

The spikes between (365, 395) and (725-755) are probably due to the default job lengths that can be selected in the user interface.

X. Challenges and Future Work

The biggest challenge for the LUNARC HPC Desktop is making users aware of using a remote desktop environment with our resources. There are many reasons for this. First, the center's user documentation should feature the desktop option prominently. If there are multiple ways of accomplishing a task, the desktop option should be the first option for users. Secondly, outreach is vital. Make sure that the desktop is marketed in the organization. Talk to communicators. Convince staff that publishes monthly newsletters and journals to write about the services provided by the center. Third, the use of desktops in education, both at the graduate and PhD levels. The desktop environment is very well suited to provide a teaching environment for subjects with a heavy emphasis on computing, visualization, and storage. We have started to improve our documentation, and LUNARC resources are currently used by three larger courses, both on graduate and PhD levels.

The desktop environment is continuously developed to handle upcoming user requirements. One of the first tasks being worked on is to package the installation of GfxLauncher as a pip package that can be easily installed. Currently, a lot of manual work is involved in installing the software. One challenge for the future is ensuring we can support upcoming standards for hardware-accelerated graphics. Currently, there are some limitations in the VirtualGL package for using Vulkan [27] as a graphics API. Some work is ongoing in the TigerVNC project to natively support hardware-accelerated rendering directly in VNC. We are currently evaluating this as a proof of concept.

XI. Conclusion

The development of the LUNARC HPC Desktop environment has been a long journey. In the early days, pushing the idea of using remote desktop solutions for interactive HPC was challenging when all you needed was an SSH-based terminal. The solution that could be used on all platforms was a good choice. This enabled many more users actually to try this kind of environment. Another important conclusion was that developing solutions with a manageable complexity level is essential. Many solutions require a lot of work to configure and maintain. By using the tools available in our resources, such as Thinlinc and SLURM, it is possible to provide a solution that both scales and is easy to maintain. Also, making it easy for research engineers to add new applications is vital to keep the platform modern and relevant for our users.

Another benefit of the LUNARC HPC resource compared to other solutions such as Open OnDemand and UCloud is that adding support for additional applications is just a matter of creating a start script with meta tags describing the application's requirements. Existing scientific software installation repositories such as EasyBuild [28] and Spak [29] can be used in these scripts. Supporting additional application versions is

often just a question of copying the existing launcher script and updating the version number.

Another aspect is that providing the source for GfxLauncher as open source has also opened up the possibility of getting feedback from other centers and users. Currently, the system is deployed at the NAISS [30] Dardel [31] resource at KTH Royal Institute of Technology and the MAX IV synchrotron [32] at Lund University.

ACKNOWLEDGEMENTS

We want to acknowledge SNIC for funding hardware for the prototype developed and the development effort to adapt the GfxLauncher framework for the NAISS Dardel system. As a branch of NAISS, LUNARC will continue developing the framework and making it available on upcoming NAISS resources.

REFERENCES

[1] “Linux Remote Desktop based on open-source — ThinLinc by Cendio.” Cendio. https://www.cendio.com/ (accessed Aug 21, 2024).
[2] ”GFX Launcher - An application launcher framework for SLURM 0.9.8 documentation” GFX Launcher. https://gfxlauncher-documentation.readthedocs.io/en/latest/ (accessed Sep 23, 2024).
[3] “Project Jupyter — Home” Jupyter. https://jupyter.org/ (accessed Aug 21, 2024).
[4] “Center for Scientific and Technical Computing — LUNARC” LUNARC. https://www.lunarc.lu.se/ (accessed Aug 21, 2024).
[5] “SLURM Workload Manager - Overview” SLURM. https://slurm.schedmd.com/overview.html (accessed Aug 21, 2024).
[6] ”PCoIP — Remote Desktop — Virtual Workstation — Teradici” Teradici. https://www.teradici.com/ (accessed Aug 21, 2024).
[7] ”NoMachine - Free Remote Desktop For Everybody” NoMachine. https://www.nomachine.com/ (accessed Aug 21, 2024).
[8] ”RealVNC - Remote access software for desktop and mobile” RealVNC. https://www.realvnc.com/ (accessed Aug 21, 2024).
[9] ”TigerVNC” TigerVNC. https://tigervnc.org/ (accessed Aug 21, 2024).
[10] ”NICE DCV - Deliver high-performance remote desktop and application streaming” Nice DCV. https://aws.amazon.com/hpc/dcv/ (accessed Sep 23, 2024)
[11] ”NoMachine - Fast, secure, easy way to get to your stuff. Wherever you are.” NoMachine. https://www.nomachine.com/ (accessed Sep 23, 2024)
[12] ”JupyterHub - A multi-user version of the notebook designed for companies, classrooms and research labs” JupyterHub. https://jupyter.org/hub (accessed Sep 23, 2024)
[13] ”Open OnDemand - Connecting Computing Power With Powerful Minds” Open OnDemand. https://openondemand.org/ (accessed Sep 23, 2024)
[14] ”UCloud UCloud User Guide” UCloud. https://docs.cloud.sdu.dk/ (accessed Sep 23, 2024)
[15] ”VirtualGL - Main / VirtualGL” VirtualGL. https://virtualgl.org/ (accessed Aug 21, 2024).
[16] ”COSMOS — LUNARC” COSMOS. https://www.lunarc.lu.se/-resources/cosmos/ (accessed Aug 21, 2024).
[17] ”Welcome to Python.org” Python. https://www.python.org/ (accessed Aug 21, 2024).
[18] ”Qt for Python - Qt for Python” Qt for Python. https://doc.qt.io/qtforpython-5/
[19] ”X.Org” X.Org. https://www.x.org/ (accessed Aug 21, 2024).
[20] ”MathWorks - Makers of MATLAB and Simulink” MathWorks. https://www.mathworks.com/ (accessed Aug 21, 2024).
[21] ”ParaView - Open-source, multi-platform data analysis and visualization application” ParaView. https://www.paraview.org/ (accessed Aug 21, 2024).
[22] ”Next generation HPC Desktop” Next generation HPC Desktop. https://next-generation-hpc-desktop.readthedocs.io/en/latest/ (accessed Aug 22, 2024)
[23] ”Lmod: A New Environment Module System” Lmod. https://lmod.readthedocs.io/en/latest/ (accessed Aug 21, 2024).
[24] ”Download RStudio — The Popular Open-Source IDE from Posit” RStudio. https://posit.co/products/open-source/rstudio/ (accessed Aug 21, 2024).
[25] ”Abaqus/CAE — SIMULIA - Dassault Systèmes” Abaqus/CAE. https://www.3ds.com/products/simulia/abaqus/cae (accessed Aug 21, 2024).
[26] ”Home — Spyder IDE” Spyder IDE. https://www.spyder-ide.org/ (accessed Aug 21, 2024).
[27] ”Home — Vulkan — Cross platform 3D Graphics API” Vulkan. https://www.vulkan.org/ (accessed Aug 21, 2024).
[28] ”EasyBuild - Building software with ease.” EasyBuild. https://easybuild.io/ (accessed Aug 21, 2024).
[29] ”Spack - A flexible package manager supporting multiple versions, configurations, platforms, and compilers.” Spack. https://spack.io/ (accessed Aug 21, 2024).
[30] ”NAISS — The National Academic Infrastructure for Supercomputing in Sweden” NAISS. https://www.naiss.se/ (accessed Aug 21, 2024).
[31] ”Dardel — NAISS” Dardel. https://www.naiss.se/resource/dardel/ (accessed Aug 21, 2024).
[32] ”MAX IV - We make the invisible visible” MAX IV. https://www.maxiv.lu.se/ (accessed Aug 21, 2024).