Towards a GPU-accelerated Open Source VDI for OpenStack

Manuel Bentele, Dirk von Suchodoletz, Manuel Messner, and Simon Rettberg

Computer Center, University of Freiburg, Germany
{manuel.bentele,dirk.von.suchodoletz,manuel.messner,simon.rettberg}
@rz.uni-freiburg.de
https://www.rz.uni-freiburg.de

Abstract. Starting from summarizing preexisting work and technologies, this paper introduces the necessary considerations and steps to develop a fully Open Source Virtual Desktop Infrastructure (OSVDI) on top of OpenStack. Unlike other already available open-source solutions, a Virtual Desktop Infrastructure (VDI) with access for 3D or video rendering using Graphics Processing Units (GPUs) caters to various use cases like GPU-accelerated desktop environments, remote visualization, or large-scale desktop virtualization. Such use cases require the integration of special-purpose hardware in cloud servers and the sharing of devices as well as resource scheduling, a user interface for session or resource selection, and efficient remote transport. The envisioned OSVDI is yet in its infancy growing from previous work to provide efficient large-scale remote access to existing PC pools in the bwLehrpool service for various teaching and learning environments. This preliminary cloud provides insights and experience to design the necessary (additional) OpenStack components and configurations.

Keywords: VDI · OpenStack · Desktop Virtualization · Remote Access.

1 Motivation

Like other IT services, the traditional desktop computer and workstations are getting centralized and cloud-operated as well. This paradigm shift supersedes decentralized, distributed machine deployment and operation. The advantages cover flexible provisioning of a wide range of software environments, reduced administration, better access control, higher security to more efficient and flexible hardware and software utilization as well as fostering of green IT efforts. Plus, it caters to the expectations of modern home office schemes offered by an increasing number of employers.

The modern-day term widely used for this kind of machine operation is Virtual Desktop Infrastructure (VDI). In an abstract and generic view, a VDI is composed of a set of interlinked components, requiring a (virtual) machine or compute node with graphic rendering capabilities (software rendering or dedicated GPU). For scalable infrastructures, the focus lies on Virtual Machines

(VMs) since they allow the most flexible deployments and allow the implementation of security designs as well as the separation of different access and data domains. Further components provide the preprocessing of rendered desktop content: Grab a framebuffer from a VM, encode it as a video stream, and transport video streams to the remote (thin) client over the LAN. To host multiple clients, various management and multiplexing services are required. Finally, modules for authentication, Quality of Service (QoS) for networks, or stream encryption may play a role as well, but will not be covered in this paper. Any VDI requires an interplay of several components and building blocks in hardware and software. Typically, the commercial vendors bundle those modules and attach a label to it to market them as a product.[1] There is no direct equivalent in the open-source domain but a range of components that could be combined to achieve similar objectives are mostly there [17]. Often these approaches lack seamless integration and ease of setup.

The authors got involved with the VDI topic when developing remote access to original (emulated) computer environments [19] or when providing remote access to hundreds of PC pool machines offering a wide range of teaching and learning software environments provided through the bwLehrpool service [2]. The latter provides a perfect baseline and playground to discuss and develop ideas for an OSVDI. Further use cases can profit from an OSVDI. This includes remote teaching scenarios on standard IT setups, access to high-performance analysis workstations for graphical workflows in various domains in science, as well as standard desktop environments in labs and offices. Additionally, the typical remote visualization scenarios in High Performance Computing (HPC) clusters would profit, where VirtualGL[2] lacks appropriate support. A VDI provides a building block to (re)centralize computer infrastructures and allow a much more flexible utilization of resources. Further, we got involved in the PePP project,[3] promoting the idea of using controlled IT environments in electronic assessments requiring VDI solutions to host up to several hundred students in parallel.

The paper will be structured as follows. We will provide an updated overview on existing implementations of GPU virtualization, relevant aspects around remote access protocols, and desktop access including provided interaction and transport channels. From the summary of the state-of-the-art, we try to identify all relevant gaps and provide a first outline of the envisioned setup and structure of the OpenStack VDI extension to be developed.

[1] Desktop virtualization is dominated for the time being by Citrix or VMware products [6]. These commercial solutions are seldom attractive for research and education purposes as they involve proprietary components and come with significant costs.

[2] See https://www.virtualgl.org

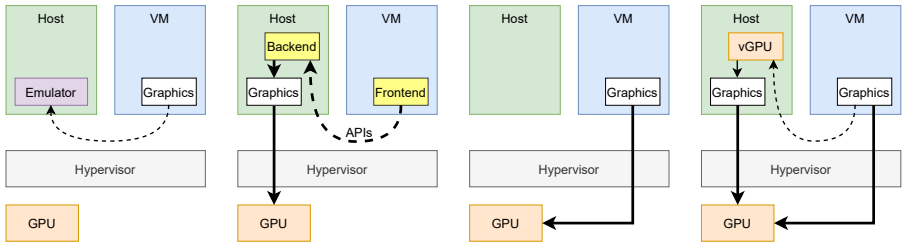
[3] Partnership for innovative E-Assessments – Joint Project of the Baden-Württemberg Universities, see https://www.hnd-bw.de/projekte/pepp

2 Related work

The general idea of remote access to desktop environments is nothing new, dating back to the era of X11/XDM, LTSP, Virtual Network Computing (VNC), XEN/Citrix, VMware, and the Windows Terminal Server. A couple of different implementations evolved to provide the graphical output of the desktop over LAN or WAN connections to the user and user input back to the central infrastructure. Depending on the actual protocol, additional channels like uni- or multi-directional audio, USB redirection, or optical drive access are implemented as well. Before suggesting further development and improvement of existing components, we summarize the state-of-the-art for relevant modules required for an OSVDI.

2.1 GPU virtualization

Virtualization of GPUs has become mandatory, starting with the evolution of full machine virtualization to render and obtain a machine's graphical output. Nowadays, there are several approaches to virtualize GPUs for VMs as presented in [23,22,17] and depicted in Fig. 1. All these approaches aim to abstract GPUs either by emulation of full GPU hardware or by virtualization of GPUs.



(a) Graphics emulation. (b) Paravirtualization. (c) Direct GPU passthrough. (d) Mediated GPU passthrough.

Fig. 1. GPU virtualization approaches.

In the early beginning, the *emulation* of graphic devices, especially VGA adapters, evolved to provide virtual GPUs to VMs. Most virtualizer solutions, like Quick Emulator (QEMU), Virtualbox, or VMware Workstation, provide legacy support for graphics emulation according to the approach visualized in Fig. 1a. The graphics emulation is provided by an emulator on the host system that emulates a framebuffer for graphics rendering. The emulator does not require access to any physical GPU resulting in increased flexibility and scalability. However, the low graphics performance through missing GPU acceleration is a major limitation for graphics-intensive applications [22].

In order to improve the graphics performance, optimizations like *paravirtualization* (Fig. 1b) have been developed [8]. Paravirtualization improves the emulation approach by the use of predefined Application Programming Interfaces (APIs), which are exposed by the host system. Those APIs provide the functionality to offload graphic commands from a graphics frontend driver in a VM to a graphics backend on the host system (which is often referred to as API forwarding). The computation of graphic commands takes place in the graphics backend, which itself uses a GPU of the host system for GPU-accelerated rendering. GPU-accelerated rendering increases the overall graphics performance compared to pure emulation [8], but requires the host system to provide access to a physical GPU through a suitable graphics driver. This approach does not allow a VM to acquire a physical GPU directly.

To accelerate graphics performance further, a *direct GPU passthrough* can be implemented as shown in Fig. 1c. This approach allows a VM to directly access a physical GPU using hardware capabilities for direct assignment of I/O devices, e.g. an I/O Memory Management Unit (IOMMU). These capabilities provide a hardware-based memory address mapping to map GPU-related device addresses and interrupts into the memory space of a VM without the need for any software-based mapping implementation. A major advantage is the fast and transparent access to the entire GPU. Therefore, a VM is only required to use the native graphics driver to be able to directly access the physical framebuffer and rendering capabilities of the GPU. A drawback here is the fact that the GPU resources (e.g. framebuffer) are entirely and statically assigned to the VM and cannot be shared with the host system. This limitation can be circumvented explicitly using dedicated shared memory technologies (e.g. in *Looking Glass*[4]). Nevertheless, results from [3] show that the graphics performance of the direct GPU passthrough approach accelerates VDI performance and improves user experience.

A rather novel concept to achieve GPU resource sharing among a host system and several VMs while preserving direct access to the GPU is implemented as *gVirt*, also known as *Intel GVT-g* [22]. This approach is often referred to as *mediated GPU passthrough* and is visualized in Fig. 1d. Following this approach, resources of a physical GPU can be partitioned into Virtual GPU (vGPU) instances. Each of these vGPU instances is a fully virtualized GPU providing its own framebuffer and rendering capabilities. All vGPU instances are managed by the host system and can be acquired and accessed by any VM or the host system itself (e.g. to share a framebuffer). A VM has direct access to the physical GPU resources of its acquired vGPU. The partitioning can either take place temporal or spatial. Intel GVT-g implements temporal partitioning in a time-shared manner (scheduling), whereas the latest dedicated GPU products (e.g. A100 MIG) from Nvidia implement spatial partitioning without any scheduling. Both partitioning methods increase flexibility, scalability, and efficiency, especially for graphic workloads that do not fully saturate the entire compute capacity of a

[4] See https://looking-glass.io

GPU. Besides flexibility, evaluation results for the mediated GPU passthrough approach show, that GPU workloads achieve almost native performance [22].

2.2 Video encoding and decoding

Any graphics output of each VM as part of a VDI is available as a stream of images in a raw framebuffer format. Without proper preprocessing and compression, such a stream requires an increasingly high network bandwidth on an increasing resolution during the real-time transport to a remote thin-client (e.g. 3+ Gbit/s for Full HD). But network bandwidth is often limited in a WAN, even in a LAN, especially if the network requirements for a VDI are underestimated [16]. Therefore, a preprocessing and adjustable encoding of the framebuffer as a compressed video stream is necessary to lower the overall network throughput per virtual desktop instance while preserving a high display quality for an acceptable Quality of Experience (QoE).

While the traditional remote transport protocols mainly use variants of JPEG encoding, this is less suitable to encode fast-changing video or 3D content from a virtual desktop session [10]. For that purpose, variants of video encoding are used, most widely the Advanced Video Coding (AVC) [4] and the High Efficiency Video Coding (HEVC) [5]. AVC, also referred to as H.264 or MPEG-4 Part 10, is a video compression standard based on block-oriented, motion-compensated integer-DCT coding. HEVC, also known as H.265 and MPEG-H Part 2, is a video compression standard designed as part of the MPEG-H project as a successor to AVC, optimized for high resolutions beyond high definition video formats [20]. Both codecs achieve high compression rates resulting in low network bandwidth. Like many other codecs, AVC and HEVC can be configured and adjusted by several parameters. For example, there is a parameter for video quality that influences the network bandwidth and QoE.

For an OSVDI, it is important that there are free and open-source implementations of video codecs. The open-source library *libavcodec*[5] implements encoding and decoding for AVC and HEVC based on the *x264*[6] and *x265*[7] software library. In addition to that, libavcodec contains decoder and sometimes encoder implementations of several other proprietary codecs, for which no public specification has been released. As such, a significant reverse engineering effort is part of libavcodec development.

Video encoding could be sped up significantly involving specialized hardware, which can be utilized by libavcodec, too. Most GPUs on the market for the server-side, as well as the client-side, contain already one or several built-in video encoder and decoder units. These units perform video encoding or decoding based on AVC or HEVC without wasting compute capacity of a CPU. If further video codecs are considered for the development of an OSVDI, the hardware acceleration for that codec should be checked first on all intended VDI devices.

[5] libavcodec is part of the FFmpeg project, http://ffmpeg.org
[6] See https://www.videolan.org/developers/x264.html
[7] See https://www.videolan.org/developers/x265.html

This scrutiny ensures an efficient video encoding or decoding with a high QoE even on low-power remote (thin) clients. Alternatively, if hardware acceleration for a specific codec is missing, an automatic codec selection is conceivable. Such an automatism preserves compatibility, especially for legacy remote (thin) clients as well as flexibility for different type of clients like browser-based or dedicated remote (thin) clients.

2.3 Remote desktop transport

Transport protocols. Remote desktop transport protocols specify the communication between a remote (virtual) machine and a (thin) client. Such transport protocols have in common, that they establish a bi-directional communication for data exchanges. One direction of the communication is used to transfer graphic output from a desktop session on a remote (virtual) machine to a (thin) client for visualization purposes. The second direction is used to transfer user inputs (e.g. mouse and keyboard events) from a user's (thin) client to the remote (virtual) machine. Both directions of the communication allow a seamless remote desktop interaction while facing the challenge for a low latency to achieve an acceptable QoE.

Transport protocols for remote desktops can be characterized by their supported amount of graphics primitives for a transfer to a client. The VNC protocol [15] implements the Remote Framebuffer Protocol (RFB) [14]. RFB works at the framebuffer level and watches for bitmap changes on a (virtual) machine's framebuffer. Then, the protocol streams those bitmap updates block by block to the client. Therefore, RFB uses a single graphics primitive to update bitmaps on a certain screen location which results in high network bandwidth and poor video performance [12]. To improve video performance, a VNC setup can be extended with a VNC proxy accelerator as shown in [21]. The OpenStack cloud platform contains a built-in VNC implementation called *noVNC*.[8] noVNC provides a web client for VNC and performs worse than the external Guacamole VNC implementation [3].

A similar protocol to RFB is the Thin-Client Internet Computing (THINC) protocol. THINC implements more low level graphic primitives which improves RFB and results in a better video performance [1], which even outperforms older Remote Desktop Protocol (RDP) versions. RDP is an proprietary remote desktop transport protocol from Microsoft and supports a significant number of high-level graphic primitives including optimizations like caching of already transferred primitives or support for glyphs. These optimizations offer a high QoE while preserving low network bandwidth for normal productivity desktop work [12] compared to VNC [24].

The Simple Protocol for Independent Computing Environments (SPICE)[9] is an open-source alternative to the proprietary RDP. Similar to RDP, SPICE supports high-level graphic primitives and is intended and optimized for remote

[8] See https://novnc.com
[9] See https://www.spice-space.org

access to VMs. Other optimizations include an additional display mode to improve QoE [11] and further interaction features like audio support, folder sharing, USB redirection, and reduced response time [9].

Both protocols, RDP and SPICE, benefit from the transport of high-level graphic primitives and the rendering of those primitives on the (thin) clients. However, this is problematic if a user interacts with a graphic-intense application but its (thin) client is not equipped with the required rendering capabilities [10,17,13]. In addition to that, both protocols are not very suitable for large desktop screen areas which change rapidly (e.g. during video playback) [10,7]. Therefore, work in [7] improves the QoE of SPICE by a motion-based JPEG compression for high-resolution video playbacks while lowering network bandwidth.

A completely different approach [18] for optimizing video playbacks is the detection of video streams within a desktop session. The detected video streams are directly transferred to the (thin) client. On client-side, those video streams are decoded and visualized using hardware acceleration. Using this approach, the network bandwidth is limited drastically during video playback. A similar concept pushes the detection of video streams one step further and encodes the entire remote desktop screen with the AVC/H.264 codec as video stream [25]. Compared to VNC, THINC, and RDP, this enhancement achieved the lowest latency in WAN environments while preserving a high QoE, even for graphic-intensive multimedia applications. A full remote desktop screen encoding is available for SPICE, too. It can be enabled using the additional *SPICE Streaming Agent*[9] which runs in a VM. The agent captures and encodes the entire screen for a subsequent transport via SPICE. As of the writing of this paper, the agent based approach is marked as experimental and requires the guest system to be prepared.

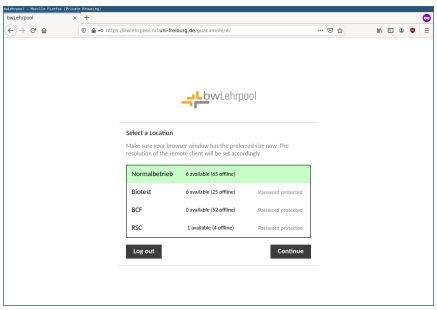
(Thin) clients. The term client refers to a device that allows a user to interact with a remote (virtual) desktop session. Therefore, a client is equipped with an application to receive desktop content from a remote (virtual) machine and send user input back to this machine. Such an application implements one or several remote desktop transport protocols like VNC, SPICE, or proprietary protocols (e.g. RDP) and is often realized as a web application (e.g. noVNC) or as a native (standalone) program (e.g. *virt-viewer*[10] which implements VNC and SPICE).

A client device is mostly a PC optimized for remote interaction, but can be a laptop, mobile phone, tablet, or single-board computer, too. If such a device is a low-performance computer, we call this type of client a *thin client* (e.g. a low-power tablet). Nowadays, mobile devices are equipped with a web browser. Therefore, a browser-based solution (web application) supports more client devices, whereas dedicated (thin) clients provide more interactions features, like USB redirection and multi-directional audio exchange. Nevertheless, most client devices support hardware-accelerated decoding of video streams (e.g. AVC or HEVC) and 3D rendering capabilities.

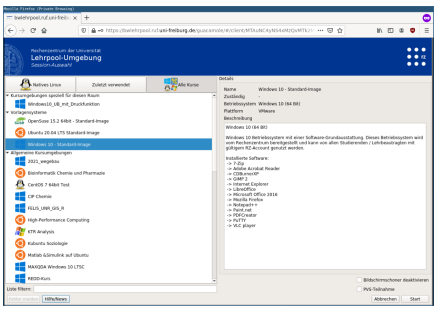
[10] See https://virt-manager.org

2.4 Preliminary work in bwLehrpool

The current pandemic demonstrated the need for adequately scaling online desktop solutions and thus pushed the implementation of a remote access for the bwLehrpool service based on Guacamole/VNC for both remote teaching and electronic exams. As an ad-hoc solution, the existing computer labs — closed due to the pandemic — were re-purposed as tiny cloud nodes, hosting one student session each. This posed the challenge of adequately assigning resources to students depending on workload; it could not be done dynamically as is possible on large servers in a cloud environment, but must be decided beforehand, since scheduling a student requiring a 3D-intense environment on a small PC with integrated graphics would lead to a sub-par experience. The solution was to partition the available machines by their specifications, and prompting the student with a selection screen (see Fig. 2a) when logging into the service, optionally protecting the more capable systems with a password.



(a) Step 1: PC pool selection.



(b) Step 2: VM image selection.

Fig. 2. Central entry point to bwLehrpool remote access through the browser and consecutive selection of a desired software environment.

Moving parts of the PC lab infrastructure to the cloud offers an expanded service based on familiar and longer-established environments, but ensures business continuity after students return to the labs post-pandemic. Resources can be provided very quickly on an ad-hoc basis so that the requirements of courses and the needs of students can be matched more precisely. These developments allow for ubiquitous teaching in presence and at home. Lecturers could benefit significantly if computer-based teaching infrastructure could be provided on-demand and tailored to students at any time, not limited to on-campus PC labs. Triggered by the increasing demand for high-performance and easy-to-use GPU resources, the bwGPUL project[11] focused on extending the existing bwLehrpool setup to access General Purpose GPUs (GPGPUs) from within VMs for tools

[11] See https://www.bwlehrpool.de/bwgpul. The project focused on the utilization of existing hardware to avoid too expensive installations of special server hardware.

that require hardware acceleration in the field of artificial intelligence and machine learning.

2.5 OpenStack – basis and missing pieces

OpenStack is a free cloud platform, most commonly deployed as Infrastructure-as-a-Service and licensed under the Apache License 2.0. The platform is composed of several components and services. Each component and service is responsible for a certain set of tasks and provides a RESTful API for communication. OpenStack's main focus is to provide an infrastructure for VMs, their storage and their network. Although, there are many components which expand this functionality in adjacent areas, like container management or bare-metal computation (see Table 1).

Table 1. Important OpenStack components

Nova	Management of VMs	**Neutron**	Software Defined Networks
Keystone	Authentication	**Horizon**	Official dashboard
Cinder	Block Storage	**Glance**	Base images and metadata

Relevant for our considerations is Nova. It manages the lifecycle (including scheduling) of the single VMs and therefore uses many APIs of other components. Nova offers a backend with a common interface to various lower level technologies (e.g. libvirt with Kernel-based Virtual Machine (KVM) infrastructure, XEN and Hyper-V) for providing VMs.

An integration of the mediated GPU passthrough technology is part of the OpenStack platform since version Queens,[12] although it's listed as experimental until version Train.[13] But even in the most up-to-date version, as of this writing, this module has severe caveats and are only available for the libvirt/KVM backend of Nova.[13] Besides the mediated GPU passthrough support in OpenStack, there are other frameworks and tools, such as *LibVF.IO*,[14] available to orchestrate VMs and vGPU instances. LibVF.IO automates the creation and configuration of VMs and vGPU instances, but cannot provide any resource scheduling for cloud computing as OpenStack does.

There are VDI plugins available for OpenStack, such as plugins for Citrix XenDestop, Microsoft RDS, or Apache Guacamole.[15] Most plugins only address commercial VDI solutions, or in the case of Apache Guacamole, just support the VNC or RDP transport protocol. Native GPU acceleration and SPICE support are missing while using these plugins.

[12] See https://docs.openstack.org/nova/queens/admin/virtual-gpu.html
[13] See https://docs.openstack.org/nova/train/admin/virtual-gpu.html
[14] See https://libvf.io
[15] See OpenStack Summit – Boston, MA (2017): https://www.openstack.org/videos/summits/boston-2017/virtual-desktop-infrastructure-vdi-with-openstack

3 Proposed system architecture

A cloud-based desktop infrastructure is more efficient and flexible than the earlier presented approach using commodity hardware from (unused) computer labs, as it allows the assigning and sharing of system resources between multiple VMs. Only in rare cases, where students require comparatively expensive computing resources, a more careful approach to resource allocation must be taken, e.g. for GPGPU tasks that have high Video RAM (VRAM) requirements that could only be fulfilled by a few expensive, specially equipped cloud nodes. For this reason, relevant information is pinned as meta-data to the VM in question, and the student's session is scheduled to a node depending on its meta-data. In the first Guacamole-based implementation of the bwLehrpool remote access, this was simply not possible, as the workflow required the student to first select the machine type they wanted to use (Fig. 2a), and only then be presented with the list of available VMs (Fig. 2b), due to technical reasons and time constraints. Our vision for the next version is to have a public VDI web application where users log in, select a VM they want to use, and finally be scheduled to an appropriate cloud node. Additional logic could be added, e.g. limiting or skipping the VM selection for a student or user group depending on the time of day, day of the week, etc. This can be useful for conducting cloud-based e-exams, to prevent students from selecting a wrong VM, and also preventing users not belonging to the group of examinees from booting into that VM.

Our focus is mainly on the mediated GPU passthrough for further development of an OSVDI because this approach combines the flexibility of emulation and paravirtualization with the performance boost of direct GPU passthrough. Since an OSVDI is built on open-source software, we use the Linux operating system on the host system. Linux already provides the *mdev* subsystem and tools for mediated devices (vGPUs) and their device drivers. Using this subsystem has the major advantage that the Linux host system can manage all vGPUs and mediate shared access. This shared access allows the Linux host to access the framebuffer of a vGPU directly in a read-only manner with low overhead and latency (e.g. with *dma-buf*), which does not work out well with the direct GPU passthrough approach as part of an OSVDI. The direct access to a framebuffer of a vGPU means in terms of an OSVDI that the Linux host system can obtain the graphics output of any VM (virtual desktop) and can control those output for further processing and transfer to remote (thin) clients.

Access to a VM session can then be implemented via two methods: A browser-based approach, using modern technologies like WebAssembly, WebUSB and MediaDevices, resulting in immediate access from a wide range of devices like laptops, tablets and mobile phones. Still, the alternative approach of using a dedicated native application the user has to install first can offer even greater integration with the user's system, as well as yielding better performance depending on its use case.

We see at least three distinct use cases for an OSVDI in conjunction with OpenStack supplied through a suitable orchestration framework:

1. OpenStack user dedicated interactive VM in stateful operation as already implemented to get started via the dashboard and either using the native noVNC or some guest system remote access built-in like VNC or RDP.
2. (Large scale) virtual PC pool setups like offered in stateless mode by the bwLehrpool service with remote access. Users do not have an associated project or personal VM in OpenStack and thus requiring a dedicated entry point (Fig. 2a) and the possibility to choose the desired VM (Fig. 2b) kicked-off from a template. This scenario matches to the objectives of the respective sub-project in PePP.
3. Special purpose (powerful) virtual workstations offering tools for interactive image analysis dedicated through a booking system preallocating resources upon request (no direct relation between OpenStack users and persons requiring such a VM). Those virtual workstations could be offered through a selection list and mapped into the project concept of OpenStack owned by the lab requiring such software environments.

The first use case is already available in the standard setup, but the other two need some consideration regarding scheduling, resource allocation and means of access. A VDI integration into OpenStack would require to implement two modules: A service for managing all relevant VDI aspects and a Nova plugin connecting Nova and the VDI service.

VDI service. Like the other OpenStack services, the VDI service offers a RESTful API for inter-service-communication. It's task is to manage jobs, their requirements and the lifecycle of VDI VMs via Nova. This can be broken down further into different aspects as follows.

Reservations. A common problem in clouds used in teaching is that often VMs are started once, being used once a week for an hour and idle in the meantime. As GPUs are a comparable expensive resource, a job based scheduling scheme, like in HPC, is more efficient. So GPUs can be reused during the idle times by other jobs.

Priorities. When looking at jobs, there are jobs with user interaction (e.g. classes), thus having a time dependency and jobs that just have to calculate some results. The service's task would be to prioritize the first category over the second one and make sure all needed resources are available when e.g. classes start by killing or pausing lower priority VMs and rescheduling them after e.g. the class has finished.

Job Handling and Registration. The last job of the service would be to handle the different jobs. Some discussion is needed whether an existing job scheduler should be included, or whether it should be implemented from scratch. Also, the service would offer a usable interface for job administration, as well as, registration and for passing all relevant requirements. This can be extended and simplified with an user interface, e.g. a dashboard. It does not necessarily need

to be included into Horizon, because the job based scheduling suggested here is orthogonal to the normal usage of VMs in OpenStack.

Nova plugin. The second module would be a Nova plugin which connects Nova and the VDI service during the VM creation process. It's task is to provide the service with all relevant metadata and properties as well as to respect (upcoming) jobs during normal VM scheduling.

4 Work program and planned efforts

For the imagined OSVDI we plan three to five major development cycles and a minimum viable product approach. In a precursor the existing Guacamole bwLehrpool remote access should get improved through hardware rendering and stream encoding deploying the Intel GVT-g desktop graphic architecture together with the KVM infrastructure as a Linux-based hypervisor and produce an assessment of ease to use and stability. This will get implemented as an enhanced bwLehrpool service and prove the capabilities of the existing kernel drivers regarding GPU virtualization and hardware partitioning. We will use the SPICE client and Looking Glass as a prove and performance measure when accessing the virtual framebuffer for AVC/H.264 encoding and transport. Upon this we will explore how to encode with low latency, and how to send it to browsers and display the content there with low latency. This provides a possible baseline to check certain expectations and features before delivering similar services like those for an OpenStack cloud.

In a second milestone, we focus on a basic VNC model (leaving further improvements of remote access to parallel or later developments) in the cloud including orchestration of resources which covers the scope of our contribution to the PePP project. This milestone starts to extend the OpenStack framework for missing components and modules. First, we develop concepts for PC pool scheduling on shared and non-shared hardware resources. Further, this milestone deals with the challenges of a suitable access broker to distribute users requesting certain types of desktops onto a suitable VM. The access broker includes the provisioning of basic interaction channels starting from a single PC pool setup.

While the previous step focused on a basic integration and the outline of strategic components the next milestone focuses on the special hardware virtualization and integration parts both from the viewpoint of the guest systems and as encoding devices from the host perspective (Fig. 3). The remote access should enjoy at least an enhanced hardware-backed video stream transport model for the remote visual cloud. Later milestones should deal with further remote interaction channels and further features and improvements for typical VDI setups like suspend and resume of interactive desktop sessions.

Starting during the second milestone measures should be taken to form a sustainable community and financing concept around the proposed service. Both

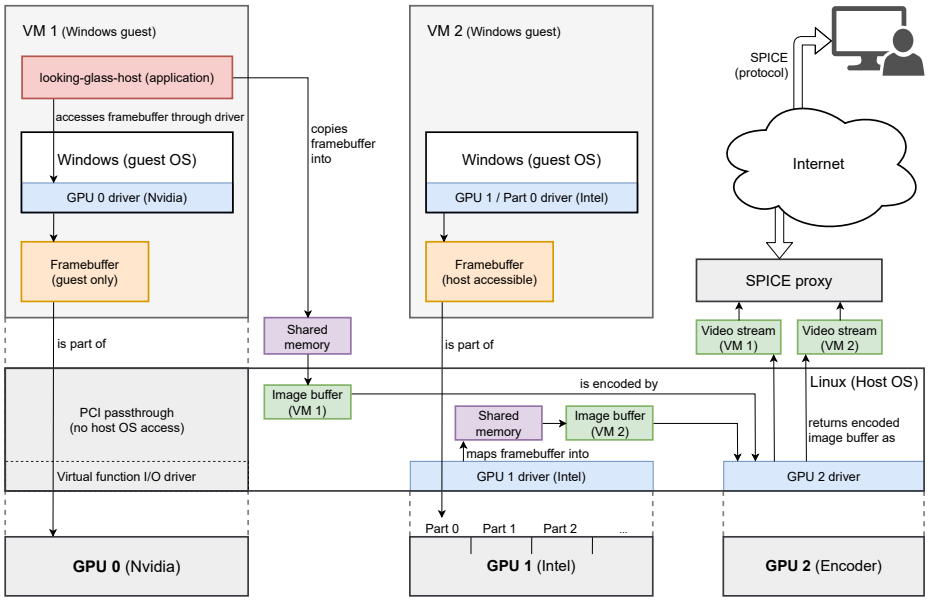


Fig. 3. GPU virtualization options and remote transport: Direct GPU passthrough (left) or mediated GPU passthrough (middle) for Windows guests, where a framebuffer (virtual desktop) of each Windows guest is encoded as video stream (right), which is transferred to a remote client using the SPICE protocol.

ongoing support, code maintenance and future development are to be supported through some stable organizational structure.

5 Conclusion and outlook

This paper intended not only to provide an exhaustive update on technology development around OSVDI but being at the same time a call for collaboration and feedback from further interested parties.[16] Up to now we were able to progress significantly in our first milestone exploring the foundations of the envisioned OSVDI. The bwLehrpool remote access including resource allocation upon demand runs smoothly with good user feedback. Starting into our new project on enabling large scale e-assessments we hope to run tasks in parallel to a certain degree to speed up development if resources permit. Certain tasks can get outsourced, e.g. the programming of well-defined software components, if additional funding is acquired. With the start of the PePP project we work on improving the project management wrt. an OSVDI solution by consolidating the code repository, pushing developments upstream to the benefit of the wider community including hardware vendors and software developers.

[16] See project information and resources at https://github.com/bwLehrpool/osvdi

As a provider of large scale research infrastructures the computer center strives to integrate the activities into other evolving infrastructures like the German National Research Data Infrastructure (NFDI) and participates in further grant applications to bolster the efforts. Like in other software projects, we are standing on the shoulders of giants and depend on developments like hardware virtualization in the Linux kernel, the SPICE protocol, and OpenStack. 1.5 FTE working at the endeavor at the moment and are financed for the coming two years. To gain sustainability, we offer proper support, maintenance of the code and collaboration with the relevant software projects and hardware vendors.[17]

On the hardware side, the VDI market strongly evolved around Nvidia hardware [17] which is unfortunately riddled with an incomplete or fragmented open-source Linux driver support and/or prohibitive software licenses on core features like GPU partitioning. A future chance stems from the tendency of hardware vendors to create more focused products for computational purposes and gaming or visualization. If there is e.g. a dedicated adapter just for video encoding this might simplify setups as no virtualization/partitioning is required.

Acknowledgments

Part of the activities and insights presented in this paper were made possible through preliminary work in the bwGPUL project supported by the Baden-Württemberg Ministry of Science, Research, and the Arts, the collaboration in the PePP project (FBM2020-VA-77-8-01241), and the German NFDI initiative (NFDI 7/1).

References

1. Baratto, R.A., Kim, L.N., Nieh, J.: THINC: A Virtual Display Architecture for Thin-Client Computing. In: Proceedings of the Twentieth ACM Symposium on Operating Systems Principles. pp. 277–290. SOSP '05, Association for Computing Machinery, New York, NY, USA (2005). https://doi.org/10.1145/1095810.1095837
2. Bauer, J., Rettberg, S., Ritter, S., Rößler, C., von Suchodoletz, D., Münchenberg, J.: BWLEHRPOOL – A JOINTLY MANAGED AND FINANCED INTER-UNIVERSITY IT PROJECT. In: EDULEARN19 Proceedings. pp. 5548–5555. 11th International Conference on Education and New Learning Technologies, IATED (July 2019). https://doi.org/10.21125/edulearn.2019.1360
3. Chang, C.H., Yang, C.T., Lee, J.Y., Lai, C.L., Kuo, C.C.: On Construction and Performance Evaluation of a Virtual Desktop Infrastructure With GPU Accelerated. IEEE Access **8**, 170162–170173 (2020). https://doi.org/10.1109/ACCESS.2020.3023924
4. ITU-T: Advanced video coding for generic audiovisual services. Recommendation H.264 and ISO/IEC 14496-1, International Telecommunication Union (2003)
5. ITU-T: High efficiency video coding. Recommendation H.265 and ISO/IEC 23008-2, International Telecommunication Union (2013)

[17] We try to open communication channels to the relevant hardware vendors through ongoing procurements for our cloud and high performance computing infrastructures.

6. Jeong, D., Park, J., Lee, S., Kang, C.: Investigation Methodology of a Virtual Desktop Infrastructure for IoT. Journal of Applied Mathematics **2015**, 1–10 (2015). https://doi.org/10.1155/2015/689870
7. Lan, Y., Xu, H.: Research on technology of desktop virtualization based on SPICE protocol and its improvement solutions. Frontiers of Computer Science **8**(6), 885–892 (2014). https://doi.org/10.1007/s11704-014-3410-5
8. Li, H., Jin, H., Liao, X.: Graphic Acceleration Mechanism for Multiple Desktop System Based on Virtualization Technology. In: 2011 14th IEEE International Conference on Computational Science and Engineering. pp. 447–452 (2011). https://doi.org/10.1109/CSE.2011.82
9. Li, W., Wang, B., Yu, J., Zhu, C., Xiao, S., Sheng, J.: The optimization of Transparent-Desktop service mechanism based on SPICE. Concurrency and Computation: Practice and Experience **28**(18), 4543–4556 (2016). https://doi.org/10.1002/cpe.3858
10. Lin, Y., Kämäräinen, T., Di Francesco, M., Ylä-Jääski, A.: Performance evaluation of remote display access for mobile cloud computing. Computer Communications **72**, 17–25 (2015). https://doi.org/10.1016/j.comcom.2015.05.006
11. Liu, X., Zhu, M., Xiao, L., Jiang, Y.: A VM-shared desktop virtualization system based on OpenStack. AIP Conference Proceedings **1955**(1), 040137 (2018). https://doi.org/10.1063/1.5033801
12. Magaña, E., Sesma, I., Morató, D., Izal, M.: Remote access protocols for Desktop-as-a-Service solutions. PLOS ONE **14**(1), 1–28 (2019). https://doi.org/10.1371/journal.pone.0207512
13. Nehra, S., Kumar, C.: Enterprise Virtual Desktop Infrastructure Architecture on OpenStack Cloud with Lightweight Directory Access Protocol. In: 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO). pp. 1050–1055 (2020). https://doi.org/10.1109/ICRITO48877.2020.9197996
14. Richardson, T., Levine, J.: The Remote Framebuffer Protocol. RFC 6143, RFC Editor (2011), https://www.rfc-editor.org/rfc/rfc6143.txt
15. Richardson, T., Stafford-Fraser, Q., Wood, K., Hopper, A.: Virtual network computing. IEEE Internet Computing **2**(1), 33–38 (1998). https://doi.org/10.1109/4236.656066
16. Rot, A., Chrobak, P.: Benefits, Limitations and Costs of IT Infrastructure Virtualization in the Academic Environment. Case Study using VDI Technology. In: Proceedings of the 13th International Conference on Software Technologies - ICSOFT. pp. 704–711. INSTICC, SciTePress, Porto, Portugal (2018). https://doi.org/10.5220/0006934707380745
17. Smirnov, V.A., Korolev, E.V., Poddaeva, O.I.: Cloud Environments with GPU Virtualization: Problems and Solutions. In: International Conference on Data Mining, Electronics and Information Technology (DMEIT). pp. 147–154 (2015)
18. Su, K., Wang, Z., Lu, X., Chen, W.: An original-stream based solution for smoothly replaying high-definition videos in desktop virtualization systems. Journal of Visual Languages & Computing **25**(6), 676–683 (2014). https://doi.org/10.1016/j.jvlc.2014.09.009
19. von Suchodoletz, D., Rechert, K., Valizada, I.: Towards emulation-as-a-service: cloud services for versatile digital object access. International Journal of Digital Curation **8**(1), 131–142 (2013). https://doi.org/10.2218/ijdc.v8i1.250
20. Sullivan, G.J., Ohm, J.R., Han, W.J., Wiegand, T.: Overview of the High Efficiency Video Coding (HEVC) Standard. IEEE Transactions on

Circuits and Systems for Video Technology **22**(12), 1649–1668 (2012). https://doi.org/10.1109/TCSVT.2012.2221191
21. Taylor, C., Pasquale, J.: Improving video performance in VNC under high latency conditions. In: 2010 International Symposium on Collaborative Technologies and Systems. pp. 26–35 (2010). https://doi.org/10.1109/CTS.2010.5478527
22. Tian, K., Dong, Y., Cowperthwaite, D.: A Full GPU Virtualization Solution with Mediated Pass-Through. In: Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference. pp. 121–132. USENIX ATC'14, USENIX Association (2014), https://www.usenix.org/conference/atc14/technical-sessions/presentation/tian
23. Wang, Z.: An Introduction to Intel GVT-g (2017), https://01.org/sites/default/files/documentation/an_introduction_to_intel_gvt-g_for_external.pdf
24. Wei, W., Zhang, Y., Lu, Y., Gao, P., Mu, K.: A VDI System Based on Cloud Stack and Active Directory. In: 2015 14th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES). pp. 151–154. Guiyang, China (2015). https://doi.org/10.1109/DCABES.2015.45
25. Wu, J., Wang, J., Qi, Z., Guan, H.: SRIDesk: A Streaming based Remote Interactivity architecture for desktop virtualization system. In: 2013 IEEE Symposium on Computers and Communications (ISCC). pp. 281–286 (2013). https://doi.org/10.1109/ISCC.2013.6754960