IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 6, 2020, accepted September 8, 2020, date of publication September 14, 2020,
date of current version September 25, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3023924

On Construction and Performance Evaluation of a Virtual Desktop Infrastructure With GPU Accelerated

CHIH-HUNG CHANG[1], (Member, IEEE), CHAO-TUNG YANG[2], (Member, IEEE), JHENG-YUE LEE[2], CHUAN-LIN LAI[3], AND CHIA-CHEN KUO[3]

[1]College of Computing and Informatics, Providence University, Taichung 433719, Taiwan
[2]Department of Computer Science, Tunghai University, Taichung 40704, Taiwan
[3]National Center for High-Performance Computing, Hsinchu 30076, Taiwan

Corresponding author: Chao-Tung Yang (ctyang@thu.edu.tw)

This work was supported in part by the Ministry of Science and Technology (MOST), Taiwan under Grant 108-2221-E-029-010, Grant 108-2745-8-029-007, Grant 108-2622-E-029-007-CC3, Grant 109-2221-E-029-020, and Grant 108-2221-E-126-002; and in part by the National Center for High-performance Computing under Project 03108F1106.

ABSTRACT More engaging and accessible solutions that offer outstanding user experience are needed. The virtual desktop infrastructure (VDI) is the first user access part of the cloud. Therefore, the performance of the VDI has a significant influence on the network. The graphics processing unit (GPU) can accelerate performance by assigning each virtual machine a dedicated GPU for performance-boosting to improve the user experience. In this paper, we propose a high-performance VDI using an integration of a GPU on OpenStack with a PCI pass-through. It might have certain benefits over using virtualized hardware, such as lower latency, better performance, and more functionality. The evaluation and comparison of the computer processing unit (CPU) and GPU are presented among the different benchmark tools.

INDEX TERMS Cloud service, virtual desktop interface, OpenStack, graphics processing unit, software as a service.

I. INTRODUCTION

Virtualization supports a data center by increasing the utilization of computing resources, which may otherwise be wasted. For instance, a typical physical server with a single workload may only use 10% to 15% of the processor cycles or memory space on the server, wasting the remaining 85% to 90% [1]. How to effectively use and manage information equipment investment is a constant concern, and the issue is even more pertinent for businesses.

Server prices are much lower than before, and the performance is similar to that of a workstation. As a result, many companies have moved their systems and services to cloud computing environments to reduce the cost of investment for the management of information equipment.

In cloud computing, all resources are virtualized, such as the computer processing unit (CPU), storage, and network resources. Resources are allocated according to the needs of the users. Cloud computing is a network-based computing model that provides access to a variety of resources, including operations, storage, and so on, usually in the data center, and delivers massive resources through virtualization and dynamic scaling. There is no need to understand the infrastructure of the cloud or to master the technology directly, but it can also use the resources of the cloud.

Providing an easy-to-use environment with a better user experience for users is a challenge for information technology (IT) managers. The part closest to the average user is the virtual desktop. However, when the user uses applications that rely heavily on graphical computing, the user experience is far inferior to the traditional high-performance standalone computer with a dedicated graphics processing unit (GPU).

In a recent study by the Lakeside Software [2], GPUs have enabled the rise of graphics through efficient processing. More and more modern applications running on adequate graphics provide an excellent user experience. Today, more than 60% of business users use at least one of these applications. Moreover, GPUs are designed to be used on computers. However, in the virtual environment, there is now similar technology to accelerate graphics processing. The GRID

The associate editor coordinating the review of this manuscript and approving it for publication was Fan-Hsun Tseng.

GPU allows multiple virtual machines (VMs) that are running any application to share the GPU. The GRID GPU technology distributes the GPU on the server to multiple VMs, so that the GPU and user no longer have a one-to-one relationship but a one-to-many relationship. The GRID GPU features are as follows:

- GPU virtualization: The NVIDIA Kepler architecture provides VMs designed to offer GPU hardware virtualization capabilities, which means that multiple users can use the GPU together.
- Low latency remote display: The GRID has a low latency remote display technology that allows the user to reduce the resource requirement to connect to a virtual desktop infrastructure (VDI) protocol.

A VM can be configured through GPU virtualization or through a pass-through approach to configure GPU resources, and pass-through practice can be interpreted as a GPU dedicated to a VM. With this technique, we can make a specific VM a vGPU. The difference between the vGPU and pass-through is that the vGPU is the complete GPU resource sliced into smaller vGPU resources to provide for the needs of the VM, and pass-through is making the resource of a GPU unit totally occupied by a single VM. Therefore, a high-performance VDI is built using high-load machines coupled with VDI, which becomes an important organizational issue for IT managers. Using VDI, it is possible to reduce the investment in IT equipment procurement without affecting user experience, improving the utilization of equipment performance and reducing hardware purchase costs and the time and effort required to manage large volumes of IT equipment.

Moreover, VDI is a desktop virtualization technology in which the desktop operating system (OS) is run and managed on premises or in a cloud data center. The virtual desktop image is delivered through a network to an endpoint device, which allows the user to interact with the OS and its application as it runs locally. The endpoint may be a traditional personal computer (PC), a thin client device, or a mobile device. Virtual desktop OSs can be configured with virtualized resources to the user's needs, and different applications can be installed on the OS. However, virtual desktops are limited by the network or for other reasons, which can lead to sluggish user experience.

Using virtualization solutions to simplify IT infrastructure can help system administrators reduce the risk of capital expenditures, reduce operating costs through automation, reduce planned and unplanned losses, and significantly reduce revenue through less downtime. Capital and operating costs are reduced by improving energy efficiency while reducing hardware requirements through server consolidation.

In recent years, the OpenStack [3]–[5] has been one of the most widely used open-source technologies to construct cloud platforms. OpenStack-supported hypervisors include KVM, QEMU, VMware, VSphere, XenServer, Hyper-V, and so on. Different hypervisors have their own VDI. There are generally noVNC, Spice, remote desktop protocol (RDP), and so on. Users use VDI to connect to the hypervisor, but not all VDIs can support the GPU display.

In our previous work [6], we revealed partial results of network delay and benchmark comparison results for this issue. In this paper, we further reveal the complete system architecture and more experimental results. According to the PCI pass-through technology to leverage GPU resources to accelerate virtual desktop performance and improving the smoothness of virtual desktops. We use OpenStack integrated with XenServer as the VM monitoring program layer. By means of the pass-through technology, a VM can be created on the GPU with an RDP environment to achieve accelerated VM image processing. Then we evaluate performance through different benchmark software to assess the GPU and the effect of the vCPU virtual desktop performance.

The organization of this work is as follows. In the Section II, we review the background knowledge for later use, regarding the system design and implementation. In Section III, we present the proposed system implementation of our work. The fourth section presents the experiments methodologu of our work. In Section V are the results of the experiments and a short discussion of our work. We introduced the related works in Section VI. In the last section, we summarize and organize our contributions and future work.

II. BACKGROUND REVIEWS

In this section, we review the background knowledge for later use regarding the system design and implementation.

A. VIRTUALIZATION

In cloud computing technology, virtualization [7]–[10] is the abstraction and transformation of computer entity resources, such as servers, networks, memory, and storage. The virtual part of the system is not affected by the existing resources or setup of the geographic or physical limitations. This refers to virtualized resources, including computing power and data storage. In computer science, virtualization is a technology for a computer or OS. Virtualization hides the real hardware devices for the user and presents another virtual computing platform. Virtualization technology transforms the entity into a virtual computing environment (i.e., VM) available to the user.

The user uses a client application to operate the VM. The VM does not restrict any applications or OSs and is like a direct run on the same machine. Moreover, VMs are the unified management of hardware resources (e.g., network, screen, and hard drive) and are more restrictive than processors or memory. The client is restricted from accessing the entity's peripherals, depending on the access policy adopted by the entity.

B. OpenStack

Among the technologies related to cloud services, virtualization technology plays a decisive role, distributing many kinds of virtualization projects from the beginning with VMware

vSphere and Hyper-V deployments to OpenStack [11]–[13] platforms. OpenStack is NASA's open-source software developed jointly by NASA and Rackspace. The Apache license is free software and is an open-source project for building infrastructure as a service (IaaS). OpenStack has three modules, a network access module, a storage module, and a centrally managed dashboard module to form a set of OpenStack shared services with VMs, external operational resources, and easy and flexible scaling or scheduling. Users can use the open-source OpenStack to build their own Amazon EC2-like services. The OpenStack specification is also compatible with Amazon EC2, so regardless of developing a system on it, using the system still helps people develop systems and use them. OpenStack can do it all, which is why IaaS is popular. In this work, we used OpenStack to construct the infrastructure of the environment.

C. VIRTUAL DESKTOP INFRASTRUCTURE

The VDI [14], [15] is a software technology that separates the desktop environment and associated application software from the physical client device used to access it. Desktop virtualization can be used in conjunction with application virtualization and user profile management systems, now known as "user virtualization." It provides a comprehensive management system for the desktop environment.

In this model, the components required for the desktop are virtualized. This allows for greater flexibility and a more secure environment for virtual desktops. This approach supports multiple disaster recovery strategies because all components are stored in the data center and maintained by the system backup. It is also easy to recover if a user's components or files are lost because all desktop components can be registered on the machines in other entities. The data are not stored in the user's device. The loss can also be reduced if the user's device is lost, and the data are stored in the data center. Software technology separates client devices. The types of desktop virtualization technologies used in typical deployments are described in detail below.

D. DESKTOP VIRTUALIZATION TECHNOLOGY

The work in [16]–[19] can be divided into the following stages: the mainframe era. The mainframe was very expensive when it first appeared, but the computing power of the mainframe is good. Thus, the idea of sharing a machine for multiple users was proposed. This is not virtualization, but rather a form of multi-tasking that relies on multiple users of the system.

Desktop virtualization is commonly used in the following scenarios.

- In distributed environments with high availability requirements where desktop technical support is not readily available,
- In environments where high network latency degrades the performance of traditional client/server applications,
- In environments where remote access and data security requirements create conflicts that can be resolved by retaining all (application) data in the data center.

Remote Desktop Protocol (RDP) is a desktop virtualization protocol that was originally developed by Microsoft from the Citrix bought the technology and developed it themselves. The RDP server is built into the Windows operating system; there are also RDP servers for Unix and OS X.

III. SYSTEM DESIGN AND IMPLEMENTATION

Due to the popularity of cloud computing, people are increasingly exposed to the environment. Most people use partial virtual desktops, and virtual desktops generally have lower performance, making them more difficult to operate. This work is about integrating the GPU into OpenStack and accelerating the performance of the VDI by allowing the VM to use GPU resources.

A. SYSTEM ARCHITECTURE

This section describes the overall architectural design and use of open-source software. As illustrated in Fig. 1, we used OpenStack as the basis for the entire virtualization. We integrated the GPU into OpenStack, allowing the VM to use the GPU resources by delivery, and tested it in multiple VDIs to discover a VDI protocol that works for OpenStack.

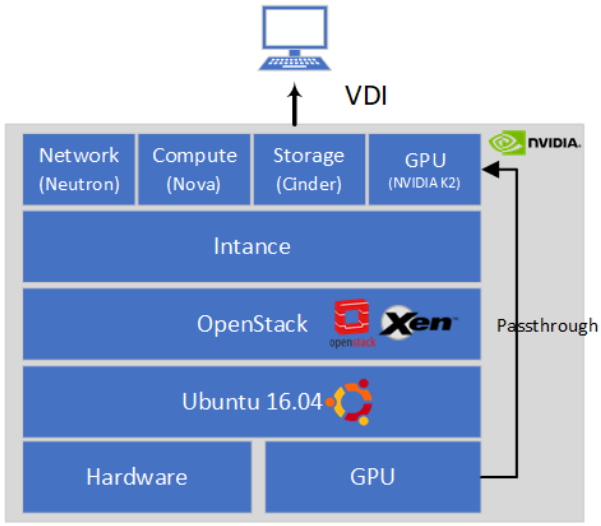

FIGURE 1. System architecture.

The framework proposed in this paper is examined in several parts. First, we integrate OpenStack with XenServer. In addition, Fig. 2 is the concept model of OpenStack nova with KVM and XenServer.

The OpenStack conceptual architecture is depicted in Fig. 3. The VM generation is one of the most important use cases in a cloud environment. We described the steps to configure the instance in the OpenStack cloud, which includes the order of the requests and interaction between various OpenStack components to successfully start the VM.

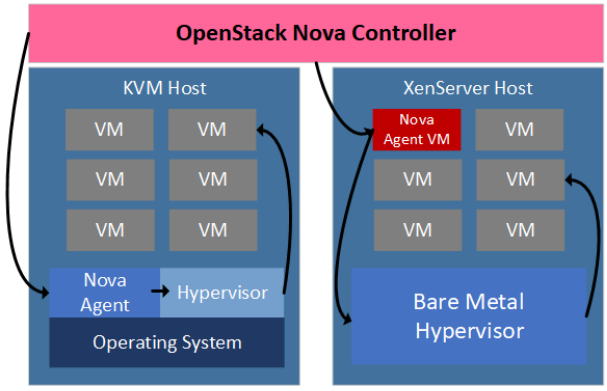

FIGURE 2. OpenStack nova with KVM and XenServer.

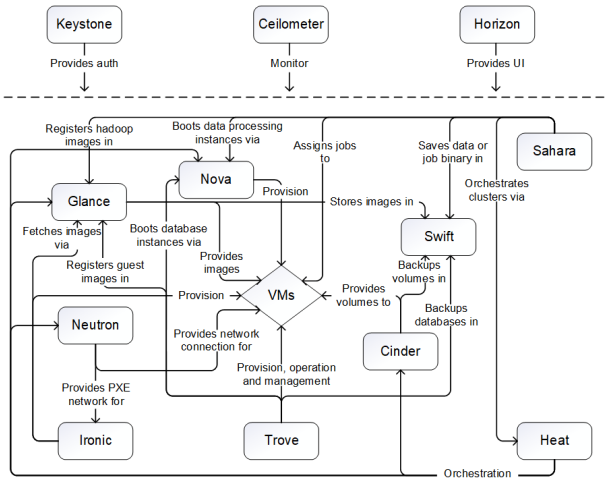

FIGURE 3. OpenStack conceptual architecture.

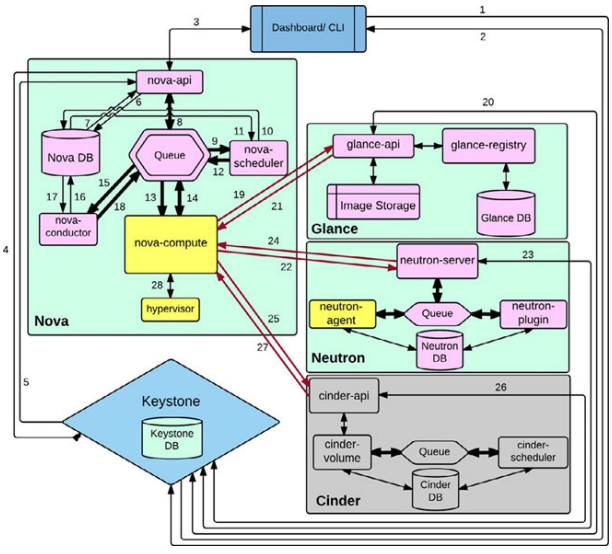

FIGURE 4. Virtual machine spawning sequence in OpenStack.

In Fig. 4, the communication process is presented for OpenStack. These connections are initiated using the associated APIs as remote procedure calls (RPCs), when the tenant communicates via the command-line interface or dashboard when issuing an instance request. It can be converted into a nova-boot command. The nova-api server sends the user's credentials to Keystone for authentication (1 and 2). After successful authentication, the nova-api contacts nova-db to initialize the new instance's initial configuration information in the database (4, 5, 6, and 7). Then, the nova-api sends an RPC to the nova-scheduler requesting the host of the boot instance of the ID (8 and 9). The nova-scheduler fetches information from nova-db and uses filters and the weight function to select the best (or least loaded) host and returns its ID (10, 11, and 12).The scheduler selects the appropriate compute node as the host and sends a message to start the new instance (12 and 13). Nova-compute runs, then the RPC calls nova-conductor, and nova-conductor accesses nova-compute to nova-db to obtain the host ID, flavor disk, vCPU, and other information (14, 15, 16, 17, and 18). Using authentication tokens, nova-compute makes representational state transfer (REST) calls to glance-api from the image. The image is retrieved from the library and uploaded to the selected host (19, 20, and 21). This uploaded image is cached for future use. Subsequently, nova-compute calls neutron-api to retrieve network allocation and configuration information for the assigned fixed internet protocols (IPs) to the new instances (22, 23, and 24). If the user requests that some volumes be attached to the instance, nova-compute uses REST calls and additional volumes (25, 26, and 27) for cinder-api. Finally, nova-compute puts all the information forwarded to the virtualization driver, and an instance request is generated on the hypervisor (28). At each stage of the provisioning process, the corresponding instance status can be observed from the Horizon dashboard as Dispatch > Networking > Spawning > Running.

We built an OpenStack cluster with eight physical machines, one of which is a controller and four of which compute nodes equipped with K2 GPUs with different hypervisors: KVM, XenServer, ESXi, and so on. Moreover, Fig. 5 illustrates the experimental environment architecture of the system. This node uses XenServer as the endpoint of the experiment. The other two nodes are the network and storage nodes. The network node is responsible for network packet transfer and VLAN management, whereas, in the storage part, Cinder and Swift suite are used. Cinder is the block storage service. In this paper, we split the space of Cinder as a VM hard disk. Swift is the object service used as the storage file of XenServer. Table 1 is the software specification and version of the system implementation.

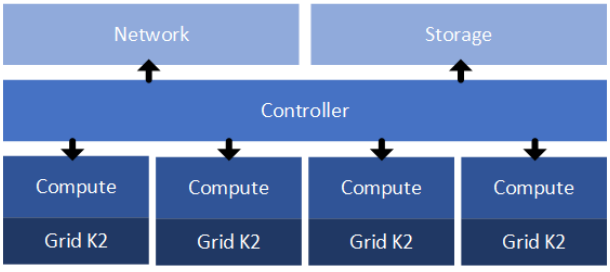

FIGURE 5. Experimental environment.

TABLE 1. Software specification.

Software	Version
Ubuntu	16.04.02 LTS
Windows	10
OpenStack	Newton
Python	2.7.6
MariaDB	10.1.1.14
XenServer	7.0
NVIDIA Driver	369.95

When creating the VM, we used the Ubuntu OS. OpenStack was applied to build and manage this cloud environment. Moreover, Fig. 6 and Fig. 7 illustrate the graphical user interface of the OpenStack management dashboard and VM instance information for the environment.

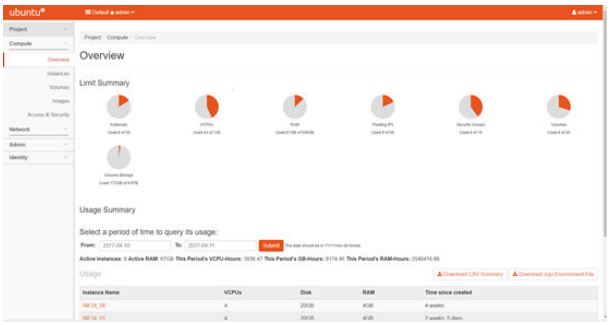

FIGURE 6. OpenStack overview.

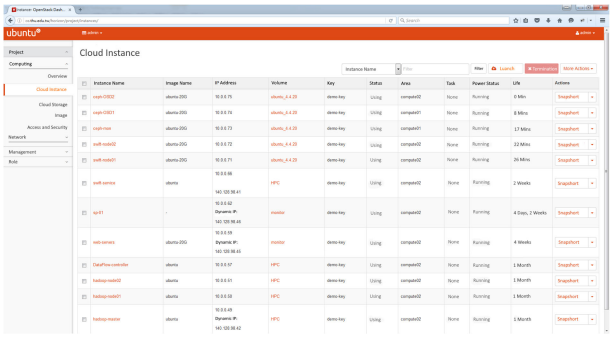

FIGURE 7. OpenStack virtual machine instances.

1) VDI DEPLOYMENT noVNC

OpenStack uses noVNC to implement a user interface for virtual desktops using the web client for Socket and HTML5 Canvas. However, the use of noVNC can cause delays and lag in the use of virtual desktops, which is very inconvenient for users. Because noVNC was not smooth enough, we integrated Spice (a standalone computing environment) into the OpenStack cloud platform, which allows users to view the "desktop" environment. The Spice environment in the system are depicted in Fig. 8 and Fig. 9.

2) GPU AND GRID GPU

We expect to use GPUs to accelerate the VDI smoothness and the ability of VMs to handle graphics. The K2 is a GPU that performs graphical operations on a GPU, server, or another device. In this section, we use the NVIDIA GRID K2 as our GPU device, and K2 is a device that uses the NVIDIA Kepler architecture designed to deliver a rich design experience in a virtual environment.

FIGURE 8. OpenStack Spice web console.

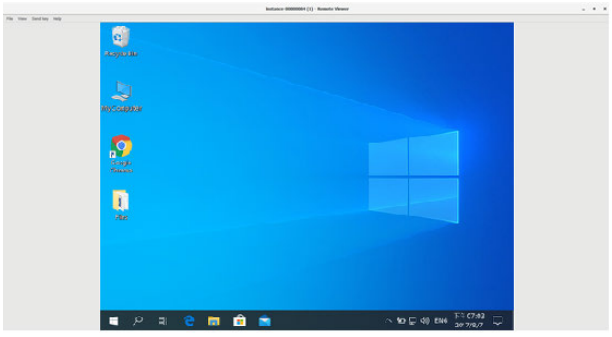

FIGURE 9. OpenStack Spice client.

A VM can be configured through GPU virtualization or through a pass-through approach to configure the GPU resources. The pass-through practice can be interpreted as a GPU dedicated to a VM. With this technique, a specific VM vGPU can be created. The various parts of the vGPU comprise the configuration of all GPU resources. In addition, the vGPU is the complete GPU resource sliced into smaller vGPU resources to provide for the needs of the VM and pass-through, which comprise the complete resources allocated to the same VM.

IV. EXPERIMENTS METHODOLOGY

In this work, we expected to use GPUs to accelerate the VDI smoothness and the ability of VMs to handle graphics. We used OpenStack and the integrated XenServer, which was delivered via a pass-through method and was installed on the compute node of the GPU (NVIDIA K2) assigned to the VM (XenServer). OpenStack and XenServer were used with the experimental data to determine whether the GPU affects the display performance of the VM. They were also employed

in testing in different situations, including correlating memory size, the number of vCPU cores, and so on with the GPU, to determine the most effective method to enhance the VM desktop performance settings. Therefore, we designed some experiments.

We simultaneously executed the ping command through the host and VM and recorded the return value during this period to determine whether network problems exist, such as latency on the VM. The influence of the GPU on the display performance of the machine was tested via the Heaven Benchmark in three scenarios, as follows:

1) Installing the GPU on the physical machine.
2) Opening and obtaining GPU resources for the VM via OpenStack and the PCI pass-through (for) hypervisor for XenServer.
3) Opening directly through XenServer and obtaining the VM through the PCI direct GPU resources.

To testing all three cases above, in the VM section, we set the memory to 8 G and the virtual CPU section is divided into three sections: 1) two slots per slot, two cores per slot (four vCPU), 2) two slots per slot, four slots per slot nuclei (eight vCPU), and 3) two slots, four cores per slot (16 vCPU). Four results were obtained from the Heaven Benchmark: score, FPS, MinFPS, and MaxFPS.

For the third experiment, we used the CineBench Benchmark tool [20]. The CineBench test obtains two return values: one for the GPU-related OpenGL FPS and the other related to CPU Score. Thus, we performed the CineBench test. As a second experiment, the test was conducted using three different scenarios, and some of the VMs were tested. The first test was performed under different CPU numbers, and then two values were returned to determine whether the GPU can accelerate the effect in a VM.

V. EXPERIMENTAL RESULTS

In this section, we present the experimental environment and the results of the experiments. In the first subsection, we describe the experimental environment, and in the following subsection, we present the experimental results.

A. EXPERIMENTAL ENVIRONMENT

We used OpenStack to build the cloud platform and then used it to create and manage the storage distributions. As a simple example, we integrated two heterogeneous storage technologies. We built our storage system through several VMs, where HDFS was built from three VMs with a four-core spec. CPU, 4 GB of random access memory (RAM), and a total of 200 GB of storage. Table 2, 3, and 4 present the experimental environment specifications.

In our experiments, we will use the NVIDIA K2 GPU as a device for accelerating VDI.

B. NETWORK DELAY EXPERIMENT

The first experiment used the ping instruction in the host and VM at the same time. The command is to ping 8.8.8.8, to test whether the VM and external machine between the network are delayed. We tested for 60s and 120s, and the experimental results are presented in Fig. 10 and Fig. 11.

TABLE 2. Experimental environment hardware specification.

Host name	CPU	Memory	Disk	GPU	OS
Openstack Controller	12 cores	64 GB	2 TB		Ubuntu 16.04.02
Openstack Network	24 cores	64 GB	2 TB		Ubuntu 16.04.02
Openstack Compute(KVM)	20 cores	24 GB	2 TB	GRID K2	Ubuntu 16.04.02
XenServer	20 cores	24 GB	2 TB	GRID K2	XenServer 7.0
Openstack Block Storage	64 cores	48 GB	8TB		Ubuntu 16.04.02

TABLE 3. Experimental environment virtual machine specifications.

Host name	CPU	Memory	Disk	OS
Instance 01	4 cores vCPU	8 GB	100 GB	Windows 10
Instance 02	8 cores vCPU	8 GB	100 GB	Windows 10
Instance 03	16 cores vCPU	8 GB	100 GB	Windows 10

TABLE 4. Experimental environment software specifications.

Software	Version
OpenStack	Newton
Python	2.7.6
MariaDB	10.1.1.14

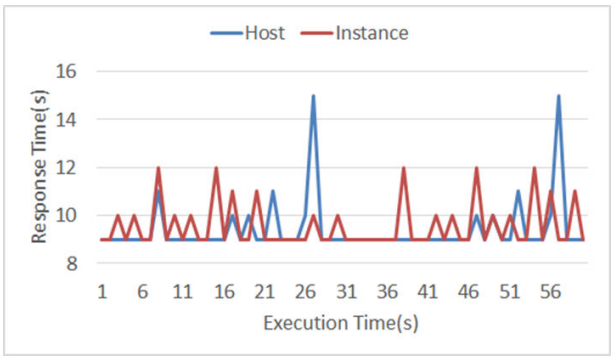

FIGURE 10. Ping 60s.

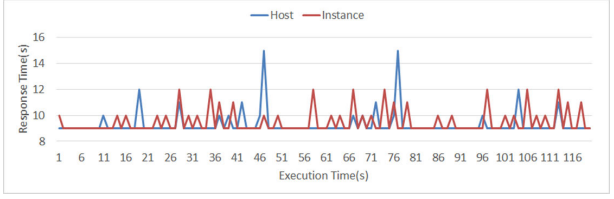

FIGURE 11. Ping 120s.

From this experiment, no significant difference was found between the VMs and physical machines outside of the network, indicating that the VM is unaffected by OpenStack and that the Windows two-tier OS causes latency on the network. Therefore, subsequent parts of the experiment can be ignored.

C. HEAVEN BENCHMARK EXPERIMENT

This experiment uses the free benchmark tool from Heaven Benchmark to measure the physical machine and the performance of the VM, using pass-through technology to allocate GPUs to the VM by modifying the virtual. The CPU core part is used to measure whether it affects the display performance of the vCPU for the VM.

First, testing OpenStack open VMs through the Heaven Benchmark yields four values: FPS, Min FPS, Max FPS, and score. We tested three cases: 4vCPUs, 8vCPUs, and 16vCPUs, all with 8 G RAM, and the results are illustrated in Figs. 12, 13, 14, and 15.

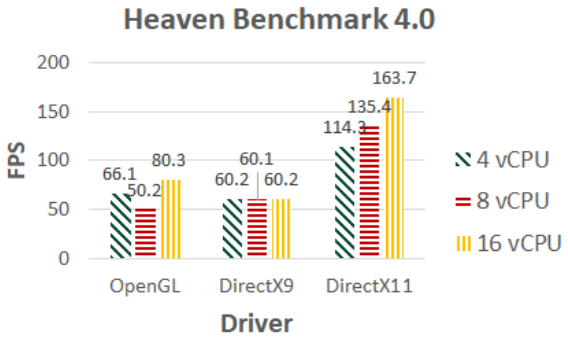

FIGURE 12. OpenStack virtual machine Heaven Benchmark FPS.

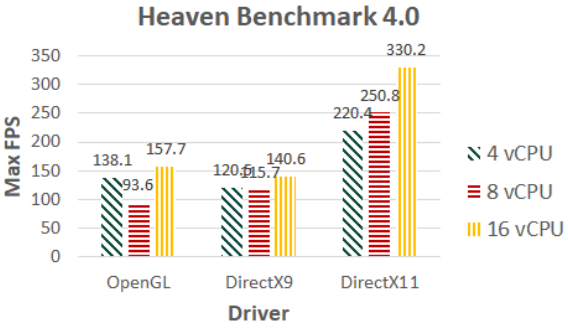

FIGURE 13. OpenStack virtual machine Heaven Benchmark MaxFPS.

The four figures display the FPS, Max FPS, Min FPS, and score. The data are plotted to reveal that the FPS averages of DirectX9 are all about the same, and with fixed memory, the number of virtual CPUs does not affect the performance of DirectX9 when the average FPS is around 60. DirectX11 affects the performance significantly because of the difference in CPU cores, which are superior to the other two drivers. However, the OpenGL part of the performance is not particularly outstanding. Moreover, we found in the tests that different screen resolutions also affect the performance; thus, we fixed the 1280*720 resolution for testing. When we used 1920*1080, the FPS and score were reduced to 50% of the figures, which are usually between 20 to 30.

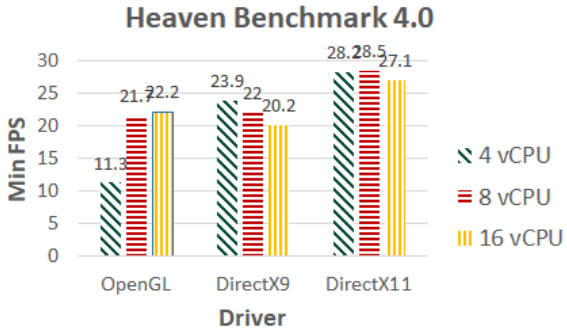

FIGURE 14. OpenStack virtual machine Heaven Benchmark MinFPS.

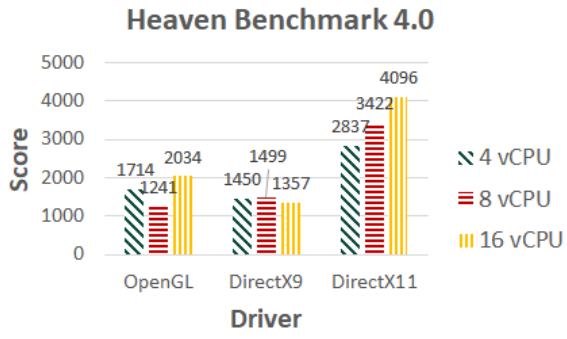

FIGURE 15. OpenStack virtual machine Heaven Benchmark Score.

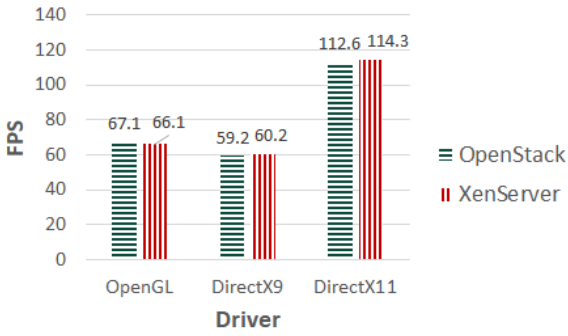

FIGURE 16. OpenStack and XenServer instance performance comparison (4 vCPUs).

In the second part of the VM test, we compared OpenStack integrated with XenServer with only the XenServer environment to test the performance by modifying the vCPU for the experiments. As in the last part of the experiment, the three VMs are 4vCPUs, 8vCPUs, and 16vCPUs. Three VMs, and all configured with 8 GB of memory, were configured to the VMs via PCI pass-through technology K2 and the previous part of the experiment. The following results were obtained using the Heaven Benchmark test. The 4vCPU results are illustrated in Figs. 16, 18, 19, and 17. The 8vCPU results are displayed in Figs. 20, 22, 23, and 21. The The 16vCPU results are depicted in Figs. 26, 24, 25, and 27.

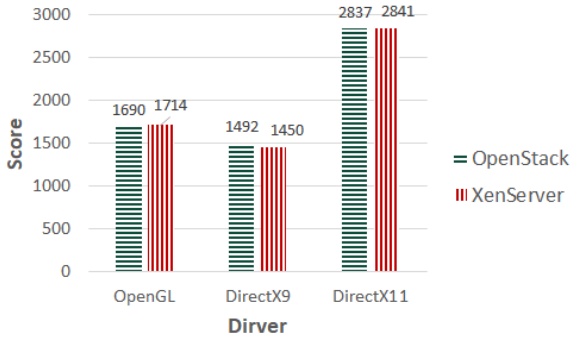

FIGURE 17. OpenStack and XenServer instance performance comparison (4 vCPUs).

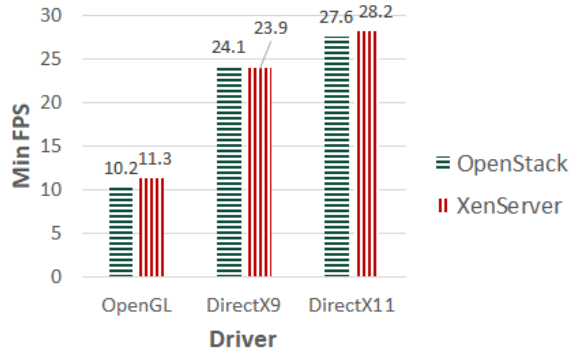

FIGURE 18. OpenStack and XenServer instance performance comparison (4 vCPUs).

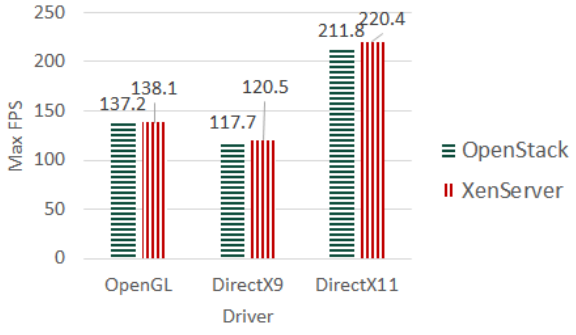

FIGURE 19. OpenStack and XenServer instance performance comparison (4 vCPUs).

The results of this second experiment test reveal that the Heaven Benchmark test part shows that DirectX11 still has the best performance. OpenGL and DirectX11 are affected by the number of CPUs in the test results for better or worse, whereas DirectX9 is not. With a fixed RAM size, the performance of DirectX11 is far higher than the other two by about two to three times, and the performance results from OpenStack and XenServer open VM testing were not what a difference.

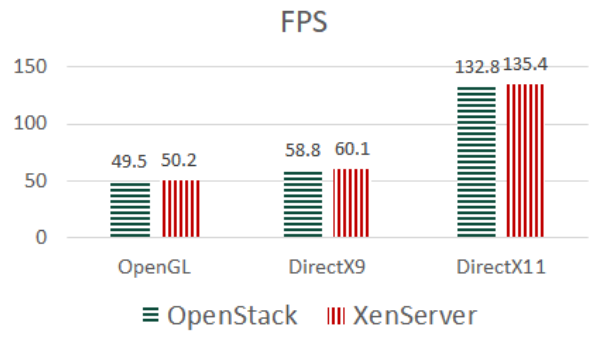

FIGURE 20. OpenStack and XenServer instance performance comparison (8 vCPUs).

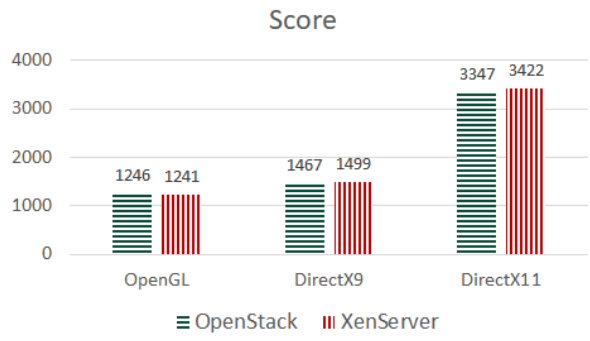

FIGURE 21. OpenStack and XenServer instance performance comparison (8 vCPUs).

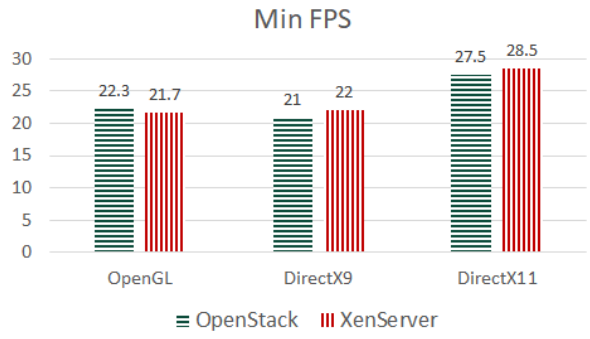

FIGURE 22. OpenStack and XenServer instance performance comparison (8 vCPUs).

In the comparison above, Figs. 28, 29, 30, and 31 present the VM performance test section, using the OpenStack or OpenStack directly between two data through VMs with very small gaps between them. Almost no gap exists; thus, OpenStack does not affect the performance display of the virtual desktop. The reason for this could be that our experimental environment integrates OpenStack and XenServer, where OpenStack only opens VMs and communicates with XenServer via the nova-api. Then, we do not need to

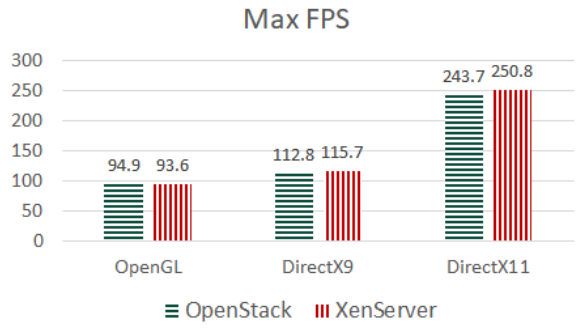

FIGURE 23. OpenStack and XenServer instance performance comparison (8 vCPUs).

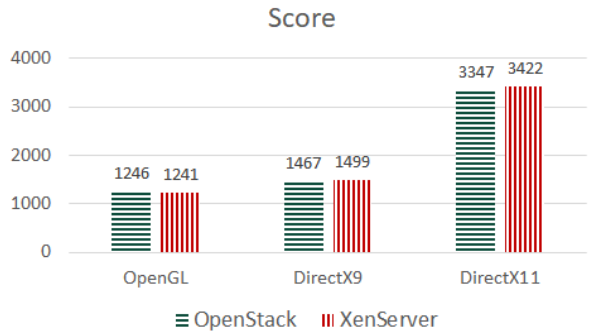

FIGURE 24. OpenStack and XenServer instance performance comparison (16 vCPUs).

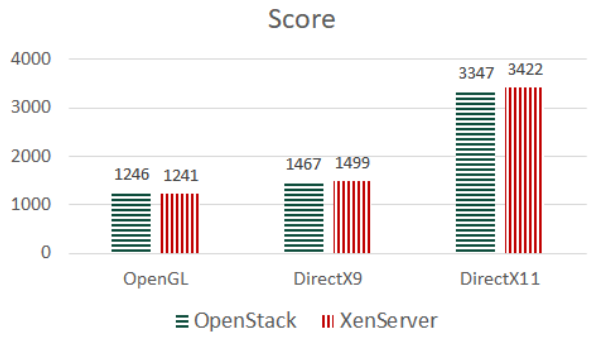

FIGURE 25. OpenStack and XenServer instance performance comparison (16 vCPUs).

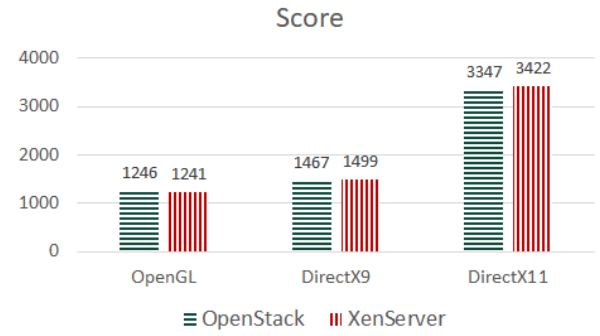

FIGURE 26. OpenStack and XenServer instance performance comparison (16 vCPUs).

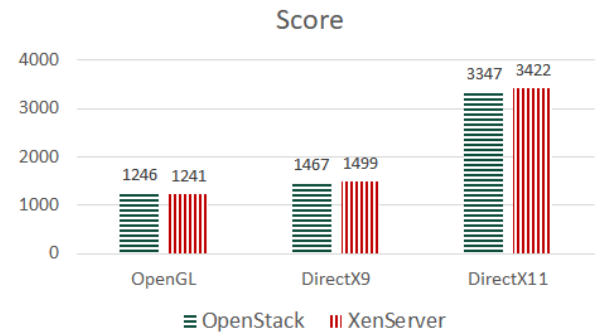

FIGURE 27. OpenStack and XenServer instance performance comparison (16 vCPUs).

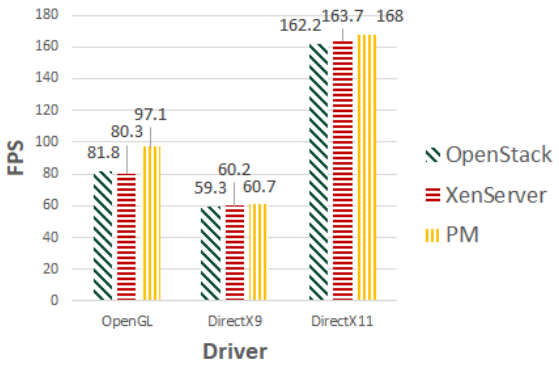

FIGURE 28. Comparison of the physical FPS.

manipulate the VM through OpenStack. Therefore, when testing the performance, we can simply ignore the influence of OpenStack.

D. CineBench EXPERIMENT

In this experiment, we experimented with the VM using the CineBench Benchmark tool, which graphically tests the CPU and GPU. We divided the results into three VMs, as in the previous experiments: four vCPUs, eight vCPUs, and 16 vCPUs. All three VMs have a fixed memory of 8 GB.

The results of the test are presented in Figs. 32 and 33. The test data reveal a significant increase in the number of different CPUs based on the CPU part values. Eight CPUs take about twice as long as four CPUs for the score, with 16 CPUs taking about 1.5 times as long to score as 8 CPUs. The relationship is presented in the figures. The difference is too much, almost always around 120 or so and not much more. How the CPU count affects the OpenGL is depicted. The effect of the VM is not that significant, and this result can be verified with the results of the Heaven Benchmark.

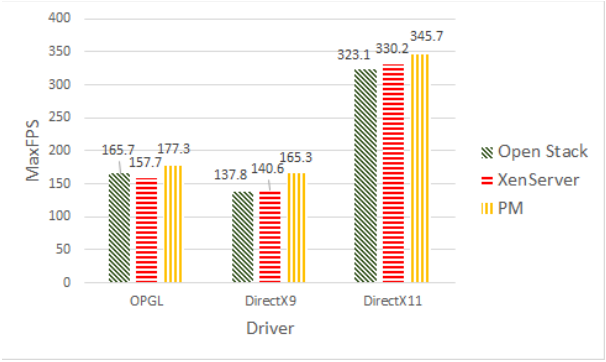

FIGURE 29. Comparison of the physical Max FPS.

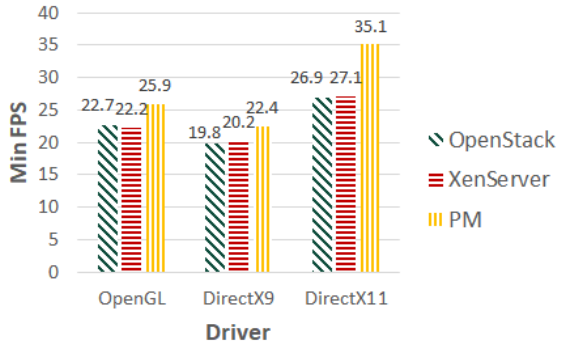

FIGURE 30. Comparison of the physical Min FPS.

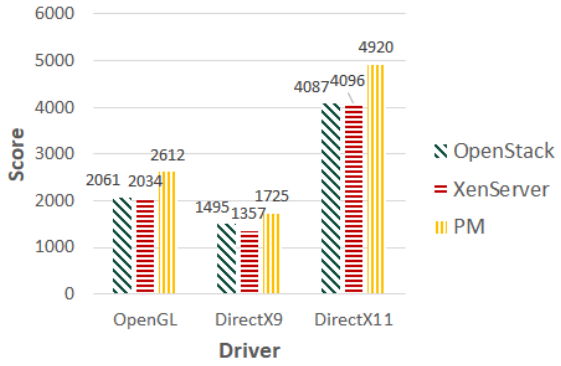

FIGURE 31. Comparison of the physical score.

E. DISCUSSION

The above three experiments can be analyzed to conclude that no significant difference exists between the VM and host network; therefore, the network part is almost negligible. Second, the Heaven Benchmark revealed that the number of different CPU cores indirectly affects the GPU and screen display. Different drivers have different effects on performance, especially in Direct X11, where the influence is the greatest, whereas DirectX9 had no effect and performed the worst. The third part of the experiment was performed using CineBench to discover the combined effects of the CPU.

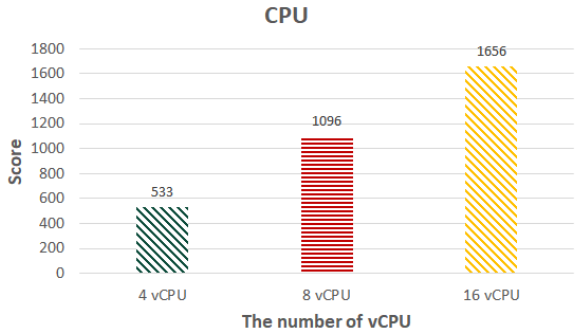

FIGURE 32. CineBench comparison of CPU.

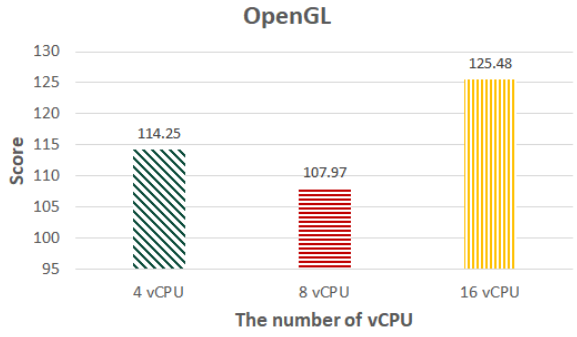

FIGURE 33. CineBench comparison of OpenGL.

VI. RELATED WORKS

Song *et al.* [21] proposed new high-performance encoding and decoding for hierarchical RAID accelerated by the GPU in a multiple VM environment. For each VM, pass-through GPU technology was used to provide dedicated access to GPU cores. For a virtual desktop, it often allows better encoding and decoding efficiency than conventional vGPU technology. The proposed hierarchical RAID eliminates the overhead for the GPU and avoids the failure of nodes. Their experimental findings reveal that the average encoding efficiency of the proposed hierarchical RAID 55 increases by 11.03% compared with hierarchical RAID 51 compared to different file sizes. In comparison, the average disk-based decoding efficiency of the proposed hierarchical RAID 55 also increases by 59.61%.

Xue *et al.* [22] introduce gScale, an open source GPU virtualization solution based on gVirt, which is scalable and realistic. GScale offers a sharing system incorporating partitioning and sharing to overcome global graphics memory space hardware constraints. In particular, we propose two approaches for gScale: (1) the private shadow graphics translation table (GTT) which allows global graphics memory space sharing between virtual GPUs, (2) ladder mapping and fence memory space pool allowing CPU access to host physical memory space (serving graphics memory) bypassing global graphics memory space. Evaluation reveals that

gScale scales up to 15 instances of virtual GPU guests in Linux or 12 instances of virtual GPU guests in Windows, which is 5x and 4x respectively that of gVirt. At the same time, when hosting multiple virtual GPU instances, gScale brings in a slight but acceptable runtime overhead.

Tan *et al.* [23] proposed a multi-channel GPU virtualization architecture (VMCG), modeled the related credit allocation and transfer processes, and redesigned the GPU fair-scheduling virtual multi-channel algorithm. For each guest VM (DomU) competing with other VMs for the same physical GPU resources, VMCG provides a separate V-channel, and each DomU submits command request blocks according to the corresponding DomU ID to their respective V-channels. Not only can multiple DomUs make full use of native GPU hardware acceleration through the virtual multi-channel GPU fair-scheduling algorithm, but the fairness of GPU resource allocation is also greatly improved during GPU-intensive workloads from multiple DomUs running on the same host. The experimental results reveal that performance is similar, at 96% of that of the native GPU for 2D/3D graphics applications. The performance is improved by around 500% for parallel computing applications, and the resource-allocation equity of the GPU is improved by about 60% to 80%.

Comparison with the related work above, we not only implement by pass-through to allow VMs to use the host's GPU resources. But also make completed experimental testing include network latency, performance comparison for a variety of VDIs configurations such as OpenStack integrated with XenServer, according to Heaven Benchmark and Cinebench Benchmark tools.

VII. CONCLUSION AND FUTURE WORK

To accelerate the smoothness of virtual desktops and their ability to handle graphics. In our previous work of Yang *et al.* [24] use graphics processing units to build a high-performance computing cloud cluster. We use GPU passthrough technology, InfiniBand virtual, and 10 Gb Ethernet network to improve the performance of the virtual cluster. In [25], they propose two algorithms of a dynamic resource allocation strategy and the energy-saving method, to increase energy utilize consumption efficiently. In this work, we used pass-through to allow VMs to use the host's GPU resources. According to testing, including the network latency tests, the OpenStack VM difference in terms of network latency with the host in the neutron architecture is so small that parts of the network are almost negligible.

We tested a variety of VDIs before experimenting, and there are few VDIs that OpenStack can support on a GPU. Neither noVNC nor Spice worked properly; thus, we used RDP as our experimental VDI. By conducting benchmark experiments after selecting the VDI, we tested and compared the CPU and GPU parts. Among the different benchmark tools, DirectX11 is the one for NVidia GRID K2. The strongest compatibility and DirectX9 is the least efficient, and the number of CPUs for DirectX9 is not affected. DirectX11 outperforms both DirectX9 and OpenGL by about two to three times for Min FPS, Max FPS, and FPS AVS.

During our experiments, we also found that K2 would not be used if the entity was displayed directly through another graphics card and that it would not be used if it was displayed through the RDP connected to the physical machine. In addition, K2 starts running with 10% to 25% up and down utilization during that time to show the action. The GPU via RDP accelerates the display. However, because the GPU is VDI accelerated, the performance via the RDP test is worse than the direct physical machine test above, almost as bad as 20% or so. The GPU also accelerates faster than a direct running physical machine via the RDP connection test. While the performance is not as good as a physical machine, the GPU does accelerate VDI performance, and the GPU can be used for virtual desktops using authentication that can significantly improve user experience.

In this paper, we conducted experiments on the GPU pass-through on XenServer. In future work, we will replace different hypervisors, such as VMware ESXi or KVM, and compare with them to determine what kind of hypervisor works for the GPU to obtain the best performance according to pass-through support. OpenStack does not support the vGPU mode, only through pass-through, but this approach may result in a waste of GPU resources because, as powerful as the GPU is, a mere virtual desktop may not be able to perform all of the features. Using virtualization technology to allocate GPUs to different VMs so that each VM has access to different VM resources is probably a better approach.

In other future work, we plan to implement a vGPU in various ways, including the existing VMware full virtualization, the XenServer vGPU, KVM, and KVM-GT, and so on. These are all options we plan to try in the future to incorporate the vGPU functionality above into OpenStack and to determine where virtual desktop performance can be maximized.

ACKNOWLEDGMENT

The authors are grateful to the National Center for High-performance Computing for computer time and facilities.

REFERENCES

[1] S. J. Bigelow. (2020). *How Can Virtualization Reduce Power Consumption?* Accessed: Jan. 15, 2020. [Online]. Available: https://searchservervirtualization.techtarget.com/answer/How-can-virtualization-reduce-power-consumption

[2] L. Software. (2017). *Elevating User Experience Through GPU Acceleration*. Accessed: Jan. 15, 2020. [Online]. Available: https://www.lakesidesoftware.com/sites/default/files/Elevating_User_Experience_through_GPU_Acceleration.pdf

[3] M. Zakarya and L. Gillam, "Energy efficient computing, clusters, grids and clouds: A taxonomy and survey," *Sustain. Comput., Informat. Syst.*, vol. 14, pp. 13–33, Jun. 2017.

[4] P. Mishra, E. S. Pilli, V. Varadharajan, and U. Tupakula, "Intrusion detection techniques in cloud environment: A survey," *J. Netw. Comput. Appl.*, vol. 77, pp. 18–47, Jan. 2017.

[5] H.-J. Hong, P.-H. Tsai, and C.-H. Hsu, "Dynamic module deployment in a fog computing platform," in *Proc. 18th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Oct. 2016, pp. 1–6.

[6] C.-T. Yang, J.-C. Liu, J.-Y. Lee, C.-H. Chang, C.-L. Lai, and C.-C. Kuo, "The implementation of a virtual desktop infrastructure with GPU accelerated on OpenStack," in *Proc. 15th Int. Symp. Pervas. Syst., Algorithms Netw. (I-SPAN)*, Oct. 2018, pp. 366–370.

[7] C. de Alfonso, A. Calatrava, and G. Moltó, "Container-based virtual elastic clusters," *J. Syst. Softw.*, vol. 127, pp. 1–11, May 2017.

[8] A. Pietrabissa, F. Priscoli, A. D. Giorgio, A. Giuseppi, M. Panfili, and V. Suraci, "An approximate dynamic programming approach to resource management in multi-cloud scenarios," *Int. J. Control*, vol. 90, no. 3, pp. 508–519, 2017.

[9] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "Topology-aware prediction of virtual network function resource requirements," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 1, pp. 106–120, Mar. 2017.

[10] C.-T. Yang, J.-C. Liu, S.-T. Chen, and K.-L. Huang, "Virtual machine management system based on the power saving algorithm in cloud," *J. Netw. Comput. Appl.*, vol. 80, pp. 165–180, Feb. 2017.

[11] M. Marks and E. Niewiadomska-Szynkiewicz, "Hybrid CPU/GPU platform for high performance computing," in *Proc. ECMS*, May 2014, pp. 508–514.

[12] S. Boob, H. Gonzalez-Velez, and A. M. Popescu, "Automated instantiation of heterogeneous fast flow CPU/GPU parallel pattern applications in clouds," in *Proc. 22nd Euromicro Int. Conf. Parallel, Distrib., Netw.-Based Process.*, Feb. 2014, pp. 162–169.

[13] T. Rosado and J. Bernardino, "An overview of openstack architecture," in *Proc. 18th Int. Database Eng. Appl. Symp. (IDEAS)*, 2014, pp. 366–367.

[14] A. Natanzon and S. Cohen, "Replicating in virtual desktop infrastructure," U.S. Patent 9 619 543, Apr. 11, 2017.

[15] K. Makoviy, D. Proskurin, Y. Khitskova, and Y. Metelkin, "Server hardware resources optimization for virtual desktop infrastructure implementation," in *Proc. CEUR Workshop*, 1904, 2017, p. 178.

[16] B. Kim and B. Lee, "Integrated management system for distributed microdatacenters," in *Proc. 18th Int. Conf. Adv. Commun. Technol. (ICACT)*, Jan. 2016, pp. 466–469.

[17] A. Paradowski, L. Liu, and B. Yuan, "Benchmarking the performance of OpenStack and CloudStack," in *Proc. IEEE 17th Int. Symp. Object/Component/Service-Oriented Real-Time Distrib. Comput.*, Jun. 2014, pp. 405–412.

[18] J. Zhang, S. Han, J. Wan, B. Zhu, L. Zhou, Y. Ren, and W. Zhang, "IM-dedup: An image management system based on deduplication applied in DWSNs," *Int. J. Distrib. Sensor Netw.*, vol. 9, no. 7, Jul. 2013, Art. no. 625070.

[19] M. Izumi and K. Horikawa, "Toward practical use of virtual smartphone," in *Proc. 9th Asia–Pacific Symp. Inf. Telecommun. Technol. (APSITT)*, Nov. 2012, pp. 1–5.

[20] *Cinebench*. Accessed: Aug. 15, 2020. [Online]. Available: https://www.maxon.net/en/products/cinebench-r20-overview/

[21] T.-G. Song, M. Pirahandeh, C.-J. Ahn, and D.-H. Kim, "GPU-accelerated high-performance encoding and decoding of hierarchical RAID in virtual machines," *J. Supercomput.*, vol. 74, no. 11, pp. 5865–5888, Nov. 2018.

[22] M. Xue, J. Ma, W. Li, K. Tian, Y. Dong, J. Wu, Z. Qi, B. He, and H. Guan, "Scalable GPU virtualization with dynamic sharing of graphics memory space," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 8, pp. 1823–1836, Aug. 2018.

[23] H. Tan, Y. Tan, X. He, K. Li, and K. Li, "A virtual multi-channel GPU fair scheduling method for virtual machines," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 2, pp. 257–270, Feb. 2019.

[24] C.-T. Yang, S.-T. Chen, Y.-S. Lo, E. Kristiani, and Y.-W. Chan, "On construction of a virtual GPU cluster with InfiniBand and 10 Gb Ethernet virtualization," *J. Supercomput.*, vol. 74, no. 12, pp. 6876–6897, Dec. 2018.

[25] C.-T. Yang, S.-T. Chen, J.-C. Liu, Y.-W. Chan, C.-C. Chen, and V. K. Verma, "An energy-efficient cloud system with novel dynamic resource allocation methods," *J. Supercomput.*, vol. 75, no. 8, pp. 4408–4429, Aug. 2019.

CHIH-HUNG CHANG (Member, IEEE) received the Ph.D. degree in computer science from Feng Chia University, in 2004. He is currently an Associate Professor with the College of Computing and Informatics, Providence University, Taichung, Taiwan. His research interests include software engineering, cloud service, big data, and deep learning. He is a member of the IEEE Computer Society.

CHAO-TUNG YANG (Member, IEEE) received the Ph.D. degree in computer science from National Chiao Tung University, in July 1996. In August 2001, he joined the Faculty of the Department of Computer Science, Tunghai University, Taiwan, where he is currently a Distinguished Professor in computer science. He has published more than 300 papers in journals, book chapters, and conference proceedings. His current research interests include cloud computing, big data, parallel computing, and deep learning. He is a member of the IEEE Computer Society and ACM. He is serving in a number of journal editorial boards, including *Future Generation Computer Systems*, the *International Journal of Communication Systems*, *KSII Transactions on Internet and Information Systems*, and the *Journal of Cloud Computing*.

JHENG-YUE LEE received the master's degree in computer science from Tunghai University, in 2017. His current research interests include cloud computing and virtualization.

CHUAN-LIN LAI received the master's degree in computer science from Tunghai University, in 2004. He is currently a Principle Engineer with National Applied Research Laboratories, National Center for High-Performance Computing (NCHC). His current research interests include cloud computing, distribute computing, virtualization, render farm, and point cloud rendering.

CHIA-CHEN KUO received the Ph.D. degree in environmental engineering from National Chiao Tung University (NCTU), in 2011. She is currently a Research Fellow and the Director of the Arts Technology Computing Division, National Applied Research Laboratories, National Center for High-Performance Computing (NCHC). She is also a Jointly Appointed Professor with the Department of Computer Science, National Chiao Tung University. Her research interests include cloud computing, render farm, real-time rendering, point cloud rendering, and interactive technology in art.