Characterization and Prediction of Performance Interference on Mediated Passthrough GPUs for Interference-aware Scheduler

Xin Xu, Na Zhang, Michael Cui, Jiayuan He*[1], Ridhi Surana

VMware Inc, The University of Texas at Austin*

Abstract

Sharing GPUs in the cloud is cost effective and can facilitate the adoption of hardware accelerator enabled cloud. But sharing causes interference between co-located VMs and leads to performance degradation. In this paper, we proposed an interference-aware VM scheduler at the cluster level with the goal of minimizing interference. NVIDIA vGPU provides sharing capability and high performance, but it has unique performance characteristics, which have not been studied thoroughly before. Our study reveals several key observations. We leverage our observations to construct models based on machine learning techniques to predict interference between co-located VMs on the same GPU. We proposed a system architecture leveraging our models to schedule VMs to minimize the interference. The experiments show that our observations improves the model accuracy (by 15% ~ 40%) and the scheduler reduces application run-time overhead by 24.2% in simulated scenarios.

1. Introduction

An increasing number of workloads are being migrated to cloud, such as virtual desktop infrastructure (VDI), big data analytics for business intelligence, facial recognition, natural language processing, etc [1]. General Purpose Graphics Processing Unit (GPGPU) is one of the major hardware accelerators that has been commonly used to speed up these workloads. All major cloud providers, such as Google [2], Amazon [3], Microsoft [4], and IBM [5], offer their solutions with GPUs. Even large enterprises and research institutions are also building their own private cloud data center to accommodate such use cases internally.

In private or public cloud, such solutions usually provision dedicated GPGPUs for VM instances to meet the high performance requirement. But this may not be cost-effective. The consumption of GPGPUs may follow the same diurnal pattern like other resources in data centers [6–8]. That is, the resource may be underutilized during certain time periods. From cloud providers' perspective, sharing those expensive GPGPUs among multiple users is a more cost effective solution. From users' perspective, these dedicated GPUs in cloud are still quite expensive. For example, an Amazon EC2 Instance with one dedicated NVIDIA Tesla V100 costs $3.06/hour, and a regular Amazon EC2 Instance with similar configuration without GPU only costs $0.34/hour. In many cases, such as VDI, ML inferencing and development, dedicated GPUs may not be required. Sharing GPUs at a lower cost could fit a lot of such cases for cloud customers.

However, sharing a GPU may cause interference between co-located workloads and lead to performance degradation. Previous works either focus on GPU performance problem in bare-metal environments [9–11], or providing functional support to share GPU in bare-metal [12, 13] or virtual environment [14–16]. The performance problems on GPUs in virtual environments has not been thoroughly studied.

We designed an interference-aware scheduler that operates at the cluster level to reduce interference between co-located VMs that share GPUs. We characterize the interference, and use machine learning techniques to model the interference between the co-located VMs. And the scheduler uses this model to place VMs to minimize interference. Specifically, in this paper, we made two contributions:

- We conducted a thorough study to understand the performance characteristics of NVIDIA vGPU, which has not been thoroughly studied before. Such a study revealed interesting characteristics of NVIDIA vGPU. The experiments show that our observations from this study help improve the prediction accuracy on the interference.
- We proposed an interference aware scheduler for cloud virtual environments that schedules VMs to reduce the interferences between co-located vGPU VMs. We constructed several machine learning models to estimate the interference. Experiments show that our models reduce the overall workload run time by 24.2%.

2. Background and Motivation

Cloud computing and virtualization for emerging workloads. Workloads like HPC and machine learnings requires high performance. Recently, as hardware and software technology keeps advancing, we observe a trend where these workloads are migrating to both private cloud [17, 18] and public cloud [19]. On hardware side, an increasing num-

[1] Jiayuan worked on this project while he was an intern with VMware

ber of hardware accelerators become available to provide high performance for these workloads, such as GPGPU, FPGA [20–22], Intel QuickAssist Technology [23] and Remote Direct Memory Access (RDMA) devices. On software side, private and public cloud platforms provide many well-know benefits such as provisioning flexibility, management efficiency, and security isolation. The core technology in cloud computing, virtualization, is also advancing to better leverage these hardware accelerators for performance gain. For example, VMware ESXi provides VMDirectPath I/O mode (or passthrough mode) that allows these hardware accelerators to be directly accessed from guest VM [24] to achieve near bare-metal machine performance. Traditionally, passthrough mode is not compatible with virtualization features such as live migration. But this issue is being addressed in both academia and industry [25–27]. Therefore, we believe that cloud and virtualization now can provide both performance and functionalities that are required for these emerging workloads.

GPGPU sharing. In bare-metal environments, GPU sharing is mainly controlled by OS level driver framework as shown in Figure 1 (a). GPUs support concurrent kernel (CUDA functions that are executed on GPUs) executions within the same application. NVIDIA provides Multiple Process Service (MPS) [13] further improves concurrency by allowing multiple applications to execute kernels concurrently on GPUs. In MPS mode, kernels from different applications are not isolated, so it has potential issues about security, e.g., side channel attack in GPU [28]. API remoting is another approach to share GPUs among remote clients [12, 29], but clients are limited by the APIs that are provided by the frameworks.

In virtual environment, a common way to share GPUs is virtualizing the hardware [14, 15]. The common architecture is shown in Figure 1 (b). Virtualizing GPUs are non-trivial work. It also add an layer of indirection in the hypervisor, and cause performance overhead. NVIDIA vGPU uses a quite unique mechanism, called mediated passthrough [30], to share GPUs as shown in Figure 1 (c). In vGPU mode, the hypervisor manages the control path of the GPU to achieve device sharing, VMs can directly access the data path as if they were accessing a dedicated GPU. Our performance evaluation on eight ML benchmarks shows that vGPU has comparable performance with passthrough GPU when running a single VM. Six of them do not have performance difference, and two of them have difference within 5%. Also, on VMware ESXi, VMs with vGPU are compatible with virtualization features such as live migration and suspend/resume [31, 32]. Therefore, in this paper, we choose to use this platform.

vGPU interference problem. In virtual environments, multiple VMs may use vGPU at the same time, and causing interference to each other. We conduct the experiments using applications and the setup in Section 6. Figure 2 shows

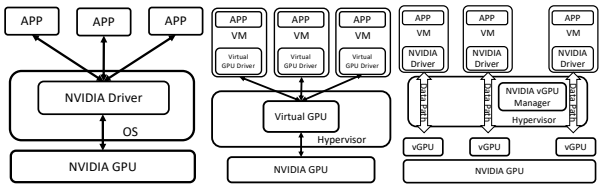


Figure 1. Bare-metal vs virtual vs Mediated Passthrough

the increased runtime of machine learning workload *dcgan* when it is co-located with another workload in vGPU mode. There are two observations from the results: 1) the interference can be very high - 220% when dcgan is co-located with word language processing. We should design our system to reduce such interference; 2) interference varies by workloads - when co-located with alexnet or mnist, the overhead for dcgan is almost negligible. This provides an opportunity to reduce interference by carefully selecting co-located workloads to minimize interference. This result motivated us to design an interference-aware VM scheduler at the cluster level to minimize interference.

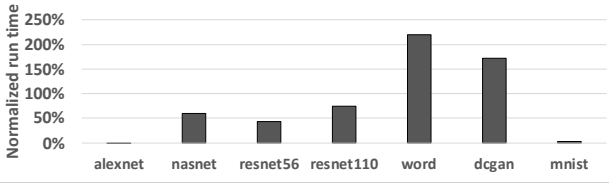


Figure 2. Interference for dcgan

One unique behavior of NVIDIA vGPU is that VMs time-shares GPU cores, and the context switch between VMs is enforced by hardware preemption. Such hardware preemption is not support before the Pascal architecture. Our experiment with simple GPU kernels show that context switches can cause up to 30% overhead. Due to space limitations, we omit the experiment details and figures.

3. Related Work

Resource scheduling in multi-tenant environments is an important topic, and various solutions have been proposed in both academia [33, 34] and industry [35]. Many work are focusing on CPU, memory, networking and storage resource. GPUs as a different compute resource become increasingly commonly for workloads requiring high performance. Therefore, resource scheduling should also taking GPUs into account.

In bare-metal environments, Prophet [10] and Baymax [9] focus on the application QoS problem for non-preemptible GPUs. It constructs performance models to predict kernel run times so that it can better schedule kernel executions to improve QoS. Mystic [11] focuses on the shared GPUs using rCUDA and MPS. It uses an analytical model to predict workload performance, and develops a scheduler based on the model to reduce interference between applications.

The high level idea of our paper is similar to these works - modeling the interference for better GPU job scheduling. But the approach to constructing models is quite different because the unique performance characteristics of vGPU. In MPS, concurrent kernel executions create interference at very fine-grain level such as L1/L2 cache because kernels accessing these resources at the same time. But vGPU has a better context separation, and therefore interference characteristics is expected to be different. For non-preemptible GPUs, kernels are not preempted as often as preemptible vGPU. Therefore, the context switch overhead is expected to be different as well.

In virtual environments, Pegasus [16] proposed a scheduler that is implemented in the hypervisor (driver domain) for virtualized GPUs to improve performance. It applies to para-virtualization mode GPUs only, not for mediated passthrough. Because the datapath in mediated passthrough are directly exposed to VM, not managed by the hypervisor (which provides the performance advantage). Therefore, the hypervisor cannot control the job submission. Prophet and Baymax proposed the schedulers that are on the CUDA job submission path, so they are not suitable for mediate passthrough GPU for the same reason. Therefore, we need to conduct new studies to understand the inference mechanism for mediated passthrough vGPU particularly. Also, we need scheduler at a higher level that works for mediate passthrough GPUs.

4. System Architecture

We proposed a cluster-level interference-aware VM scheduler. The architecture is shown in Figure 3. The system consists of an offline application profiling stage, and an offline model training stage, and an online VM scheduler. We leverage analytical techniques and machine learning techniques to construct models that quantify the application interference. We assume that each VM only host one application. At the application profiling stage, we run the applications to collect the statistics when each application is running alone without any interference. These characteristics are selected to reflect intrinsic system runtime behaviors (CPU, memory, GPU utilization, etc). The statistics are stored in a database with a label for the application so that they can be easily retrieved when by the scheduler. At the model training stage, we run applications when interference presents to collect application run times. The inputs of models are application statistics, and the output is the interference (quantified based on the run time of applications). At scheduling stage, the scheduler obtain statistics from the database and uses the models to evaluate the interference on all possible hosts to find the host with least interference. The key components in the system are identifying the appropriate statistics to reflect GPU performance, and constructing the models. In the following section, we discuss how to achieve that.

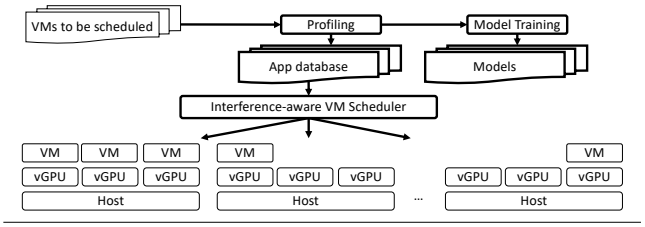


Figure 3. System Architecture

5. Experiment Setup

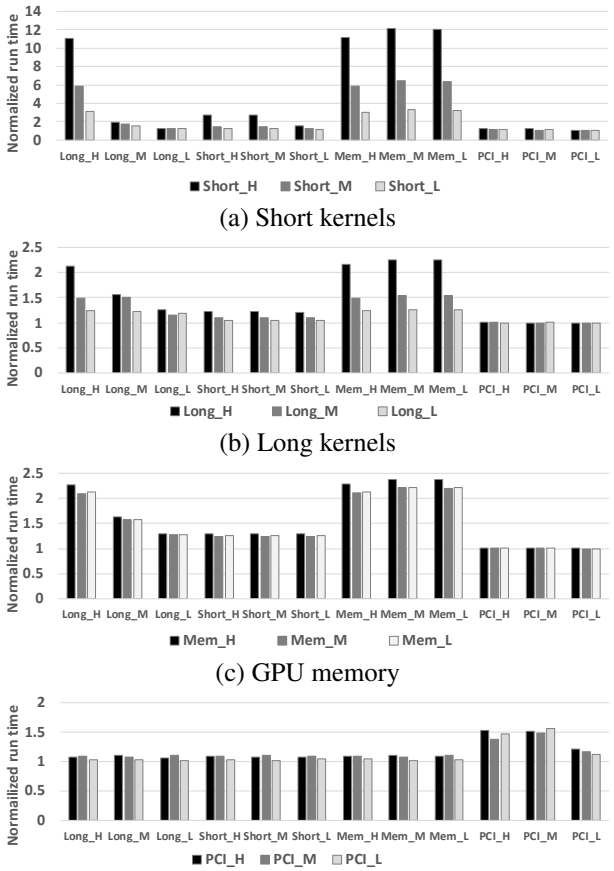
Experiments are done on a 2-socket Supermicro machine with two 12-core Intel Xeon E5-2680 v3 2.5GHz GPU, 128 GB memory, and one NVIDIA Tesla P100 16GB GPU with a 390.42 NVIDIA vGPU driver. We set up two VMs, each with 8 VCPUs and 16GB memory, 64 GB disk, Ubuntu 16.04, and CUDA 9.0. VMs use either 8GB profile (2 VMs per GPU) or 4GB profile (4 VMs per GPU) depending on experiments. The vGPU scheduler is set to best effort if not explicitly specified since this is the default scheduler and enable maximum device utilization. We use this same setup for all experiments throughout this paper.

6. Performance Characterization

The interference model must be able to capture the intrinsic system level behaviors that can reflect application interference on GPUs. We develop microbenchmarks with different intensity levels of GPU core utilization, GPU memory utilization and PCIe bandwidth utilization. We conducted experiments and measured that the context switch time slice is about 1ms. To see how this time slice affect kernels with different lengths, we use two microbenchmarks to stress GPU cores: short kernel (kernel length $<$1ms) and long kernel (kernel length $>$1ms) to evaluate the impact of context switches on kernels with different lengths. Each type of microbenchmark has three stress levels: high (H), medium (M) and low (L). For GPU core (or GPU memory) utilization, the three levels are 100%, 50% and 25% for reported GPU (or GPU memory) utilization. For PCIe bandwidth, the three levels are 10GB/s, 7GB/s and 5GB/s.

We run one microbenchmark in each VM, and run two VMs simultaneously with vGPUs. With one microbenchmark repeatedly running in one VM (background), we measure the other microbenchmark's run time in the other VM (foreground). The results are shown in Figure 4. Each bar in the figure represents the normalized run time of that foreground microbenchmark, based on its run time without interference. The X-axis is the background microbenchmark that causes interference. We summarize the observations from the figures below.

In figure (a), the short kernels are severely affected by the long kernels and the memory benchmarks, and the run times increase by 2X ˜ 11X. The short kernels are also affected by the short kernels, and the run times increase by 13% ˜ 2.7X.



(a) Short kernels

(b) Long kernels

(c) GPU memory

(d) PCIe bandwidth

Figure 4. Microbenchmarks under interference

The short kernels are not affected by PCIe benchmarks. In figure (b), the long kernels also show the similar trend as the short kernels, but the scale is smaller. The maximum increase in the run time is around 2X. The reason why the short kernels experience a higher overhead is because of the way the best effort scheduler works. Figure 5 illustrates how the short kernels are executed on the GPU when there are or no co-located long kernels.

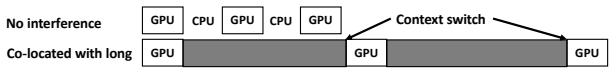


Figure 5. Short kernels running in best effort scheduler

When there is no interference (the baseline of this experiment), multiple short kernels can be finished consecutively in one time slice. When there are co-located long kernels, once a short kernel finishes, the short kernel benchmark switches to CPU task and long kernel starts to run on the GPU for 1ms. During this time, even if the short kernel benchmark finishes its CPU task and wants to run its next short kernel on the GPU, it must wait until the next context switch. On the other hand, when the long kernels are affected by the short kernels, they are only interrupted briefly by short kernels. Therefore, the short kernels show higher overheads than the long kernels. Note that in real applications the overhead also depends on the timing when kernels are issued to the GPU. For example, if this short kernel workload has a longer CPU task, the kernel execution on the GPU may be perfectly interleaved between two kernels, and the overhead won't be so high. The exact amount of overhead for real world applications is difficult to determine through analyzing applications' each kernel execution. This motivates us to use modeling techniques to capture the high-level characteristics of the applications and estimate the overhead. **Observation: Interference that is caused by context switches depends on the kernel lengths of co-located workloads and the timing when kernels are issued to the GPU.**

In figure (c), the GPU memory benchmarks can be affected by the same memory activities, and the run times increase to more than 2X. Different memory utilizations cause similar overhead. Kernel executions also cause overhead, and longer kernels with higher utilizations cause high overhead. **Observation: Memory accesses can be affected by both kernel executions and memory activities.** In figure (d), the PCIe benchmarks are not affected by kernel executions or memory activities, but are only affected by PCIe benchmarks. **Observation: PCIe data transfer is less likely to be affected by GPU core and memory activities.**

To summarize our observations, the mechanisms that cause the interference to co-located workloads are complicated. It is difficult to quantitatively deduce the interference just based on these observations. This motivates us to use machine learning techniques to construct mathematical models to predict the interference. These observations are essential for accurate models.

7. Model Interference

To model vGPU interference, we first select the features as model inputs based on our observations, and then construct classification and regression models. We demonstrate how to leverage our observations to improve the model accuracy.

Feature Selection. To construct models, we first need to identify a set of software and hardware characteristics that are closely related to interference based on our observations. The selected features are in Table 1.

Table 1. Three feature sets

	Features
Base	GPU util, GPU mem util, PCIe read/write b/w
Extend	GPU util, GPU mem util, PCIe read/write b/w, VCPU, VM mem, avg num threads, avg kernel length, avg long kernel length, long/short kernel ratio
Select	GPU util, PCIe read/write b/w, PCIe write b/w, VCPU, VM mem, avg kernel length

Intuitively, the features in the *base* set can be used to construct models. Based on our observations above, we add a

feature to characterize the kernel compute intensity: the average number of threads, features to characterize both kernel lengths and kernel execution's timing, features to characterize kernel lengths: the average kernel length, the average long kernel length, and the ratio of the total number of the long kernels and the total number of the short kernels. The timing of kernel executions is difficult to represent. We choose to include vCPU utilization and VM memory utilization as 2 features, because these two statistics, combined with GPU statistics, represent the interaction between CPU tasks and GPU kernels. So they can reflect the timing of kernel executions at the high level. We also define the *select* set which is a refined set with fewer features, but we believe they may still present workloads characteristics based on our previous observations.

We collect features either through ESXi, Nvidia-smi, or nvprof, when the workload runs alone. Since nvprof is not supported in vGPU mode, we run workloads in passthrough mode to get detailed statistics about kernels. Note that the purpose of nvprof is to collect workloads' intrinsic software characteristics, so it does not have to be in vGPU mode.

Model Construction. The model output is interference which is represented by the run time overhead. We collect a workload's run time when it is co-located with every other workload running repeatedly. That is, we collect workload A's run time with workload B running repeatedly, and repeat this experiment by replacing B with other workloads. This overhead is normalized based on its run time when it runs alone. We select seven machine learning workloads from Pytorch [36] and Tensorflow [37], and run two batch sizes for each workload, so in total we have 14 workloads. We did 5-fold cross validation with 80% training data and 20% testing data. We constructed a random forest regression model and a linear regression model. Table 7 shows the mean square errors (MSE) when using different feature sets for the two models.

Table 2. Regression models' MSEs

	Base	Extend	Select
Random forest	0.151	0.129	0.122
Linear	0.355	0.213	0.207

The random forest model achieves lower MSE than the linear model. This is not surprising given the complicated interference mechanism. For each model, the extend feature set achieves a better accuracy than the base feature set - 15% reduction in MSE for the random forest model and 40% reduction for the linear model. It means the additional features improves the models, and demonstrates that our characterization helps the modeling process effectively. The select feature set achieves similar as the extend feature set, suggesting the select features are the most relevant ones to interference. This also aligns with our observations. We also constructed classification models using random forest, decision tree, naive Bayes, and k-nearest neighbors. These models also show good accuracy with the average F-scores around 0.6 to 0.8. F-scores is a measure of model accuracy considers both the precision and the recall. Its best value is 1 and its worst value is 0. The good accuracy of these models suggests that there is a strong correlation between the features that we selected based on our observations and the interference, and it is possible to predict the interference.

Note that we construct models for vGPU, not for workloads. The purpose of the workloads is stressing various components of the GPU. While having more data from more workloads are useful, it is not necessary to run all ML applications to build models. It is possible to use a set of microbenchmarks to create synthetic workloads to stress the GPU. We construct the models for one vGPU profile. We can use the same approach to construct a model for each vGPU profile. Or we can consider the number of current running VMs as a model feature to just construct one model that fits all possible profiles. This is worth further investigation.

Evaluation. To further demonstrate the benefit of characterization and our prediction models, we design an interference-aware scheduler that leverages our models for VM placement. Using outputs from the random forest regression model, the interference-aware scheduler places VMs on hosts in a way that minimizes interference. In addition, we design a simple round-robin scheduler and an ideal scheduler that are based on the actual and predicted interference data respectively. Using the actual interference data allows us to achieve the lowest overhead under the same scheduling policy. If predicted data is effective, the results should be close to this theoretical limit. The experiment simulates a cluster with 50 hosts, each of which has one GPU that supports two VMs, for a total of 100 VMs that utilize the full capacity of the cluster. The interference is evaluated by aggregating the individual interference of each co-located VM. The workloads are randomly generated from the aforementioned 14 workloads, and we repeat this experiment 100 times. The average overhead reduction is 28.5% for the ideal scheduler and 24.2% for the random forest scheduler.

8. Conclusion

In this paper, we proposed a cluster level interference-aware VM scheduler. We show our methodology to characterize and predict the interference between workloads that use NVIDIA vGPU. We made observations that help us employ machine learning techniques to construct prediction models. The results show that our interference-aware scheduler based on our models can effectively reduce application run-time overhead by 24.2%.

9. Acknowledgment

The authors would like to thank Josh Simons for the support and guidance of this project and providing valuable feedback on the paper. The authors also like to thank Bryan Tan for his contribution to the project.

10. Discussion Topic

Note that there are more run time statistics that can be considered in the performance characterization such as L1/L2 cache miss/hit ratio, GPU TLB hit/miss ratio, etc. Those architectural statistics may or may not be relevant because we do not know exactly how context switch impacts those components, e.g., whether it cleans up GPU TLB or L1 cache. Since nvprof in vGPU mode is not supported, it is difficult to do the detailed analysis. We may use more microbenchmarks to evaluate the high-level performance characteristics. This is worth further exploration.

Current modeling process produce one model per vGPU profile, because different profile support different number of VMs. In fact, the profile can be a feature in the model (quantified by the number of running VMs and vGPU memory size). Then, we only need one model per one type of GPU. The applications in the experiments are used to exercise GPUs, so we want to have reasonable amount of data to cover various behaviors, e.g. high and low utilizations, number of threads, etc. But the modeling approach does not require us to run all applications in all possible combinations to be effective. We can periodically re-construct the model to make sure new applications with new patterns are included.

We assume one application per VM. This assumption may not be true for all clusters. If the workload can change over time while a VM is running and generate a different interference behavior, the new interference pattern won't be predicted by scheduler. Such dynamic behaviors must be monitored by an online interference detector. The detector can rely the aforementioned infrastructure to collect the system run-time statistics. It may be as simple as monitoring GPU resource utilization and send the notifications when they are heavily utilized. Or a more intelligent detector may be built to detect interference by examining various system states.

This paper is based on NVIDIA vGPU and VMware ESXi, which are proprietary products. However, we believe that the framework proposed here are generally applicable to other products. Because our characterization approach treat GPU as a black/grey box, and the statistics we collected are at high level.

References

[1] Google Cloud. Cloud AI products. [Online]. Available: https://cloud.google.com/products/ai/

[2] Google. Leverage GPUs on Google Cloud for machine learning, scientific computing, and 3D visualization. [Online]. Available: https://cloud.google.com/gpu/

[3] Amazon. Amazon EC2 Instance Types. [Online]. Available: https://aws.amazon.com/ec2/instance-types/

[4] Microsoft. Introduction to NVIDIA GPUs in Azure. [Online]. Available: https://azure.microsoft.com/en-us/resources/videos/build-2016-introduction-to-nvidia-gpus-in-azure/

[5] IBM. GPUs for cloud servers. [Online]. Available: https://www.ibm.com/cloud/gpu

[6] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 123–137, Aug. 2015. [Online]. Available: http://doi.acm.org/10.1145/2829988.2787472

[7] L. A. Barroso and U. Hoelzle, *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 1st ed. Morgan and Claypool Publishers, 2009.

[8] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 267–280. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879175

[9] Q. Chen, H. Yang, J. Mars, and L. Tang, "Baymax: Qos awareness and increased utilization for non-preemptive accelerators in warehouse scale computers," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '16. New York, NY, USA: ACM, 2016, pp. 681–696. [Online]. Available: http://doi.acm.org/10.1145/2872362.2872368

[10] Q. Chen, H. Yang, M. Guo, R. S. Kannan, J. Mars, and L. Tang, "Prophet: Precise qos prediction on non-preemptive accelerators to improve utilization in warehouse-scale computers," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17. New York, NY, USA: ACM, 2017, pp. 17–32. [Online]. Available: http://doi.acm.org/10.1145/3037697.3037700

[11] Y. Ukidave, X. Li, and D. Kaeli, "Mystic: Predictive scheduling for gpu based cloud servers using machine learning," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 353–362.

[12] J. Duato, A. J. Pea, F. Silla, R. Mayo, and E. S. Quintana-Ort, "rcuda: Reducing the number of gpu-based accelerators in high performance clusters," in *2010 International Conference on High Performance Computing Simulation*, June 2010, pp. 224–231.

[13] NVIDIA. NVIDIA Multi-Process Service. [Online]. Available: https://docs.nvidia.com/deploy/mps/index.html

[14] Y. Suzuki, S. Kato, H. Yamada, and K. Kono, "Gpuvm: Why not virtualizing gpus at the hypervisor?" in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIX ATC'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 109–120. [Online]. Available: http://dl.acm.org/citation.cfm?id=2643634.2643646

[15] M. Gottschlag, M. Hillenbrand, J. Kehne, J. Stoess, and F. Bellosa, "Logv: Low-overhead gpgpu virtualization," in *2013 IEEE 10th International Conference on High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, Nov 2013, pp. 1721–1726.

[16] V. Gupta, K. Schwan, N. Tolia, V. Talwar, and P. Ranganathan, "Pegasus: Coordinated scheduling for virtualized accelerator-based systems," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 3–3. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002181.2002184

[17] InsideHPC. VMware HPC Virtualization Enables Research as Service. [Online]. Available: https://insidehpc.com/2015/11/video-vmware-hpc-virtualization-enables-research-as-service/

[18] VMware Technical White Paper. ENABLING MACHINE LEARNING AS A SERVICE (MLAAS) WITH GPU ACCELERATION USING VMWARE VREALIZE AUTOMATION. [Online]. Available: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/solutions/vmware-mlaas-vrealize-automation-white-paper.pdf

[19] InsideHPC. Case Study: Deep Learning For Fluid Flow Prediction In The Cloud. [Online]. Available: https://insidehpc.com/2019/01/case-study-deep-learning-for-fluid-flow-prediction-in-the-cloud/

[20] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–13.

[21] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," *SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 13–24, Jun. 2014. [Online]. Available: http://doi.acm.org/10.1145/2678373.2665678

[22] Microsoft. Project Brainwave. [Online]. Available: https://www.microsoft.com/en-us/research/project/project-brainwave/

[23] Intel. QuickAssist Technology. [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/intel-quick-assist-technology-overview.html

[24] VMware Blog. Performance and Use Cases of VMware DirectPath I/O for Networking. [Online]. Available: https://blogs.vmware.com/performance/2010/12/performance-and-use-cases-of-vmware-directpath-io-for-networking.html

[25] Z. Pan, Y. Dong, Y. Chen, L. Zhang, and Z. Zhang, "Compsc: Live migration with pass-through devices," in *Proceedings of the 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*, ser. VEE '12. New York, NY, USA: ACM, 2012, pp. 109–120. [Online]. Available: http://doi.acm.org/10.1145/2151024.2151040

[26] X. Xu and B. Davda, "Srvm: Hypervisor support for live migration with passthrough sr-iov network devices," in *Proceedings of the12th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '16. New York, NY, USA: ACM, 2016, pp. 65–77. [Online]. Available: http://doi.acm.org/10.1145/2892242.2892256

[27] Intel, "Achieving fast, scalable i/o for virtualized servers," *White Paper*.

[28] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: Gpu side channel attacks are practical," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 2139–2153. [Online]. Available: http://doi.acm.org/10.1145/3243734.3243831

[29] L. Shi, H. Chen, J. Sun, and K. Li, "vcuda: Gpu-accelerated high-performance computing in virtual machines," *IEEE Transactions on Computers*, vol. 61, no. 6, pp. 804–816, June 2012.

[30] VMware. Machine Learning using Virtualized GPUs on VMware vSphere. [Online]. Available: https://blogs.vmware.com/apps/2018/05/machine-learning-using-virtualized-gpus-on-vmware-vsphere.html

[31] NVIDIA. At VMworld: NVIDIA vGPU with vMotion Delivers the Agile Data Center. [Online]. Available: https://blogs.nvidia.com/blog/2018/08/27/gpu-live-migration-vmotion-virtualization/

[32] VMware vSphere Blog. vSphere 6.7: Suspend and Resume of GPU-attached Virtual Machines. [Online]. Available: https://blogs.vmware.com/vsphere/2018/07/vsphere-6-7-suspend-and-resume-of-gpu-attached-virtual-machines.html

[33] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," *SIGPLAN Not.*, vol. 48, no. 4, pp. 77–88, Mar. 2013. [Online]. Available: http://doi.acm.org/10.1145/2499368.2451125

[34] C. Delimitrou, N. Bambos, and C. Kozyrakis, "Qos-aware admission control in heterogeneous datacenters," in *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*. San Jose, CA: USENIX, 2013, pp. 291–296. [Online]. Available: https://www.usenix.org/conference/icac13/technical-sessions/presentation/delimitrou

[35] VMware. Distributed Resource Scheduler, Distributed Power Management. [Online]. Available: https://www.vmware.com/products/vsphere/drs-dpm.html

[36] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[37] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous

systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/