An Economy-Oriented GPU Virtualization With Dynamic and Adaptive Oversubscription

Jianguo Yao, *Senior Member, IEEE*, Qiumin Lu, Run Tian, Keqin Li, *Fellow, IEEE*, and Haibing Guan

Abstract—GPU is becoming attractive around multiple academic and industrial area because of its massively parallel computing ability. However, there are still some obstacles which the GPU virtualization technologies should overcome to reach their maturity. These obstacles mainly include the problem of resource allocation strategy to guarantee possible higher yield. This shortage has already become an obvious barrier to the practical GPU usage in the cloud for satisfying business and academical requirements. There are many mature pieces of research in the area of oversubscribed cloud computing to enhance economic efficiency. However, the study on GPU oversubscription is almost blank for the just started use of GPU in cloud computing. This paper introduces gOver, an economy-oriented GPU resource oversubscription system based on the GPU virtualization platform. gOver is able to share and modulate GPU resource among workloads in an adaptive and dynamic manner, guaranteeing the QoS level at the same time. We evaluate the proposed gOver strategy with designed experiments with specific workload characteristics. The experimental results show that our dynamic GPU oversubscription solution improves the economic efficiency by 20% over traditional GPU sharing strategy, and outperforms the static oversubscription method by much better stability in QoS control.

Index Terms—Economy, GPU virtualization, QoS, resource oversubscription

1 Introduction

The cloud-style systems based on the virtualization technologies have been widely used based on the resource scheduling, sharing and isolating functionalities. A physical server usually divides its finite resources into multiple combinations including CPU, GPU, disk and memory, then offers cloud services based on virtual machines (VMs) [1]. Such virtualized computing environments enable convenient provision and release of shared resources. Since cloud customers usually tend to overestimate their resource requirement and occupy only a portion of the offered resource allocation in reality, the resource utilization is maintained under 20% in most datacenters [2]. The low-level utilization means that VMs remained idle and resources were wasted in most time. As a result, resource oversubscription becomes an appealing strategy to improve cloud resource utilization. Cloud service providers (CSPs) purposely claim more VMs than the number they can fully support to achieve an appreciable rate of return on investment. Many mature researches about the oversubscription of cloud resources are focusing on memory [3], CPU [4], network bandwidth [5], [6] and power [7]. However, there appears to be a gap in the study on GPU oversubscription due to the complex scheduling of virtualized GPU in the cloud.

GPU presents a rapid adoption in more general-purpose applications rather than just graphics rendering for its excellent parallel computing ability, especially the tasks involving large-scale floating-point arithmetic because the computation speed of these workloads can be largely accelerated by employing GPU. Moreover, cloud providers start offering GPU computing services in recent years. As examples of Amazon EC2 Elastic GPU [8] and Alibaba GPU Cloud Services [9], GPU-available cloud services are now serving as a new parallel computing option. With such a rising trend in cloud computing, GPU-intensive workloads from cloud customers generate a growing requirement for more powerful virtualized GPU with satisfying performance and efficient scheduling strategy.

The previous works on GPU scheduling based on a full virtualized solution for GPU with the mediated pass-through GPU virtualization technology. This design enables almost-native performance when running a single VM. Its scheduling is time-based and also an improved version of the round-robin strategy, so it achieves concurrency in a static environment. However, this scheduling mechanism is hardly efficient when being applied in the GPU cloud with huge and disparate applications running concurrently. It is impossible to dynamically modulate GPU allocation among VMs according to their load status. So, if the active context accomplishes its load but allocated time span remains, then the GPU device should stay idle before the scheduled context switch is triggered. There are always light VMs wasting resource but heavy VMs suffering low performance, let alone maintain a stable performance level for unpredictable peaks. As a result, GPU cannot be fully utilized under the cloud circumstance. For this reason, it has practical significance to

• *Jianguo Yao, Qiumin Lu, Run Tian, and Haibing Guan are with Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: {jianguo.yao, luqiumin, tianrun, hbguan}@sjtu.edu.cn.*
• *Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA. E-mail: lik@newpaltz.edu.*

Manuscript received 7 June 2021; revised 8 June 2022; accepted 4 August 2022. Date of publication 18 August 2022; date of current version 7 April 2023.
This work was supported in part by the Program for NSFC under Grant 62032008, in part by STCSM Project under Grant 20510712400, in part by the Ministry of Education-China Mobile Research Foundation under Grant MCM20180703, and in part by Alibaba Innovative Research (AIR) Program and SJTU-Enflame Joint Lab.
(Corresponding author: Jianguo Yao.)
Recommended for acceptance by A.C.M. Melo.
Digital Object Identifier no. 10.1109/TC.2022.3199998

apply GPU oversubscription in the cloud to improve economic efficiency.

Since the various motivations for GPU resource oversubscription are evident, it faces large challenges. First, it is meaningless to statically configure the GPU overbooking ratio for the unpredictable cloud workloads. Too large overbooking ratio will cause severe resource overload and even system crash, while too small overbooking ratio cannot achieve efficient resource utilization and considerable return on investment. As a result, it is better to take dynamic and adaptive oversubscription according to actual load conditions to wins both benefits and service quality. So GPU driver should be equipped with the function of workload monitoring to master the real-time load information and then take them as a reference to decide overbooking degree. Second, we need to design a timely efficient strategy to mitigate the GPU overload on possible one of the oversubscribed VMs. Oversubscription is based on the phenomenon that not all customers fully use their requested resources at the same time, CSPs must transfer resources from well-resourced VMs to overloaded VMs in case users find that the resource supply is not in accordance with the contract [10]. Namely, the GPU scheduler must be able to modulate resource allocation across VMs relying on their varying workloads in common case rather than statically allocated.

We then implement the economy-oriented GPU virtualization solution with dynamic oversubscription based on an open source GPU virtualization. For the most critical function, we present an intelligent and autonomous, performance-aware GPU overselling strategy. To fulfill this function, we design a complete self-adaptive GPU scheduling system, including a continuity of a series of stages, from workload monitoring to overload detection to overload mitigation. After mitigation measures being worked out, we then assess the oversubscription potentially and perform overselling according to current GPU usage condition. This complete solution is named as gOver by us in this study. To summarize, our work includes the following major contributions:

- gOver provides a strategy to adaptively oversell new VMs with virtualized GPU based on real-time monitoring, timely detection and mitigation for GPU overload. First, this dynamic scheduling and controlling mechanism offer a better arrangement on the sharing of required resources among users, which obviously improve the utilization of the system. Also, the dynamic oversubscription schema satisfies interests of cloud providers and provides QoS guarantee for customers at the same time. For cloud providers, overselling amplifies the available GPU resources for selling and brings significant economic benefits. From the perspective of customers, the performance demands are guaranteed all the time because the actual resource allocation is always being adjusted according to the requirement monitoring by our mechanism.
- gOver equips a GPU native scheduler with dynamic oversubscription. Since the introduced virtual GPU platform has offered fine-grained GPU time slot controlling and monitoring, its original scheduling strategy is static and trivial. Our mechanism upgrade its design with adaptive capacity allocation according to vGPU utilization and QoS of VMs. The maximum GPU resource authorized to each VM will be calculated at the beginning of every scheduling period. Thus, GPU reallocation happened in time to mitigate overload under oversubscription.
- Finally, we design the experiments to evaluate our proposed gOver solution on economic efficiency and QoS performance. We first analyze GPU utilization characteristics of some GPU-intensive workloads to show the necessity of GPU oversubscription. Then we compare the performance of gOver with non-oversubscribed GPU virtualization and static oversubscription method. The results show 20% improvement on economic efficiency compared to traditional GPU virtualization and much better QoS control over static oversubscription method.

The rest of this paper is organized as follows. Section 2 introduces background knowledge and motivation. The detailed design and implementation of gOver are described in Section 3. Section 4 analyzes the evaluation methodology and the experiment results. We give a discussion about the imperfection of gOver and give future work in Section 5. Section 6 reviews the related works and Section 7 concludes the paper.

2 Background and Motivation

In this section, we first summarize the principle and background in the area of resource oversubscription and the virtualized GPU resources. These descriptions include GPU virtualization technology and GPU scheduling mechanism aiming at QoS. Considering the default GPU scheduling strategy, we then come to the conclusion that there are inefficient factors about the performance and the resource utilization in the current virtualized GPU allocation and then raise the motivation.

2.1 Resource Oversubscription

Designed as a paradigm that enables flexible access to a pool of configurable sharing resources (e.g., applications, space, computing, bandwidth), the cloud has been the best solution for tenants to migrate local services and then construct an online deployment [11]. Cloud service providers should face the explosive requests for cloud resources. But it is an extremely expensive and difficult work to upgrade hardware facilities in datacenters. Furthermore, clients tend to demand the resources far beyond necessity even if only a fraction of these allocated share actually be utilized during active time, so the resource wasting is serious. The utilization rate of servers in datacenters is maintained at below 20% in general [2], [12], [13]. From the standpoint of economic efficiency, it is crucial for the future success of cloud providers to efficiently host more clients on a restricted group of physical instances.

Considering such mentioned reasons, it is appropriate to utilize hardware facilities as much as possible within its maximum capacity limitation to amortize the cost of nonrecurring investments. This solution results in resource oversubscription, which entails publishing more resources than are actually available on the cloud to the markets [10]. Oversubscription can be beneficial to the reduction of idle resource, improve utilization of resource and increase revenue. Many academic researches focus on optimal oversubscription ratio and the resource scheduling problem in oversubscribed datacenters [14]. A lot of resources have been put into oversubscribe, including memory [3], bandwidth [5], [6], CPU [4], storage [15] and power [7]. Along with the listed advantages, the resource oversubscription also causes disadvantages, which include high resource contention possibility, resource overload, and inter-user interference. It might decline service quality and even cause performance loss. CSPs should apply effective mechanisms to prevent and alleviate resource overloads, such as VM quiescing, resource stealing and service live migration [10].

2.2 GPU Virtualization

GPU virtualization technology enables each running VM to access physical GPU device directly (or partially), rather than emulating GPU computing by CPU, resulting in a significant performance boost. By now, there are device emulation researches like [16]. Also, the API forwarding studies also include rCUDA [17], [18], [19], [20], Gvim [21], FairGV [22], AvA [23] and etc. [24], [25], [26], [27] The direct pass-through [28], [29] and mediated pass-through [30] have been raised to implement full GPU functions in a virtualized manner. Among them, the mediated pass-through solution achieves the best performance with sharing capability and full features, also scalability and live migration. Under mediated pass-through virtualization, the VM commands directly access GPU in performance critical operations, while being trapped and emulated by hypervisor when executing privileged instructions.

Most GPU products are closed source without any exposed implementations of their virtualized solutions for commercial reasons. One of the few open-source GPU drivers is named gVirt which is embedded in the Linux kernel by Intel [31]. gVirt is a product level full GPU virtualization implementation with detail specifications about scheduling logic and communication protocols. This project acted as a research phase in the Intel GPU virtualization solution, and the whole solution is now published with the name GVT-g. Based on this solution, it is possible to modify or add features in the source code to realize extra functions about virtualized resource sharing and isolating [32]. gVirt employs mediated pass-through technology as shown in Fig. 1 as its full GPU virtualization solution to achieve good performance, scalability and also secure isolation among VMs. Multiple VMs share a single physical GPU but each VM believes its ownership of this physical GPU. Currently, since the original gVirt contents have already been merged into the GVT-g product, this project is still being maintained and extended with more good features such as QoS guarantee and migrations [33], [34].

2.3 QoS in GPU

In the research area, we usually use the notion of QoS to refer to the performance of cloud services. The improvement and optimization on such a metric is one of the central research topics in the area of cloud computing. In the most initial gVirt solution, there is a completely competitive scheduling strategy applied to the virtual GPU sharing. A VM with a

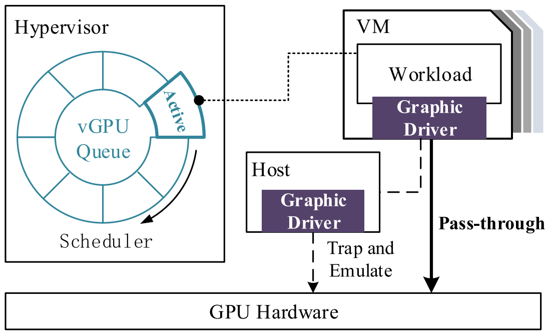


Fig. 1. The overall design of current GPU virtualization.

resource-intensive workload will request more GPU resource to reduce the share for others. So, the administrator is not able to guarantee personalized QoS to different users. Some applications may require higher performance but lower priority, while others show the opposite pattern. So the control of resources allocation is quite important for GPU QoS services.

In the improved version of gVirt, the QoS of virtual GPU can be tuned through two parameters: cap and weight. The parameter cap refers to the limitation of the vGPU resource which a guest VM can utilize in percentage. So, the configuration on this parameter can manipulate the vGPU resource consumption. However, the weight is another parameter which is orthogonal with cap. This factor can maintain the fairness among guest VMs and then guarantee the minimum resource consumption of a single VM through load balancing.

2.4 Scheduling in GPU

gVirt is a full-virtualized vGPU implementation with detailed scheduling strategy and scope communication tunnels, which makes it possible to modify or add features in the source code to realize extra functions about virtualized resource sharing and isolating [32]. gVirt emulates each vGPU and conducts GPU device resource scheduling among vGPUs in a time-sharing scheduling strategy. The vGPUs are all linked together in a scheduling queue for the round-robin scheduling. GPU uses time slices to decide which VM should be activated in this period of time. And the GPU scheduling strategy triggers context switch in a fixed time interval. gVirt saves the active vGPU states, then restores the following picked vGPU context at the end of every time slice.

The time interval is critical for balancing performance and efficiency of vGPU. The short scheduling interval can cause unacceptable scheduling overhead. However, an excessively long interval often have negative effect on scheduling granularity and block the fluency of the workloads. The scheduling interval in gVirt is 1 ms, and at the end of it the context switch will be triggered. The active context in every interval under scheduling is shown in Fig. 2. When a context is switched into the inactive state, the activated time of the VM should be recorded and then subtracted from the scheduled time limitation. Then the scheduler validates the candidate VM whether the allocated interval is exhausted. For example, according to the last third part shown in Fig. 2, VM1, and VM3 run out their time

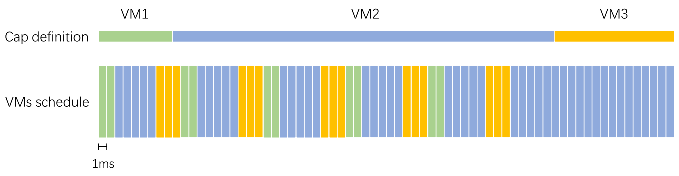


Fig. 2. VM context switch scheduling among time slices.

allocation, while VM2 still have some allocated time. As a result, VM2 is the only schedulable guest in the following time. The scheduler will suspend running applications on VM1 and VM3 in the current scheduling cycle.

2.5 Motivation

Cloud service users tend to have a high-level requirement against availability and performance in a clustered environment and usually overestimate their resource needs. Following the terms of contracts which the service users have agreed, service providers submit VMs with a superfluous capacity to guarantee the QoS level. However, for the most part, cloud workloads tend to utilize just a fraction of the assigned resources. Then many GPU cycles are just occupied with no works to run. However, current GPU virtualization technology achieves concurrency in a static environment. It is impossible to temporarily adjust resource allocation to serve the intensive workloads, or maintain a stable performance level for instantaneous peaks. The original GPU scheduling mechanism is hardly efficient when applied in the GPU cloud where huge applications with disparate performance demand running concurrently. So, the workloads with different levels of GPU resource requirement never get an appropriate allocation and then the cloud GPU cannot have an efficiency utilization with less idle resource share.

Then we can summarize the challenges we should face in designing an efficient economic-oriented mechanism for optimizing the utilization, economic profit and QoS performance of virtualized GPU platform with oversubscription. First, the static oversubscription is impractical, and only the dynamic resource allocation can support the execution of real workloads. Second, the GPU resource utilization with multiple workloads performs unacceptable under traditional scheduling strategies since many time slots remain idle. As a result, a new resource allocation mechanism is required to improve resource utilization. Finally, since the target environment should support virtualized GPU resources, our proposed mechanism should adapt its specific characteristics.

Facing such a challenge, we investigate the specified implementation of a GPU virtualization and then support GPU oversubscription based on it. Our purpose is to implement an adaptive GPU sharing strategy under cloud oversubscription to improve economic efficiency. The time slot allocation to every VM should be dynamically and automatically configurable according to runtime status of workloads. Also, the uncontrollable competition on oversold resources should be overcome to satisfy the user resource allocation under oversubscription and avoid resource support violation.

Actually, in GPU virtualization, the general resource sharing method is to allocate the computation time slots among different clients under some scheduling strategies and then virtualize the allocated time slots into virtualized GPU contexts. Since here some overheads may be caused by the virtualization implementation, our strategy can certainly improve the resource utilization. The fact is, even in the situation that a single workload exclusively occupies the GPU device, there should still be many time slots keeping idle with no rendering command executed in the whole execution time period. However, in our mechanism, the vGPU contexts can keep receiving and caching rendering commands no matter they are activated or switched down. Since they are scheduled and activated in a round-robin manner, the GPU device can keep processing the commands and reduce the idle time percentage. Under this solution, the idle overhead which may be caused by resource requirement variation or over-provision can all be relieved. Also, with this proposed mechanism, the service providers can offer more virtual GPU resources than they actually possess with the user-defined QoS demand satisfied, which can obviously bring economic profits to them.

3 Design and Implementation

This section mainly includes the description about gOver, our GPU oversubscription strategy design. Resource oversubscription comes in many forms. In this study, we consolidate the guest VMs into a server node. The VM count is more than the fitting level, which means that the total occupied resources claimed by all the VMs are more than the physical device capacity.The target of gOver solution can be summarized as follows: 1) maintaining the parallel performance of the concurrent workload executions efficiently on GPU device. 2) guaranteeing the QoS level for all VMs with oversubscription. There are two important points that require attention in designing and implementing the GPU virtualization with dynamic oversubscription.

- How the GPU oversubscription potential to evaluate. The load characteristics of every workload are always different, and therefore resource utilizations can be changing in tendencies. It needs to design a suitable method to infer the GPU requirement for each VM and then accurately assess the oversubscription potential.
- How to mitigate performance interference among VMs. Oversubscription signs a blank cheque on the GPU capacity supply. All customers have the right to access the oversold resources. Given the over-allocated circumstance, resource contentions increase a lot among co-hosting VMs. Disorganized contention brings adverse impacts on performances. As a result, it needs to carefully alleviate the interference issue on performance.

3.1 Overall Design

We then have the description about the complete solution to construct the GPU virtualization with dynamic oversubscription. The top-level structure of gOver is illustrated in Fig. 3. There are three independent, yet cooperating modules, which are responsible for monitoring, oversubscribing and scheduling respectively. The whole architecture is based on the Xen system.

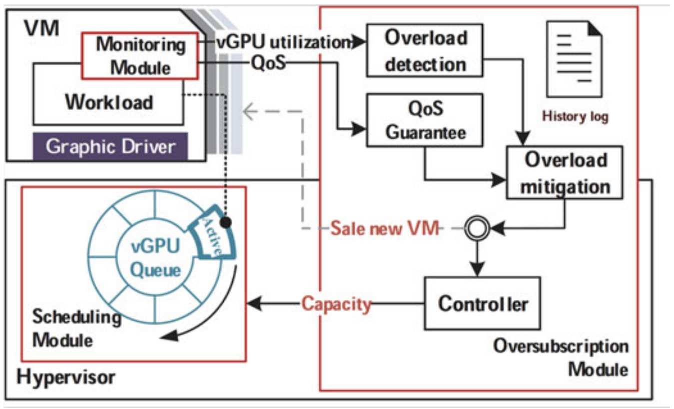


Fig. 3. The overall design of gOver.

For the essential first step, we need to monitor the runtime information on VMs. There are two manifestations of vGPU overload, one is high vGPU utilization, which reflects insufficient cap allocation, and the other is low QoS, which includes goals of availability, throughput, and consistency. QoS has different forms in different business scenarios. For this paper, we take the frames per second (FPS) as an example measurement of GPU QoS. An application scenario of this measuring indicator is the Cloud gaming [35], [36], which has been successfully developed due to the speed-boosting network. A piece of high-performance GPU can be distributed to multiple game players. Cloud game also put forward an urgent requirement of GPU oversubscription.

The oversubscription module is mainly in charge of making the overselling decision with the help of five subcomponents. It receives periodic reports of statistical data from the monitoring module and maintains history data for a limited time. The overload detection component receives the information of vGPU utilization and judges if an overload occurs on any VM according to historical data. The QoS guarantee component receives the QoS information and judge if QoS on VMs is broken. The overload mitigation component receives their judgment results and computes a suitable cap value to each VM through a comprehensive analysis of vGPU load and QoS. The key point is that it may trigger an overselling behavior if all vGPUs work easily. The controller is in charge of resetting cap in the vGPU scheduled info.

The scheduling module in kernel driver recalculates the max timeslice authorized to each VM according to the new cap value at the beginning of each scheduling period. Then all active guest servers run under the adjusted resource capacity limits in every scheduling period to cooperate with oversubscription strategy. By applying this architecture, we realize the dynamic and adaptive GPU oversubscription in safety.

3.2 Monitoring Module

Resource overallocation is a risky technique for the unpredictable workloads of cloud applications. If the load on a server suddenly explodes, it would be easy to break the promise on performance to customers for cloud providers. So we first propose the monitoring module to master load fluctuation such that we can take countermeasures in advance to avoid service collapse. Also, we need to know the FPS to inspect whether it meets the contract requirements with customers. As the name implies, the monitoring module is responsible for calculating and recording runtime information of all active virtual machines. In the top-level structure, the monitoring module is an interface through which the application workloads can be watched. To realize the recording of FPS, the LD_PRELOAD linux environment variable is set to replace the default rendering functions, then the module can record the rendering timestamp and then account for the FPS metric. Because FPS does not require extreme time precision, some discrepancy is allowed as long as FPS accomplishes on a macro level. There are no ready-made tools to ask for GPU real-time utilization. So, the GPU driver should be modified to realize the calculation of vGPU utilization.

The GPU scheduling is periodic and the percentage value of cap also need a denominator to do the arithmetical operation. That is to say, we need a proper period to help calculate the resource ratio that the guest has token. A short period has better precision but too-short period is meaningless as most context cannot be finished during that time. A longer period has worse precision then end-user may feel that the performance is not stable, sometimes fast sometimes slow. And as a denominator, it should be easy to help the percentage calculation, a right value can achieve higher numerical accuracy and computing efficiency. As a result, 1 s is the choice, considering most benchmark and user experience, the performance accuracy at per-second level is enough. We modify the struct of vGPU schedule information to maintain a list of variables as Listing 1, including when the vGPU is scheduled in and out the active queue. Note that if the task queue is empty in a vGPU, it will not be scheduled into active and then the next vGPU in order is checked. So the difference between every pair of inside time and outside time is fully occupied by vGPU to conduct computing. We add the sum of all difference value in a scheduling period into the busy time. The max timeslice is equal to the product of cap and the scheduling period. So we can get the vGPU utilization by dividing the busy time by the max timeslice.

Listing 1. vGPU Schedule Information

```
struct vGPU_sched_info {
  tslice_t sched_in_time;
  tslice_t sched_out_time;
  tslice_t busy_time;
  int32_t cap;
  int32_t initial_cap;
  int64_t timeslice;
  int64_t max_timeslice;
  //more{\ldots}
};
```

3.3 Oversubscription Module

The oversubscription module is mainly deployed in the hypervisor for the requirement of privileged command execution. According to the system structure, then the oversubscription module can maintain the connections with all the active guest VMs. The runtime status then can be collected through the port-like interfaces. At the beginning of the physical machine release, we have no knowledge about the customer applications to initialize an oversubscription ratio.

So we first sell the physical machines in non-oversubscribed mode. And then, the overload detection component and the QoS guarantee component receive the vGPU utilization and FPS of each VM. We refer to the ith VM as vm_i. We define the vGPU utilization and FPS on vm_i as u_i and fps_i. The cloud customer will negotiate with the provider about their QoS demands and the application running on vm_i will set its FPS requirement as a constant value fps_t (t means target) in advance. Then we initialize the initial cap in vGPU scheduled info which is mentioned in Listing 1 according to FPS demands and GPU hardware performance. The initial cap is a pessimistic expectation for satisfying heaviest workloads under QoS demands of the customer, so the dynamic cap value which is authorized to vGPU in reality should be always less than it.

In this module, we mainly propose two critical points to meet the challenge of dynamic oversubscription. The first point we propose is the assessment method of GPU oversubscription potential. We take the overload line of vGPU utilization as U_o (o means overload). On the one hand, if any u_i exceeds U_o three times in succession according to history logs, it will send vGPU utilization of all VMs at this moment with an overload signal to the mitigation component. On the other hand, if any fps_i does not satisfy its fps_t, the QoS guarantee component will also send the difference value as e_i ($e_i = fps_t - fps_i$) with an overload signal to the mitigation component. As for the mitigation part, there are two kinds of behaviors. First, if it does not receive any overload signal three times in succession, then the physical machine probably over-provision its GPU resource in the current state. Suppose there are n VMs, we define the following formula to evaluate whether to oversell VM: $total\_load = \sum_{i=1}^{n} u_i \times cap_i$, where cap_i is the capacity percentage allocated to vm_i. If the total load is less than U_l (l means lowload), we then publish a new VM to markets. Second, if it receives an overload signal, then it will take mitigation measures. We steal some capacity form the lightest VM to the overloaded VM. The stolen cap amount depends on real-time situations. To make performance smooth and ensure user experience, we set is as small as one-fifth of the unused cap of the lightest VM. The lightest VM is defined by comprehensive information analysis on all VMs, taking care of both vGPU utilization and FPS, whose weights to measure overload degree is set as w_u and w_e respectively. The detailed implementation of mitigation method is shown in Algorithm 1. Note that if the load on one machine is too heavy, more than one virtual guest send the overload signal, it should be a severe warning. Because there are hundreds of thousands of physical machines in datacenters, VM migration can be done to handle this critical situation.

The second point we propose is the elimination method of performance interference among oversubscribed VMs. As mentioned above, we use the cap to restrict the share limitation of the GPU resource which the guest could be allocated in a scheduling period. Our cap means the resource should occupy even if the guest is idle in the current period. By setting the cap of each VM appropriately, we can completely achieve isolation of resources among VMs and then avoid disorganized contention. The capacity-restricted scheduling is based on the default round-robin strategy. For the detailed implementation of capacity, we add more members to schedule info structure in QEMU and Xen as Listing 1 to become a clear framework that can adapt to the dynamic requirement. When a VM is initialized, the scheduler gets its initial cap from the driver interface in the file system, where each created VM can be queried for its configuration parameters including the capacity, then store it in the schedule managing information. The initial cap is a threshold reference of the dynamic cap. Every time we write the cap value in the scheduled info, its GPU share capacity configuration will be affected in the next scheduling interval. The mechanism of resetting schedule info also enables that the capacity of a VM can be changed even if the VM is running. Then the target of the oversubscription module can be realized that guaranteeing the performance requirement of VMs with the reasonable amount of resources as little as possible.

Algorithm 1. The Overload Mitigation Procedure

```
Input: $u_i$, $e_i$ of all $vm_i$; OverSignal;
 1: if (OverSignal == null) {
 2:   releaseNewVM();
 3:   return;
 4: }
 5: if (OverSignal == warning) {
 6:   vmMigration();
 7:   return;
 8: }
 9: struct vGPU_sche_info *lightest_vgpu;
10: int lightest = $MAX\_INT$;
11: for $vm_i$ in VMs {
12:   int factor = $w_u * u_i + w_e * e_i$;
13:   if (factor $<$ lightest)
14:     lightest_vgpu = $vm_i$;
15: }
16: stolen_cap = 0.2*lightest_vgpu → unused_cap;
17: lightest_vgpu → cap -= stolen_cap;
18: OverSignal.vgpu → cap += stolen_cap;
```

3.4 Scheduling Module

The scheduling module directly conducts time slice allocation and context switch among vGPU. All the behaviors happen in the kernel mode, so we modify the loaded GPU driver functions which implement these features in the Linux kernel. We first read the initial GPU cap value of the VM from its configuration file and initialize the vGPU schedule info including the initial cap. Then we add an interface to set a timer to regularly change the vGPU schedule information in memory according to commands from the oversubscription module. Through this interface, the scheduling process can then update the max timeslice in the next period so that the kernel can respond to dynamic oversubscription demands. The time interval should be less than the scheduling period to respond quickly and be well adjusted considering overhead. To make performance smooth during the second time slot, we divide 1 s into 10 slot that everyone has 100 ms as shown in Fig. 4. Then we add a function as in Algorithm 2 executed every 100 ms to allocate timeslice for next 100 ms. The dedicate time slice for vGPU in every 100 ms will also plus the remaining time slice from last 100 ms to make more fairness. To ensure that the schedule time residue does not

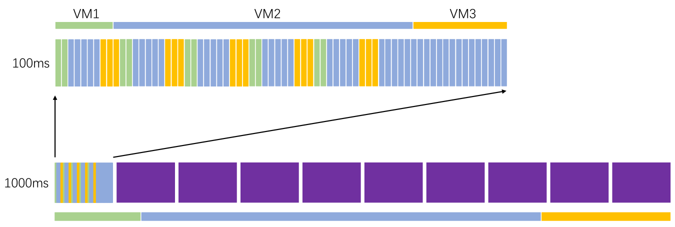


Fig. 4. The scheduling period design.

accumulate so much that it messes the scheduling, the schedule info of VMs will be reset according to capacity every 1 s, as the target is accurate at the one-second level, no need further incremental. Our algorithm about when to pick up next vGPU is the same to original logic. In the scheduling logic, the triggered scheduler selects the first VM that still has allocated time slice left as the activating target. And the skipped VM should wait for the next 100 ms scheduling cycle to be served.

Algorithm 2. The Dynamic Timeslice Allocation Procedure

```
Input: struct vGPU_list vgpu_queue; SCHED_PERIOD = 1 s;
       SMOOTH_PERIOD = 100 ms;
 1: static u64 stage_check;
 2: int stage = stage_check++%10;
 3: for vgt in vgpu_queue {
 4:   if (stage == 0) {
 5:     vgt → max_timeslice = SCHED_PERIOD ∗ vgt→cap;
 6:     vgt → timeslice = SMOOTH_PERIOD ∗ vgt→cap;
 7:   } else {
 8:     vgt → timeslice += SMOOTH_PERIOD ∗ vgt→cap;
 9:   }
10: }
```

Algorithm 3. The Next vGPU Pickup

```
Input: struct vGPU_sche_info *cur_vgpu
 1: struct vGPU_sche_info *next_vgpu = cur_vgpu;
 2: do {
 3:   next_vgpu = next_vgpu → next;
 4:   if (next_vgpu → timeslice > 0) {
 5:     break;
 6:   }
 7: } while (next_vgpu ! = cur_vgt);
 8: if (cur_vgpu → cap > 0 && next_vgpu == cur_vgpu) {
 9:   return dom0;
10: }
11: return next_vgpu
```

4 Evaluation

We then evaluate the gOver solution to verify the function of our GPU virtualization and the performance of our oversubscription method in this section. First, we list the system configurations about the hardware and software adopted in the following experiments. Second, we analyze the typical characteristics of GPU workloads to illustrate the reason why it is necessary to introduce our gOver strategy. After that, the proposed experiments compare our dynamic GPU oversubscription with non-oversubscribed GPU virtualization and static

TABLE 1
System Configurations

	Mainboard Platform	Configuration
Hardware	Platform	Intel NUC Kit NUC5i5MYHE
	CPU	Intel Core i5-5300 U 2.30 GHz
	Memory RAM	16GB
	Graphics Card	Intel HD Graphics 5500
Software	Platform	Xen 4.3.0
	Kernel	Linux 4.3.0
	Operating System	Ubuntu 16.04 LTS
VM	Memory RAM	2 GB
	Kernel	Linux 4.3.0
	Operating System	Ubuntu 16.04 LTS

oversubscription as follows. Unlike these two approaches, the modified GPU virtualization with dynamic oversubscription emphasizes a tradeoff between benefits and quality of services in a shared system. Our strategy has better economic efficiency and QoS guarantee under the dynamic control.

4.1 Experimental Setup

Configurations are all listed in Table 1. We use the Intel NUC Kit NUC5i5MYHE with extra AGP aperture as the basic platform for experiments. The CPU processor is the 5th generation Intel Core processor i5-5300 U 2.30 GHz based on the Broadwell architecture with the integration of Intel HD Graphics 5500. The memory ram is 16 GB. Our enhanced version is developed over the Intel vGPU solution GVT-g 2016q4 in Xen 4.3.0. Also, newer Xen versions are also acceptable for our mechanism after we patch them with the GVT-g modification. The kernel part of this project is forked from Linux kernel v4.3.0 to support the GPU virtualization. We install Ubuntu 16.04 LTS as the operation system of hypervisor for virtualization system management. We choose this long-term support version for its stability, although newer OS version can hardly affect the evaluation behavior. Each guest is configured with 2 GB memory and the same Ubuntu system. In this GPU virtualization solution, the device emulation environment is QEMU-based, which provides the device model and the display console.

Benchmarks are chosen as glxgears [37], glmark2 [38] and plot3D [39] under GVT-g, which is an open-sourced GPU virtualization platform enabling modification for supporting fine-grained context scheduling and monitoring. Only in such GPU virtualization implementation that our gOver strategy can be precisely evaluated and sampled. The three benchmarks represent different GPU workloads in cloud computing. The first benchmark display continuous scene with three rotating gears and no other changes. It is an ultra-lightweight test program. If both the graphical card and driver work well, the gears will rotate fast. The second 3D benchmark compares the different features and extensions of OpenGL by rendering more than a dozen scenes and output undulating FPS values. It has an unstable performance with irregular loads of heavy and light tasks. The third workload renders a revolving 3D terrain shape through an overlook perspective. This workload acts as a periodic task with a regular pattern. The graphics display window is scalable. To ensure that the window size does

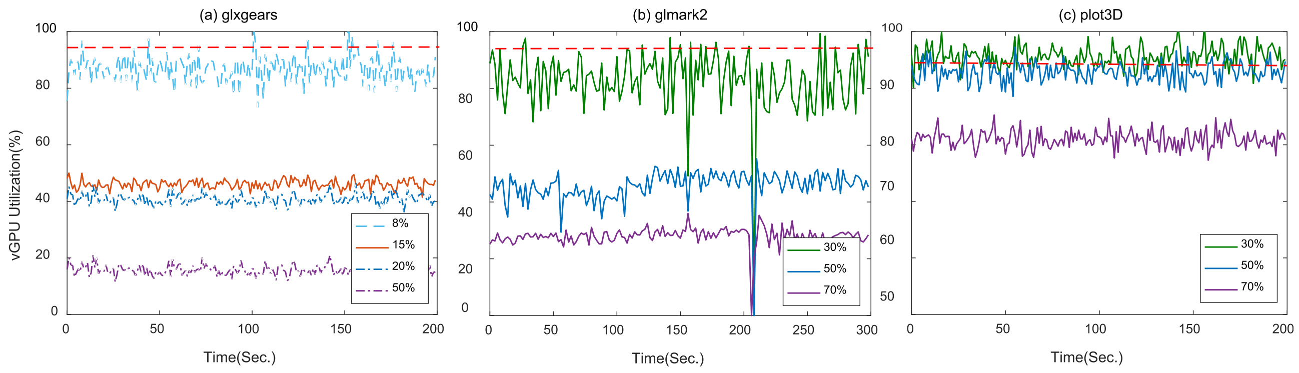


Fig. 5. vGPU resource share of a) glxgears, b) glmark2 and c) plot3D under different capacity constraints. The red dotted line indicates the vGPU overload.

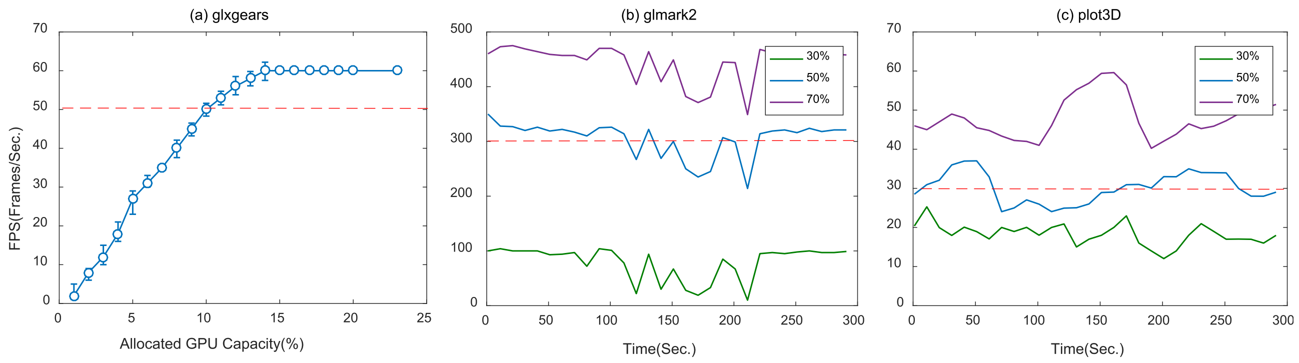


Fig. 6. Workload FPS of a) glxgears, b) glmark2 and c) plot3D under different capacity constraints. The red dotted line indicates the FPS demands of three benchmarks.

not affect FPS, we take $960 * 720$ resolution as the default option. These mentioned benchmark workloads are chosen under the limitation of Intel GPU virtualization platform on performance and interface support. As a result, there are only a limited group of workloads with appropriate complexity which can be selected.

Methodology is designed from the unpredictable point of cloud workloads. We have implemented a testing mechanism that continues submitting random tasks to each idle VM as an interference effect. In such a situation, all the VMs contend with each other for the resource of physical GPU. And the scheduling strategy ensure that task processing procedures of all the VMs are concurrent. We test the average GPU utilization and service QoS under three virtualization solutions, the original gVirt, our gOver modification, and the static oversubscription. During running time, we collect the vGPU utilization and QoS values for analysis on economic efficiency. Apart from the initial cap definition, the other configuration parameters are all set the same in all the following evaluations. Also, to avoid the initialization disturbance may happen at the beginning period, all the testing workloads are repeated three times in the execution. Then the evaluation results illustrated below are the average record of multiple repetitions. In addition, other prior solutions about GPU scheduling can hardly be compared with our mechanism in a normalized situation because these solutions were usually constructed in a non-virtualized environment, or realized virtualization through API forwarding on closed-sourced platform.

4.2 Workload Property Analysis

Before the experiment properly starts, we need a detailed analysis of the properties of possible workloads to prove the necessity of adopting oversubscription in GPU virtualization. We only create one virtual machine on the physical server and allocate a series of GPU capacity values to the VM to run benchmarks, meanwhile, we record the vGPU utilization and FPS every second. Figs. 5 and 6 show the changes of vGPU utilization and FPS in different GPU capacities respectively. From the data analysis of three representative GPU benchmarks, we can easily find that different workloads are significantly different in resource requirement to reach the same QoS level and the QoS exceeds their necessity in most case.

First we come to the vGPU utilization curve of glxgears in constant GPU capacity levels of 8%, 15%, 20%, 50% as depicted in Fig. 5a. In the graph, the X-axis refers to the runtime time point record, and the Y-axis refers to the corresponding utilization record of the vGPU status. Obviously, there is an inverse correlation relation between the vGPU utilization and GPU capacity of the VM. The larger GPU capacity, the lower vGPU utilization. When we allocate half of the physical GPU resource to run glxgears, its utilization even cannot reach 20% all the time. That means it does not take that many resources to render graphics. We then come to the FPS curve of glxgears in Fig. 6a. In this figure, the X-axis refers to the vGPU capacity allocation recorded as percentage, and the Y-axis refers to the relevant FPS record. We have conducted a series of capacity values to observe the FPS changes along with time. The broken line in this figure refers to the average FPS level under different capacity constraints. In addition, the error bar refers to the recorded upper and lower bound of FPS value in the whole execution period. As the error bars show, it seems that the FPS record still has vibrations even if the GPU capacity is a constant configuration.

According to the figure, the relationship between the capacity allocation and the workload FPS follows a high-fitting linear relation if the capacity share is under 15%, which is an appropriate configuration range. However, workload FPS becomes stable when capacity exceeds 15%, which is another piece of evidence for the fact that glxgears are a low-footprint application. If GPU virtualization is in the form of average resources allocation among VMs, the VM running glxgears tasks will waste a lot of GPU time slices.

And the second workload example is glmark2. The Fig. 5b illustrates the vGPU utilization record under 3 constant GPU capacity configuration levels, that is, 30%, 50% and 70%. The vGPU utilization is at a good level when the allocated capacity is more than 50%. Form the curve of 30%, we can find that the vGPU utilization is generally higher than 80% and even exceeds 95% several times which is a vGPU overload signal. At the same time, according to the corresponding runtime FPS in Fig. 6b, the FPS is obviously vibrating and always lower than 100 which is far away from the standard value of glmark2. So the demand for GPU resources of glmark2 is at a relatively high level than glxgears. That is also the reason why fair GPU allocation is detrimental for heavy tasks in cloud computing. We can see in the 50% situation in Fig. 6b, the average FPS is 304.9, but the vibration can be from 214 to 350. It violates the service quality requirement for quite a while where should be provisionally allocated more than 50% capacity. That is the reason why our gOver dynamic oversubscription is necessary for GPU virtualization.

Also, compared with the vGPU utilization of plot3D in Fig. 5c, we can see that for the same 50% capacity, the glxgears uses only 20% and the glmark2 use less than 50%, but plot3D uses more than 90% vGPU resources which is a serious danger sign. Also, under the same QoS level that the FPS target is 30, we know that the workload plot3D requires about 50% GPU share from the data in Fig. 6c, which is a required value about 7 times comparing to the glxgears workload. Supposing that we allocate the same share of GPU resource to the three workloads mentioned above, the plot3D never reaches the pleasant QoS target. However, the workload glxgears usually occupies more resource than enough. This is a clear evidence of the necessity to the gOver.

4.3 Performance Analysis

We then focus on the experiment results about our gOver GPU capacity allocation strategy with dynamic oversubscription reflected from the graphs and the descriptions in this section. After that, the improvement compared to the resource scheduling strategies based on traditional GPU virtualization with the static GPU oversubscription should be discussed. Based on the illustrated records, the necessity and the advantage of our solution with dynamic oversubscription can be solidly reflected.

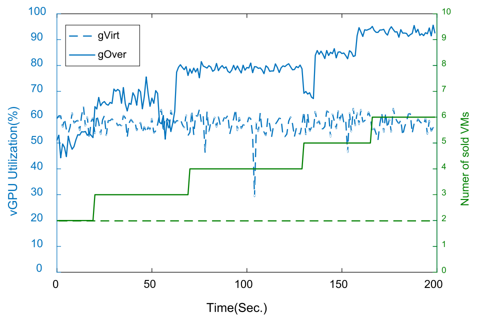


Fig. 7. The comparison on vGPU utilization between gVirt (traditional allocation) and gOver.

4.3.1 Comparison With gVirt

The traditional gVirt solution schedules the vGPU contexts under a fair strategy, where each context get the same resource allocation share. First, we can have a glance at Fig. 7, which illustrates the total physical GPU utilization variations in a time period under our gOver system and traditional gVirt solution. Based on the gVirt kernel system, we generate three virtual machines and randomly assign one workload from benchmarks mentioned above to each of them. In this scenario, we use the Intel GPU tools [40] to select the total physical GPU utilization on the hypervisor. From the gVirt line in Fig. 7, we can see that although the utilization is relatively flat, the overall level is lower than 60%, which indicates that the physical GPU is not fully leveraged. The random workloads assignment conforms to the real cloud environment. The average GPU utilization of three workloads is 57.4%, which indicates GPU is idle and wasted for nearly half the time. Based on our gOver modification to gVirt, we generate two virtual machines with a 50% capacity at first and then conduct monitoring and adaptive overselling. From the gOver line, we know that the overselling behavior happens at about 20 s, 70 s, 140 s, and 160 s. The co-hosting VMs in the physical server is growing from 2 to 6 at the end. From the GPU utilization varying under gOver system, we can find that it is lower than gVirt at the beginning, but soon rise to 60%, then continue to climb along with the increased overbooking ratio, and achieve about 95% at last. The average GPU utilization is about 76.4% under gOver system, achieving an improvement of 20 percent. The overbooking ratio means the over published GPU capacities over 100%, to be specific, the ratio 3 scenario represents we publish 300% GPU capacity with 6 VMs. In our gOver system, the GPU overbooking ratio increases from 1 to 3 and the income surpasses traditional gVirt solution by nearly 20%.

4.3.2 Comparison With Static Oversubscription

The offline static oversubscription method usually sets the overbooking ration based on experience before the physical server releases. We conduct four groups of experiments on the overbooking ratio at 1.5, 2, 3 and 4. And then we assign random workloads to run under these four ratios. We calculate the average GPU utilization and QoS broken time proportion in 5 minutes which is enough to represent the overall result. From the experimental results showed in Fig. 8, we see that the GPU utilization is unsatisfactory when overbooking

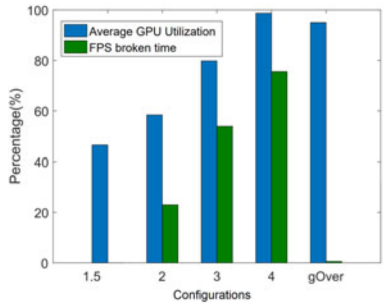


Fig. 8. Average GPU Utilization and FPS broken time under different static oversubscription ratios and gOver stragety.

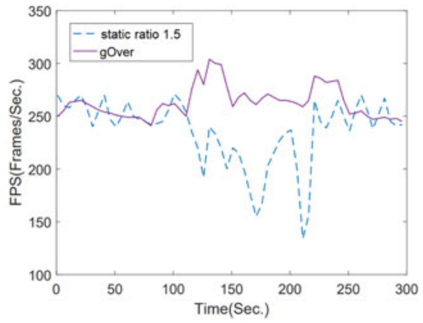


Fig. 9. The FPS variation in comparison of gOver and 1.5 overbooking ratio.

ration is lower than 3, but when we set a higher ratio above 3, it seems that the QoS guarantee during more than 50% of the five-minute evaluation time may be broken, which means that the service QoS level is severely failed. Since the vGPU resource utilization has reached a considerable level under 3 and 4 overselling ratios, the QoS contravention possibility is unacceptable for the cloud service providers, because the possible oversell profit can hardly cover the contract violation compensation claimed by the customers. By our overload detection and immediate mitigation to do adaptive oversubscription in GPU driver, we can both achieve high-level vGPU resource utilization up to 95% and keep the FPS value in control with a broken time less than 1%, which is imperceptible to customers. We select the FPS curve of glmark2 under our gOver system and the overbooking ratio of 1.5 which is seemly optimal to be a static setting in Fig. 9. The glmark2 is allocated 33% GPU resources under overselling factor of 1.5. In our gOver solution, the GPU capacity allocated to glmark2 is always changing along with the workloads environments on the physical server. From the curve of over ratio 1.5, we find out the result that the recorded execution FPS of workload glmark2 declines during the period between 110 s an 230 s. This reflects the fact that the per-frame rendering tasks in this time period require more vGPU resource than the allocated 33% vGPU share. Although oversubscription is exploited to earn more economic interests, the static overselling ratio is inflexible to the changeful cloud workload. So our adaptive overselling strategy results in good performance with well-pleasing FPS hold as the gOver curve shows. Obviously, this evaluation results illustrate that this system can cope with significant workload behavior change in the whole time period. Since our gOver solution provides dynamic GPU resource allocation adjustment during the execution, the resource utilization can be improved and the user QoS level can be guaranteed according to the optimization target.

4.4 Analysis on Economic Efficiency and QoS

Our next experiment compares the economic efficiency and QoS of the light-fit, heavy-fit and random workloads placement at the physical server of our gOver system. The light-fit placement allocates glxgears on all published VMs and the heavy-fit placement allocates glmark2 on all published VMs. The overselling number of VMs is up to 6 when running lightweight tasks at all VMs, but there is no overselling behavior happening when running heavyweight tasks on VMs. So the economic efficiency of the light-fit case can achieve six times over heavy-fit case. By building our gOver kernel system, we can get very high economic efficiency within one physical server when there are light workloads in the cloud environment. But in reality, there are always co-hosting heavy and light VMs on a physical machine. The cloud workloads are fluctuating along with time as random workload placement method. We conduct five times of random benchmark allocation experiments and get the average overselling number of VMs is also up to 4, which is a considerable benefit over non-oversubscribed GPU virtualization solution.

5 Discussion

Since the GPU efficiency can have significant improvement in the situation that multiple workloads running on one GPU with the proposed gOver solution, it may be difficult to estimate precisely the actual average performance benefit that a cloud platform with our strategy applied. The fact is that even the same benchmark with slightly modified parameters may result in different practical evaluation data record. To ensure effectiveness and safety, some parameters of the overload mitigation procedure need to be adjusted according to a real cloud environment. Our experiments are based on one physical GPU which can only accommodate very few VMs and cannot conduct VM migration to handle overload emergency. For plenty of servers in the cloud, gOver can perform better due to more resource coordination. With the help of oversubscription, the resource allocation efficiency of the virtualized GPU can have a significant improvement in the cloud.

In this paper, our introduced benchmarks are all graphic workloads and then the evaluations are all processed with the scheduling in graphic GPU mode. However, the Intel GPU supports the GPGPU functionality interfaces such as OpenCL to act as a GPGPU device. In this scenario, there is no change on the behavior of the GVT-g GPU virtualization. That is to say, our proposed gOver mechanism allows the GPGPU workload scheduling and performance optimization if necessary. But the FPS can no longer be used as the performance quality metric for monitoring in this case. In this situation, processed unit task count per interval should be a better choice for performance metric.

Currently, only the Intel graphics processor platform can be supported by the proposed gOver implementation. Other industrial GPU vendors include Nvidia and AMD never publish the open-source version of their GPU driver software. As a result, we cannot modify the proposed gOver implementation for the compatibility on such GPU platforms. However, the principle and the concept which construct our design are applicable even if the architectures are different. It is obvious that the concept of GPU oversubscription can always be practical in improving scheduling efficiency among multiple VM contexts. Since we just aim at virtual server GPU resources in this paper, it can be a general scheduling strategy among various accelerator computation platforms such as GPGPUs and AI-specific chips. Actually, if some loss on the controlling granularity is tolerable, it is able to migrate and deploy our mechanism to the platforms provided by other vendors with no support of open-source device driver. In the platforms from the vendors like NVIDIA and AMD, we can still construct similar scheduling system with user-space function-library level control under the task level granularity.

Also, according to our knowledge, currently vendors like NVIDIA provide GPU virtualization functionalities, but the available resource scheduling policies in such virtualization still remain trivial, including FIFO and round-robin, and all the resource share configurations are statically fixed. Obviously, with the development of GPU performance, the significance of gOver method will become more and more important for general-purpose applications.

6 Related Work

To support multi-user sharing in business applications, many pieces of research have been conducted to realize GPU resource virtualization and improve its performance. The Intel GPU virtualization solution offers the virtual GPU contexts to the virtual machines with GPU resource requirement by employing mediated pass-through technology, where VMs can directly access hardware source in performance critical operations while being trapped and emulated by hypervisor when executing privileged instructions [31]. Next, the Ballooning technique [41] and the gScale [42] are the examples which improve the GPU virtualization performance by memory optimizations. Along with these researches, [43], [44], [45] are examples of other GPU resource allocation optimization works. Even though the GPU virtualization technology has already got some practical design and come to mature implementation in some area, reaching a high-level virtual GPU resource utilization is still challenging in the cloud for customer's overestimation of their demands. There are many mature researches in the area of oversubscribed cloud computing about memory [3], CPU [4], [15], bandwidth [5], [6], storage [15] and power [7]. However, there appears to be a gap in the study on GPU oversubscription due to the complex scheduling of virtualized GPU.

Compared with the similar researches recently proposed on GPU scheduling, resource oversubscription and GPU resource virtualization, the strategy proposed in our research aims at the effect on economic profit and resource utilization performance when introducing oversubscription schema into GPU resource scheduling. Also, in our research, the scheduled GPU resources are virtualized and shared among multiple virtual machine instances. With the help of the open-sourced Intel platform, our scheduling strategy can dynamically control the GPU device in a fine-grained time-slot level. The mentioned studies above for GPU resource scheduling usually pay more attention to the non-virtualized task-level GPU scheduling, or implement the virtual GPU controlling on the function library level in a solution based on API forwarding design. These projects cannot offer the virtual GPU controlling in the time-slice granularity. Also, the oversubscription schema that our research introduces takes the economic profit of service provider as the target while considering resource utilization and user QoS improvement when designing the optimization. This optimization target may have some differences from other researches.

7 Conclusion

In this paper, we have developed an adaptive GPU oversubscription technology gOver according to the open-source Intel GPU virtualization solution. This technology applies to the datacenters which provide virtualized GPU computing service. We introduce three collaborated modules to fulfill dynamic capacity allocation under GPU oversubscription. First, the monitoring module is in charge of calculating real-time vGPU load conditions. Then, the oversubscription module decides whether to oversell new VMs based on monitoring data. It also takes timely measures to mitigate overload by adjusting GPU allocation among VMs. We design and implement a controller to reset the cap authorized to each VM according to mitigation measures. At last, the static time-based scheduling is upgraded with adaptive capacity allocation according to real-time vGPU utilization and QoS of VMs. The maximum GPU resource authorized to each VM will be calculated at the beginning of every scheduling period. This adaptive and dynamic oversubscription schema satisfies interests of cloud providers and provides QoS guarantee for customers at the same time. The evaluation shows that our gOver performs 20% improvement on economic efficiency compared to traditional GPU virtualization and much better QoS control over static oversubscription method.

Acknowledgments

We would like to thank the anonymous reviewers for their constructive comments and feedback that greatly helped us improve the final paper.

References

[1] F. Lombardi and R. Di Pietro, "Secure virtualization for cloud computing," *J. Netw. Comput. Appl.*, vol. 34, pp. 1113–1122, 2011.
[2] L. Tomás and J. Tordsson, "Improving cloud infrastructure utilization through overbooking," in *Proc. 2013 ACM Cloud Autonomic Comput. Conf.*, 2013, Art. no. 5.
[3] M. R. Hines, A. Gordon, M. Silva, D. Da Silva, K. Ryu, and M. Ben-Yehuda, "Applications know best: Performance-driven memory overcommit with ginkgo," in *Proc. IEEE 3rd Int. Conf. Cloud Comput. Technol. Sci.*, 2011, pp. 130–137.
[4] X. Zhang, Z.-Y. Shae, S. Zheng, and H. Jamjoom, "Virtual machine migration in an over-committed cloud," in *Proc. IEEE Netw. Operations Manage. Symp.*, 2012, pp. 196–203.
[5] N. Jain, I. Menache, J. S. Naor, and F. B. Shepherd, "Topology-aware VM migration in bandwidth oversubscribed datacenter networks," in *Proc. Int. Colloq. Automata Lang. Programm.*, 2012, pp. 586–597.
[6] D. Breitgand and A. Epstein, "Improving consolidation of virtual machines with risk-aware bandwidth oversubscription in compute clouds," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2012, pp. 2861–2865.

[7] X. Fu, X. Wang, and C. Lefurgy, “How much power oversubscription is safe and allowed in data centers,” in *Proc. 8th ACM Int. Conf. Autonomic Comput.*, 2011, pp. 21–30.

[8] Amazon EC2 elastic GPU. 2016. [Online]. Available: https://amazonaws-china.com/cn/ec2/elastic-gpus/

[9] Alibaba cloud elastic GPU service. 2016. [Online]. Available: https://www.alibabacloud.com/product/gpu

[10] S. A. Baset, L. Wang, and C. Tang, “Towards an understanding of oversubscription in cloud,” in *Proc. 2nd USENIX Workshop Hot Topics Manage. Internet, Cloud, Enterprise Netw. Serv.*, 2012, Art. no. 7.

[11] H. Huang et al., “Towards exploiting CPU elasticity via efficient thread oversubscription,” in *Proc. 30th Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2021, pp. 215–226. [Online]. Available: https://doi.org/10.1145/3431379.3460641

[12] I. S. Moreno and J. Xu, “Neural network-based overallocation for improved energy-efficiency in real-time cloud environments,” in *Proc. IEEE 15th Int. Symp. Object/Component/Service-Oriented Real-Time Distrib. Comput.*, 2012, pp. 119–126.

[13] H. Zhou, J. Yao, H. Guan, and X. Liu, “Comprehensive understanding of operation cost reduction using energy storage for IDCs,” in *Proc. IEEE Int. Conf. Comput. Commun.*, 2015, pp. 2623–2631.

[14] I. S. Moreno and J. Xu, “Customer-aware resource overallocation to improve energy efficiency in realtime cloud computing data centers,” in *Proc. IEEE Int. Conf. Service-Oriented Comput. Appl.*, 2011, pp. 1–8.

[15] R. Yang et al., “Performance-aware speculative resource oversubscription for large-scale clusters,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 7, pp. 1499–1517, Jul. 2020.

[16] F. Bellard, “QEMU, a fast and portable dynamic translator,” in *Proc. USENIX Annu. Tech. Conf.*, 2005, Art. no. 41.

[17] J. Duato, A. J. Pena, F. Silla, R. Mayo, and E. S. Quintana-Ortí, “rCUDA: Reducing the number of GPU-based accelerators in high performance clusters,” in *Proc. Int. Conf. High Perform. Comput. Simul.*, 2010, pp. 224–231.

[18] L. Shi, H. Chen, and J. Sun, “vCUDA: GPU accelerated high performance computing in virtual machines,” in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2009, pp. 1–11.

[19] C. R. F. Silla, S. Iserte, and J. Prades, “On the benefits of the remote GPU virtualization mechanism: The rCUDA case,” *Concurrency Comput.: Pract. Exp.*, vol. 29, no. 13, pp. 1–29, 2017.

[20] C. R. S. Iserte, J. Prades, and F. Silla, “Transparent I/O-aware GPU virtualization for efficient resource consolidation,” in *Proc. 16th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2016, pp. 131–140.

[21] V. Gupta et al., “GViM: GPU-accelerated virtual machines,” in *Proc. 3rd ACM Workshop Syst.-Level Virtualization High Perform. Comput.*, 2009, pp. 17–24.

[22] C. H. Hong, I. Spence, and D. S. Nikolopoulos, “FairGV: Fair and fast GPU virtualization,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3472–3485, Dec. 2017.

[23] H. Yu, A. M. Peters, A. Akshintala, and C. J. Rossbach, “AvA: Accelerated virtualization of accelerators,” in *Proc. 25th Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, 2020, pp. 807–825. [Online]. Available: https://doi.org/10.1145/3373376.3378466

[24] C. Smowton, “Secure 3D graphics for virtual machines,” in *Proc. 2nd Eur. Workshop Syst. Secur.*, 2009, pp. 36–43.

[25] H. A. Lagar-Cavilla, N. Tolia, M. Satyanarayanan, and E. De Lara, “VMM-independent graphics acceleration,” in *Proc. 3rd Int. Conf. Virtual Execution Environ.*, 2007, pp. 33–43.

[26] M. Dowty and J. Sugerman, “GPU virtualization on vmware’s hosted I/O architecture,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 43, pp. 73–82, 2009.

[27] N. M. Gonzalez and T. Elengikal, “Transparent I/O-aware GPU virtualization for efficient resource consolidation,” in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2021, pp. 131–140.

[28] D. Abramson et al., “Intel virtualization technology for directed I/O,” *Intel Technol. J.*, vol. 10, no. 3, pp.179–192, 2006.

[29] Y. Dong, J. Dai, Z. Huang, H. Guan, K. Tian, and Y. Jiang, “Towards high-quality I/O virtualization,” in *Proc. Israeli Exp. Syst. Conf.*, 2009, Art. no. 12.

[30] L. Xia, J. Lange, P. Dinda, and C. Bae, “Investigating virtual passthrough I/O on commodity devices,” *ACM SIGOPS Operating Syst. Rev.*, vol. 43, pp. 83–94, 2009.

[31] K. Tian, Y. Dong, and D. Cowperthwaite, “A full GPU virtualization solution with mediated pass-through,” in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2014, pp. 121–132.

[32] Igvtg-kernel. 2016. [Online]. Available: https://github.com/intel/Igvtg-kernel/tree/2016q4–4.3.0

[33] Q. Lu et al., “gMig: Efficient vGPU live migration with overlapped software-based dirty page verification,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 5, pp. 1209–1222, May 2020.

[34] Q. Lu, J. Yao, H. Guan, and P. Gao, “gQoS: A QoS-oriented GPU virtualization with adaptive capacity sharing,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 843–855, Apr. 2020.

[35] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui, “Cloud gaming: Architecture and performance,” *IEEE Netw.*, vol. 27, no. 4, pp. 16–21, Jul./Aug. 2013.

[36] S. Wang and S. Dey, “Rendering adaptation to address communication and computation constraints in cloud mobile gaming,” in *Proc. IEEE Glob. Telecommun. Conf.*, 2010, pp. 1–6.

[37] glxgears, 2019. [Online]. Available: https://github.com/mattn/go-glxgears

[38] glmark2, 2020. [Online]. Available: https://github.com/glmark2/glmark2

[39] plot3d, 2019. [Online]. Available: http://www.geeks3d.com/gputest/

[40] intelgputools, 2020. [Online]. Available: https://github.com/mkuoppal/intel-gpu-tools

[41] Y. Dong, M. Xue, X. Zheng, J. Wang, Z. Qi, and H. Guan, “Boosting GPU virtualization performance with hybrid shadow page tables,” in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2015, pp. 517–528.

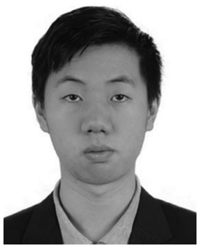
[42] M. Xue et al., “gScale: Scaling up GPU virtualization with dynamic sharing of graphics memory space,” in *Proc. USENIX Conf. Annu. Tech. Conf.*, 2016, pp. 579–590.

[43] J. Yao, Q. Lu, H.-A. Jacobsen, and H. Guan, “Robust multi-resource allocation with demand uncertainties in cloud scheduler,” in *Proc. IEEE 36th Symp. Reliable Distrib. Syst.*, 2017, pp. 34–43.

[44] 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 *Proc. 22nd Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, 2017, pp. 17–32.

[45] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo, “Quality of service support for fine-grained sharing on GPUs,” in *Proc. 44th Annu. Int. Symp. Comput. Archit.*, 2017, pp. 269–281.

Jianguo Yao (Senior Member, IEEE) received the BE, ME and the PhD degrees from the Northwestern Polytechnical University (NPU), Xian, Shaanxi, China, in 2000, 2007, and 2010, respectively. Currently, he is a professor and associate dean of the School of Software, Shanghai Jiao Tong University, and the director of SJTU-Enflame Joint Lab. His research interests are cloud computing, virtualization, and industrial Big Data.



Qiumin Lu received the ME degree from Jiao Tong University, Shanghai, China, in 2017. Currently, he is working toward the graduate degree with the Shanghai Key Laboratory of Scalable Computing and Systems, School of Software, Shanghai Jiao Tong University. His research interests mainly include GPU virtualization, feedback control applications, and concurrent programming.

Run Tian received the ME degree from Jiao Tong University, Shanghai, China, in 2019. She is currently working toward the graduate degree with the Shanghai Key Laboratory of Scalable Computing and Systems, School of Software, Shanghai Jiao Tong University. Her research interests mainly include GPU virtualization and oversubscription resource scheduling.

Keqin Li (Fellow, IEEE) is a SUNY distinguished professor of computer science with the State University of New York. He is also a National distinguished professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, Big Data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored more than 860 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently an associate editor of the *ACM Computing Surveys* and the *CCF Transactions on High Performance Computing*. He is a Member of Academia Europaea (Academician of the Academy of Europe).

Haibing Guan received the PhD degree from Tongji University, in 1999. He is a professor of the School of Electronic, Information and Electronic Engineering, Shanghai Jiao Tong University, and the director of the Shanghai Key Laboratory of Scalable Computing and Systems. His research interests include distributed computing, network security, network storage, green IT, and cloud computing.