Future Generation Computer Systems 63 (2016) 25–36

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

A user mode CPU–GPU scheduling framework for hybrid workloads

Bin Wang, Ruhui Ma, Zhengwei Qi *, Jianguo Yao, Haibing Guan

School of Software, Shanghai Jiao Tong University, Shanghai, China

HIGHLIGHTS

- We propose a user mode framework for CPU-GPU hybrid resource management.
- We analysis the drawbacks of the existing GPU and CPU scheduling systems.
- vHybrid is integrated by one GPU algorithm and two algorithms for CPU scheduling.
- The result outperforms existing SLA-aware GPU and CPU scheduling systems.

ARTICLE INFO

Article history:
Received 17 July 2015
Received in revised form
25 January 2016
Accepted 16 March 2016
Available online 11 April 2016

Keywords:
Hybrid workloads
CPU–GPU resource management
Scheduling
Switched control

ABSTRACT

Cloud platforms composed of multi-core CPU and many-core Graphics Processing Unit (GPU) have become powerful platforms to host incremental CPU–GPU workloads. In this paper, we study the problem of optimizing the CPU resource management while keeping the quality of service (QoS) of games. To this end, we propose vHybrid, a lightweight user mode runtime framework, in which we integrate a scheduling algorithm for GPU and two algorithms for CPU to efficiently utilize CPU resources with the control accuracy of QoS. vHybrid can maintain the desired QoS with low CPU utilization, while being able to guarantee better QoS performance with little overhead. Our evaluations show that vHybrid saves 37.29% of CPU utilization with satisfactory QoS for hybrid workloads, and reduces three orders of magnitude for QoS fluctuations, without any impact on GPU workloads.

1. Introduction

There is a virtually unanimous consensus in the community that Graphics Processing Unit (GPU) is one of the most important resources for cloud computing. This is because many of today's cloud graphic applications, such as cloud gaming, video rendering, and social networking, have very demanding GPU requirements, and poor GPU scheduling can directly affect application performance and degrade user experience. To provide adequate GPU resources, GPU virtualization has been widely studied. For example, the work in [1–7] leveraged GPU virtualization for general purpose computing on GPUs. In our previous work, VGRIS and vGASA [8,9], achieved a Service Level Agreement (SLA)-aware GPU resource scheduling policy. They accelerated the running times of GPU-intensive workloads and guarantee the Quality of Service (QoS) in the cloud gaming platform.

However, only the GPU scheduling mechanism is not sufficient to effectively enhance the overall computing resource utilization in a CPU–GPU hybrid platform, where GPU-intensive and CPU-intensive workloads coexist. For example, in Amazon EC2 GPU instances [10], each Virtual Machine (VM) can either run the graphic rendering applications that need GPU resources (e.g., cloud gaming), or have the CPU-intensive workloads consume the CPU resources. In addition, cloud users need to make sure their data remain intact after uploading to the remote server. Thus, a physical server runs client applications such as Apache [11]. They also require high computation in the CPU. As a result, a strategy to allocate fair-share CPU resources among CPU workloads while guaranteeing the QoS of GPU computation is deemed as a crucial demand. Unfortunately, neither CPU nor GPU resources are efficiently utilized in the cloud. One likely reason is that the workloads running in the virtualized environment are significantly affected by other workloads. The default scheduling mechanisms are usually designed and embedded by the hardware vendors who do not consider the use of cloud computing in their design nor productions. Therefore, the scheduling policies would allocate excessive CPU and GPU resources for a workload, but starve the other. On the other hand, most of the previous work only considers proposals for the scheduling policy of CPU or GPU individually. For

* Corresponding author.
E-mail addresses: binqbu2002@sjtu.edu.cn (B. Wang), ruhuima@sjtu.edu.cn (R. Ma), qizhwei@sjtu.edu.cn (Z. Qi), jianguo.yao@sjtu.edu.cn (J. Yao), hbguan@sjtu.edu.cn (H. Guan).

http://dx.doi.org/10.1016/j.future.2016.03.011
0167-739X/© 2016 Elsevier B.V. All rights reserved.

example, under the use of vGASA in a hybrid platform, each GPU demand attains its basic QoS but the response time of the CPU workload still fluctuates hugely every second (Section 2).

Motivated by these problems, this paper proposes vHybrid, a user mode lightweight CPU–GPU resource management framework with new open loop and adaptive control algorithms, optimizing the CPU utilization while keeping the QoS of GPU-intensive workloads. All of the work is implemented by library API interception, without modifying either the architecture or the source code of the Operating System (OS), applications, or VMs.

vHybrid consists of two CPU control policies and a GPU scheduling policy for specific resource management. To meet SLA requirements for all the GPU resources, vHybrid borrows the SLA-aware scheduling policy from vGRIS. For CPU-intensive workloads, the first CPU scheduling policy, open-loop control allocates CPU resource using a preset model calculated by a relationship between our scheduling intensity and application performance. It can maximize the CPU resource usage and vacate as many resources as possible for the other waiting applications. The second policy is adaptive scheduling, which dynamically and instantaneously modifies the CPU resource required proportionally by analyzing SLA performance of applications. This process costs additional CPU resources, but ensures a substantial increase in control accuracy of QoS.

The main contributions of this paper are as follows:

- We propose vHybrid, a user mode framework for CPU–GPU hybrid resource management, without source code level changes in applications, guest OSes, or host OSes. Moreover, by leveraging the mature library API interposition, the guest OS and the network applications remain unmodified. This enables vHybrid to be deployed in any clusters to effectively schedule CPU and GPU resources.
- vHybrid is integrated by one existing algorithm for GPU scheduling and two new algorithms for CPU scheduling to trade off overhead and control accuracy of the CPU–GPU hybrid workloads. Open loop scheduling is a straightforward implementation that roughly minimizes the CPU resource occupation. Adaptive scheduling achieves better control accuracy of QoS for CPU-intensive workloads, with a small overhead.
- We conduct experiments of real-world workloads, showing that open-loop scheduling saves 37.29% of CPU utilization with satisfactory QoS for hybrid workloads, and adaptive scheduling achieves better QoS performance (i.e., reducing QoS fluctuations by three orders of magnitude), without compromising GPU workloads. The result outperforms existing SLA-aware GPU scheduling systems such as vGASA.

The rest of the paper is organized as follows. Section 2 gives the motivation of this paper. Section 3 introduces the architecture of vHybrid. Based on the architecture above, two CPU scheduling policies, open-loop scheduling and adaptive scheduling, are proposed in Section 4. Comprehensive experiments and performance evaluation are presented in Section 5. We discuss the related work in Section 6, followed by conclusions in Section 7.

2. Motivation

To motivate the need for both GPU and CPU scheduling policies, we start by introducing a few cases of CPU–GPU hybrid workloads for cloud computing in which QoS is desired. Subsequently, we show that both the default and SLA GPU scheduling would sometimes waste GPU and CPU resources for hybrid workloads. Finally, we conduct the experiments on our machine testbed and show how the SLA-aware GPU scheduling fails to provide good performance for CPU-intensive workloads.

2.1. CPU–GPU hybrid workloads

The CPU–GPU hybrid workloads have been employed as core roles to provide cloud services. For example, virtualization technology provides numbers of VMs with a software interface that is different from its underlying hardware counterpart in the cloud. Each VM can choose between running either CPU-intensive or GPU-intensive workloads. These workloads are sometimes mission-critical, which require high QoS of CPU and GPU resources according to SLA. Meanwhile, the hosts, which run the hypervisors, would sometimes launch other GPU and CPU computation instances. Therefore, poor resource sharing among different workloads would degrade some applications whose workloads become starved.

In this paper, our CPU–GPU hybrid workloads include three GPU-intensive instances (DiRT 3, Far Cry 2, StarCraft 2), and one CPU-intensive workload (Apache). The former three games will generate intensive graphic computational loads. During the process of gaming, the pictures usually have widely varying Frames Per Second (FPS). The FPS may continuously vary as the game scenes change, in which the QoS estimation is more complex than other GPU-intensive workloads. On the other hand, the Apache read and write [12] will continuously generate numbers of tasks for CPU response, which received 2000 requests per second and completed 200 requests simultaneously.

2.2. Native GPU scheduling for hybrid workloads

The native GPU scheduling as well as the CPU computation determines the FPS. Some CPU computation prepares the data for the GPU, e.g., calculating objects in the upcoming frame according to the game logic. The data are next uploaded to the GPU buffer, and then the GPU performs the computation. Finally, the calculation result is sent back to the main memory for the next iteration, or output to the screen. The GPU computation library depends on the application, e.g., Direct3D [13] or OpenGL [14] for gaming and rendering, or DirectCompute [15], OpenCL [16], or CUDA [17] for general-purpose GPU computation. The detailed API depends on the graphics library, e.g., `glutSwapBuffers` from OpenGL and `Present` from Direct3D.

We now present some experiments to observe the performance of a native scheduling policy (Direct3D) in our hybrid platform while guaranteeing the QoS of each workload. Each GPU workload concurrently runs in a separate VM which is configured with Windows 7[1] as the guest OS supporting the Direct3D graphics library, while the host concurrently launches the Apache service. Fig. 1(a) shows the experimental results.

Compared to their original performance under the same game configuration, the FPS of the GPU-intensive workloads is reduced dramatically due to the poor scheduling. The three games have FPS that fluctuate between 20 and 60. The most likely reason is the asynchronous and non-preemptive nature of the GPU process. For example, the default GPU scheduling mechanism in the Direct3D runtime library tends to allocate resources on a first-come first-serve manner, which results in an excessive FPS rate for low-end games and unplayable FPS rate for GPU intensive games when they are running concurrently in separate VMs. Graphics APIs also typically work in an asynchronous way to maximize hardware performance. APIs such as `DisplayBuffer` immediately return

[1] Note that many CPU scheduling policies can easily be implemented in Linux, e.g., `cgroups`, but we choose Windows as the basic OS for our implementation because today's most GPU computational applications are usually developed in the Windows platform. For example, the Direct3D framework provides many rendering functions for games in Windows.

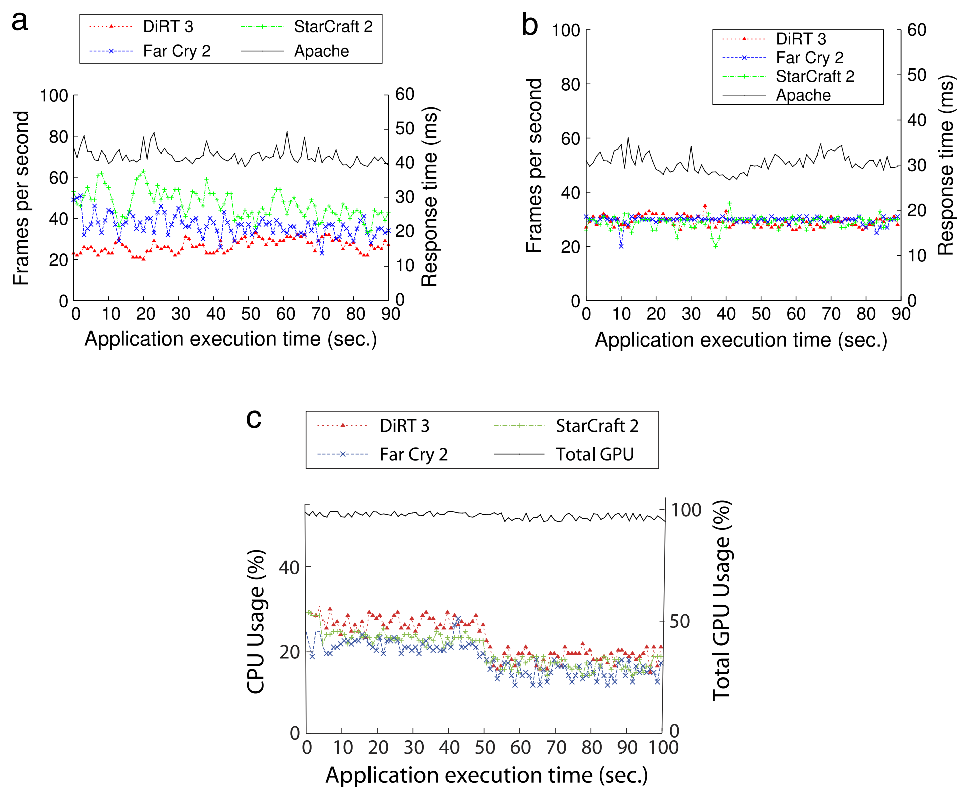


Fig. 1. Existing CPU–GPU resources scheduling results in the poor QoS under heavy contention: (a) Native default scheduling; (b) SLA-aware scheduling. (c) Shows the CPU usage that can be utilized after scheduling.

when they issue and submit a rendering command to the GPU. The GPU maintains a command buffer for upcoming user space requests. Therefore, if the underlying command buffer is full, the 3D application has to be blocked for some time.

On the other hand, the response time of Apache fluctuates hugely (from 39 to 51 ms). That is because the GPU computation tasks from the three VMs share the CPU resources with CPU-intensive workloads. Since the FPS may continuously vary as the game scene changes, the CPU allocation from the GPU computation fluctuates as the frames switch, and the native GPU scheduling does not address this problem as was mentioned above. In the example of Fig. 1(a), the higher the FPS that GPU-intensive workloads incur, the longer the response time that the Apache service attains. This might cause the starvation of the QoS of Apache.

2.3. *SLA-aware GPU scheduling for hybrid workloads*

To overcome the aforementioned problems of the default GPU scheduling for GPU-intensive workloads, our previous contribution is the design of an SLA-aware GPU scheduler called vGASA. vGASA features a feedback control loop using a proportional–integral (PI) controller [18]. We now use our experiment testbed to observe the performance of vGASA for scheduling hybrid workloads. Fig. 1(b) shows the observation results.

After using the SLA-aware policy of vGASA, compared with Fig. 1(a), all the games run much faster and meet their SLA requirements when scheduled. Their curves also remain stable, where the average FPS rates of DiRT 3, Starcraft 2 and Farcry 2 are 30.1, 30.5, and 30.3, respectively. Similarly, as shown in Fig. 1(c), the GPU usages from the three games typically range from 70%–80% and seldom exceed 90%. Moreover, vGASA enables an extra game concurrently run with these three workloads to improve GPU utilization. In this way, the SLA-aware scheduling policy can improve the GPU utilization while also insuring the QoS of the GPU-intensive workloads.

However, although the average response time of Apache changes from 28 to 35 ms, both Fig. 1(a) and (b) show that the response time of Apache fluctuates hugely every second. And according to our experiments, both native and SLA-aware scheduling incur a high CPU utilization totally with these hybrid workloads, i.e., 83.34% and 68.49%, respectively. These results reflect that CPU resources are excessively occupied by these workloads, which can lead to the starvation of QoS to other CPU-intensive workloads. In summary, the SLA-aware GPU scheduling cannot obtain an efficient CPU utilization of CPU-intensive workloads, or achieve an accurate QoS.

2.4. *Design guidelines*

Based on the aforementioned observation, we can arrive at the following conclusions. Firstly, the default GPU scheduling is imbalanced when it allocates either CPU or GPU resources to the workloads. Secondly, the SLA-aware GPU scheduling consumes excessive CPU resources such that other CPU workloads might become starved, but nonetheless insures that all the GPU-intensive workloads have their basic GPU allocation for QoS. Therefore, in this paper, our design guideline is to address these two problems. Similarly to our previous work that demonstrated the ability to efficiently allocate GPU resources while guaranteeing the QoS of each workload, vHybrid borrows such policy for GPU scheduling from vGASA. On the other hand, we integrate two CPU scheduling policies into our vHybrid framework. One allocates CPU resources using a preset model calculated by a relationship between scheduling intensity and application performance. The other dynamically and instantaneously modifies the CPU resources required proportionally by analyzing SLA performance of applications.

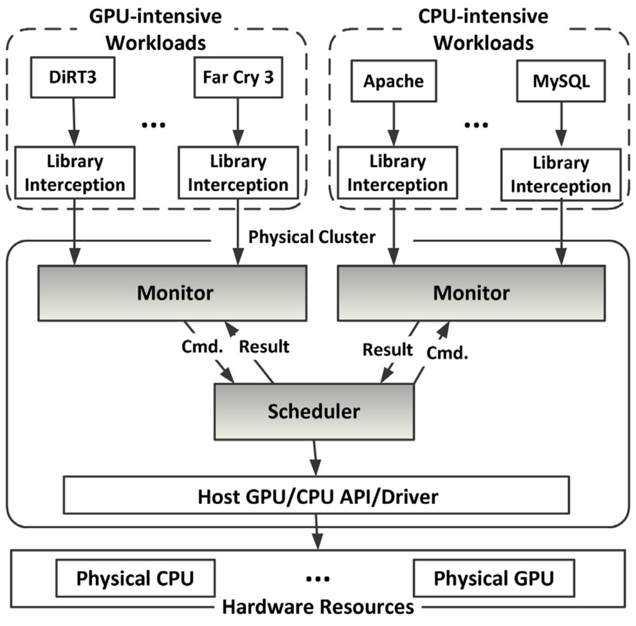


Fig. 2. Architecture of vHybrid framework.

3. vHybrid architecture

This section mainly presents the design and implementation of vHybrid architecture and the scheduling policies. First, we show the basic idea of vHybrid and the challenges to implement it in a CPU–GPU hybrid platform. Then, the overall architecture of vHybrid framework is proposed. Finally, we briefly introduce the SLA-aware GPU scheduling and two CPU policies: an open loop control algorithm and an adaptive control algorithm, for different CPU performance goals, respectively. Other CPU scheduling policies are applicable to vHybrid framework as well.

3.1. Key idea

In contrast to previous CPU and GPU scheduling schemes that modify or patch the guest OS kernel or hypervisor, vHybrid enables different workloads to utilize CPU and GPU resources with no modification. At its heart, vHybrid exploits library interposition to insert a `Sleep` function in the loops of both GPU and CPU computation models. In this way, vHybrid can control the resource utilization rates of games by setting a proper sleep time for each workload. For example, if a CPU-intensive application runs a bit too fast, vHybrid can slow it down such that other CPU applications have more access to the CPU. Likewise, a GPU-intensive workload will be rate-limited in FPS by vHybrid when its allocation exceeds the SLA. This leads to a simple scheduling technique where each workload takes care of its QoS and is not starved by others.

However, there are several concrete challenges to consider in order to make vHybrid truly effective. First, how should one insert the `Sleep` function for each workload to intercept the system call? Second, when and how long should we `Sleep` the workload? Third, how should we acquire information from the QoS of the workloads? Fourth, should we make vHybrid perform efficiently and stably in CPU–GPU hybrid platform? In Section 4, we will explain the details of the mechanism we have designed to address all these challenges.

3.2. Architecture design

Fig. 2 demonstrates the architecture of vHybrid within the CPU and GPU computational models, where modules introduced by vHybrid are highlighted in grey. It is designed in a cluster. Every workload has a monitor, which is responsible for monitoring its running parameters and communicating with a scheduler. The scheduler serves two purposes. First, it receives running parameters from monitors and regards them as the inputs to scheduling. Second, it automatically allocates CPU–GPU computation tasks as the output under the hybrid workloads. Thus, the central scheduler specifies the performance feedback from each monitor. In our current implementation, the scheduler is a user mode runtime engine that intercepts responding API invocations and schedules CPU–GPU resources according to the feedback in the cluster. To this end, we integrate one existing policy for GPU scheduling [8] and two new control algorithms for CPU to make a trade-off between overhead and accuracy of QoS.

The brief introduction of one GPU and two CPU control algorithms in vHybrid framework are listed as follows.

- **SLA-aware scheduling for GPU** allocates enough GPU resources to each running game to fulfill their SLA requirements. However, the CPU–GPU resources may be not fully used under this policy.
- **Open-loop control algorithm for CPU** allocates CPU resources as little as possible while still fulfilling the QoS requirements by a preset model. As the complexity of CPU-intensive workloads changes, higher stability of this scheduling policy may be incurred.
- **Adaptive control algorithms for CPU** allocate CPU resources using QoS feedback, and adjust applications' performance proportionally and dynamically. Thus, the calculation of resource allocation will cost additional CPU resources.

4. Implementation details

We now provide a detailed implementation of vHybrid framework. As described in Section 3, our framework mainly integrates three scheduling policies which can address all the challenges.

4.1. Library interposition

One way to achieve library interposition is to modify the hypervisor of the VMs. Hypervisors supporting the paravirtualization technology redirect graphics and socket API calls from 3D applications and I/O read/write directly to the corresponding API on the host. We can modify the redirection procedure and insert the code of the three scheduling policies. However, under such circumstances, the whole architecture should be fully re-implemented, which makes vHybrid framework inflexible when choosing different VMs. Besides, since the workloads are not restricted in VM instances, the hypervisor-based approach is not suitable in a non-virtualized environment.

Another way is to use a message processing mechanism that monitors the message loop of applications running under Windows. Such mechanism is called a hook. Using this approach, an application is able to intercept a particular type of messages and specify a function to deal with them before they are sent to other Windows applications. Hence, the greatest advantage of leveraging a hook as a graphics API interceptor is that the hypervisor, host OS, and underlying device driver do not need to be modified, and thus the approach is much more flexible when switching to a different solution. Therefore, this paper uses the hook method to implement the library interposition in the host. In our current implementation, we achieve the interception of both VM and host on a single server. Note that vHybrid is designed for a cluster, and the single server implementation can also be applied to the deployment to clusters.

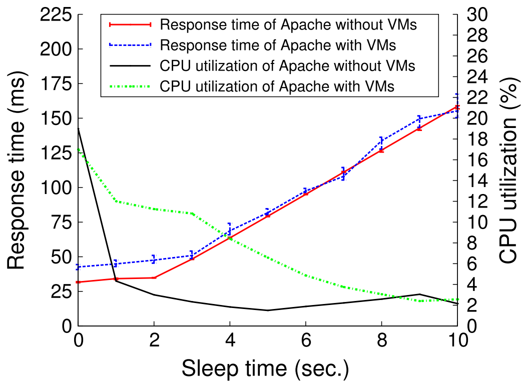


Fig. 3. The measurements of response time and CPU utilization with respect to varying sleep times.

Algorithm 1 The open-loop control algorithm.

```
1: set $T_{target}$ to *criteria*
2: while TRUE do
3:     ComputeApacheResponseTime()
4:     get responding sleep time $T$ according to the model
5:     $sleep\_time$ = CalcSleepTime($T$)
6:     Sleep($sleep\_time$)
7:     SocketRecv()
8: end while
```

4.2. CPU computation model

Our control policies for CPU mainly focus on saving CPU resources, while ensuring stable QoS of applications. Fig. 1(b) and (c) show high CPU utilization and large fluctuations of the QoS. Such expensive CPU utilization is caused by CPU-intensive applications getting more resources than necessary. Given this, we introduce two control algorithms to resolve the problem. Since our main objective is to control the stability of the QoS of applications, the response time of the Apache server must be converged to the reference of the system. Our system controls the response time by delaying its last call, `SocketRecv`.[2] This is achieved by inserting a `Sleep` call before `SocketRecv`. The time of the delay is decided by the control algorithm, and is preset in the case of the open-loop control and self-adjusted in the case of the adaptive control. Both algorithms come from the open-loop control theory [19,20]. Such theory has become an essential theory in the electronic communication area. The detailed design and implementation of the two control algorithms are discussed below.

Open-Loop Control Algorithm: The open-loop control algorithm allocates to CPU-intensive applications as little resources as possible only by inserting sleep calls to reduce the QoS slightly. We implement the control policy by using a preset model responsible for the response time and sleep time in CPU-intensive workloads.

The algorithm is shown in Algorithm 1. We take the previous Apache application as an example. First, we provide a preset model for Apache in each different case (Fig. 3). As the sleep time increases, the response time grows in a linear manner showing a dramatic dropdown of the CPU utilization of Apache. This implies that the relationship between our policy and performance can be modeled by a mathematical formula, fitting the open-loop control.

We can easily deduce the model as follows:

$$r(k) = \begin{cases} \lambda_1 s(k) + b_1, & s(k) > \eta, \\ \lambda_2 s(k) + b_2, & s(k) \le \eta, \end{cases} \tag{1}$$

where $r(k)$ is the output measurement of the response time at the control interval k, $s(k)$ is the action of the sleep time at the control interval k, λ_1, λ_2 and b_1, b_2 are the fitting parameters using input–output measurements, and η is the threshold where switching between linear models occurs.[3] Note that this model is partially linear and the response time is always controllable, since a change in the sleep time always affects the response time.

Our open-loop control pre-calculates a model that is able to find a balance between the QoS requirements and CPU resource allocation. The QoS requirements can be modified by a request of the cloud service provider. For responsive application performance, the open-loop control policy obtains the demand for the QoS in advance, using the current state of performance of applications and the previously calculated model to compute the necessary latency immediately. While controlling, the open-loop control policy can take the following strategy:

$$s_{req}(k) = \begin{cases} \dfrac{r_{ref}(k) - b_1}{\lambda_1}, & r_{ref}(k) > \gamma, \\ \dfrac{r_{ref}(k) - b_2}{\lambda_2}, & r_{ref}(k) \le \gamma \end{cases} \tag{2}$$

where $r_{ref}(k)$ is the response time reference at the control period k, $\gamma = \lambda_1\eta + b_1 = \lambda_2\eta + b_2$ is the threshold for model switching and $s_{req}(k)$ is the sleep time action taken at the control period k.

The open-loop control policy focuses on the CPU resource utilization. This does not promise the control accuracy in the applications' performance. Therefore, we provide the second algorithm, the adaptive control policy, which is capable of maintaining the control accuracy of the SLA performance in complex CPU–GPU hybrid environments.

Adaptive Control Algorithm: As shown in Algorithm 2, the adaptive control policy for CPU uses a feedback-based control theory. Every time before making a schedule, we obtain the latest performance of applications, then, the `response time controller` introduced below works out the time of delay for the request. In addition, the controller makes the feedback gradually approach the desired response time. The developed mechanism adapts to complicated systems and ensures the stability of the SLA performance of applications, even in the case of unpredictable workloads.

Regarding the response time controller, it is designed to be an independent single in single out (SISO) controller for each of the sets of VMs hosting CPU-intensive applications. The role of such controllers is to compute the required sleep time for each VM such that, (1) the hosted CPU-intensive application component gets enough resources so that it does not become a bottleneck in a multi-tier application; (2) the VM maintains a relatively high resource utilization of the CPU. In our design, the way to achieve these goals is to maintain a specified level of quality of experience for each web user. Therefore, such controllers are referred to as *response time controllers*.

To design the response time controller, we study the relationship between the sleep time and response time. In such a model, we can refer to the response time as the system output and the sleep time as the system input. To obtain the input–output model, we apply the parameter fitting method using actual measurements, as

[2] Note that `SocketRecv` is a system call only for Apache service [11]. We can also observe that many other CPU-intensive applications usually have `SocketRecv`-like system calls, because `SocketRecv` is a socket function responsible for reading contents and replying with a value to the client, the `Sleep` call in this paper could easily be inserted in other socket-like functions.

[3] In Fig. 3, we use two 2 periods of linear fitting [20] to emulate the algorithm for sleep time. So there is only one switching point at open-loop and adaptive algorithms.

Algorithm 2 The Adaptive control algorithm.

```
1:  set $T_{target}$ to criteria
2:  while TRUE do
3:      ComputeApacheResponseTime()
4:      get feedback, $T_{out}$, from last iteration
5:      $Offset = T_{target} - T_{out}$
6:      $F$ = PICalAndControl($Offset$)
7:      $sleep\_time$ = CalcSleepTime($F$)
8:      Sleep($sleep\_time$)
9:      SocketRecv()
10: end while
```

shown in Fig. 3. At the end of each control period, the response time is measured as the system output. Hence, the measurement transfer function is $H(z) = 1$. We adopt a PI controller which is sufficient to achieve the adaptive performance of a closed-loop system [18]. Hence, we can obtain the transfer function of the PI controllers for the discrete system as follows:

$$G_c(z) = \begin{cases} G_{c1}(z) : k_p^1 + k_i^1 \dfrac{z}{z-1}, & r(k) > \gamma, \\ G_{c2}(z) : k_p^2 + k_i^2 \dfrac{z}{z-1}, & r(k) \le \gamma \end{cases} \tag{3}$$

where k_p^1 and k_p^2 are the proportional gains and k_i^1 and k_i^2 are the integral gains. They should be chosen properly to ensure the system's stability and convergence.

Next step is to obtain the transfer function ($G_p(z)$) of the response time control. According to (1), we can get the transfer function $G_p(z)$ as follows:

$$G_p(z) = \begin{cases} G_{p1}(z) : \lambda_1, & r(k) > \gamma, \\ G_{p2}(z) : \lambda_2, & r(k) \le \gamma. \end{cases} \tag{4}$$

The control block diagram is shown in Fig. 4.

Then we analyze the system's stability. The first closed-loop transfer function of Fig. 4, denoted by $T_{c1}(z)$, is given by

$$T_{c1}(z) = \frac{G_{c1}(z)G_{p1}(z)}{1 + G_{c1}(z)G_{p1}(z)H(z)} = \frac{(\lambda_1 k_p^1 + \lambda_1 k_i^1)z - \lambda_1 k_p^1}{(\lambda_1 k_p^1 + \lambda_1 k_i^1 + 1)z - (\lambda_1 k_p^1 + 1)}. \tag{5}$$

From (5), we can get the pole of the first closed-loop discrete system at

$$z_1 = \frac{\lambda_1 k_p^1 + 1}{\lambda_1 k_p^1 + \lambda_1 k_i^1 + 1}. \tag{6}$$

From control theory [18], if the pole is within the unit circle centered at the origin, i.e.,

$$\left| \frac{\lambda_1 k_p^1 + 1}{\lambda_1 k_p^1 + \lambda_1 k_i^1 + 1} \right| < 1, \tag{7}$$

then, the closed-loop discrete system is stable.

Similarly, we can get the pole of the second closed-loop discrete system at

$$z_2 = \frac{\lambda_2 k_p^2 + 1}{\lambda_2 k_p^2 + \lambda_2 k_i^2 + 1} \tag{8}$$

and get the stability condition for the second controller G_{c2}:

$$\left| \frac{\lambda_2 k_p^2 + 1}{\lambda_2 k_p^2 + \lambda_2 k_i^2 + 1} \right| < 1. \tag{9}$$

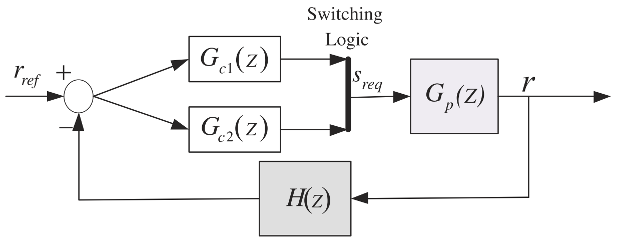


Fig. 4. Blocks of response time controller.

Note that the response time controller exerts gain scheduling control approach to stabilize the partial linear system. The two controllers are switched between G_{c1} and G_{c2} with the logic of $r(k) > \gamma$ or not. The conditions of (7) and (9) only ensure the stability for the controllers G_{c1} and G_{c2} separately. However, the stability for the whole adaptive scheduling should be discussed based on the linear switched systems.

Note that when the integral parameter k_i and proportional parameter k_p are both 0, the system is an open-loop transfer function. However, when only the integral parameter K_i is 0, the closed-loop transfer function (7) and (9) can be rewritten as $z_1 = \frac{\lambda_1 k_p^1}{\lambda_1 k_p^1 + 1}$ and $z_2 = \frac{\lambda_2 k_p^2}{\lambda_2 k_p^2 + 1}$ respectively. Hence the adaptive system with proportional control is always stable without any poles.

We consider discrete-time switched systems with a collection of discrete-time linear time-invariant (LTI) systems as

$$r(k+1) = A_q r(k), \quad k \in \mathbb{Z}^+,\ q \in Q, \tag{10}$$

where $\mathbb{Z}^+$ is a set of non-negative integers. The set $Q = \{1, 2\}$ is an index set and stands for the collection of discrete models, i.e., $A_1 = z_1$ and $A_2 = z_2$.

A switched linear system (10), where $A_q \in \mathscr{A} \equiv Conv\{A_1, A_2\}$, $Conv\{.\}$ means convex combination, is globally asymptotically stable under arbitrary switchings if and only if there exists a finite integer n such that

$$\|A_{i_1} A_{i_2} \cdots A_{i_n}\| < 1, \tag{11}$$

for all n-tuple $A_{i_j} \in \{A_1, A_2\}$, where $j = 1, \ldots, n$. More stability analysis for switched systems can be found in [21].

4.3. GPU computation model

To design a GPU computation model for scheduling GPU-intensive workloads, our previous work, VGRIS [8,22], proposes several scheduling policies for cloud based gaming platforms, such as the SLA-aware, proportional-share scheduling and hybrid scheduling with a compromise. The SLA-aware scheduling is designed to guarantee the minimum FPS required and maximum latency for a cloud games service provider. It allocates just enough resources for each VM to ensure its SLA. Otherwise, VM needing less GPU may get more resources than necessary, even if GPU demanding ones cannot meet the requirement. Note that the goal of cloud providers is usually to guarantee a basic SLA for each VM, so the SLA-aware scheduling should be an appropriate choice for them.

To achieve the same goal, we also slow down less demanding applications to free extra GPU resources for others. In the example of our CPU–GPU hybrid platform, after the `Present` call returns, we have no direct control over the time when the frame becomes visible. However, the extra delay is negligible in the case of a local display. If the frame is displayed remotely, we assume some fixed time of network delay. Therefore, we consider the frame latency as the time in-between the returns of two consecutive `Present` calls, as illustrated in Fig. 5(b).

To stabilize the frame latency according to a given SLA, we extend each frame by delaying its last call, `Present`. This is achieved via inserting a `Sleep` call before the `Present` call. The amount of the delay should be equal to the desirable latency minus the time of computation of `ComputeObjectsInFrame`, `DrawPrimitive`, and `Present` altogether.

The SLA-aware scheduling can also contribute to improving CPU resources usage. As GPU utilization needs APIs, such as Direct3D or OpenGL, a lower GPU resource usage can also lead to a release of CPU resources. Moreover, the results of GPU calculations are sent to the CPU, which can also affect the performance of related applications. Therefore, the SLA-aware scheduling provides a method that allows cloud service providers to use their CPUs more efficiently.

5. Evaluation

In this section, we provide a detailed quantitative evaluation of vHybrid by comparing it with the native performance of the system. The environment setting of our experiments is introduced in Section 5.1. We conducted a series of experiments to illustrate the performance of two CPU scheduling policies that are designed. Then, another group of experiment is conducted to estimate the leftover CPU resource usage in vHybrid. With a comprehensive analysis at last, we evaluated the efficiency of CPU–GPU resource management and control accuracy of QoS in vHybrid. To evaluate the performance, we focused on two aspects: (1) The efficiency of CPU resource management for hybrid workloads. (2) The QoS (response time, throughput, or FPS) of workloads, including its accuracy.

The experimental processes in four fields are as follows:

- The influence on the performance of existing GPU-intensive workloads in VMs with two new CPU scheduling policies.
- How two new CPU scheduling policies affect the accuracy of the QoS of CPU-intensive workloads (i.e., Apache).
- How other workloads (i.e., MySQL) utilize the saved CPU resource in vHybrid.
- The overall efficiency of the CPU utilization of vHybrid under CPU- and GPU-intensive hybrid workloads.

5.1. Experimental settings and workloads

The experiments are conducted on the testbed with iCore7-2600k 3.4 GHz CPU, 16 GB RAM, and ATI HD6750 graphics card. Each hosted VM is assigned dual cores and 2 GB RAM. Windows7 x64 is selected as the operating system for both host OS and guest OSes. The hypervisor is VMware Workstation [23]. The configurations of GPU-intensive and CPU-intensive workloads are the same with Section 2. Furthermore, when measuring the response time of Apache, we sent a data file along the request, so that an increase of response time will be convenient for us to adjust our scheduling.

5.2. Effectiveness of proposed scheduling algorithms

This experimental section starts with the system condition in which three gaming workloads run in separate VMs and share one graphics card. Furthermore, an Apache host is also run together with these VMs, sharing the same CPU with them. Note that all the operations of this experiment are conducted together with SLA-aware scheduling for GPU, thus making sure that the GPU resource is legitimately utilized and the workloads in VMs have a stable FPS performance, as shown in Fig. 1(b), Section 2.

5.2.1. GPU workloads performance and stability

We have evaluated the stability of the performance of the GPU workloads under the open-loop scheduling approach. Fig. 6 illustrates that without any of the two algorithms, the average values of the CPU utilization for the three workloads in VMs are 15.26% (Far Cry 2), 16.72% (StarCraft 2), and 11.25% (DiRT 3), with a total CPU utilization of 43.24%. With open-loop scheduling, the values of the three workloads become 14.34% (Far Cry 2), 17.14% (StarCraft 2), and 12.03% (DiRT 3), adding up to a total CPU utilization of 43.54%. With adaptive scheduling, these three workloads occupy 11.82% (DiRT 3), 16.25% (StarCraft 2), and 15.33% (Far Cry 2) of the CPU utilization, respectively, which adds up to a total of 43.40%. The CPU utilization of the workloads in VMs has been changed with 0.34% after open-loop scheduling is used. Whereas, the total CPU utilization after applying adaptive scheduling is 0.16% higher than the CPU utilization without using any scheduling. Furthermore, the CPU usages of the workloads in VMs after adaptive scheduling are even more accurate than with open-loop scheduling.

As shown in Fig. 7, the open-loop scheduling for the CPU will merely affect the FPS of the GPU-intensive workloads. Each FPS of the three workloads maintains a level of 30 after applying our proposed scheduling. Thus, it can be concluded in accordance to the above evaluation that the open-loop scheduling hardly influences the performance of the GPU workloads, including FPS and CPU utilization. Whereas, in case of adaptive scheduling, not only a stable CPU utilization is achieved, but also the FPS of the workloads remain at a level of 30. Therefore, the adaptive scheduling algorithm also makes sure that the performance of the GPU-intensive workloads will be marginally influenced.

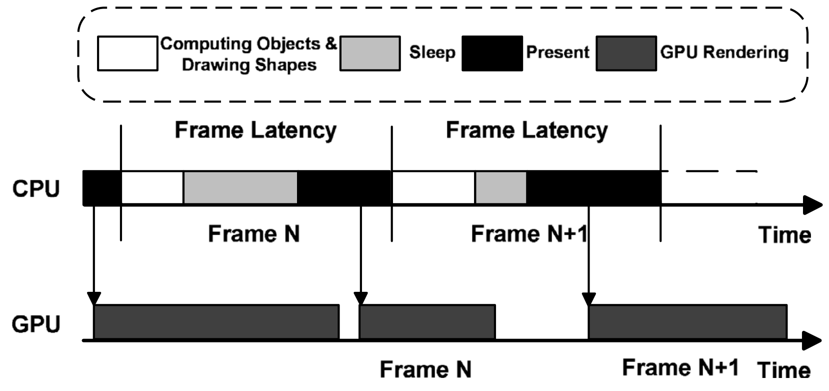
5.2.2. CPU workloads performance and stability

In addition, we have evaluated the efficiency of the open-loop scheduling for CPU in controlling the response time of Apache. In order to justify the effectiveness of the scheduling algorithm in different user scenarios, the response times for the experiment have been set to 50, 80 and 100 ms. Fig. 8 gives the results of different scenarios. We also list the max and min response times of Apache as well as its average CPU utilizations under three policies in Table 1.

Without any of the two scheduling algorithms, Apache response time varies significantly from second to second as shown in Fig. 8. Also, the rang of response time is 11.5 ms (Table 1), which makes the standard deviation of response time sequence to be 3.84. With open-loop scheduling, the rang of response time becomes 11.2 ms (Table 1) and the standard deviation of response time sequence becomes 2.21. The result is almost 3 times more accurate than that with no scheduling. When adaptive scheduling is used, the average response time of Apache has been slowed down to 49.83 ms, which is only 0.17% smaller than what we expected. This is an acceptable error as compared to the open-loop scheduling error of 1.2%. This is because the opening and writing files to Apache incur an extra overhead in our adaptive scheduling, causing a longer interval in calculating the sleep time dynamically than that expected. In conclusion, both the open-loop and adaptive scheduling can

```
While(1) {
  ComputeObjectsInFrame();
  DrawPrimitive(&VGA_Buffer);
  Sleep(calculated_sleep_time);
  Present(&VGA_Buffer);
}
```

(a) Pseudocode under Direct3D.



(b) Frame latency.

Fig. 5. SLA-aware scheduling approach [8].

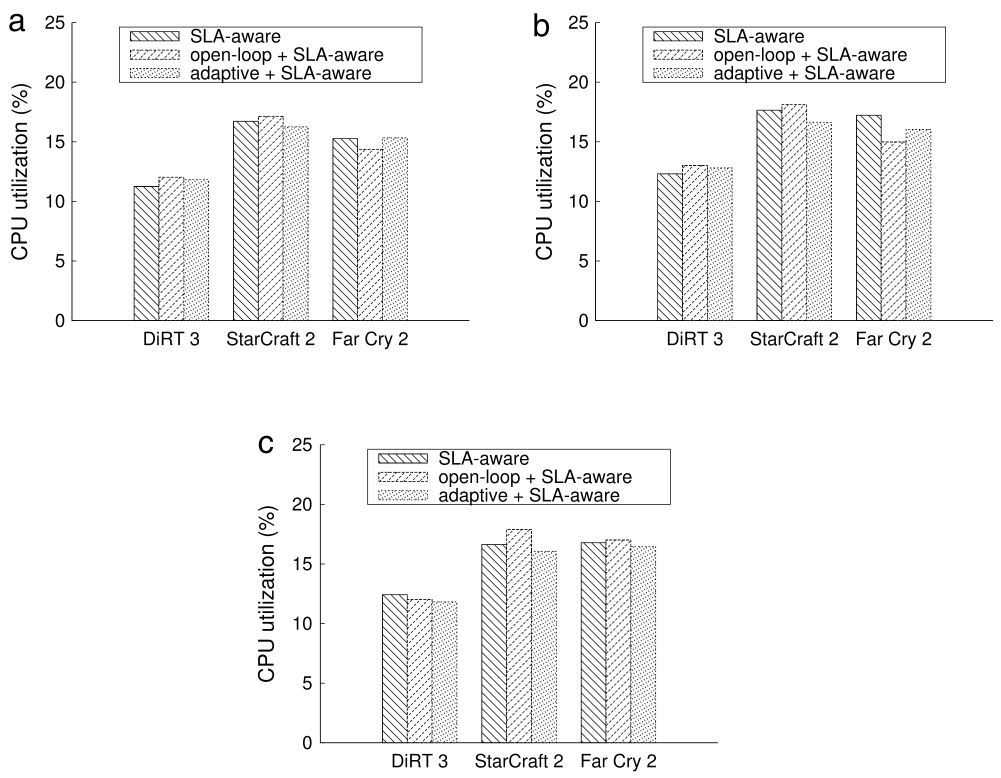


Fig. 6. (CPU utilization of game workloads) response time = (a) 50 ms; (b) 80 ms; (c) 100 ms.

Table 1
Response time and CPU utilization of Apache in different scheduling policies.

Scheduling policies	Response time (ms)			Average CPU utilization
	Max	Min	Rang	
No scheduling	38.6	27.1	11.5	19.14%
Open-loop	56.3	45.1	11.2	13.87%
Adaptive	52.3	47.8	4.5	15.06%

set the response time at different values to satisfy the user requirements.

5.2.3. CPU utilization of Apache with two algorithms

As shown in Fig. 9 and Table 1, considering the response time is equal to 50 ms, the average CPU utilizations of Apache after the open-loop and the adaptive scheduling dramatically drop, which are only 72.4% and 78.6% of the CPU utilization with no scheduling (Table 1). This clearly confirms our design idea of the two scheduling algorithms: CPU resources are saved substantially as shown by our control model. The adaptive scheduling occupies a bit more CPU resources than the open-loop scheduling. The reason is that we need to obtain the response time and adjust the sleep time dynamically, which will eventually cost slightly more CPU resources. Briefly, both scheduling algorithms occupy less CPU resources than without any scheduling. It manifests that they avoid unnecessary CPU resource cost. And with response time being set higher, the CPU utilization is cut down.

5.3. Utilizing leftover CPU resource

In this part of evaluation, the usage efficiency of the CPU resource released by the formerly mentioned two scheduling policies is demonstrated. We selected MySQL [24], one of the most popular open-source database software, to fulfill the CPU together with three VMs and Apache. In this group of experiments, MySQL query sending is tested. Each time, we use benchmark "*mysqlslap*" [25] to simulate sending 2000 queries along with

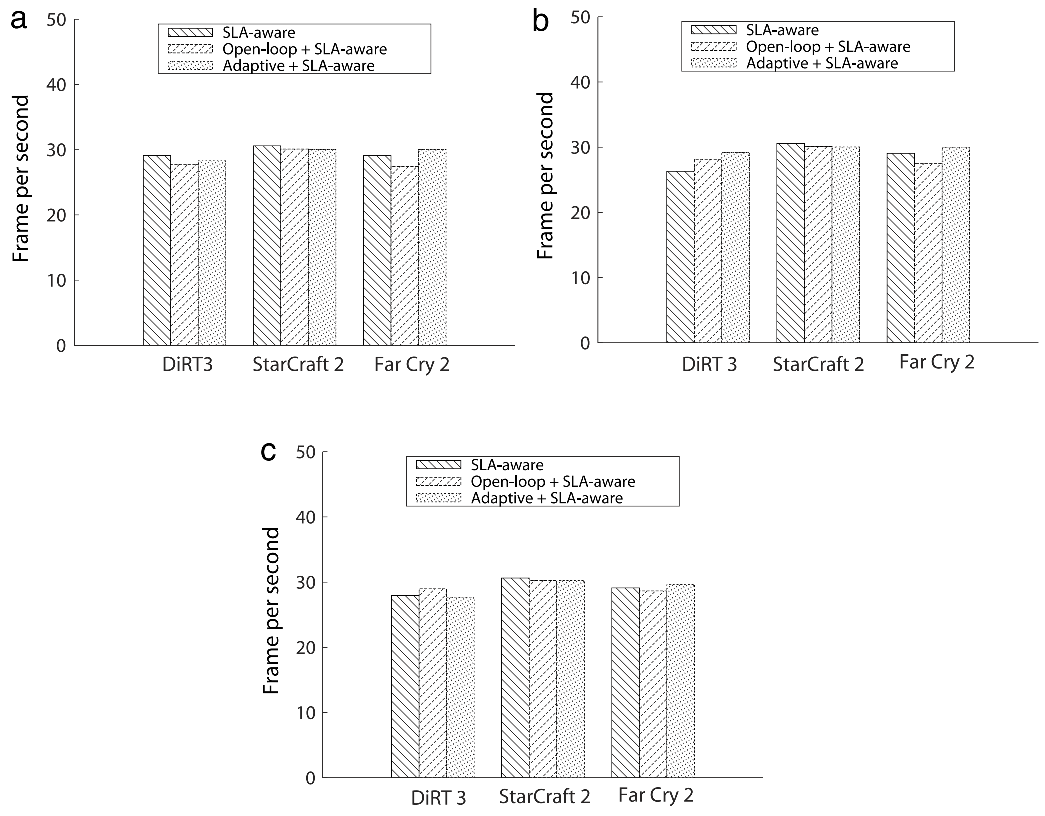


Fig. 7. (FPS of game workloads) response time = (a) 50 ms; (b) 80 ms; (c) 100 ms.

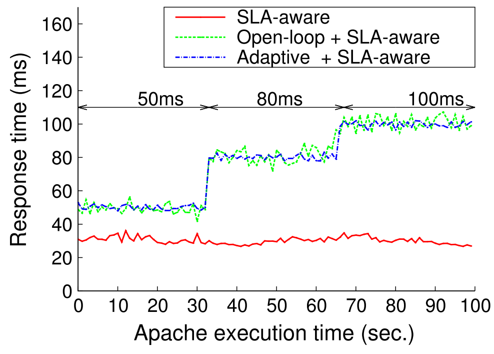


Fig. 8. Response time of Apache in different scheduling policies.

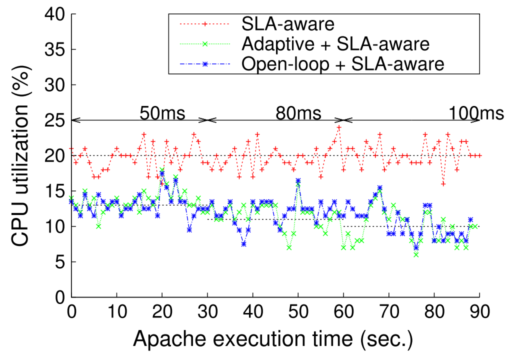


Fig. 9. CPU utilization of Apache in different scheduling policies.

a concurrency of 2000 clients to MySQL server. *mysqlslap* is a diagnostic program designed to emulate client load for a MySQL server and to report the timing of each stage. This test lasts for a few minutes, and the throughput of the server is collected each second with three different scheduling policies.

5.3.1. *MySQL throughput performance*

As shown in Fig. 10, our experiment lasts for three stages, and each stage lasts for 120 s. The first stage is the throughput of MySQL after SLA-aware scheduling for the GPU. At that time, we have just enough CPU resource to run this workload barely, leading to a low throughput of MySQL. Thus, in the first 120 s, the average throughput of MySQL only remains at a level of 2414.19. After the first 120 s, we spend 120 s to run the workloads after open-loop and adaptive scheduling, respectively. Fig. 10 shows that the average throughput of MySQL respectively reached 3738.28 and 3193.89. As the result shows, both the throughputs increase significantly by our new CPU scheduling policies.

In particular, although the CPU utilization of Apache decreases very slightly (only 1.73%) after adaptive scheduling, still a large increase in throughput for MySQL is executed. Our scheduling policies decrease the CPU resource occupied by Apache; thus, the

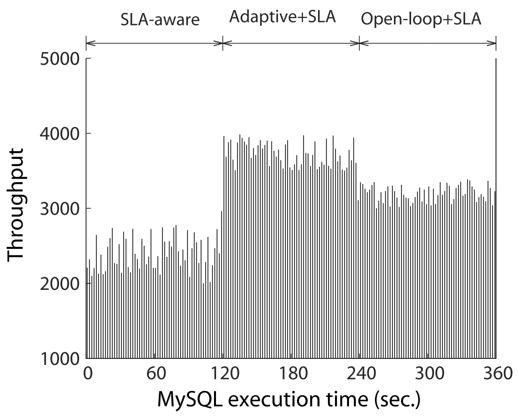


Fig. 10. MySQL throughput performance with different scheduling policies.

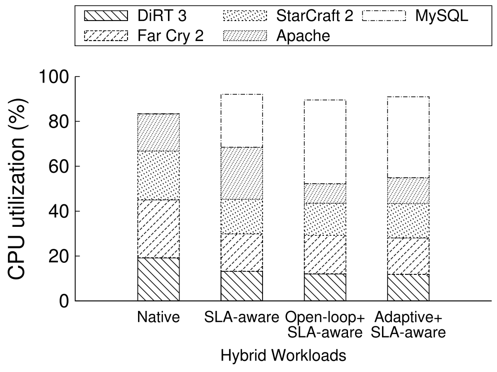


Fig. 11. CPU utilization of hybrid workload.

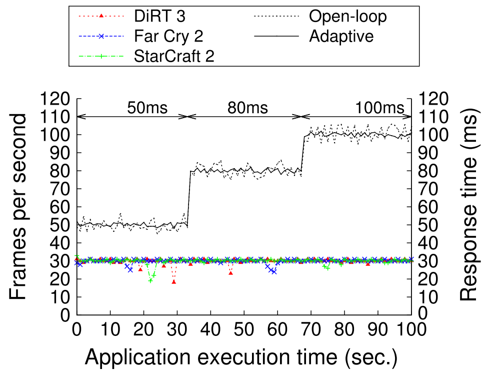


Fig. 12. Overall system performance of QoS.

leftover computing resource can be used by MySQL. Therefore, vHybrid provides an efficient CPU resource management.

5.3.2. *MySQL throughput stability*

Not only the throughput of MySQL improved greatly, its stability is also guaranteed by our scheduling policies. As shown in Fig. 10, with a standard deviation of 2475.33, the throughput of MySQL without our CPU scheduling varied from 2797 to 2003. During that situation, the CPU performance is poor because of the resources scrambled by Apache and MySQL.

Both the scheduling policies restrict the greedy occupation of CPU-intensive applications, as we have a series of control methods to limit the unnecessary cost of CPU resources. After the open-loop scheduling, the standard deviation of throughput decreases to 1953.83, ranging from 3999 to 3301. Adaptive scheduling shows even a better stability, with a standard deviation value of 1190.82 and the range differed from 3386 to 3000. Thus, our polices achieve 21.06% and 51.89% improvement, respectively.

5.4. *Comprehensive analysis*

Finally an overall evaluation of the system in vHybrid is made. We illustrate our experimental results in the CPU utilization and QoS of hybrid workloads, compared to the performance shown in Fig. 1(a) and (b).

5.4.1. *The CPU utilization of hybrid workloads*

With two new scheduling policies, the system can host five workloads well as shown in Fig. 11, while it can only host four workloads under the vanilla environment and five workloads reluctantly with SLA-aware scheduling. Then, the CPU utilization is evaluated. Fig. 11 shows the CPU utilization in four cases based on different resource scheduling policies. According to the first bar, the games and Apache occupy 83.34% of CPU resources. Furthermore, Apache do not have enough CPU resource to achieve a stable response time. This is because without SLA-aware scheduling, the games occupy extra resources when their FPS is far beyond 30 and inevitably affects the performance of Apache. Compared to the first, the second bar shows that SLA-aware scheduling lowered the CPU utilization of games. Further, Apache slows the response of CPU resources released by games. Meanwhile, all these games run on an FPS of 30, meeting a smooth user experience.

To evaluate the leftover 31.51% CPU utilization after SLA-aware scheduling, MySQL is introduced into the system. By observing the performance of added workload MySQL, it can be concluded that MySQL can obtain the CPU resources saved from the games. However, although MySQL occupies a certain proportion of CPU resources, it does not work well because Apache occupies excessive CPU resources. That is what our scheduling policies for CPU aim to solve as shown in the last two bars.

5.4.2. *The QoS performance of hybrid workloads*

Fig. 12 shows the comparison between our two CPU scheduling policies. The response time of Apache is stabilized at 50 ms by both open-loop and adaptive scheduling policies, indicating a guarantee of QoS of CPU-intensive workloads in vHybrid. For the control accuracy of QoS, the stability of QoS is both promised by open-loop and adaptive scheduling policies, while the latter performs better.

Moreover, because of the SLA-aware scheduling for GPU, all the games are run at an FPS of 30 and fluctuated in a small range. As the fluency of GPU-intensive applications is guaranteed, it can be concluded that in vHybrid, all the hybrid workloads (GPU- and CPU-intensive) fulfill the QoS accurately, with efficient CPU resource management.

6. Related work

CPU–GPU resource management is not yet well studied, but is becoming a hot topic these days [5,7,26–28]. However, resource management as a scheduling policy in cloud systems has been widely studied over the years. Recent efforts on job scheduling in distributed clusters minimize the response time for network services in a cloud environment [29,30]. Maguluri et al. [31] proposed a stochastic model for a cloud computing cluster which focuses on the allocation issue for resources such as the CPU, the memory and the storage space. Wong et al. [32] introduced fairness functions for multi-resource allocations, including CPU and memory. Ko et al. [33] proposed a hardware-in-the-loop simulation technique that integrates the GPU hardware into a full system simulator. The work described in these references mainly uses a multi-core CPU and possibly one or more GPU(s) in their cluster or cloud environment; however, it lacks a QoS-aware scheduling mechanism for hybrid CPU–GPU computing resources.

Previous studies on GPU virtualization have used the interposition technology to export GPU features to VMs [10,4,5]. Kato et al. [34] introduced a GPU resource sharing solution with a priority scheduling algorithm for GPU-accelerated applications in the X Window system. Phull et al. [35] presented a time-share framework to handle interference and to schedule GPU resources in GPU-based clusters. Sajjapongse et al. [36] proposed a runtime system with a preemptive mechanism to schedule multi-process GPU applications including synchronization. Gdev [2] provided a GPU scheduling scheme and virtual GPU support in real-time systems, in order to achieve GPU utilization and isolation among multiple tasks. TimeGraph [3] provided a real-time GPU scheduler that offers prioritization and isolation features in dealing with GPU workloads. vCUDA [5] and rCUDA [4] interpose Compute Unified

Device Architecture (CUDA) APIs in VMs to schedule GPU resources. VGRIS [8] also uses library interposition to achieve a virtualized GPU resource isolation and scheduling in cloud gaming. In contrast, this paper demonstrates the inefficient CPU utilization under CPU–GPU hybrid workloads and provides a lightweight framework to improve the CPU resource management without reducing GPU resources' QoS.

Several recent projects have studied the utilization of workload consolidation on CPU and GPU [1,2,27]. Luk et al. [37] proposed Qilin that implements automatic mapping technology on distributed systems with hybrid CPU–GPU resources. Vignesh et al. [38] considered two CPU/GPU node scheduling: CPU/GPU specific node and hybrid node. All the above work achieve a sound mathematical analysis; however, it requires modifications in order to be applied to the kernel OS. Piao et al. [39] proposed a new framework for efficient work sharing between CPU and GPU for data-parallel JavaScript workloads. In this case, the work-sharing scheduler partitions the input data into smaller chunks and dynamically dispatches them to both CPU and GPU for concurrent execution. By comparison, vHybrid can be efficiently deployed to any hybrid platform because it only traps the system and graphic calls in order to avoid any kernel modifications.

7. Conclusion

The cloud platform usually employs on a large scale multi-core CPUs and many-core GPUs, and consequently faces the key challenge of efficiently utilizing the CPU–GPU resources for diverse games and other applications. In order to address this challenge, first we analyzed real CPU- and GPU-intensive hybrid workloads. We found that CPU utilization is usually inefficient since the typical GPUs are just used to accelerate computing, with a limited support for CPU–GPU resource sharing. We proposed vHybrid, a user mode lightweight CPU–GPU resource management framework that is SLA-aware, and which utilizes open-loop and adaptive control algorithms. We conducted performance evaluations without modifying the native OS and applications and found that vHybrid achieves efficient resource utilization and response time with a desirable control accuracy of QoS for CPU–GPU hybrid workloads in cloud platforms. Our future work will implement vHybrid for large scale workload. Also, our future work will include the analysis of more APIs from other CPU intensive workloads and the addition of hook functions for interception.

Acknowledgments

This work was supported by National Science and Technology Major Project (No. 2013ZX03002004), National R&D Infrastructure and Facility Development Program (No. 2013FY111900), and NRF Singapore CREATE Program E2S2.

References

[1] A.J. Younge, J.P. Walters, S.P. Crago, G.C. Fox, Supporting high performance molecular dynamics in virtualized clusters using IOMMU, SR-IOV, and GPUdirect, in: Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE'15, 2015.

[2] S. Kato, M. McThrow, C. Maltzahn, S. Brandt, GDEV: first-class GPU resource management in the operating system, in: Proceedings of the 2012 USENIX Conference on USENIX Annual Technical Conference, 2012.

[3] S. Kato, K. Lakshmanan, R. Rajkumar, Y. Ishikawa, TimeGraph: GPU scheduling for real-time multi-tasking environments, in: Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference, 2011.

[4] J. Duato, A.J. Peña, F. Silla, R. Mayo, E.S. Quintana-Ortí, rCUDA: Reducing the number of GPU-based accelerators in high performance clusters, in: Proceedings of the International Conference on High Performance Computing and Simulation, 2010.

[5] L. Shi, H. Chen, J. Sun, K. Li, vCUDA: GPU-accelerated high-performance computing in virtual machines, IEEE Trans. Comput. 61 (6) (2012) 804–816.

[6] I. Tanasic, I. Gelado, J. Cabezas, A. Ramirez, N. Navarro, M. Valero, Enabling preemptive multiprogramming on GPUs, in: Proceeding of the 41st Annual International Symposium on Computer Architecuture, ISCA'14.

[7] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, R.R. Rajkumar, RGEM: A responsive GPGPU execution model for runtime engines, in: Proceedings of the 2011 IEEE 32nd Real-Time Systems Symposium, 2011.

[8] M. Yu, C. Zhang, Z. Qi, J. Yao, Y. Wang, H. Guan, VGRIS: virtualized GPU resource isolation and scheduling in cloud gaming, in: Proceedings of the 22nd International Symposium on High-Performance Parallel and Distributed Computing, 2013.

[9] Z. Qi, J. Yao, C. Zhang, M. Yu, Z. Yang, H. Guan, VGRIS: virtualized GPU resource isolation and scheduling in cloud gaming, ACM Trans. Archit. Code Optim. 11 (2) (2014).

[10] M. Becchi, K. Sajjapongse, I. Graves, A. Procter, V. Ravi, S. Chakradhar, A virtual memory based runtime to support multi-tenancy in clusters with GPUs, in: Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing, 2012.

[11] AS Foundation, Apache HTTP Server Reference Manual—for Apache Version 2.2.17, 2010.

[12] W.A. Knaus, J.E. Zimmerman, D.P. Wagner, et al., APACHE-acute physiology and chronic health evaluation: a physiologically based classification system, Crit. Care Med. 9 (8) (1981) 591–597.

[13] D. Blythe, The direct3D 10 system, in: ACM SIGGRAPH 2006 Papers, SIGGRAPH'06, 2006, pp. 724–734.

[14] OpenGL.org, OpenGL-The Industry Standard for High Performance Graphics. https://www.opengl.org/.

[15] Nvidia, Directcompute. https://developer.nvidia.com/directcompute.

[16] Khronos, The Open Standard for Parallel Programming of Heterogeneous Systems. https://www.khronos.org/opencl/.

[17] J. Sanders, E. Kandrot, first ed.

[18] K. Ogata, Discrete-Time Control Systems, Prentice-Hall, 1995.

[19] C.D.L.J. Collins, Open-loop and closed-loop control of posture: a random-walk analysis of center-of-pressure trajectories, Exp. Brain Res. 95 (2) (1993) 308–318.

[20] D. Jacobs, Linear fitting with missing data: Applications to structure-from-motion and to characterizing intensity images, in: 1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1997. Proceedings, IEEE, 1997, pp. 206–212.

[21] H. Lin, P. Antsaklis, Stability and stabilizability of switched linear systems: A survey of recent results, IEEE Trans. Automat. Control 54 (2) (2009) 308–322. http://dx.doi.org/10.1109/TAC.2008.2012009.

[22] C. Zhang, Z. Qi, J. Yao, M. Yu, H. Guan, vGASA: Adaptive scheduling algorithm of virtualized GPU resource in cloud gaming, IEEE Trans. Parallel Distrib. Syst. (2013).

[23] M. Rosenblum, Vmware's virtual platform, in: Proceedings of Hot Chips, Vol. 1999, 1999, pp. 185–196.

[24] Oracle Co., Ltd., MySQL: The World's Most Popular Open Source Database, https://www.mysql.com/.

[25] Oracle Co., Ltd., MySQL 5.7 Reference Manual, http://dev.mysql.com/doc/refman/5.7/en/mysqlslap.html.

[26] B. Van Werkhoven, J. Maassen, H.E. Bal, et al., Optimizing convolution operations on GPUs using adaptive tiling, Future Gener. Comput. Syst. 30 (2014) 14–26.

[27] A. Pathania, Q. Jiao, A. Prakash, T. Mitra, Integrated CPU–GPU power management for 3D mobile games, in: Proceedings of the 51st Annual Design Automation Conference, DAC'14, 2014.

[28] V.T. Ravi, M. Becchi, G. Agrawal, S. Chakradhar, ValuePack: value-based scheduling framework for CPU–GPU clusters, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2012.

[29] C. Sonia, D. Marco, G. Mehdi, G.-V. Horacio, P. Alina, Madalina, T. Massimo, Parallel patterns for heterogeneous CPU/GPU architectures: Structured parallelism from cluster to cloud, Future Gener. Comput. Syst. 37 (2014) 354–366.

[30] Z. Fan, C. Junwei, U.K. Samee, L. Keqin, H. Kai, A task-level adaptive mapreduce framework for real-time streaming data in healthcare applications, Future Gener. Comput. Syst. 43–44 (2015) 149–160.

[31] S.T. Maguluri, R. Srikant, L. Ying, Stochastic models of load balancing and scheduling in cloud computing clusters, in: IEEE INFOCOM, 2012.

[32] C. Joe-Wong, S. Sen, T. Lan, M. Chiang, Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework, in: IEEE INFOCOM, 2012.

[33] Y. Ko, T. Kim, Y. Yi, M. Kim, S. Ha, Hardware-in-the-loop simulation for CPU/GPU heterogeneous platforms, in: Proceedings of the 51st Annual Design Automation Conference, DAC'14, 2014.

[34] S. Kato, K. Lakshmanan, Y. Ishikawa, R. Rajkumar, Resource sharing in GPU-accelerated windowing systems, in: IEEE Real-Time and Embedded Technology and Applications Symposium, 2011.

[35] R. Phull, C.-H. Li, K. Rao, H. Cadambi, S. Chakradhar, Interference-driven resource management for GPU-based heterogeneous clusters, in: Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing, 2012.

[36] K. Sajjapongse, X. Wang, M. Becchi, A preemption-based runtime to efficiently schedule multi-process applications on heterogeneous clusters with GPUs, in: Proceedings of the 22nd International Symposium on High-Performance Parallel and Distributed Computing, 2013.
[37] C.-K. Luk, S. Hong, H. Kim, Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping, in: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, 2009.
[38] R. Vignesh, T.B. Michela, J. Wei, A. Gagan, C. Srimat, Scheduling concurrent applications on a cluster of CPU–GPU nodes, Future Gener. Comput. Syst. 29 (2013) 2262–2271.
[39] X. Piao, c. Kim, Y. Oh, H. Kim, J.W. Lee, Efficient CPU–GPU work sharing for data-parallel javascript workloads, in: Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion, WWW Companion'14, 2014.

Bin Wang is currently a Ph.D. student of Computer Science and Engineering at Shanghai Jiao Tong University (China). He received the M.S. degree in Software Engineering from Xi'an Jiao Tong University (China) in 2009. In 2005 he received his B.S. degree at School of Mechanical and Electrical Engineering from Xidian University (China). His research interests include virtual machine, data center network and cloud computing.

Ruhui Ma received his Ph.D. degree in computer science from Shanghai Jiao Tong University, China, in 2011. He has worked as a postdoc at Shanghai Jiao Tong University (2012 and 2013), at McGill University, Canada (2014), respectively. He is an Assistant Professor in the Department of Computer Science and Engineering at SJTU. His main research interests are in virtual machines, computer architecture and network virtualization.

Zhengwei Qi is an assistant professor in the School of Software at the Shanghai Jiao Tong University (China). He received his B.Eng. and M.Eng. degrees from Northwestern Polytechnical University in 1995 and 1999, respectively. He received his Ph.D. in Computer Science and Engineering from Shanghai Jiao Tong University in 2005. His research interests are distributed computing, virtualized security, model checking, program analysis and embedded systems.

Jianguo Yao is an assistant professor in the School of Software at the Shanghai Jiao Tong University (China). He received his Ph.D. in Northwestern Polytechnical University.

Haibing Guan now is the vice dean of School of Electronic, Information and Electronic Engineering, Shanghai Jiao Tong University, also in charge of the Ministry-province jointly-constructed cultivation base for state key lab and the Shanghai Key Laboratory of Scalable Computing and Systems. He obtained his Ph.D. degree from Tongji University in 1999 and worked in Shanghai Jiao Tong University since 2002. His major research interests include computer system, Cloud Computing. He hosted several national and ministry-province joint programs in network security, virtualization technology, network storage, data communication, etc. Research works of Prof. Guan are funded by Ministry of Science and Technology, National Natural Science Foundation, Ministry of Education and Shanghai government many times. Moreover, he keeps touch with institutes from the industry like Research Laboratory of IBM, Intel, Microsoft, HP and native large enterprises.