CoCG: Fine-grained Cloud Game Co-location on Heterogeneous Platform

Taolei Wang, Chao Li, Jing Wang, Cheng Xu, Xiaofeng Hou, Minyi Guo
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
Email: {sjtuwtl, jing618, jerryxu}@sjtu.edu.cn, {lichao, hou-xf, guo-my}@cs.sjtu.edu.cn

***Abstract*—Cloud games have received widespread attention and exponential growth recently as a key technology for building metaverse. Unlike general tasks in the cloud, the scene-complex, latency-critical, and interaction-intensive features make it challenging for cloud game co-deployment on heterogeneous platforms. Game-grained resource allocation leads to low resource effectiveness. Although previous work tries to explore individual game partitioning methods, they still face the problem of inefficient game hosting decisions and ultimately QoS violations. In this paper, we propose a fine-grained game characteristic and scheduling strategy to co-locate games together for high resource usage effectiveness. First, we fully explore the relationships between game scenes and resource usage behaviors by breaking the cloud game into stages with multiple frames and clustering them. We adopt machine learning methods to predict game resource consumption in real-time. To further improve multi-game parallelism, we co-locate games in a complementary way and steal time from the loading stage to avoid oversubscribing. The evaluation shows that our work increased the throughput of the cloud game deployments by 23.7% with low overhead compared to previous work.**

***Index Terms*—Cloud Game, Stages, Prediction, Co-location**

I. Introduction

Cloud games, as the name implies, are games that clients run on the cloud and with remote user operations. The concept of cloud game was put forward in 2016 [3]. It has been highly concerned by academia and industry as a key technology for building metaverse [19]. In the past, cloud games require expensive computing resources due to the strict requirements of network speed (<3ms) of visual display and execution latency of player interaction [40]. Additionally, in order to ensure the user experience, only one game has to be run on the same machine at the same time [29] [33]. Without parallelism and reuse, the cost will be greatly increased.

Cloud games are noteworthy because this category of applications is completely different from other workloads that have been studied before. First, cloud game is latency-critical, however, unlike the usual latency-critical task, once a game is deployed, it is difficult to migrate or stop to plan resource allocation [26]. Second, cloud game has not only high throughput but also high interactivity, so the batch processing method is not suitable for it. Third, bursty workloads like web servers or graph processing tend to focus more on parallelism design when utilizing the GPU, while cloud game tasks mainly use the GPU for rendering, using completely different channels.

Due to the particularity of cloud games, the existing scheduling algorithm cannot co-locate games well. Parties [5], a type of scheduling algorithm filling BE (Best Effort) tasks into LC (latency-critical) tasks as much as possible is not suitable for cloud games. The lack of game feature extraction results in long-term GPU resource exclusivity with low utilization in the most time, significantly wasting CPU and GPU resources. Furthermore, due to the high user impact of cloud games, the overall resource utilization will break the limit once multiple games are co-locating. The previous work [2] [15] used to solve the problem of peak overlap through fusion and migration, which is completely impossible to achieve in the game.

Dynamic design of cloud game resource scheduling is essential to handle emergencies. On the one hand, the phasing of cloud games needs to be dynamic in order to fully express the characteristics of cloud games. The length and use of these stages change dynamically, and the scheduling method needs to be adaptive to the changing stages. On the other hand, the scheduling process for cloud games makes it difficult to deal with emergent issues through a pre-defined scheduling sequence and needs to take full account of contingencies and be tailored to actual resource performance. An important question is how to avoid peak game encounters.

In this work, we present a novel methodology that enables efficient and adaptive game co-location for improving throughput in cloud games. We leverage fine-grained game characteristic extraction and adaptive prediction-based scheduling strategy to co-locate games together for high resource usage effectiveness. We break the cloud game into stages with multiple frames and clustering frames and stages according to multi-dimensional resource behavior. To further improve multi-game parallelism, we avoid oversubscription by stealing time from the loading phase.

This paper makes four important contributions:

- We take the first step to study the intra-game scenes and extract the resource usage features of cloud games.
- We propose an ML-based stage predictor for accurate stage-level resource usage prediction.
- We enhance multi-game co-location throughput by complementary deployment and peak overlap.
- We validate our work in a real cloud game environment using real games. We show that our work can greatly improve resource effectiveness.

The remainder of this paper is organized as follows. Section II introduces the background. Section III further motivates our design. Section IV proposes our system including the game

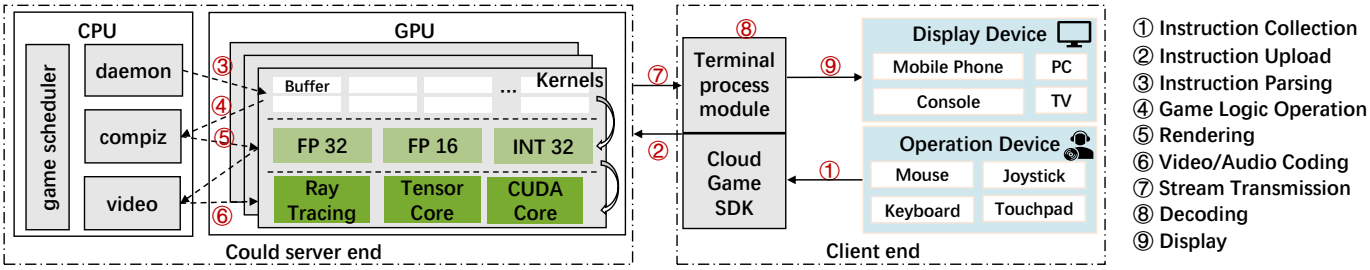


Fig. 1: The workflow of cloud game applications.

profiler in Section IV-A, ML-based stage predictor in Section IV-B, and complementary resource scheduler in Section IV-C. Section V details the evaluation methodology and shows our experimental result. Section VII concludes this paper.

II. Background

A. Workflow of Cloud Games

As is shown in Fig. 1, the deployment framework [30] of cloud games is roughly divided into three parts: cloud server end, client end, and network connection. The cloud server end is maintained locally by cloud service providers, and the entire cloud platform often consists of one cloud game scheduler and multiple cloud game backend servers. The client end is a device that players run independently locally, including a display device and an operate device, so there are various terminal devices due to different gameplay and environments. The network connection is generally managed by the operator, providing services such as SDK(Software Development Kit) and faster transmission through encoding and decoding. Fig. 1 also illustrates the workflow of cloud games. After the game is launched, every action taken by the player is recorded in the operate device and transmitted to the Terminal process module provided by the cloud game service operator. After receiving the instructions, the CPU compiles them and preloads them into the GPU buffer. Then execute the command on the game interface and perform calculations on the future development of the game, importing the results into the GPU for rendering. Generate corresponding video images from the generated game graphics, encode them, and send them back to the cloud game SDK in the player's hands. After local decoding, videos are output on the display device.

B. Complexity of Cloud Games

As shown in Fig. 1, the whole process of running a cloud game involves a variety of different hardware and software. As a result, cloud game is much more complex than other cloud applications. Through our observations, we have concluded 3 major difficulties:

- **Multiple scenes.** Due to the current computer's inability to support games loading all the models and maps involved into memory at once, the game is clearly divided into a number of scenes [22], all of which are clearly divided by loading period. As the game continues to evolve, the scenarios are becoming more complex and consume more resources such as CPU and GPU.
- **No-delay experience.** Most cloud game tasks are not only latency-critical and resource-intensive, they are also very pressure-sensitive [25]. Once the server cannot provide the resources to meet the actual demands of the game, it is easy to experience frame dropping and lagging. Users cannot tolerate this because it affects the player's operation and brings negative feedback.
- **User influence.** We cannot accurately determine the duration of each scene like other cloud applications, as this is entirely determined by the user's gaming experience [14]. Players can choose to stay in a certain scene for a long time for interaction, or quickly skip a certain scene and enter the next game scene for play. These uncertain factors have caused great trouble for us to pursue fine-grained resource scheduling.

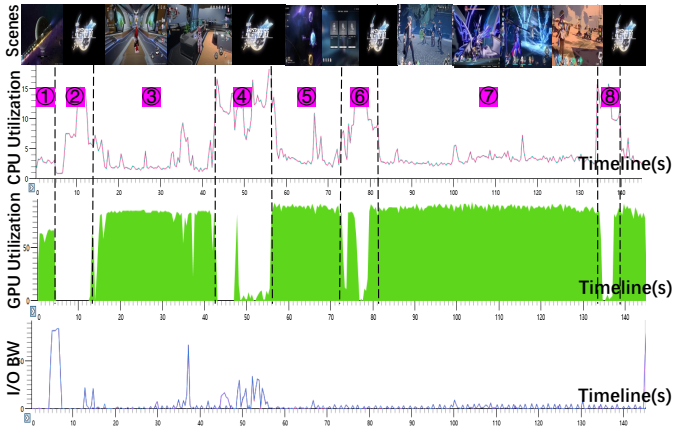


Fig. 2: The resource utilization of different game stages (Honkai: Star Rail)

Existing works only address one of the above issues; PilotFish [39] and Quasar [8] focus exclusively on the resource consumption of the game itself, hoping to reduce the overall resource utilization of the game through optimized rendering, while Ghobaei-Arani [12] and Amiri Maryam [1] focus on game delays, trying to find a balance between resource pressure and execution latency.

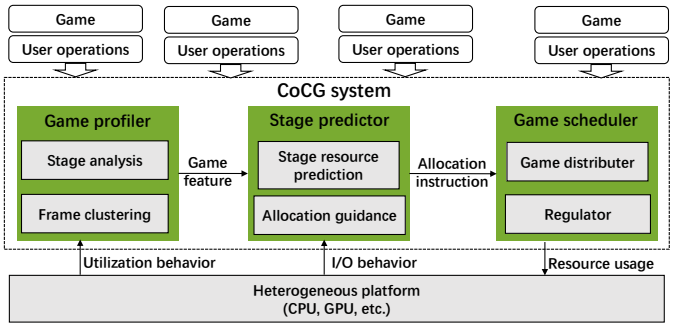


Fig. 3: CoCG system overview.

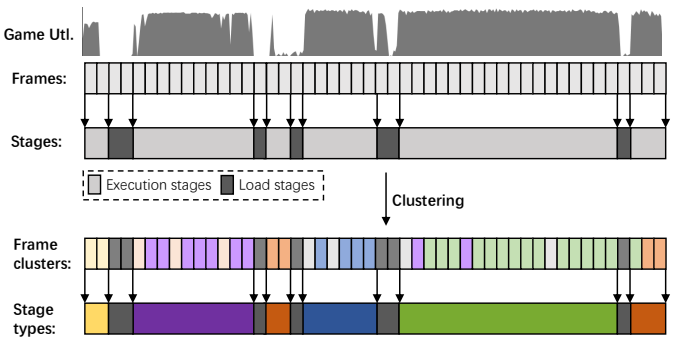


Fig. 4: The clustering of frames. Stages have different types that is the combinations of different frame clusters.

III. Preliminary and Motivation

We conducted a preliminary survey of popular games on the market and recorded the consumption of various resources during the operation of cloud games. We have the following observations.

Observation 1: Different scenes in games have distinguished resource behaviors. In the game shown in Fig. 2, there are 3 different main scenes that the game character experiences: walking in the main world (③), fighting in the instance Zones (⑤) and interacting with NPCs (⑦). The difference in resources consumed by these three scenarios is significant. This means we cannot simply treat a game as an application that consumes resources at the maximum demand all the time but need to perform a fine-grained analysis of the game itself.

Observation 2: Game stage can be separated by detecting resource loading. For user-relevance, based on Fig. 2, although it is not known how long the player will stay in a certain scene, once one leaves the current scene, the game will inevitably enter a very obvious transition stage, namely the loading stage shown in the third and fifth stages in the figure. It is inevitable to take some time to clear the previous scene data and preload the model for the next scene when entering a new scene. This gives us a basis for dividing the stage in real-time.

Observation 3: We can improve resource usage effectiveness by fine-grained resource co-location. As shown in Fig 2, the third, fifth, and seventh stages are three different game scenarios, with different CPU and GPU resource consumption characteristics. The second, fourth, sixth, and eighth stages are the loading stages, which are the loading interfaces when the game switches scenes. At this time, the CPU consumption is the highest because all content in the next scene needs to be pre-calculated. At the same time, GPU resource consumption is relatively low, as the screen appears black and rendering is not necessary. Co-locating these different resource usage stages can utilize more computing resources that improve system effectiveness.

Observation 4: We can steal time from loading stages to avoid peak usage encounters. In most cases, there are loading stages that prepare for the entire game to load resources from disks. While any bit of lag is not tolerated when the game is running at its peak and players are actively interacting, reducing the resource supply at the loading stage and extending the time appropriately is not overly offensive to players [32]. So when there is a shortage of resources, we prefer to extend multiple loading stages rather than cut the resource allocation of a game that is at its peak [35].

For a level-based game, these observations are easy to understand. The clusters with low resource consumption correspond to the loading in the game, while the high consumption of different resources corresponds to these different levels. For open-world games, stages don't seem to exist. However, with further research, we can still discover the game's internal logic. According to Google Cloud Game Services [18], the GPU is mainly responsible for real-time rendering while the game is running. When the GPU renders an open-world game, it tends to focus on the environment around the characters, while the background mainly relies on blurring. Engaging in a boss fight or entering a new map can cause a significant change in resource consumption, which can be detected by our system. In our work, open-world games are treated as phased games with particular longer running stages.

Based on the above observations, we have designed a new cloud game characterizing and scheduling strategy to co-locate games together for higher resource usage effectiveness.

IV. System Design

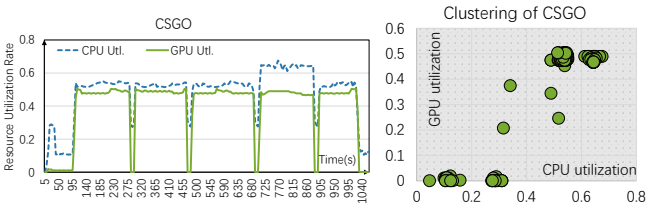
Our system maintains three components, including the frame-gained game profiler, ML-based stage predictor, and complementary resource scheduler, as shown in Fig. 3.

A. Frame-gained Game Profiler

In this section, we introduced cloud game stages, including what stages are in cloud games, the types of stages, and the specific situations corresponding to different stages.

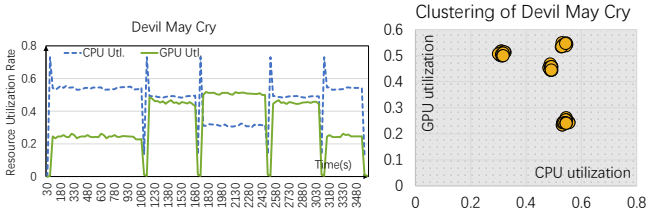
1) Game stage analysis: In section III, we observed that game resources can be clearly divided, and therefore proposed the concept of frames and stages. as shown in Fig. 4.

Firstly, there are obvious two types of stages in cloud games. The first type is the loading stage, which generally includes initialization, runtime loading, and shutdown. The second type is the execution stage, which is the time during which the



(a) CPU and GPU resource utilization. (b) Clusters

Fig. 5: Stage types of CSGO game by clustering.



(a) CPU and GPU resource utilization (b) Clusters

Fig. 6: Stage types of Devil May Cry game by clustering.

game interacts with the player normally. Both of these stages are crucial for predictive scheduling, as the first type of stage fixedly divides the second type of stage, meaning that once a game is detected as a loading stage, we should reassign appropriate resources to accommodate its next execution stage.

For the execution stage, in most cases, only one scene and one cluster are included, meaning that the resource consumption within the same game scene remains unchanged. Most games conform to this characteristic, but there are also special circumstances.

The first situation is that a stage includes multiple clusters and one scene, as shown in Fig. 4. In this stage, the game is very complex and requires multiple resource clusters to represent. It is worth noting that, generally speaking, the cluster logic inside this stage is very clear, but the order is not necessarily fixed. For example, this stage indicates that the player has entered a big secret realm where they need to defeat three fixed bosses in order to leave. Due to the different mechanisms of each boss and the different monsters summoned, the resource consumption corresponding to the three bosses varies. At this point, players will inevitably defeat three bosses and eventually leave the stage, but the order may be different for each player, completely influenced by temporary judgments.

The second situation is that a stage includes one cluster and multiple scenes, also shown in Fig. 4. This situation indicates that the game may have some seemingly different but similar resource consumption scenes during the game process, which are included in the same stage. For this stage, the general situation is that the player is indifferent to small parts of a large map. Although there is a significant difference in gameplay, the resource consumption is similar due to the consistency of the map model and background, so it can be included in the same stage for consideration. If the situation is that multiple clusters and scenes in a single stage, we can decompose them into separate stages.

After completing the division of all stages for a certain game, we can smoothly make real-time predictions for the entire gameplay.

2) Game frame clustering: We directly clustered the resources using the K-means algorithm in Fig. 5b and Fig. 6b from the date in Fig. 5a and Fig. 6a. However, there is no logical relationship between the clustering results. Therefore, we introduced game stages in the hope of guiding resource allocation through the game's internal engine. We found that there is no one-to-one correspondence between the cluster and the scene, so our cloud game stage integrates resources and content to provide a new perspective on the game.

Each frame cluster represents the amount of resources consumed in a certain 5-second slice. This information is the result of being clustered, as shown in Fig. 4. As with the above classification of execution stages, there may be multiple frame clusters that together form a single stage, and this combination of frame clusters will continue to occur in subsequent stages. If the game has N clusters, then it has at most 2^N stage types; however, the experiments in Fig. 5 and 6 show that in most cases the number of stage types does not exceed 2N, which is a great help in predicting the stage.

B. ML-based Stage Predictor

Our stage predictor is a real-time system that detects at 5-second intervals. The determination of the 5-second interval time is because, in our experiment, all loading stage times were higher than this, so a 5-second detection can definitely identify the loading stage. As shown in Fig. 8, the operation of the entire predictor is mainly divided into four steps, Real-time data collection, stage judgment, next-stage prediction, and resource adjustment. The first step, real-time data collection mainly involves collecting the CPU, GPU, and memory consumption of the game running, and transmitting the data to the predictor. Then, The predictor compares the data to determine whether the game is still in the same execution stage or has entered the loading stage. If it is still in the original stage, it will directly return to the first step and wait for detection again. If it is found that it has already entered the loading stage, it will proceed to the next step. Combine all previous stages of the game and input them into the offline trained machine learning prediction model to obtain the prediction results for the next execution stage. Finally, based on this prediction result, reallocate the resources consumed by the game.

1) Prediction algorithm: As mentioned earlier, the biggest difference between games and other applications is that users' decisions have a very significant impact on them. Although we have reduced the user's impact on game duration by identifying and loading stages, the user's choice can even affect the order of stages. How to minimize this user impact in prediction requires us to classify the game and select different data as samples for training based on different game types, in order to achieve high accuracy.

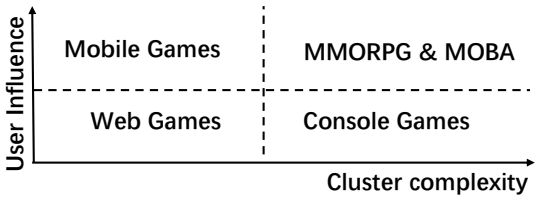


Fig. 7: Game Category

Fig.7 shows our four divisions of game types. The horizontal axis represents the complexity of the game stage type, which becomes more complex as it moves to the right. The vertical axis represents the player's level of influence on the game, with a greater impact as it moves upwards. This classification is strategically designed to filter the input dataset, thereby enhancing the accuracy of cluster predictions.

Prediction for web games. In the lower-left corner of Fig.7, it represents games with relatively simple stages and less user influence. This type of game is usually a flash or web game, characterized by a short total playing time, a single game process, less player interaction, and low resource consumption. Representative works include Contra and Raiden. This type of game is very easy to predict because the game runs with a relatively single stage (usually no more than 3) and player behavior does not affect the game process. Therefore, we can train all player's game records as a training set, and the results obtained will be relatively fixed.

Prediction for mobile games. In the upper-left corner of Fig.7, it represents a game with a relatively single stage but significant user influence. These games are usually mobile games, characterized by players logging in to complete tasks every day, but the order in which tasks are completed may vary greatly among different players. Representative works include Genshin Impact, and Arknights. For this type of application, we need to finely establish a training set for each individual player and make predictions based on this. Since players of this type of game may log in every day for several years, this prediction is also very important and can be done once and for all.

Prediction for console games. In the lower-right corner of Fig.7, it represents games with complex stages but less user influence. These games are usually big console games. Representative works include "Devil May Cry 5" and "Resident Evil 2". They are characterized by a total game duration of several days and a large number of game levels. However, players are only stuck in a few big scenes for long periods. Their choices in the game are unlikely to affect the switching of game stages. For this type of game, we can no longer use the data of a single player playing a single game as a sample. Instead, we need to connect all the processes of the player playing the game, integrate the data of the player's entire process, and incorporate it into the training set as an element.

Prediction for MMORPG & MOBA games. In the upper-right corner of Fig.7, it represents games with complex stages and significant user influence. These games are usually MMORPG or other multiplayer online games. Its characteristic is that multiple players interact in the same game area at the same time, and the interaction between players will affect the next stage. Representative works include World of Warcraft and DOTA2. For this type of game, considering only the playing process of a single player will definitely not yield a definite result. Therefore, we will package the data of several players who log in to the game at the same time, integrate them into one, and then put them into the training set.

We use 3 popular machine learning algorithms to train the models, including Decision Tree Classifier (DTC), Random Forest (RF), and Gradient Boosted Decision Trees(GBDT). The performance of these 3 algorithms is evaluated in Section V. We measure a number of real game co-locations to train the models. One thing worth mentioning is that contention feature profiling and model training only need to be performed once. After the models are trained, we can use them for prediction with negligible overhead.

2) Dynamic adjustment: No matter how accurate the prediction algorithm is, it will eventually encounter errors in prediction. For this reason, we have designed three emergency plans to revise the prediction results.

Rehearsal callback. When the prediction error occurs, a third situation different from the normal result will be found in the next stage adjustment, that is, the real-time operation data collected at this time is quite different from the current stage, and it is not in the loading stage. At this time, the callback is divided into two possibilities. The first is the stage error caused by the prediction error. At this time, the real-time running data will be re-matched to the correct stage, and jump to a new stage immediately after the next detection is completed. The second error is that the real-time running data result is similar to the loading stage, which leads to the misjudgment of entering the next stage. At this time, the resource allocation is returned to the previous stage immediately.

Redundancy allocation. When a prediction error occurs, the utilization of callback resources cannot simply be set to a regular value, but rather a redundant S should be added to this foundation. This redundancy is determined by the predicted accuracy P (%) of the game and the peak resource consumption M of the game shown as Eq. 1:

$$S = (1 - P) \times M \tag{1}$$

which means when the accuracy is higher, the redundancy is smaller, and the specific value of redundancy depends on the consumption of the game itself.

Replacing model. If there are still many errors after using the above two methods, the other prediction algorithm among DTC, RF, and GBRT should be used to make a new judgment. For tasks with a large amount of computation and a long running time, the algorithm of DTC is more suitable. For simple, small tasks, it is more suitable for RF algorithms. GBRT is relatively stable, so it is more suitable for games with a large impact on users.

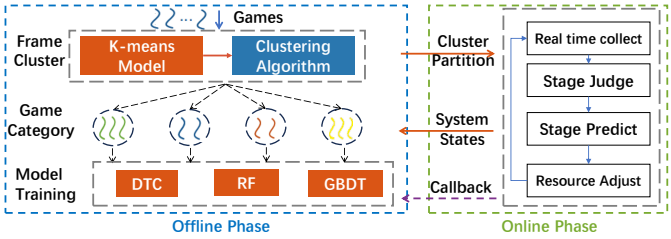


Fig. 8: Design of ML-based stage predictor

C. Complementary Resource Scheduler

Typically, each game is deployed on a single GPU device rather than across multiple GPUs to satisfy Quality of Service (QoS) requirements. [18]. With a single game timeline, there is more space for optimization for multiple game co-locations than the crude approach of only consuming the maximum amount of resources [16]. Our timeline-aware multi-game co-location consists of two main components, a distributor and a regulator. The distributor determines whether a pending game can be assigned to a server that is already running games, while the regulator is used to resolve spikes as soon as they occur.

1) Game distributor: When a game needs to be allocated to a running server, it makes sense only when they will consume fewer resources than the server can provide for all the time when running together. Previous practices have either sacrificed the QoS of some games in exchange for the overall resource usage to meet the condition [12] or pulled up the resource requirements of individual games to ensure they are all met, which means wasting a lot of resources [31]. With fine-grained predictions, we can estimate the resources of the stage it is likely to enter, based on the stage it is currently running at. The specific details are shown in the Algorithm 1.

2) Regulator: Although our distributor has tried to avoid multiple game peaks overlapping as much as possible, it is bound to happen that peaks cannot be completely overlapped due to the inherent high uncertainty of the games [34]. This may be due to errors in prediction or due to some out-of-the-ordinary behavior of the players. When such problems occur, we need to introduce regulators to make timely adjustments. Regulators use two main strategies:

Extend loading time. Users are more tolerant of appropriately extending the loading time compared to dropping frames at peak times. So once we find that peak staggering is unavoidable, we can make the loading stage of the game take longer by reducing the resource allocation for the current stage. Although a single loading stage extension does not completely stagger the peaks, as our prediction algorithm is forward-looking, we can overlap the peaks as much as possible by spreading the time over multiple loading stages.

Distinguish game length. Although we have stressed that the feature of game length is very much influenced by the user, it is worth noting that game manufacturers often inform players of the expected play time of the game at the time of release. This suggests that it is feasible to distinguish between long-term and short-term games at a coarser granularity. While long-term game runs, when we determine that the current stage has reached a peak in resource consumption, and the next peak stage is still a long time away according to the prediction, we can deploy as many short-term games as possible between the two peaks to make better use of resources and increase the overall throughput.

Algorithm 1 Pseudocode of Distributor.

Input: the i^{th} game, denoted by S_i
Output: Whether this game can be distributed into a current server, denoted by P
tasks.group(stage, cluster)
For tasks in type.history_list **do**
    Sum += $Consumption_i$
    tasks.group(stage, cluster)
End For
If Sum $> Consumption_{Total}$ **then:**
    P = false
While tasks $\neq \emptyset$ **do**
    G = list{type.history}
    N = Total.iteration
    For each N $\in$ G **do**
        NextStage = prediction.G
        G.append = NextStage
    End For
    M = maxG.Consumption
    If M + $Consumption_{S_i} > Consumption_{Total}$ **then:**
        P = False
        break
    Else:
        P = True
    End If
    task.pop
End While

D. Discussion

Admittedly, once two games with a sum of peak consumption greater than the total resources are co-located together, no matter how finely the scheduling is done, there is bound to be performance degradation due to peak interleaving. Fortunately, players can tolerate the loss of performance to a certain extent as long as they are compensated. In practice, therefore, cloud game operators are often comfortable balancing the player experience by distributing game credits or other benefits, as long as the performance degradation occurs for less than 5% of the total time [18].

Moreover, when considering scales for larger servers with more CPUs, GPUs, and also more games that are co-located, our work is more expansive than the previous work [8] [39]. This is because when our pre-experiment analyzes the stage characteristics of the game for a specific GPU and CPU, no matter what platform the game is migrated to, the number of stages and the logical relationship between the stages will not

change. These metrics are determined by the content of the game itself. The only thing that will change is the amount of resources consumed, which can be obtained in a single experiment. So the matter of moving games between different platforms is very easy for our algorithm.

V. Evaluation

A. Experimental Setup

Software and hardware Environments. Our workloads run on a 4-core Intel i7-7700 CPU, 8GB RAM, 2 NVIDIA GeForce GTX 2080 GPUs, and Linux OS. Specifically, we consider both CPU and GPU resources. The CPU can be measured by cores or by CPU usage, using the Linux cgroup tool. GPU-Z [20] is involved in validating parameters including GPU and GPU memory utilization.

We use GamingAnywhere (GA) [17] to build the cloud game environment, which is a popular open-source cloud game platform. The GA has two components: the GA server (for running games, encoding, and streaming videos) and the GA client (for decoding and displaying videos and transmitting user commands).

Considering different CPUs and GPUs, or the impact of the version of the game, and graphical settings, the specific amount of resources consumed by the same game must be different. However, based on our pre-experiments, we know that these factors do not affect the division of stages in our system.

Workloads. We study a suite of 5 cloud games that are previously used in similar studies, which are DOTA2, CSGO, Genshin Impact, Devil May Cry, and Contra. DOTA2 is a 3D Multiplayer Online Battle Arena(MOBA) game and CSGO is a 3D First Person Shooting(FPS) game. They all belong to games with complex stages and significant user influence. Genshin Impact is currently the most popular mobile game in the world. Devil May Cry is an Action Role Playing Game(ARPG) game and one of best selling Console games. Contra is the most classic entry game for most people.

Performance measurement methodology. We have designed a total of 3 schemes for the experiment. The modest way is to default that each cloud game consumes the same resources from the start of the operation to the end of the application and allocate them based on this (as the solution in GAugur). This method is our baseline. The second scheme perceives that each game has different resource consumption stages at runtime but does not predict the next stage at the time of scheduling, and only redeploys the resource usage based on the current operation(improved version). The last method is our newly proposed scene resource model. On the premise of comprehensively considering the corresponding relationship between the scene and resource use, the machine learning prediction is made for the layout of the scene to better guide the implementation of the scheduling scheme.

B. Overall Improvement

1) Resource utilization of single games: Holding the stage prediction in Section IV-A, we can allocate resources for running games. Fig. 10 shows the comparison between the allocated resources and the actually required resources in Genshin impact. The resources we allocate basically cover the actual resources consumed by the game. Moreover, compared to always allocating space based on the maximum resource consumption of the game (65%), our solution has saved a total of 27.3% of resources in this case. Overall, for the five games tested, it can save an average of 17.5%. These flexible resources can be allocated to tasks with low latency-critical tasks such as machine learning and graph computing, thereby improving the resource utilization of the entire cloud platform.

It is worth mentioning that during the period from 300 seconds to 500 seconds in Fig. 10, our work underwent three brief resource allocation increases. The reason for this phenomenon is that there was a rapid resource fluctuation during the game period, causing the program to mistakenly assume that it had entered a new phase and made a jump. However, this misjudgment was quickly adjusted after detecting that the actual phase was different from the current judgment phase, jumping back to the original phase. This leads to strong robustness.

2) Resource utilization of multiple games: In order to continuously distribute game tasks on the real machine, we have designed scripts for each game to run automatically. In order to increase the complexity of the situation, we have designed different scripts for the same game as shown in Table I. When a game is assigned, it randomly selects one from the scripts to execute. This greatly increases the complexity of our system. At the same time, it also shows that our scheduling algorithm is not only suitable for transparent and processed data but also has good scalability.

- DOTA2. We design 2 scripts, one is conducting a match with 9 bots, and the other one is playing a tower defense game in the arcade.
- CSGO. We design 2 scripts, one is conducting a match with 9 bots, and the other one is moving in the training map without shooting.
- Devil May Cry. We design 3 scripts, one is passing the first level in simple mode, another one is passing the

TABLE I: Evaluated Workloads

Game	Scripts	Script descriptions	# of stage type
DOTA2	script 1	conducting a match with 9 bots	3
	script 2	playing a tower defense game in the arcade	3
CSGO	script 1	conducting a match with 9 bots	4
	script 2	moving in the training map without shooting	3
Devil May Cry	script 1	first level in simple mode	2
	script 2	second level in simple mode	4
	script 3	third level in simple mode	6
Genshin Impact	script 1	run + battle + fly	5
	script 2	fly + battle + run	5
	script 3	battle + run + fly	5
Contra	script 1	first level	2
	script 2	first two levels	2
	script 3	first three levels	2

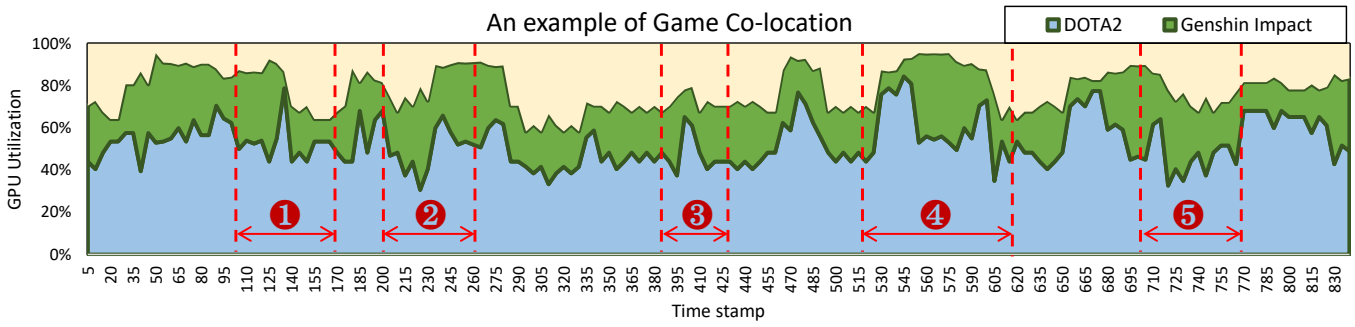


Fig. 9: Co-location of Genshin Impact and DOTA2. Our scheduler can co-locate two games with maximum utilization of 78% and 43%, and keep below the upper limit.

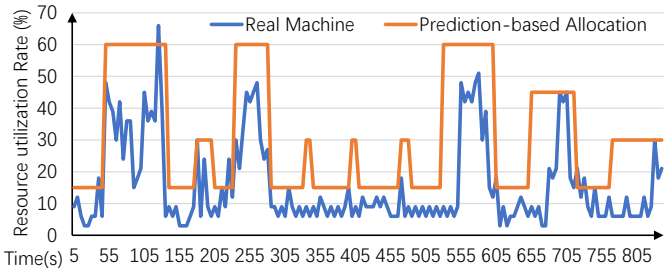


Fig. 10: Genshin Impact prediction allocation

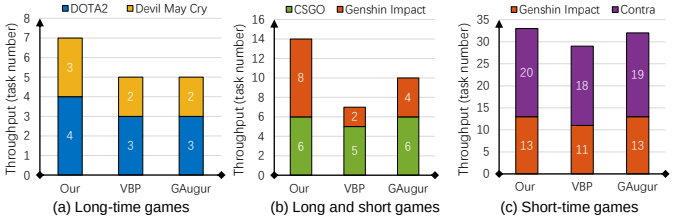


Fig. 11: Throughput of games co-location. The throughput of our work is 23.7% higher than that of others.

second level in simple mode, and the last one is passing both the first and second levels.

- Genshin Impact. We design 3 scripts, with 3 different orders to complete 3 same tasks.
- Contra. We design 3 scripts, one is passing the first level, another one is passing the first two levels, and the last one is passing the first three levels.

Fig. 9 shows one of our co-location results with Genshin Impact and DOTA2. We can see that Genshin Impact has a maximum resource consumption of 78% while Dota 2 is 43%. The overall resource consumption of the two games' co-location does not exceed 95%. This shows that our fine-grained scheme can better achieve peak shaving. In the first period, both of the games are in the running stage, while Dota 2 is in a higher consumption stage and Genshin Impact consumes less at this time. The second period is just the opposite, Genshin Impact enters a higher consumption stage and Dota 2 consumption begins to decrease. During the third period, our scheduler detects the rapid increase of DOTA2, so we switch the Genshin Impact stage to a lower one. However, the later performance shows that it is just a sudden event, not a change of stage. So the consumption of Genshin Impact returns immediately. At the beginning of the fourth period, DOTA2's resource consumption reached its peak, and in order not to break through the upper limit, the system arranged Genshin Impact to stay in the loading stage for an extra 15 seconds. Genshin Impact does not enter the next stage until the peak is over. In the fifth period, Genshin Impact took advantage of Dota 2's loading stage to start a new task.

We compare our methodology with the following state-of-the-art alternatives:

- Vector Bin Packing(VBP). VBP assumes that the game can run normally at 90% of its maximum resource consumption. At the same time, an application can be assigned to a server only when the server's remaining resources are higher than the peak of the application.
- GAugur. GAugur is a performance model proposed by [23]. it uses profiling to predict whether two games can be co-located on the same server. After allocating two games on a server, GAugur also assigns a fixed resource limit to each game through machine learning algorithms.

Combine the 5 games we have selected in pairs for co-location, and there are a total of 10 scenarios. However, there are multiple situations where both games consume a lot of resources for a long time and cannot run on the same machine. Therefore, we selected three scenarios in Fig. 11 as representatives for detailed analysis.

The experiment in Fig. 11 shows two-hour duration results of tasks completed by three game combinations (DOTA2 + Devil May Cry, CSGO + Genshin Impact, and Genshin + Contra) under different co-location strategies and the total duration of performance loss. During these two hours, the selected game will continuously run requests until the distributor passes the request and starts running.

We use throughput T in Eq.2 to measure our work

$$T = \sum_{i=1}^{k} N_i \cdot S_i \tag{2}$$

where k represents the number of different games running on the server, N_i represents the number of the i^{th} game running in 2 hours, and S_i represents the duration of the i^{th} game.

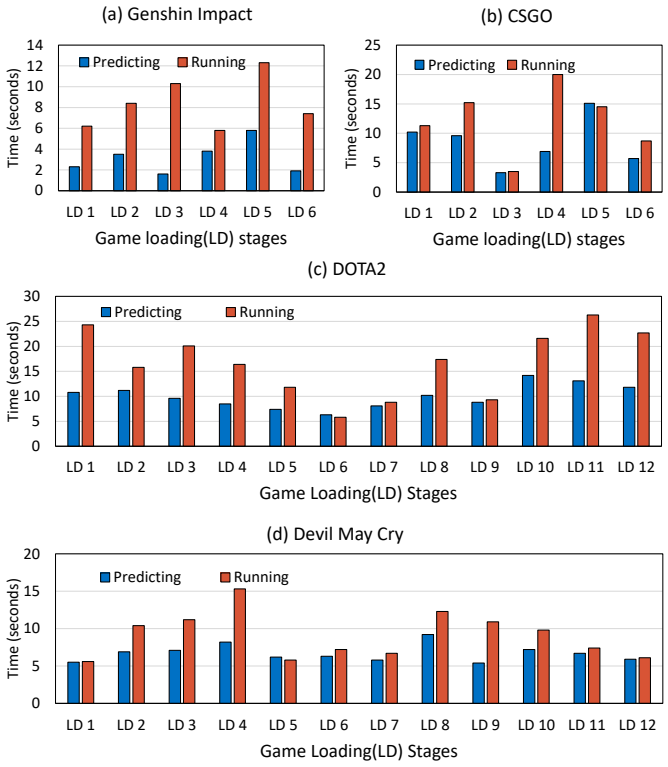


Fig. 12: Overhead of Scheduling

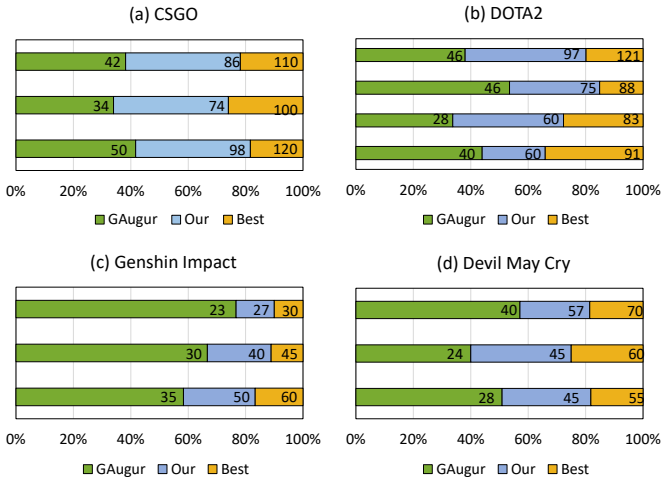


Fig. 13: FPS of Co-location Games

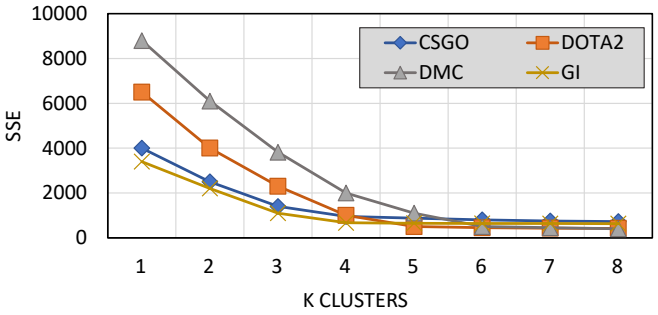


Fig. 14: Clustering result with different K value. The SSEs remain few changes when K $>$ 5.

Regarding the combination of DOTA2 and Devil May Cry, the sum of the peak resource consumption of these two games is very large. Therefore only our work co-locates them as much as possible, other solutions can only be executed individually. The combination of CSGO and Genshin Impact is a combination of long-term games and short-term games. Our work can try to insert short-term games as much as possible between the peak periods of long-term games, resulting in a significant increase in the number of runs of Genshin Impact. And the third combination, as it is two combinations with less resource consumption, all three schemes have good performance. According to the above formula, the throughput of our work is 23.7% higher than that of others. Overall, our work increased the throughput of cloud games with acceptable performance losses.

C. Overhead

1) Scheduling overhead: For our overall scheduling process, since the predictor training part is mainly offline, the main time overhead of the real-time system is the prediction between stage and stage. As mentioned in Section IV-A1, loading time tends to be 5s-30s between stage and stage, which provides the conditions for us to detect stage transitions. At the same time, it means that as long as our prediction time is shorter than the loading time, the overhead of our scheduling can be reduced as much as possible. Fig. 12 illustrates the average time for the four games to be evaluated at different stages of loading and prediction time of resource usage. The predicting time is between 3s-13s, which is basically completely covered by loading time. This means that the time overhead of our scheduling system is perfectly acceptable.

2) QoS overhead: For cloud game applications, QoS (Quality of Service) is often measured by FPS(Frames Per Second) [22]. The average gamer's demand for FPS is a minimum of 30 frames, and reaching 60 frames is considered to achieve the ideal performance. Previous work [23] has also measured cloud game applications by this standard. Fig. 13 compares our method with GAugur. For a game, the maximum number of frames it can achieve will vary depending on the stage while providing it with sufficient CPU and GPU resources. Our experiments covered all 4 games as much as possible, and the FPS of the games reached 78% of the best performance after scheduling, compared to only 43% of GAugur. For the two games Genshin Impact and Devil May Cry, the manufacturer locked 30/60 frames in order to adapt to more devices. In this case, our solution ensured that the game ran at a frame rate of more than 30 frames as much as possible, meeting the minimum standards. For CSGO and Dota 2, there is no upper frame limit, so our solution can reach more than 60 frames to obtain the ideal QoS. The experimental results show that the two strategies we used in section IV-C2 significantly improve the QoS of co-location games.

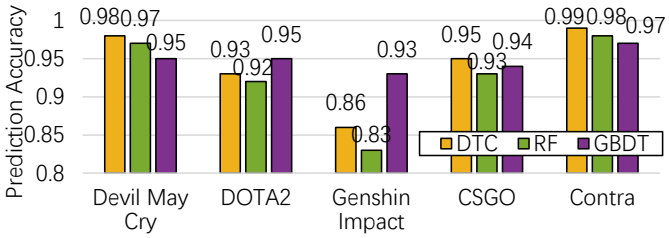


Fig. 15: Prediction Accuracy

D. Performance Breakdown

1) Impact of clustering methods: We used the K-means method to cluster each frame into different clusters. K-means demonstrated significantly higher accuracy compared to other clustering methods like Graph Partitioning, which does not require the number of clusters. Additionally, we observed a trade-off in the K-means algorithm: while a larger number of clusters leads to higher accuracy, it also incurs more execution overhead.

To determine how many clusters each game should be divided into, we conducted experiments using different k values. As shown in Fig. 14, there are obvious inflection points in all four images. The existence of these inflection points indicates that there are indeed similar characteristics of resource consumption in the game, and also guides us to choose the appropriate k value. In the end, we chose to divide Contra into 2 clusters(Resource consumption of this game changes very little during the run, so it is divided into two clusters: the loading and the running.), CSGO and Genshin Impact into 4 clusters, DOTA2 into 5 clusters, and Devil May Cry into 6 clusters. This approach not only conforms to our intuitive understanding of games but also has algorithmic support.

2) Impact of prediction algorithms: We next show the prediction accuracy of the machine learning model. Given the current execution stage and previous stage history, the model predicts the next game-running stage. As mentioned in Section IV-B, for different types of games, we select different samples for training. Our data consists of two parts. One is the user information and corresponding resource consumption collected during the running of the Alibaba Cloud game platform, and the other is the data collected through repeated games in the laboratory. Among the samples generated, we used 75% of them (randomly selected) to train the model and the remaining 25% for testing. We applied DTC, RF, and GBDT machine learning algorithms to build the machine learning model. Fig. 15 presents the mean prediction accuracy produced by different machine learning algorithms for the tested games. As can be seen, DTC achieves an accuracy of over 92% in most cases, indicating that the model can predict the next stage of a game. Also, we find that although DTC and RF have lower accuracy in Genshin Impact, GBDT remains as is. This is because Genshin Impact has a complex real-world environment and requires more in-depth iteration.

VI. Related Work

A. Cloud Game

Resource management issues in cloud games have been extensively studied, including request allocation, scheduling, server configuration [2] [9] [11] [15] [24] [28] [36]. However, the correspondence between game content and resource consumption is not taken into account in these works. Some simple scheduling strategies were employed, including Disallowing Co-location and Vector Bin Packing policies [9] [11] [24]. However, these tactics either lead to over-resource scheduling or QoS violations. Performance predictions for coexisting games were studied in previous work [2] [15]. However, their model simply assumes that performance degradation depends only on the number of co-located games, so it can produce large prediction errors.

B. Prediction-based Scheduling

Extensive research [6] [7] [8] [13] [27] [41] [37] has been conducted on the prediction of performance disturbances caused by contention of shared resources. However, since games are very different from general applications, applying existing methods to our problem is not useful for many reasons. Cuanta [13], and Bubble-Flux [37] cannot capture competing behavior between multidimensional resources. Bubble-Up [27] and SMiTe [41] can only handle co-location for two applications. Paragon [7] and Quasar [8] rely on the assumption that application capability is additive is incorrect for cloud games. Machine learning techniques are used to build performance prediction models. However, because these models do not take into account the inherent nature of cloud games, they produce a high prediction error.

C. GPU Co-location

Techniques have been proposed in previous work to improve GPU utilization. TimeGraph [21], Baymax [10], vGASA [38], and VGRIS [28] schedule GPU tasks for collaborative workloads to improve hardware utilization while guaranteeing QoS. These technologies complement our work. Prophet [4] can predict performance interference for co-located applications on GPUs. However, it does not apply to our problem since predictions are not done in real-time. DJay [14] dynamically adjusts the game settings of the shared game as the game progresses to accommodate changes in the game scene, thereby improving performance. However, it focuses on a set of well-designed games, which cannot apply in real data centers. Our approach and dJay [14] are complementary.

VII. Conclusion

In this work, we have presented a novel strategy that enables efficient and adaptive resource scheduling for cloud games. We break the cloud game into stages by frame clustering. Our work leverages machine learning technology to real-time predict game resource consumption. To improve multi-game co-location, our work steals time from the loading stage to avoid oversubscribing. The experimental results show that

our work increased the 23.7% throughput of the cloud game deployment compared to the prior work at low overhead.

ACKNOWLEDGMENT

We sincerely thank all the anonymous reviewers for their valuable comments. This work is supported by the Shanghai S&T Committee Rising-Star Program (No.21QA1404400) and the National Natural Science Foundation of China (No.62122053). The corresponding author is Chao Li.

REFERENCES

[1] M. Amiri, H. Al Osman, S. Shirmohammadi, and M. Abdallah, "An sdn controller for delay and jitter reduction in cloud gaming," in *Proceedings of the 23rd ACM International Conference on Multimedia*, ser. MM '15, 2015, p. 1043–1046.

[2] M. Basiri and A. Rasoolzadegan, "Delay-aware resource provisioning for cost-efficient cloud gaming," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 972–983, 2018.

[3] W. Cai, R. Shea, C.-Y. Huang, K.-T. Chen, J. Liu, V. C. M. Leung, and C.-H. Hsu, "The future of cloud gaming [point of view]," *Proceedings of the IEEE*, pp. 687–691, 2016.

[4] 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," *SIGARCH Comput. Archit. News*, p. 17–32, apr 2017.

[5] S. Chen, C. Delimitrou, and J. F. Martínez, "Parties: Qos-aware resource partitioning for multiple interactive services," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19, 2019, p. 107–120.

[6] Y. Cheng, W. Chen, Z. Wang, and Y. Xiang, "Precise contention-aware performance prediction on virtualized multicore system," *J. Syst. Archit.*, p. 42–50, jan 2017.

[7] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," *SIGPLAN Not.*, p. 77–88, mar 2013.

[8] ——, "Quasar: Resource-efficient and qos-aware cluster management," p. 127–144, feb 2014.

[9] Y. Deng, Y. Li, R. Seet, X. Tang, and W. Cai, "The server allocation problem for session-based multiplayer cloud gaming," *IEEE Transactions on Multimedia*, pp. 1233–1245, 2018.

[10] G. A. Elliott, B. C. Ward, and J. H. Anderson, "Gpusync: A framework for real-time gpu management," in *2013 IEEE 34th Real-Time Systems Symposium*, 2013, pp. 33–44.

[11] D. Finkel, M. Claypool, S. Jaffe, T. Nguyen, and B. Stephen, "Assignment of games to servers in the onlive cloud game system," in *2014 13th Annual Workshop on Network and Systems Support for Games*, 2014, pp. 1–3.

[12] M. Ghobaei-Arani, R. Khorsand, and M. Ramezanpour, "An autonomous resource provisioning framework for massively multiplayer online games in cloud environment," *Journal of Network and Computer Applications*, pp. 76–97, 2019.

[13] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ser. SOCC '11, 2011.

[14] S. Grizan, D. Chu, A. Wolman, and R. Wattenhofer, "Djay: Enabling high-density multi-tenancy for cloud gaming servers with dynamic cost-benefit gpu load balancing," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, ser. SoCC '15, 2015, p. 58–70.

[15] H.-J. Hong, D.-Y. Chen, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu, "Placing virtual machines to optimize cloud gaming experience," *IEEE Transactions on Cloud Computing*, pp. 42–53, 2015.

[16] X. Hou, L. Hao, C. Li, Q. Chen, and M. Guo, "Power grab in aggressively provisioned data centers: What is the risk and what can be done about it," in *International Conference on Computer Design*, 2018.

[17] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "Gaminganywhere: An open cloud gaming system," in *Proceedings of the 4th ACM Multimedia Systems Conference*, ser. MMSys '13, 2013.

[18] G. Inc., "Google cloud game," https://cloud.google.com, 2017.

[19] M. Inc., "Metaverse," https://www.pwc.com/us/metaverse, 2021.

[20] T. Inc., "Gpu-z," https://www.techpowerup.com/gpuz/, 2021.

[21] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, "Timegraph: Gpu scheduling for real-time multi-tasking environments," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, 2011.

[22] Y. Li, H. Liu, X. Wang, L. Pu, T. Marbach, S. Tang, G. Wang, and X. Liu, "Themis: Efficient and adaptive resource partitioning for reducing response delay in cloud gaming," in *Proceedings of the 27th ACM International Conference on Multimedia*, ser. MM '19, 2019.

[23] Y. Li, C. Shan, R. Chen, X. Tang, W. Cai, S. Tang, X. Liu, G. Wang, X. Gong, and Y. Zhang, "Gaugur: Quantifying performance interference of colocated games for improving resource utilization in cloud gaming," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '19, 2019.

[24] Y. Li, X. Tang, and W. Cai, "On dynamic bin packing for resource allocation in the cloud," in *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '14, 2014, p. 2–11.

[25] Y. Li, C. Zhao, X. Tang, W. Cai, X. Liu, G. Wang, and X. Gong, "Towards minimizing resource usage with qos guarantee in cloud gaming," *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[26] T. Liu, S. He, S. Huang, D. Tsang, L. Tang, J. Mars, and W. Wang, "A benchmarking framework for interactive 3d applications in the cloud," in *2020 53rd Annual IEEE/ACM MICRO*, 2020.

[27] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *2011 44th Annual IEEE/ACM MICRO*, 2011, pp. 248–259.

[28] Z. Qi, J. Yao, C. Zhang, M. Yu, Z. Yang, and H. Guan, "Vgris: Virtualized gpu resource isolation and scheduling in cloud gaming," *ACM Trans. Archit. Code Optim.*, jul 2014.

[29] R. Ren, X. Tang, Y. Li, and W. Cai, "Competitiveness of dynamic bin packing for online cloud server allocation," *IEEE/ACM Transactions on Networking*, pp. 1324–1331, 2017.

[30] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui, "Cloud gaming: architecture and performance," *IEEE Network*, pp. 16–21, 2013.

[31] S. Shi, C.-H. Hsu, K. Nahrstedt, and R. Campbell, "Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming," in *Proceedings of the 19th ACM International Conference on Multimedia*, ser. MM '11, 2011, p. 103–112.

[32] K. T. and Mark, "On frame rate and player performance in first person shooter games," *Multimedia Systems*, 2007.

[33] X. Tang, Y. Li, R. Ren, and W. Cai, "On first fit bin packing for online cloud server allocation," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016, pp. 323–332.

[34] J. Wang, C. Li, Y. Liu, T. Wang, J. Mei, L. Zhang, P. Wang, and M. Guo, "Fargraph+: Excavating the parallelism of graph processing workload on rdma-based far memory system," *Journal of Parallel and Distributed Computing*, 2023.

[35] J. Wang, C. Li, T. Wang, L. Zhang, P. Wang, J. Mei, and M. Guo, "Excavating the potential of graph workload on rdma-based far memory architecture," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2022, pp. 1029–1039.

[36] D. Wu, Z. Xue, and J.-Q. He, "icloudaccess: Cost-effective streaming of video games from the cloud with low latency," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1405–1416, 2014.

[37] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13, 2013, p. 607–618.

[38] C. Zhang, J. Yao, Z. Qi, M. Yu, and H. Guan, "vgasa: Adaptive scheduling algorithm of virtualized gpu resource in cloud gaming," *IEEE Transactions on Parallel and Distributed Systems*, pp. 3036–3045, 2014.

[39] W. Zhang, B. Chen, Z. Han, Q. Chen, P. Cheng, F. Yang, R. Shu, Y. Yang, and M. Guo, "PilotFish: Harvesting free cycles of cloud gaming with deep learning training," in *USENIX ATC 22*, Jul. 2022.

[40] X. Zhang, H. Chen, Y. Zhao, Z. Ma, Y. Xu, H. Huang, H. Yin, and D. O. Wu, "Improving cloud gaming experience through mobile edge computing," *IEEE Wireless Communications*, pp. 178–183, 2019.

[41] Y. Zhang, M. A. Laurenzano, J. Mars, and L. Tang, "Smite: Precise qos prediction on real-system smt processors to improve utilization in warehouse scale computers," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 406–418.