GAugur: Quantifying Performance Interference of Colocated Games for Improving Resource Utilization in Cloud Gaming

Yusen Li
Nankai University
Tianjin, China
liyusen@nbjl.nankai.edu.cn

Chuxu Shan
Nankai University
Tianjin, China
shanchx@nbjl.nankai.edu.cn

Ruobing Chen
Nankai University
Tianjin, China
chenrb@nbjl.nankai.edu.cn

Xueyan Tang
Nanyang Technological University
Singapore
asxytang@ntu.edu.sg

Wentong Cai
Nanyang Technological University
Singapore
aswtcai@ntu.edu.sg

Shanjiang Tang
Tianjin University
Tianjin, China
tashj@tju.edu.cn

Xiaoguang Liu
Nankai University
Tianjin, China
liuxg@nbjl.nankai.edu.cn

Gang Wang
Nankai University
Tianjin, China
wgzwp@nbjl.nankai.edu.cn

Xiaoli Gong
Nankai University
Tianjin, China
gongxiaoli@nankai.edu.cn

Ying Zhang
Nankai University
Tianjin, China
yingzhang@nankai.edu.cn

ABSTRACT

Cloud gaming has been very popular recently, but providing satisfactory gaming experiences to players at a modest cost is still challenging. Colocating several games onto one server could improve server utilization. To enable efficient colocations while providing Quality of Service (QoS) guarantees, a precise quantification of performance interference among colocated games is required. However, achieving such precise interference prediction is very challenging for games due to the complexity introduced by the contention on many shared resources across CPU and GPU. Moreover, the distinctive properties of cloud gaming require that the prediction model should be constructed beforehand and the prediction should be made instantaneously at request arrivals, which further increases the difficulty. The existing solutions are either not applicable or not effective due to many limitations.

In this paper, we present GAugur, a novel methodology that enables highly accurate prediction of the performance interference among games arbitrarily colocated. By leveraging machine learning technologies, GAugur is able to capture the complex relationship between the interference and the contention features of colocated games. We evaluate GAugur through extensive experiments using a large number of real popular games. The results show that GAugur is able to identify whether a colocated game satisfies QoS requirement within an average error of 5%, and is able to quantify the performance degradation of a colocated game within an average error of 7.9%, which significantly outperforms the alternatives. Moreover, GAugur incurs an offline profiling cost linear to the number of games, and negligible overhead for online prediction. We apply GAugur to guiding efficient game colocations for cloud gaming. Experimental results show that GAugur is able to increase the resource utilization by 20% to 60%, and improve the overall performance by up to 15%, compared to the state-of-the-art solutions.

HPDC '19, June 22–29, 2019, Phoenix, MN, USA

ACM ISBN 978-1-4503-6670-0/19/06…$15.00
https://doi.org/10.1145/3307681.3325409

CCS CONCEPTS

• **Information systems → Multimedia information systems**;
• **Computer systems organization → Cloud computing**;

KEYWORDS

Cloud Gaming; Game Co-location; Performance Interference; Performance Prediction; Machine Learning

ACM Reference Format:
Yusen Li, Chuxu Shan, Ruobing Chen, Xueyan Tang, Wentong Cai, Shanjiang Tang, Xiaoguang Liu, Gang Wang, Xiaoli Gong, and Ying Zhang. 2019. GAugur: Quantifying Performance Interference of Colocated Games for Improving Resource Utilization in Cloud Gaming. In *The 28th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '19), June 22–29, 2019, Phoenix, MN, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3307681.3325409

1 INTRODUCTION

Cloud gaming has gained great popularity in recent years. In cloud gaming, games run on cloud servers and the players interact with

games over the Internet through thin clients. Cloud servers encode the rendered game scenes into videos and stream the videos to thin clients. The thin clients decode and display the videos to players, and send the control commands by players to games running on cloud servers. Different from traditional console gaming, cloud gaming puts the entire game workload to the cloud, which greatly reduces the software and hardware requirements for players to run high-end games. In this way, cloud gaming can deliver high-quality gaming experiences to players anytime, anywhere and on any device. Due to such advantages, cloud gaming has attracted great interest from both academia and industry [4, 5, 7, 22].

For cloud gaming service providers, one of the major concerns is the high operating cost. This is because high-end games are usually resource intensive. Running the games consumes a large amount of resources such as CPU, GPU, memory etc. In order to provide satisfactory gaming experiences to players, existing cloud gaming platforms often allocate each player to a dedicated server for running the requested game. This approach causes huge resource waste and may lead to unaffordable operating cost. For example, OnLive [1], the pioneer of cloud gaming, was bankrupt just for this reason.

In fact, colocating multiple games on a single server for improving server utilization is possible in some ways. Figure 1 shows the frame rate (frames per second or FPS) of some pairs of colocated games. As can be seen, some games such as *Ancestors Legacy* and *Borderland* can still run at high frame rates when colocated with each other. If we specify a minimum frame rate requirement (or QoS requirement), say 60 FPS (which is good enough for playing most games [11]), these two games can be safely colocated with performance guarantees. If there is a way to quantify the performance of arbitrary colocation of games, efficient colocation decisions can be steered. However, it is not an easy task because the frame rate of a game could be very different when colocated with different games. For example, *Ancestors Legacy* can render 105 FPS when colocated with *Borderland*, but can only run at 57 FPS when colocated with *H1Z1*. This is because colocated games share many on-chip resources such as CPU cores, GPU cores, cache, memory bandwidth etc, and the resource contention introduces varying amounts of performance interference among games.

Profiling the performance interference of all possible game colocations beforehand is prohibitively expensive. The number of all colocations is $O(2^N)$, where N is the number of games. As a cloud gaming platform may provide services for hundreds of games, a brute-force profiling approach is not practical. A more computationally efficient approach is to learn the performance interference through prediction. A large body of prior works [8, 13, 14, 19, 27, 28, 30, 31, 35, 38, 39] have studied the prediction of interference among colocated applications due to the resource contention. However, none of them can perfectly solve our problem due to the distinctive challenges for cloud gaming.

First, the prediction must be performed before games are actually colocated. Otherwise, once games are not properly colocated in practice, it is hard to readjust by migrating games among servers due to interruption to game play. Therefore, the techniques such as

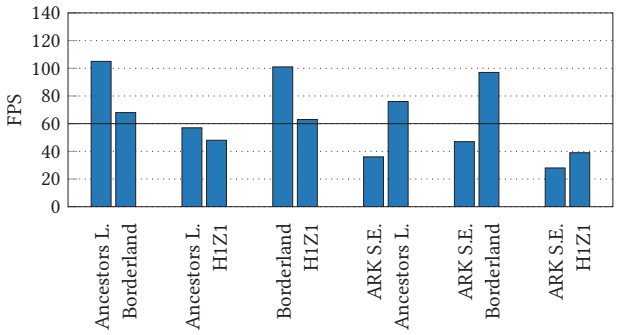

Figure 1: Performance of colocated games. X-axis denotes the colocated pairs. Y-axis is the frame rate of each colocated game.

Heracles [27], Bubble-Flux [35] and DeepDive [31] are not applicable to cloud gaming because they detect and handle the interference only for applications already colocated and running.

Second, the prediction must be instantaneous. This is because gaming requests from players must be assigned to servers immediately after they are received, to avoid long waiting time for the players. The instantaneity requirement prevents the non-real-time prediction techniques such as Prophet [8] from being applicable.

Third, the contention behaviors among colocated games are more complex. A game loop consists of many tasks (e.g., processing user input, updating game state, and rendering frames), which consume many shared resources across CPU and GPU. The interference among games may be due to contention on any type of resources shared. Therefore, the techniques such as Bubble-Up [28] and Cuanta [19] are not effective because they only consider the contention on single resource. Moreover, this complexity leads to several behaviors of games very different from general applications. For example, the performance degradation of colocated games may not be linear to the contention pressure suffered, and the aggregate intensity of multiple colocated games may not be equal to the sum of the individual intensities. So, the techniques relying on assumptions incompatible with these distinctive properties (e.g., SMiTe [39], Parogon [13] and Quasar [14]) are ineffective for games.

Four, players may choose different parameter settings (e.g., resolutions) when playing games. Different parameter settings could cause different resource consumptions and thus different contention behaviors, making efficient and precise prediction of interference even more challenging.

Prior work [6, 21] has used Sigmoid functions to predict the performance interference among games. However, this approach assumes that the performance degradation of a game is only dependent on the number of games that it is colocated with, which may produce high prediction error since the performance degradation of a game could be significantly different when it is colocated with different games, as can be seen from Figure 1.

In this paper, we propose *GAugur*, a methodology that enables precise performance interference prediction for arbitrarily colocated games. GAugur leverages machine learning techniques to build the prediction models. The entire process has four steps: contention feature profiling, model building, model training and online

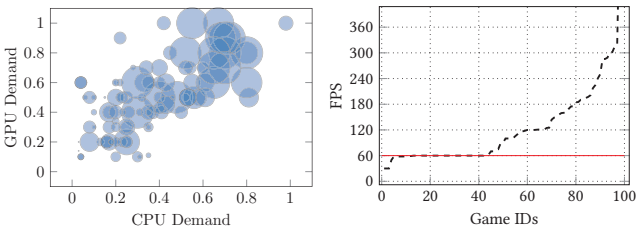

Figure 2: (a) Resource demand (bubble size represents the memory demand) and (b) Frame rate of 100 popular games when they are running alone.

prediction. The first step is to profile the contention features of games, including the sensitivity (refers to how much performance degradation a game suffers under a pressure on a shared resource) and the intensity (refers to the pressure that a game puts on a shared resource) of each game on each shared resource. This is achieved by colocating each game with a set of carefully designed benchmarks with tunable pressure. With the contention features as the input variables, the second step builds performance prediction models using machine learning technologies. Specifically, we consider two prediction models: one classification model and one regression model. The classification model is used to identify whether a game can meet the QoS requirement when it is colocated with other games. The regression model is used to quantify the exact performance degradation of a game under colocation. The models are then trained in the third step using the data collected by testing a number of real game colocations. Finally, in the fourth step, the trained models can be used to predict the performance interference for any given game colocation. Among all the four steps, the first three steps are offline and need to be performed only once while the last step (i.e., the prediction) is online and can serve continuously arriving prediction requests.

GAugur has several promising features which perfectly address the previously mentioned challenges. By using machine learning techniques, it is able to capture the complex interactions of interference on different shared resources. Moreover, the well-trained prediction models incur negligible overhead in online prediction, allowing GAugur to predict the performance interference in real time before games are actually colocated. Although contention profiling and model training have some cost, the cost is offline and is at the scale of the number of games (will be shown in Section 3). We evaluate GAugur using a large number of popular games of various genres. The results show that the average prediction error of GAugur is 5% for classification and 7.9% for regression, which significantly outperforms the alternatives. We also demonstrate through two gaming request assignment problems that GAugur is able to steer efficient game coloations for cloud gaming. Experimental results show that we are able to increase the resource utilization by 20% to 60%, and improve the overall performance by up to 15%, compared to the state-of-the-art solutions.

The rest of this paper is structured as follows. Section 2 further discusses the motivation of this work. Section 3 illustrates the detailed design of GAugur. Section 4 and Section 5 present the evaluations of GAugur. Section 6 summarizes the related work. Section 7 presents some discussions. Finally, conclusions and future work are summarized in Section 8.

2 MOTIVATION

This section introduces the existing game colocation policies. We shall show that due to inability of precise performance interference prediction, the existing policies either lead to resource overprovisioning or incur QoS violations.

2.1 Disallowing Colocation

Many commercial cloud gaming platforms disallow game colocations in order to guarantee the gaming experiences [1, 4]. Each game runs on a dedicated server, so games do not share any resource with other games. This policy delivers the best performance, however, servers run at low utilization which results in resource overprovisioning.

Figure 2a presents the resource demand for CPU, GPU and memory of 100 popular games (a full list of the games can be found in [3]) when they run alone on a server. Each resource demand is normalized to the maximum resource demand of all games for the corresponding resource type. As can be seen, the resource demand varies greatly across games and resource types, indicating that large amounts of resources will be wasted if games run alone. On the other hand, the diversity of resource demand provides great opportunity for improving server utilization by colocating games with various resource demands on the same server.

2.2 Vector Bin Packing (VBP)

This policy describes each game by a resource demand vector which is generally measured as the resource consumptions when the game runs alone on a server [24, 25]. Multiple games are allowed to run on the same server if the total resource demand of the games does not exceed the server capacity for each resource dimension. The VBP policy significantly improves resource utilization compared to disallowing colocations. However, it has two main problems.

First, resource over-provisioning still exists because when a game runs alone, it may consume more resources than it actually demands for satisfying the QoS requirement. Figure 2b presents the frame rates of the 100 games when they run alone on a server. If we specify a frame rate of 60 FPS as the QoS requirement, the games with higher frame rates will consume more resources than the actual demand.

Second, there may be QoS violations since the performance interference among colocated games is not considered. For example, we measure that the resource demands of *DDDA* and *Little Witch Academia* are $[0.45, 0.32, 0.06, 0.05]$ and $[0.33, 0.6, 0.25, 0.5]$ (each vector represents the CPU, GPU, CPU memory and GPU memory consumption of the game, each value is normalized to the server capacity of the corresponding resource) when they run alone. Under the VBP policy, the two games can be colocated on the same server as the total resource demand does not exceed the server capacity. However, we find that when the two games are actually colocated, the frame rate of *Little Witch Academia* is only 42 FPS, which is much lower than 60 FPS.

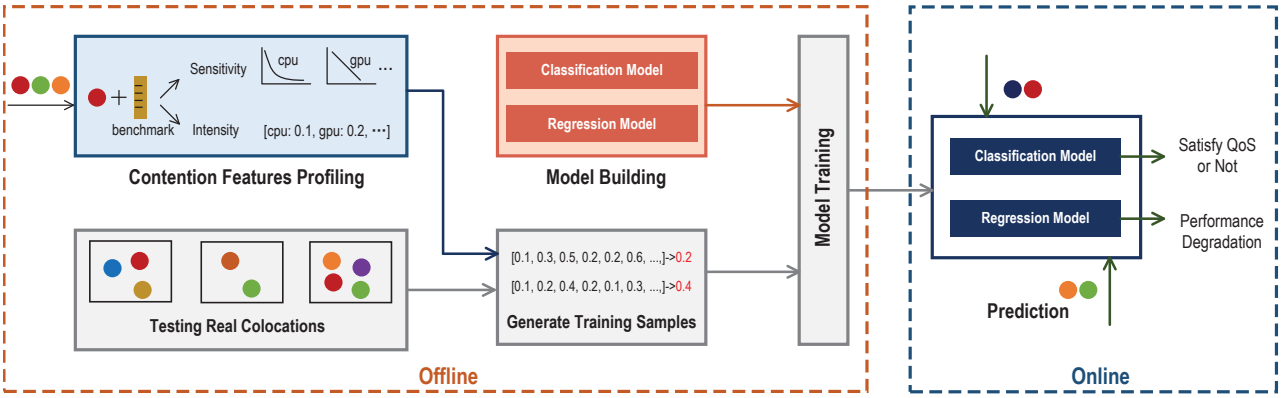

Figure 3: Design of GAugur.

3 DESIGN OF GAUGUR

3.1 Overview

The design objective of GAugur is to predict the performance interference among colocated games, in real time before the games are actually colocated. Figure 3 describes an overview of GAugur, which consists of four main steps: contention feature profiling, prediction model building, model training and online prediction. The first three steps are offline and need to be performed only once, while the last step serves prediction requests in an online manner. The details of each step are presented in the rest of this section.

3.2 Contention Feature Profiling

This section discusses how we capture the sensitivity and intensity of games. The sensitivity refers to how much performance degradation a game suffers under a pressure on a shared resource. The intensity refers to the pressure that a game puts on a shared resource. We identify seven shared resources which are most important for games, including CPU cores (CPU-CE), last level cache (LLC), memory bandwidth (MEM-BW), GPU cores (GPU-CE), GPU memory bandwidth (GPU-BW), GPU L2 cache (GPU-L2), and PCIe bandwidth (PCIe-BW). We do not account for the memories (including CPU memory and GPU memory), because we observe that the memories have almost no impact on the frame rate of games as long as the total memory demand of the colocated games does not exceed the server capacity. It is worth noting that our methodology can be extended to any shared resource.

Design of Benchmarks. In order to characterize the contention features, we develop several benchmarks, one for each shared resource. The design of similar benchmarks for the shared resources on CPU (including CPU-CE, LLC and MEM-BW) has been discussed a lot in the prior work [12, 13, 28]. Below are some important principles for the benchmark design summarized by the prior work:

- First, the benchmark should be able to progressively increase the amount of pressure for the shared resource, from no pressure to almost the maximum possible pressure;
- Second, the benchmark should not cause significant contention in other shared resources.

The benchmarks for the shared resources on GPU have not been studied before. This section briefly describes each one of them.

The Benchmark for GPU-CE. GPUs have a massively parallel architecture consisting of thousands of cores designed for handling multiple tasks simultaneously. In order to generate a pressure of x (a pressure of x indicates the cores are busy with a probability of x, $0 \le x \le 1$), we repeatedly launch a thread on each core such that all the cores execute the same kernel function in parallel. Between two successive rounds, we insert a sleep and the sleep time will thus determine the utilization of each core. In the implementation, we carefully tune the sleep time for each sampled x such that the GPU utilization (which can be observed from the performance counters) is exactly equal to x.

The Benchmark for GPU-BW. In order to generate a pressure of x on GPU-BW (a pressure of x means we occupy a portion x of the GPU memory bandwidth), we repeatedly perform streaming accesses to a fraction of the GPU memory space (copy data from one array to another one). We also insert a sleep between two rounds, and carefully tune the sleep time for each sampled x such that the bandwidth utilization is exactly equal to x. Note that there is no instruction available on modern GPUs to access memory bypassing cache (like *_mm_stream_si64x* on CPUs). Therefore, the benchmark also generates pressures on GPU caches. We argue that this makes sense because no application can occupy GPU bandwidth without using any GPU cache in practice. The benchmark for PCIe-BW is similar. The only difference is that it performs streaming data accesses between CPU memory and GPU memory. We skip the details in the interest of space.

The Benchmark for GPU-L2. A pressure of x on GPU-L2 means that a portion x of the cache capacity is occupied. In order to generate a pressure of x, we create an array of size $x \times capacity$ and issue random access to the array. To ensure that all accesses go to the L2 level cache, we set the address distance between two successive accesses to be larger than the capacity of GPU L1 cache.

Profiling. This section presents how we profile the contention features of games using the benchmarks. Consider a game A. Suppose the shared resources are indexed by $\{1, \ldots, R\}$. To profile the contention features of A for a shared resource $r \in \{1, \ldots, R\}$, we colocate game A with the benchmark of r. We tune the benchmark so that it gradually generates a set of pressures $\{0, 1/k, 2/k, \ldots, 1\}$

, where 0 indicates no pressure, 1 indicates the maximum pressure, and k is the sampling granularity. For each pressure x, we record the performance degradation of A under x, denoted by $\delta_r^A(x)$, which refers to the ratio between the frame rate when game A is colocated with the benchmark and the frame rate when game A runs alone. The frame rate of each game is measured as follows: we choose a popular game scene (e.g., Summoner's Rift for League of Legends) and run the game for several minutes. Then, we compute the average frame rate during the test period. We regard the collected performance degradations as the sensitivity curve of game A for resource r, denoted by

$$S_r^A = \left[\delta_r^A(0), \delta_r^A(\frac{1}{k}), \delta_r^A(\frac{2}{k}), \ldots, \delta_r^A(1)\right]. \quad (1)$$

For each game A, we also record the slow down of the benchmark (in terms of the running time to complete a fixed number of iterations) when it is colocated with game A compared to running alone. We regard as the average slow down of the benchmark for all pressures as the intensity of game A for resource r, denoted by I_r^A.

Results and Observations. This section presents the profiling results and the important observations on the results. We test 100 popular games with various genres [3]. We choose Windows 10 as the profiling platform, because most of modern games are for Windows. We use a server with a 4-Cores Intel i7-7700 CPU, 8GB RAM and an NVIDIA GeForce GTX 1060 GPU.

Figures 4 and 5 present the sensitivity curves and the intensity of six representative games for different shared resources (the results of other games are not shown here in the interest of space). For each shared resource, we sample 10 different levels of pressure (i.e., $k = 10$). From the results we have several observations:

- *Observation 1.* Games could be sensitive to many shared resources, and have different levels of sensitivity for different shared resources. For example, *Far Cry4* is sensitive to all the shared resources, and has rather different performance degradations for different shared resources under the same pressure.
- *Observation 2.* The sensitivity of a game is not necessarily correlated with the intensity of the game for the same shared resource. For example, *Granado Espada* is very sensitive to GPU-CE, but its intensity for GPU-CE is very light.
- *Observation 3.* Different games have different sensitivity and intensity for the same shared resource. For example, *The Elder Scrolls5* suffers 70% performance degradation for CPU-CE under the maximum pressure, while *Far Cry4* suffers only 30% degradation.

Observations 1 to *3* show the diversity of sensitivity and intensity among games for shared resources, suggesting that the sensitivity and intensity should be captured separately for each shared resource.

Moreover, we also observe that games have several different behaviors compared to general applications, including:

- *Observation 4.* The sensitivity of a game does not necessarily change linearly with the pressure for some shared resources such as GPU-CE, LLC etc.
- *Observation 5.* Game intensity on the same shared resource is not additive.

Observation 4 indicates that games are very different from the applications that have linear sensitivity to the pressure for the shared resources. As a result, the prediction methodologies relying on such properties [10, 39] are not applicable to games.

Observation 5 is identified through the following investigation. We choose two games (*AirMech Strike* and *Hobo Tough Life*) and run them together with each benchmark. We regard the slow down of a benchmark as the aggregate intensity generated by the two games for the corresponding shared resource. Figure 6 compares the aggregate intensity of the two games with the sum of the individual intensities of the two games. As can be seen, the aggregate intensity could be very different from the sum of individual intensities for some shared resources, which draws *Observation 5*.

Observation 5 indicates that when a game is colocated with more than one games, the total intensity suffered by the game is not simply equal to the sum of the intensities of the colocated games. Therefore, the prior work [13, 14] relying on such assumption to handle colocations of more than two applications is not applicable to games.

3.3 Impact of Resolution

Game players may choose different resolutions when playing a game. Profiling the sensitivity and intensity for all possible resolutions for each game is too expensive. This section studies the impact of resolution on the sensitivity and intensity of games. We identify through a systematic investigation that the impact of resolution on the sensitivity and intensity has strong similarity among games. In the interest of space, we skip the details of the investigation and only show the observations:

- *Observation 6.* The sensitivity curves of games are not affected by resolutions.
- *Observation 7.* Resolution has insignificant impact on game intensity for the shared resources CPU-CE, MEM-BW and LLC.
- *Observation 8.* There is strong linear relationship between the intensity and the number of pixels for the shared resources GPU-CE, GPU-BW, GPU-L2 and PCIe-BW.

According to *Observation 6*, we only need to profile the sensitivity curve of a game for one resolution. Note that *Observation 6* only implies that the performance degradation trends are similar when a game runs at different resolutions, while the actual performance (i.e., the frame rate) of a game may be different at different resolutions. This is reasonable because a game needs to generate more pixels at a higher resolution, causing more workload. By testing a large number of games, we identify the following relationship

$$FPS_A = -a_A * N_{pixels} + b_A, \quad (2)$$

where FPS_A is the frame rate of game A when running alone, N_{pixels} is the number of pixels of the resolution, a_A and b_A are two parameters. Equation (2) implies that if we know the frame rate of a game for two different resolutions, the frame rate of the game at any resolution can be calculated accordingly. Similarly, *Observation 7* and *Observation 8* indicate that if we know the intensity of a game for two different resolutions, the intensity of the game at any resolution can be calculated accordingly.

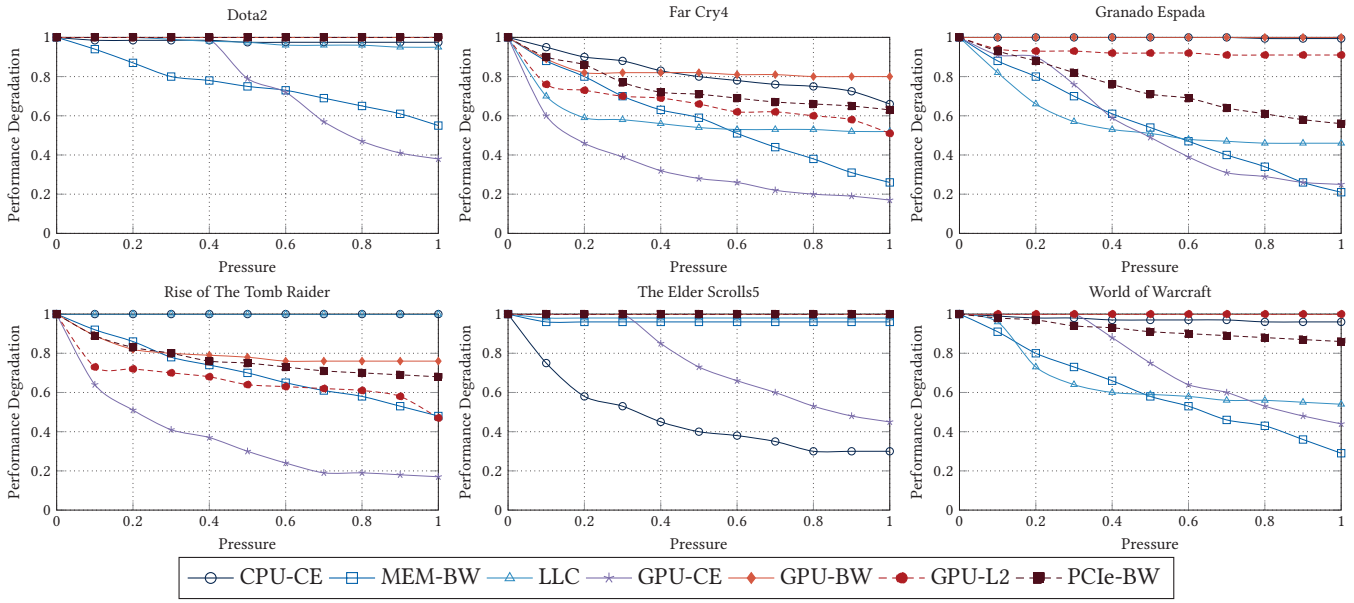

Figure 4: Sensitivity curves of selected games.

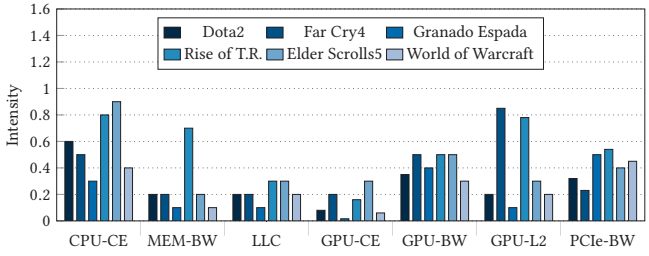

Figure 5: Intensity of selected games.

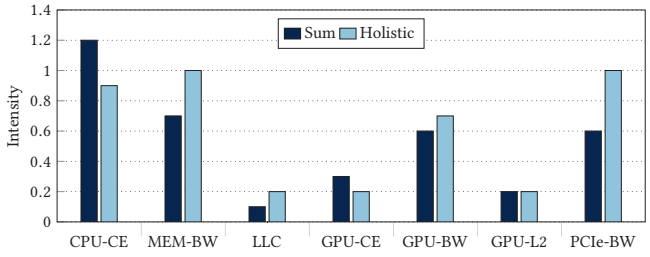

Figure 6: Aggregate intensity VS sum of intensity.

In addition to the resolution, some other graphics settings may also affect the sensitivity and intensity of games, such as antialiasing, anisotropic filtering and lighting/shadows. However, these settings are not tuned frequently in practice, so we do not consider such impacts in this paper for simplicity.

3.4 Prediction Models

Using machine learning technologies, we propose two interference prediction models for colocated games: one *classification* model and one *regression* model, targeting at different predicting objectives. This section presents the details of the models.

Classification Model (CM). This model aims to identify whether a game satisfies the QoS requirement (i.e., whether the frame rate of the game is higher than the FPS requirement) when it is colocated with a set of other games. With the CM, we are able to identify the safe game colocations (i.e., the colocations with all games satisfying QoS requirement), and use the results to improve server utilization. The model is described as follows:

$$\tilde{X}_{A\vee\{B,C,\ldots\}} = \mathrm{CM}(Q, F^A_{solo}, S^A, I^B, I^C, \ldots). \tag{3}$$

This model can predict whether game A satisfies the QoS requirement when it is colocated with games B, C etc. The inputs of the model include the QoS requirement Q (the minimum frame rate required), game A's frame rate F^A_{solo} when running alone, A's sensitivity curves S^A, and the intensities of all the colocated games I^B, I^C etc, where $S^A = [S^A_1, \ldots, S^A_R]$, $I^B = [I^B_1, \ldots, I^B_R]$ and $I^C = [I^C_1, \ldots, I^C_R]$. The output $\tilde{X}_{A\vee\{B,C,\ldots\}}$ is a binary variable, with 1 indicating a positive answer and 0 indicating a negative answer.

Regression Model (RM). The regression model aims to quantify the exact performance degradation of a game when it is colocated with other games. With the RM, we are able to predict the exact performance of colocated games, which can help to identify the colocations with less interference and improve the overall performance. The model is described below:

$$\tilde{\delta}_{A\vee\{B,C,\ldots\}} = \mathrm{RM}(S^A, I^B, I^C, \ldots). \tag{4}$$

The model can predict the exact performance degradation suffered by game A when it is colocated with games B, C etc. The inputs of the model include game A's sensitivity curves, and the intensities of all the colocated games. The output $\tilde{\delta}_{A\vee\{B,C,\ldots\}}$ is a ratio describing the performance degradation of game A compared to A's solo-run performance in terms of frame rate.

In the two models described above, the number of variables characterizing the intensities of the colocated games is not fixed, which is dependent on the number of colocated games. This is

not allowed by machine learning models which require a fixed number of input variables. We use a transformation technique to solve this problem. Consider a set of colocated games, denoted by G. Recall that the individual intensity of each game $g \in G$ is $I^g = [I_1^g, \ldots, I_R^g]$. For each shared resource $r \in \{1, \ldots, R\}$, we calculate the mean and variance of the intensity of all the games in G for resource r, denoted by $mean_r^G$ and var_r^G respectively, where $mean_r^G = \frac{1}{|G|} \sum_{g \in G} I_r^g$ and $var_r^G = \frac{1}{|G|} \sqrt{\sum_{g \in G} (I_r^g - mean_r^G)^2}$. Then, the aggregate intensity of all the games in G is represented by

$$I^G = \left[|G|, (mean_1^G, var_1^G), \ldots, (mean_R^G, var_R^G) \right], \tag{5}$$

where $|G|$ is the size of G. After this transformation, the aggregate intensity of colocated games is represented by a fixed number of $2R+1$ variables. We shall show in Section 4 that this approximation incurs insignificant error. Note that according to *Observation 5*, we cannot simply use the sum of the individual intensity of each colocated game to represent the aggregate intensity.

It is worthing noting that the RM also can be used to identify whether a game satisfies the QoS requirement according to the predicted performance degradation. The reason why we still consider the CM is that the CM achieves higher prediction accuracy compared to using the RM for classification. This is because classification is generally easier than regression for machine learning. We will show the results in Section 4.

Based on the models defined in (3) and (4), we use several popular machine learning algorithms to build the models, including Decision Tree Classifier/Regression (DTC/DTR), Random Forest (RF), Gradient Boost Decision/Regression Tree (GBDT/GBRT) and Support Vector Clustering/Regression (SVC/SVR). The performance of these algorithms are evaluated in Section 4.

3.5 Model Training and Online Prediction

Before the prediction models can be used, we need to train them first. The training samples are collected through testing real game colocations. For example, suppose we test a colocation of games A, B and C, and measure that their frame rates are 40, 50, and 60 FPS respectively when colocated. Assume the solo-run frame rate is 100 FPS for all the three games. Suppose the QoS requirement is 60 FPS. In terms of game A, we can generate the following training sample for the CM:

$$\left[60, 100, S^A, I^B, I^C \right] \rightarrow 0, \tag{6}$$

where 0 indicates that game A does not satisfy the QoS requirement. For the RM, we can generate the following training sample:

$$\left[S^A, I^B, I^C \right] \rightarrow 0.4, \tag{7}$$

where 0.4 is the performance degradation of game A, calculated according to 40/100. Similarly, we can generate corresponding training samples for B and C. By testing one game colocation of k games, we can generate k training samples for each model.

After the models are trained, we can use them for prediction. Different from the contention feature profiling and model training which are performed offline, the prediction is done online. Given a game colocation, with the contention features as the input, the models can output the corresponding results instantaneously.

3.6 Overhead Analysis

This section analyzes the overhead of GAugur. Contention feature profiling is performed for each game, which incurs $O(N)$ overhead, where N is the number of games. In order to train the models, we need to measure a number of real game colocations. We shall show in Section 4 that measuring several hundred game colocations is sufficient. As a cloud gaming platform generally has hundreds of games, it indicates that the overhead of model training is also $O(N)$. The overhead of online prediction is negligible. As contention feature profiling and model training need to be performed only once, the total overhead of GAugur is thus $O(N)$.

4 EVALUATIONS

We conduct extensive experiments using the 100 popular games [3] to evaluate GAugur. For each game, we profiled the contention features and the solo-run frame rate of the game for two different resolutions. The contention features and solo-run frame rates for other resolutions of each game used in the evaluations are computed according to *Observations 6-8* and (2). We examined a large number of real game colocations for model training and testing, including 500 colocations of two games, 100 colocations of three games and 100 colocations of four games. The game colocations with more than four games are not considered, as we find that most of the games run at very low frame rate when they are colocated with four other games on our server. The games in each measured colocation are randomly selected from the 100 games. Each game runs at a randomly selected resolution. For each measured colocation, we record the frame rate of each game and compute the performance degradation compared to the solo-run frame rate of the game. Among the 700 game colocations measured, we randomly select 400 game colocations and use the samples generated by the selected colocations (a game colocation of k games can generate k samples) as the *training set*. The samples generated by the other 300 game colocations are used as the *test set*.

All the experiments are performed on a server configured with a 4-Cores Intel i7-7700 CPU, 8GB RAM, an NVIDIA GeForce GTX 1060 GPU, and Windows 10 OS. In order to run multiple games concurrently, we connect multiple monitors to the server. We use ASTER [2], a multiseat software for Windows 10, which can provide an individual desktop for each monitor. All workplaces run independently as if each of them has its own server.

4.1 Alternative Prediction Approaches

We compare our methodology with two state-of-the-art alternative interference prediction approaches.

Sigmoid [6, 21]. This approach assumes the performance degradation of a game is only dependent on the number of games colocated. The frame rate of game A when colocated with n games is modeled by $\frac{\alpha_{A,1}}{1 + e^{-\alpha_{A,2} \times n + \alpha_{A,3}}}$, where $\alpha_{A,1} \sim \alpha_{A,3}$ are the parameters. In the implementation, for each game A, we derive the parameters by regression using the game colocations containing game A among the selected 400 game colocations for training.

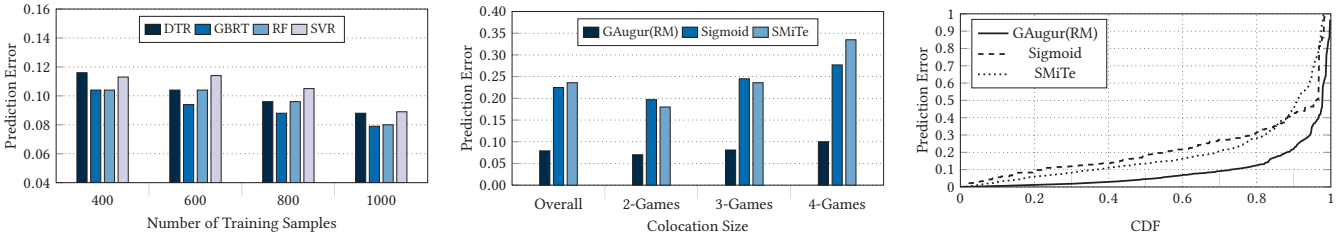

Figure 7: (a) Prediction error of GAugur using different machine learning algorithms. (b) Breakdown of prediction errors for different colocation sizes. (c) CDF of prediction errors for different prediction methodologies.

SMiTe [39]. SMiTe is a methodology that enables performance interference prediction on simultaneous multithreading (SMT) multicore processors. Suppose application A is colocated with application B. SMiTe models the performance degradation of A as

$$\tilde{\delta}_{A\vee\{B\}} = \sum_{r=1}^{R} \left(c_r \times \delta_r^A(1) \times I_r^B \right) + c_0, \tag{8}$$

where $\delta_r^A(1)$ is the sensitivity score of A for the shared resource r (refers to the performance degradation that A suffers from the maximum pressure for resource r), c_0 is a constant and c_r is the coefficient for resource r.

So far SMiTe is not able to handle colocations with more than two applications. To solve this problem, we leverage the methodology given by Paragon [13], assuming game intensity is additive. Suppose game A is colocated with B, C, ..., the new model predicts the performance degradation of A according to

$$\tilde{\delta}_{A\vee\{B,C,\ldots\}} = \sum_{r=1}^{R} \left(c_r \times \delta_r^A(1) \times (I_r^B + I_r^C + \ldots) \right) + c_0. \tag{9}$$

In the implementation, the coefficients are derived by regression using the samples in the training set.

4.2 Prediction Accuracy

Accuracy for Regression. We first present the results for regression. The objective is to predict the performance degradation of a game when colocated with other games. Figure 7a presents the mean prediction error produced by GAugur using different machine learning algorithms with different numbers of training samples. The training samples are randomly selected from the training set. The prediction error is defined as $|\tilde{\delta} - \delta|/\delta$, where $\tilde{\delta}$ and δ denote the predicted and the actual performance degradation respectively. For GAugur(RM), we applied DTR, GBRT, RF and SVR machine learning algorithms to build the model. As can be seen, using more training data generally produces smaller prediction errors for all the algorithms. The prediction error is within 10% for all the algorithms when using 1000 training samples. Among all the algorithms, GBRT achieves the best performance, which produces an error of 7.9%. In the rest of discussions, GAugur(RM) particularly refers to the GBRT model trained using 1000 samples.

Figure 7b compares GAugur(RM) with Sigmoid and SMite. As can be seen, SMiTe produces an average error of 23.6%, which is much higher than that of GAugur(RM). This is because SMiTe uses a linear model to capture the relationship between the interference and the contention features of colocated games, which is inaccurate. Sigmoid produces an average error of 22.5%, which is similar to SMiTe. This is because Sigmoid assumes the interference is only dependent on the number of colocated games, which is obviously inaccurate as has been shown earlier. A breakdown of the prediction errors for different colocation sizes are also summarized in Figure 7b. As can be seen, as the colocation size increases, all the methodologies produce a higher prediction error. GAugur(RM) outperforms the alternatives significantly for each colocation size. For the colocations of four games, GAugur(RM) still keeps the prediction error within 10%, indicating that GAugur(RM) is very effective for dealing with colocations of large sizes. In contrast, SMiTe produces very large prediction errors for the colocations of four games. This is because it assumes that game intensity is additive, which is not true in practice.

Figure 7c summarizes the CDF of the prediction errors produced by each methodology. As can be seen, GAugur(RM) produces much smaller prediction error than the alternatives at each percentile range.

Accuracy for Classification. This section presents the results for classification. The objective is to predict whether a game satisfies the QoS requirement when colocated with other games. Figure 8a presents the prediction accuracy produced by GAugur(CM) with different numbers of training samples under a QoS requirement of 60 FPS. The prediction accuracy is defined as the percentage of the correct judgements among all the testing samples. We applied DTC, GBDT, RF and SVC machine learning algorithms to build the CM of GAugur. Similarly, using more training data generally achieves higher prediction accuracy for all the algorithms. When using 1000 training samples, GBDT achieves as high as 95% accuracy, which outperforms other algorithms. Figure 8b shows the results for a lower QoS requirement, and similar performance trends are observed. In the rest of discussions, GAugur(CM) particularly refers to the GBDT model trained using 1000 samples.

We also compare GAugur(CM) with Sigmoid, SMiTe and GAugur(RM). Note that Sigmoid, SMiTe and GAugur(RM) are models for regression, which can only predict the performance (degradation). To apply them for classification, we first compute the frame rate of a colocated game using the regression results, and then check whether the game satisfies the QoS requirement according to the predicted frame rate. Figure 8c shows the results. As can be seen, GAugur(CM) achieves the highest prediction accuracy among all the methodologies. It implies that GAugur(CM) can very precisely judge whether a colocated game satisfies the QoS requirement. GAugur(RM) achieves a bit worse performance compared to GAugur(CM), indicating that classification using regression models

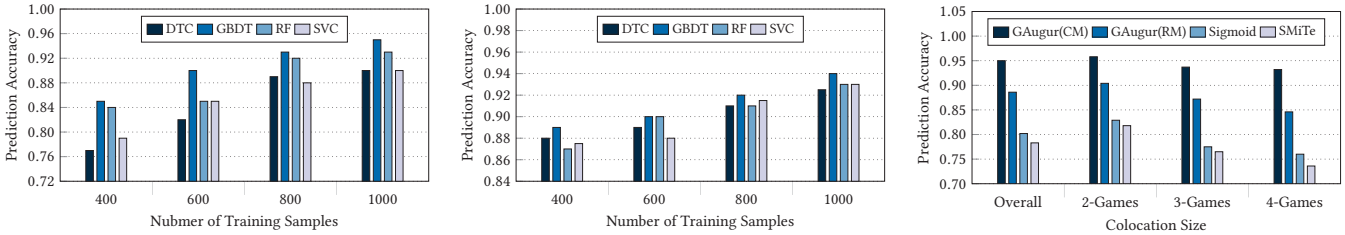

Figure 8: (a) Prediction accuracy of GAugur(CM) under a QoS requirement of 60 FPS. (b) Prediction accuracy of GAugur(CM) under a QoS requirement of 50 FPS. (c) Prediction accuracy of different methodologies.

loses some accuracy compared to using classification models directly. Sigmoid and SMiTe perform much worse than GAugur(CM), which both achieve an accuracy around 80% only. A breakdown of the prediction accuracies for different colocation sizes are also summarized in Figure 8c. It can be seen that GAugur(CM) achieves very high prediction accuracy for all colocations sizes.

5 INTERFERENCE-AWARE GAME COLOCATIONS

In this section, we present an evaluation of applying GAugur to guide efficient colocations of games in order to improve resource utilization and overall performance for cloud gaming. Specifically, we consider two problems. All the prediction models used in this section are trained as in Section 4.

5.1 Minimizing Resource Usage with QoS Guarantee

The first problem strives to minimize the number of servers used to pack a given set of gaming requests such that all the games satisfy the specified QoS requirement. The proposed GAugur(CM) can be used to solve this problem, because it is able to identify the feasible colocations (colocations with all games satisfying the QoS requirement). With this information, gaming requests can be assigned according to the feasible colocations of large sizes, so that the number of servers needed will be reduced.

We first demonstrate the performance of GAugur on judging whether a game colocation is feasible or infeasible. We divide the colocations tested into four categories: true positives (TP, actual feasible colocations that are judged as feasible colocations), false positives (FP, actual infeasible colocations that are judged as feasible colocations), false negatives (FN, actual feasible colocations that are judged as infeasible colocations), true negatives (TN, actual infeasible colocations that are judged as infeasible colocations). We are concerned with three metrics: accuracy, precision and recall. The accuracy is defined as the fraction of correct judgements among all judgements, i.e., $(|TP|+|TN|)/(|TP|+|FP|+|FN|+|TN|)$, which indicates an overall performance. The precision is defined as the fraction of the true positives among all the feasible colocations identified by the model, i.e., $|TP|/(|TP| + |FP|)$. It expresses the ability to identify only the feasible colocations. The recall is defined as the fraction of the true positives among all actual feasible colocations, i.e., $|TP|/(|TP| + |FN|)$, which expresses the ability to find all the feasible colocations. To give a complete verification, we consider a small problem size with 10 (randomly selected) games. We only consider the game colocations containing less than five games (there are 385 such colocations for 10 games). For the considered colocations, we first identify all the actual feasible colocations by measuring the actual performance of the colocated games. Then, we use GAugur(CM) to judge the colocations according to prediction. We also compare GAugur(CM) with GAugur(RM), Sigmoid, SMiTe and VBP. The feasible colocations identified by VBP refer to the colocations satisfying that the sum of resource utilizations of all games in the colocation does not exceed the server capacity for each shared resource (LLC and GPU-L2 are not included because cache is generally not characterized by utilization).

Figure 9a presents the TP, FP, FN, and TN produced by each methodology under 60 FPS QoS requirement. Figure 9b summarizes the accuracy, precision and recall of each methodology. As can be seen the accuracy, precision and recall of GAugur(CM) are all very high, indicating that GAugur(CM) is able to identify the feasible colocations very precisely. The precision of GAugur(CM) is 94%, implying that among all the feasible colocations identified by GAugur(CM), only 6% violate the QoS requirement, which is much smaller than that of the alternatives. The recall of GAugur(CM) is 88%, indicating that almost 90% actual feasible colocations can be identified by GAugur(CM). In contrast, the precisions of Sigmoid, SMiTe and VBP are much lower than that of GAugur(CM), indicating that many infeasible colocations are mistakenly judged as feasible colocations by these models. This kind of mistakes are very serious for cloud gaming because such wrong decisions could lead to unsatisfied gaming experiences to players. Moreover, the recalls of Sigmoid, SMiTe and VBP are also much lower than that of GAugur(CM). It implies that many actual feasible colocations cannot be identified by these models, which limits the potential resource utilization improvement opportunities.

We next show how GAugur(CM) can improve the resource utilization compared to GAugur(RM), Sigmoid, SMiTe and VBP. Given a set of gaming requests, we compare the number of servers used by each methodology to pack the gaming requests, such that all the games satisfy the specified QoS requirement, i.e., the game colocation at each server is feasible colocation. For each methodology, we assume the gaming requests are packed to servers according to Algorithm 1. It can be proved that Algorithm 1 has an approximation ratio of $ln(k)$ [33] (k is the maximum size of a single colocation) compared to the optimal algorithm that uses the minimum number

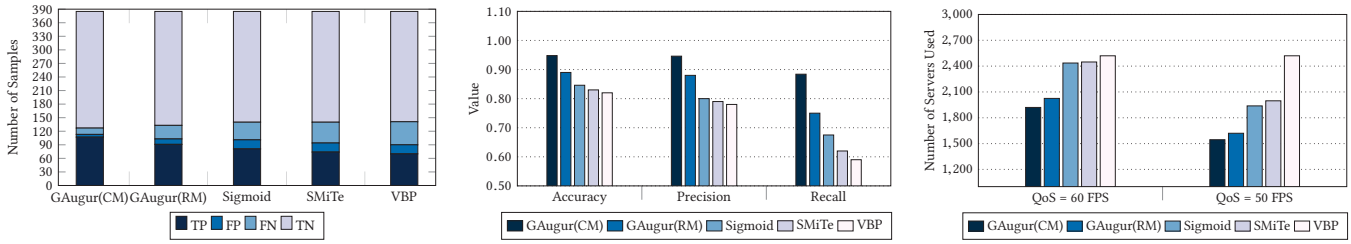

Figure 9: (a) TP, FP, FN and TN of each methodology. (b) Prediction accuracy, precision and recall of each methodology. (c) Number of servers used by each methodology.

Algorithm 1 Interference-aware Request Assignment

1: $\mathcal{F} \leftarrow$ the set of all actual feasible colocations identified
2: **while** there are requests remaining **do**
3: $c \leftarrow$ the colocation of the maximum size in $\mathcal{F}$
4: **if** each game in c has remaining requests not assigned **then**
5: Allocate a server and assign a request of each game in c to the server
6: **else**
7: remove c from $\mathcal{F}$
8: **end if**
9: **end while**

of servers to pack the gaming requests[1]. Note that we only consider the actual feasible colocations identified by each methodology as using the false positives is not meaningful because those colocations violate QoS requirement.

Figure 9c presents the number of servers used by each methodology, when there are 5000 gaming requests which are randomly distributed among the 10 selected games. As can be seen, GAugur(CM) always uses the minimum number of servers compared to other methodologies for different QoS requirements. The advantage is from 20% to 40% over Sigmoid, SMiTe and VBP. Note that if colocation is not allowed, 5000 servers will be used, indicating that GAugur(CM) can increase resource utilization by up to 60%. This is because GAugur(CM) can identify more feasible colocations than other methodologies, providing more colocation choices when assigning requests to servers.

5.2 Maximizing Overall Performance

The second problem strives to pack a given set of gaming requests to a fixed number of servers such that the average frame rate of games is maximized. The proposed GAugur(RM) can be used to solve this problem, because it is able to quantify the performance interference among colocated games. With this information, games causing high interferences can be assigned to different servers, so that the overall performance can be improved.

To evaluate the benefits, we again use the 10 selected games and consider the game colocations of size smaller than five. We compare GAugur(RM) with Sigmoid, SMiTe and VBP. For GAugur(RM), Sigmoid and SMiTe, the gaming requests are assigned to servers as follows: requests are assigned one by one according to the predicted

[1]The gaming request packing problem is NP-hard, so the optimal algorithm is not used due to high computational overhead.

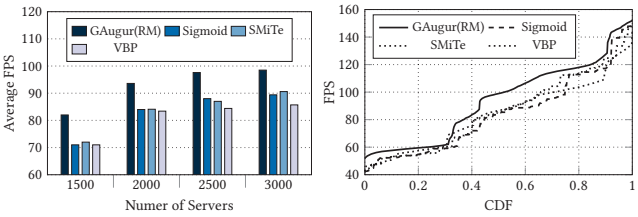

Figure 10: (a) Average FPS achieved by each methodology. (b) CDF of frame rates for different methodologies.

performance, each request is assigned to the server producing the maximum (predicted) average frame rate after assignment among all servers. For the VBP, the gaming requests are assigned in a worst-fit manner, where each request is assigned to the server with the largest remaining capacity (the remaining capacity of a server is measured by the total remaining capacity of all the shared resources except for LLC and GPU-L2).

Figure 10a presents the average frame rate produced by each methodology for different numbers of servers, when there are 5000 gaming requests which are randomly distributed among the 10 selected games. As can be seen, using more servers leads to higher frame rate for all the methodologies, because the average number of games assigned on each server is smaller. GAugur(RM) achieves the best performance and the advantage over the alternatives is up to 15%. Figure 10b summarizes the CDF of the frame rates of all games for each methodology when 2000 servers are used. An obvious higher frame rate from GAugur(RM) can be observed. This is because GAugur(RM) can predict the performance interference more precisely than the alternatives according to the results shown earlier.

6 RELATED WORK

Prior work [12, 13, 28, 39] discussed the design of benchmarks for characterizing the sensitivity and intensity of applications for the shared resources. However, all these works only considered the shared resources on CPUs. Our work is the first time to design such benchmarks for the shared resources on GPUs. Prior work [13, 14] leveraged collaborative filtering techniques to reduce the overhead of profiling the sensitivity and intensity for applications. Such techniques are complementary to our work.

Many interference-aware resource management techniques were proposed in the literature, such as Q-Cloud [30], CPI2 [38], DeepDive [31], and Heracles [27]. They detect interference between

colocated workloads and generate interference-aware schedules for improving resource utilization. However, prediction of the performance (degradation) of the colocated workloads was not discussed in these works.

The prediction of performance interference due to the shared resource contention on CPUs was studied a lot [10, 13, 14, 19, 28, 35, 39]. However, as games are very different from general applications, applying the existing approaches to our problem is nontrivial for many reasons. Cuanta [19], Bubble-Up [28], and Bubble-Flux [35] are not able to capture the contention behavior among multidimensional resources. Bubble-Up [28] and SMiTe [39] can only deal with colocations of two applications. SMiTe [39] and the work [10] assume the sensitivity is linear to the contention pressure suffered which is incorrect for games. Paragon [13] and Quasar [14] rely on the assumption that application intensity is additive which is not true for games. Machine learning techniques were utilized in the performance interference prediction models [16, 26, 29]. However, high prediction errors (> 16%) were produced because these models did not consider the sensitivity and intensity of applications.

Several techniques were proposed in the prior work to improve the utilization of GPUs. TimeGraph [23], GPUSync [17], Baymax [9], vGASA [37] and VGRIS [32] schedule GPU tasks of colocated workloads to improve the hardware utilization with QoS guarantees. These techniques are complementary to our work. Prophet [8] is able to predict the performance interference of colocated applications on GPUs. However, it is not applicable to our problem because it relies on a simulator to synthesize the execution trace of colocated applications, and thus the prediction is not done in real time. dJay [20] dynamically tunes the game settings for the colocated games during game play to adapt to changes of game scenes for improving performance. However, it focuses on a set of games already colocated. Our approach can work together with dJay and they are complementary.

The resource management issues in cloud gaming also have been extensively studied, including request assignment, scheduling, server provisioning etc [6, 15, 18, 21, 24, 34, 36]. However, the interference among colocated games was not considered in these works. Some simple game colocation policies were adopted, including the Disallowing Colocation and Vector Bin Packing policies [15, 18, 24, 34]. However, these policies either lead to resource overprovisioning or cause QoS violations as we have discussed in Section 2. The performance prediction of colocated games was studied in prior work [6, 21]. However, their models simply assume that the performance degradation is only dependent on the number of games colocated, and as such could incur large prediction errors as we have demonstrated earlier.

7 DISCUSSIONS

In this paper, we use frame rate as the metric to characterize the performance of a game. However, the frame rate may change during game play, because game scenes vary dynamically which generates different amounts of rendering workload. In our profiling, we measure the frame rate in a moderate way which computes the mean frame rate of a game during a period of time. This could lead to temporary QoS violations when all the colocated games render complex game scenes simultaneously (although this rarely happens in practice). A conservative mechanism to solve this problem is to measure the minimum frame rate instead of the mean in the profiling. Another approach is to adapt the dynamic changes of game scenes as dJay [20] does. All these solutions are compatible with GAugur.

In cloud gaming, servers not only run games, but also need to encode the outputs into videos and stream the videos to the players. Modern GPUs such as NVIDIA GRID have integrated video encoders and streaming units on their chips. Hardware encoders consume much less resource compared to the traditional software encoders, which would generate insignificant impact on game performance. Therefore, video encoding and streaming are not considered in this paper for simplicity. However, our proposed methodology can easily be extended to consider video encoding and streaming.

In addition to the frame rate, cloud gaming is also sensitive to the interaction delay, which consists of processing delay (the delay at the server side, including the time spent on processing player's command, rendering, video encoding etc) and network delay. The processing delay of colocated games can be predicted in a similar way using our methodology. One thing to be aware of is that the processing delay could also be affected by the contention on the network bandwidth. The network delay is generally not affected by game colocations, and thus does not need to be predicted.

8 CONCLUSIONS AND FUTURE WORK

In this work, we have presented GAugur, a novel methodology that enables precise prediction of performance interference among colocated games. GAugur leverages machine learning technologies to model the complex relationship between the interference and the contention on various shared resources across CPU and GPU. We evaluate GAugur using 100 real games and the results show that GAugur achieves very high prediction accuracy and significantly outperforms the alternatives. We apply GAugur to guiding efficient game colocations in cloud gaming. The results show that GAugur can improve the resource utilization by up to 60% and can improve the overall performance by up to 15%, compared to the state-of-the-art solutions.

There are several interesting directions for future work. First, GAugur is only tested on one server type in this paper. We wish to test GAugur on more server types in the future. Second, we assume the server has only one CPU and one GPU in this paper. Sharing multiple CPUs and GPUs may have scheduler related issues which we plan to address as part of our future work. Third, we only discuss the impact of resolution on the contention features of games in this paper. We would like to further study the impact of other graphics settings. Fourth, we plan to extend GAugur so that it can predict the interaction delay of colocated games.

9 ACKNOWLEDGMENT

This work is partially supported by National Science Foundation of China (61602266, 61872201, 61702521, U1833114, 61702286); Science and Technology Development Plan of Tianjin (16JCYBJC41900, 17JCYBJC15300, 18ZXZNGX00140, 18ZXZNGX00200, 18JCYBJC15600); and the Fundamental Research Funds for the Central Universities and SAFEA: Overseas Young Talents in Cultural and Educational Sector.

REFERENCES

[1] 2013. OnLive. http://onlive.com/. (2013).

[2] 2018. ASTER. https://www.ibik.ru/. (2018).

[3] 2018. Game List: A Walk in the Woods, After Dreams, AirMech Strike, Ancestors Legacy, ARK Survival Evolved, Battlerite, Black Squad, BlubBlub, Borderland2, CALL TO ARMS, Candle, Cities:SkylineCoD14s, CoD14, Cognizer, Craft The World, DARK SOULS III, Dragon's Dogma, Delicious 12, Destined, Divinity: Original Sin 2, DmC:Devil May Cry, Dota2, DRAGON BALL XENOVERSE 2, Empire Earth III, Endless Fables:The Minotaur's Curse, Far Cry 4, FAR:Lone Sails, FINAL FANTASY XII THE ZODIAC Frightened Beetles AGE, Frightened Beetles, Gems of War, Getting Over It with Bennett Foddy, Granado Espada, GUNS UP!, H1Z1, Hand of Fate 2, Heroes and Generals, Hobo:Tough Life, Human:Fall Flat, Impact Winter, Kingdom Come:Deliverance, Life is Strange:Before the Stormm, Little Nightmares, Little Witch Academia, LoLOLgout,LOL, Maries Room, NARUTO SHIPPUDEN:Ultimate Ninja STORM 4, NBA 2K17, NBA Playgrounds, Need for Speed:Hot PursuiNieR:Automatat, NieR:Automata, Northgard, Ori and the Blind Forest, Oxygen Not Included, PES2017, PlanetSide2, PES2015, Project RAT, Project CARS, Radical Heights, RiME, RimWorld, Robocraft, Russian Fishing 4, Salt and Sanctuary, Shop Heroes, Slay the Spire, StarCraft 2, Stardew Valley, Stellaris, Tactical Monsters Rumble Arena, Team Fortress 2, TEKKEN 7, The Long Dark, The Sibling Experiment, The Walking Dead:A New Frontier, The will of a single Tale, The Witcher 3 - Wild Hunt, Tiger Knight, Torchlight II, The Legend of Heroes:Trails of Cold Steel, Unturned, VEGA Conflict, War Robots, War Thunder, Warface, Warframe, World of Warships, WRC 5 FIA World Rally Championship, Assassin's Creed Origins, Rise of The Tomb Raider, Hearth Stone, Mahou Arms, World of Warcraft, Warcraft, Romance of the Three Kingdoms 11 with Power Up Kit, The Elder Scrolls V: Skyrim Special Edition, PES2012, Dynasty Warriors 5. . (2018).

[4] 2018. GeForce Now. http://www.geforce.com/. (2018).

[5] 2018. LiquidSky. https://liquidsky.com/. (2018).

[6] Mohaddeseh Basiri and Abbas Rasoolzadegan. 2016. Delay-aware resource provisioning for cost-efficient Cloud Gaming. *IEEE Transactions on Circuits & Systems for Video Technology* PP, 99 (2016), 1–1.

[7] Wei Cai, Ryan Shea, Chun-Ying Huang, Kuan-Ta Chen, Jiangchuan Liu, Victor C. M. Leung, and Cheng-Hsin Hsu. 2016. The future of cloud gaming. *Proceedings of IEEE* 104, 4 (April 2016), 687–691.

[8] Quan Chen, Hailong Yang, Minyi Guo, Ram Srivatsa Kannan, Jason Mars, and Lingjia Tang. 2017. Prophet: Precise QoS Prediction on Non-Preemptive Accelerators to Improve Utilization in Warehouse-Scale Computers. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 17–32.

[9] Quan Chen, Hailong Yang, Jason Mars, and Lingjia Tang. 2016. Baymax: Qos awareness and increased utilization for non-preemptive accelerators in warehouse scale computers. *Proceedings of the 21th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* 44, 2 (2016), 681–696.

[10] Yuxia Cheng, Wenzhi Chen, Zonghui Wang, and Yang Xiang. 2017. Precise contention-aware performance prediction on virtualized multicore system. *Journal of Systems Architecture* 72 (2017), 42–50.

[11] Kajal T Claypool and Mark Claypool. 2007. On frame rate and player performance in first person shooter games. *Multimedia systems* 13, 1 (2007), 3–17.

[12] Christina Delimitrou and Christos Kozyrakis. 2013. iBench: Quantifying interference for datacenter applications. In *IEEE International Symposium on Workload Characterization*. 23–33.

[13] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware scheduling for heterogeneous datacenters. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Vol. 48. ACM, 77–88.

[14] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: resource-efficient and QoS-aware cluster management. *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* 49, 4 (2014), 127–144.

[15] Yunhua Deng, Yusen Li, Ronald Seet, Xueyan Tang, and Wentong Cai. 2017. The server allocation problem for session-based multiplayer cloud gaming. *IEEE Transactions on Multimedia* PP, 99 (2017), 1–1.

[16] Tyler Dwyer, Alexandra Fedorova, Sergey Blagodurov, Mark Roth, Fabien Gaud, and Jian Pei. 2012. A practical method for estimating performance degradation on multicore processors, and its application to hpc workloads. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society Press, 83.

[17] Glenn A Elliott, Bryan C Ward, and James H Anderson. 2013. GPUSync: A framework for real-time GPU management. In *2013 IEEE 34th Real-Time Systems Symposium*. IEEE, 33–44.

[18] David Finkel, Mark Claypool, Sam Jaffe, Thinh Nguyen, and Brendan Stephen. 2014. Assignment of games to servers in the OnLive cloud game system. In *2014 13th Annual Workshop on Network and Systems Support for Games*. IEEE, 1–3.

[19] Sriram Govindan, Jie Liu, Aman Kansal, and Anand Sivasubramaniam. 2011. Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC)*. ACM, 22.

[20] Sergey Grizan, David Chu, Alec Wolman, and Roger Wattenhofer. 2015. dJay: enabling high-density multi-tenancy for cloud gaming servers with dynamic cost-benefit GPU load balancing. In *Acm Symposium on Cloud Computing*. 58–70.

[21] Hua-Jun Hong, De-Yu Chen, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. 2014. Placing virtual machines to optimize cloud gaming experience. *IEEE Transactions on Cloud Computing* (2014).

[22] Chun-Ying Huang, Cheng-Hsin Hsu, Yu-Chun Chang, and Kuan-Ta Chen. 2013. GamingAnywhere: an open cloud gaming system. In *Proceedings of the 4th ACM Multimedia Systems Conference*. 36–47.

[23] Shinpei Kato, Karthik Lakshmanan, Raj Rajkumar, and Yutaka Ishikawa. 2011. TimeGraph: GPU scheduling for real-time multi-tasking environments. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference (ATC)*. 17–30.

[24] Yusen Li, Xueyan Tang, and Wentong Cai. 2014. On dynamic bin packing for resource allocation in the cloud. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 2–11.

[25] Yusen Li, Xueyan Tang, and Wentong Cai. 2015. Play request dispatching for efficient virtual machine usage in cloud gaming. *IEEE Transactions on Circuits & Systems for Video Technology* 25, 12 (2015), 2052–2063.

[26] Seung-Hwan Lim, Jae-Seok Huh, Youngjae Kim, Galen M Shipman, and Chita R Das. 2012. D-factor: a quantitative model of application slow-down in multi-resource shared systems. *ACM SIGMETRICS Performance Evaluation Review* 40, 1 (2012), 271–282.

[27] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: Improving resource efficiency at scale. In *International Symposium on Computer Architecture (ISCA)*, Vol. 43. ACM, 450–462.

[28] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. 2017. Bubble-up:increasing utilization in modern warehouse scale computers via sensible co-locations. In *IEEE/ACM International Symposium on Microarchitecture*. 248–259.

[29] Nikita Mishra, John D Lafferty, and Henry Hoffmann. 2017. ESP: A machine learning approach to predicting application interference. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 125–134.

[30] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. 2010. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European Conference on Computer systems*. ACM, 237–250.

[31] Dejan Novakovic, Nedeljko Vasic, Stanko Novakovic, Dejan Kostic, and Ricardo Bianchini. 2013. Deepdive: Transparently identifying and managing performance interference in virtualized environments. In *Proceedings of the 2013 USENIX Annual Technical Conference (ATC)*.

[32] Zhengwei Qi, Jianguo Yao, Chao Zhang, Miao Yu, Zhizhou Yang, and Haibing Guan. 2014. VGRIS: Virtualized GPU resource isolation and scheduling in cloud gaming. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 2 (2014), 17.

[33] Vijay V Vazirani. 2013. *Approximation algorithms*. Springer Science & Business Media.

[34] Di Wu, Xue Zheng, and Jian He. 2014. iCloudAccess: cost-effective streaming of video games from the cloud with low latency. *IEEE Transactions on Circuits & Systems for Video Technology* 24, 8 (2014).

[35] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. 2013. Bubble-flux:precise online QoS management for increased utilization in warehouse scale computers. In *International Symposium on Computer Architecture (ISCA)*. 607–618.

[36] Miao Yu, Chao Zhang, Zhengwei Qi, Jianguo Yao, Yin Wang, and Haibing Guan. 2013. VGRIS: virtualized GPU resource isolation and scheduling in Cloud Gaming. In *Proceedings of the 22th International Symposium on High-performance Parallel and Distributed Computing (HPDC)*. 203–214.

[37] Chao Zhang, Jianguo Yao, Zhengwei Qi, Miao Yu, and Haibing Guan. 2014. vGASA: Adaptive scheduling algorithm of virtualized GPU resource in cloud gaming. *IEEE Transactions on Parallel and Distributed Systems* 25, 11 (2014), 3036–3045.

[38] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. 2013. CPI 2: CPU performance isolation for shared compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys)*. ACM, 379–391.

[39] Yunqi Zhang, Michael A. Laurenzano, Jason Mars, and Lingjia Tang. 2014. SMiTe: Precise QoS Prediction on Real-System SMT Processors to Improve Utilization in Warehouse Scale Computers. In *IEEE/ACM International Symposium on Microarchitecture*. 406–418.