Reinforcement Learning-Based Resource Partitioning for Improving Responsiveness in Cloud Gaming

Yusen Li, Xiwei Wang, Haoyuan Liu, Lingjun Pu, Shanjiang Tang, Gang Wang, and Xiaoguang Liu

Abstract—Cloud gaming has been very popular in recent years, but issues relating to maintaining low interaction delay to guarantee satisfactory user experience are still prevalent. We observe that the server-side processing delay in cloud gaming system could be heavily influenced by how the resources are partitioned among processes. However, finding the optimal partitioning policy that minimizes the response delay faces several critical challenges. First, fine-grained resource partitioning is non-trivial due to the limitations of hardwre-based resource isolation techniques. Second, game wokload is highly dynamic and unpredictable, making the design of efficient resource partitioning policy more challenging. In this article, we propose an online resource partitioning framework for reducing response delay in cloud gaming, which has several promising properties. First, we divide the processes into disjoint groups and partition resources among process groups, which greatly simplifies the resource partitioning problem while ensuring high partitioning effectiveness. Second, to tackle dynamic workload changes, we classify game workloads into several clusters and maintain separate process grouping plan for each cluster. Third, we leverage reinforcement learning to adaptively choose the best actions for minimizing response delay in real time. We evaluate the proposed framework in a real cloud gaming environment using several real games. The experimental results show that our approach can reduce the response delay by 22 to 41 percent compared to a system without resource partitioning, and outperforms other resource partitioning policies significantly.

Index Terms—Cloud gaming, interactive delay, resource partitioning, machine learning, reinforcement learning

1 Introduction

Cloud gaming has received a lot of attentions from both academia and industry in recent years [1], [2], [3], [4]. The basic idea of cloud gaming is to run the game on the cloud server and stream the video of gameplay to the remote player, while the player sends the input actions over the network to the cloud server. Compared to the traditional console gaming, this approach provides several advantages, including running the game directly without downloading and installing, and on a wider range of devices (such as tablets and mobile phones) due to lower hardware requirements.

It is well-known that game players are very sensitive to interaction delay. The interaction delay in cloud gaming includes the network delay (network round-trip latency), server-side response delay (refers to the delay between when the user input is received by the server and when the corresponding game screen is sent out from the server, which is the sum of the delays for game rendering, video encoding and transmission etc), and the client side playout delay (for video decoding and displaying). According to previous studies [5], the server-side response delay usually dominates the interaction delay in cloud gaming. This is because the main workload is processed at cloud server side. Many existing works have tried to reduce response delay in cloud gaming, e.g., through reducing the video encoding time [6], [7], speculating rendering frames [8] or adaptively streaming the game videos [9], [10]. However, none of them considers the impact of resource partitioning.

Cloud gaming system (particularly referring to the server side) is composed of a set of processes. If resources such as CPU cores and Last Level Cache (LLC) are fully shared among processes, there will be contention over the shared resources, which will slow down game rendering, video encoding or transmission, and thus increase the response delay. Some process may have a high demand for resources, but the benefit that the process gets from the resources may not directly correlate with this demand. For example, video streaming generally consumes a lot of cache space, but the cached data only saves a little amount of processing time because they are unlikely to be reused again. Therefore, it makes sense to reduce the response delay through correctly partitioning the resources among processes in cloud gaming.

Our preliminary studies (see the experimental results in Section 2) show that the benefit of resource partitioning on response delay reduction could be significant. Partitioning cache alone can reduce the response delay by up to 30 percent

- *Yusen Li, Xiwei Wang, Haoyuan Liu, Lingjun Pu, Gang Wang, and Xiaoguang Liu are with the Department of Computer Science, Nankai University, Tianjin 300071, China. E-mail: {liyusen, wangxiwei, liuhaoyuan, wgzwp, liuxg}@nbjl.nankai.edu.cn, pulingjun@gmail.com.*
- *Shanjiang Tang is with the Department of Computer Science, Nankai University, Tianjin 300071, China, and also with the School of Computer Science and Technology, Tianjin University, Tianjin 300354, China. E-mail: tashj@tju.edu.cn.*

Manuscript received 23 July 2020; revised 3 Feb. 2021; accepted 28 Mar. 2021. Date of publication 5 Apr. 2021; date of current version 7 Apr. 2022.
(Corresponding author: Yusen Li.)
Recommended for acceptance by X. Cheng.
Digital Object Identifier no. 10.1109/TC.2021.3070879

compared to fully-sharing the cache. However, despite its advantages, resource partitioning for delay reduction in cloud gaming faces critical challenges.

First, some resources can only be coarsely partitioned due to hardware limitation. For example, the LLC can only be partitioned at cache way granularity for Intel's CPUs. With this limitation, the policy of complete-isolation (i.e., partitioning the cache into isolated parts and allocating each process a separate part) might be sub-optimal, because the expected cache allocation for a process may not precisely align with the cache way size. Therefore, more effective partitioning policy should be investigated.

Second, as multiple resources can be partitioned, the number of possible partitions over all resources is likely to be substantial. To find the optimal resource partition, testing all possible partitions in a real system is impractical. Therefore, we require a mechanism to estimate the performance (in terms of response reduction) of any given partition. However, it is very challenging to build a performance model due to the complex interactions and resource contention behaviors among processes.

Third, game scenes would vary over time, leading to dynamic workload of processes. Different workloads may require different partitions for achieving minimal response delay. However, the change of game workload is fast-paced and unpredictable, and re-computing the optimal partition for each workload change is impractical. Therefore, a dynamic and adaptive partitioning strategy is required.

The resource partitioning issue has been extensively studied for improving system performance for general applications. However, none of them can address all of the previously mentioned challenges in cloud gaming. For example, DCAPS [11], EMBA [12], KPart [13] and Hypart [14] only considers one type of resource, [15], [16] assume the workloads of applications are static, [17] only considers the complete-isolation policy which is not effective, Heracles [18], PARTIES [19], CoPart [20] and Quasar [21] can only be applied in limited scenarios because they lack precise performance models to quantify performance interference.

In this paper, we present a framework that enables efficient and adaptive resource partitioning for reducing the server-side response delay in cloud gaming. The proposed framework has several promising features which adequately address the previously-mentioned challenges. To sidestep the deficiency caused by the hardware limitation, we divide the processes into disjoint groups and partition the resources among groups. This mechanism can ensure high partitioning effectiveness while greatly simplifies the resource partitioning problem. To search the optimal process groups, we build a performance model by employing machine learning technology, which is able to capture the complex relationships between resource partition and system performance. Our framework deals with dynamic workload changes from two perspectives. First, we classify game workloads into several clusters and maintain separate process grouping plan for each cluster, so that different workloads can have different grouping plans. Second, we leverage reinforcement learning to adaptively adjust resource partitioning in an online manner. The crafted designed reinforcement learning model is able to learn the impact of various resource partitioning adjustment actions on system performance, and automatically selects the best adjustment action that minimizes the response delay according to real time workload.

We validate the proposed framework in a real cloud gaming environment using real games of different genres. The experimental results show that the proposed framework can reduce the response delay by 22 to 41 percent compared to a system without resource partitioning, and also outperforms other state-of-the-art partitioning policies significantly. To the best of our knowledge, this is the first work to optimize response delay in cloud gaming through resource partitioning.

This paper extends from our preliminary work [22], with significant improvements which include:

- We classify game workloads into several clusters and maintain separate process grouping plan for each cluster, which is more adaptive to dynamic workload changes.
- We improve the performance model by adding game workload into the input features, which is able to predict the performance of a game under specific resource partition and game workload.
- We improve the Q-Learning model by using neural networks to replace Q-Tables, which is more effective to distinguish the continuous system states such as game workload.
- We compare the proposed framework with more state-of-the-art alternatives. More observations and discussions about the experimental results have been made.
- We add a comprehensive summary of the related works.

The remainder of this paper is organized as follows. Section 2 illustrates the motivation of this work. The proposed framework is proposed in Section 3. Section 4 presents the experimental evaluations. The related work is summarized in Section 5. Finally, conclusions are made and future work is discussed in Section 6.

2 Motivation

In this section, we present the motivation of this work. First, we will show in Section 2.1 that resource partitioning could have significant impact on the response delay of cloud gaming and existing resource partitioning solutions are inefficient. Second, we will discuss the challenges of designing efficient resource partitioning policy in cloud gaming.

2.1 Impact of Resource Partitioning

We first show by some preliminary experiments that how resource partitioning could affect the responsiveness of cloud gaming.

The experiments are performed in a real cloud gaming environment, built on the open source platform GamingAnywhere [1] (see more details in Section 4.1). We run the game *Nexuiz* in the system and partition the LLC of the server among seven major processes (`nexuiz`, `server`, `video`, `audio`, `xorg`, `rtsp` and `compiz`) in the system. The server where the system is running has 10 MB LLC and

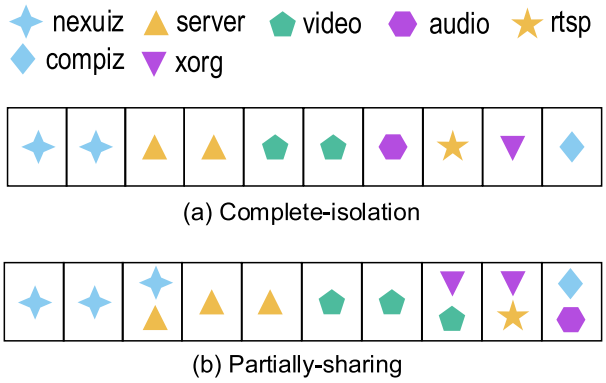


Fig. 1. (a) Complete-isolation, with a response delay of 103 ms; (b) Partially-sharing, with a response delay of 56 ms.

the LLC is partitioned with 1 MB granularity (the size of cache way) using Intel's CAT technology [23]. We let the game stay in a fixed game scene (which generates a stable game workload) and measure the server side response delay (using the approach proposed in [5]) under three different partitioning policies:

Fully-Sharing. This policy assumes the entire LLC is shared among all the processes, namely, no partitioning is considered. Therefore, the processes freely compete for the LLC according to their demands, which may lead to inefficient allocation because the LLC demands are not necessarily correlated with the response delay. In this experiment, the fully-sharing policy incurs a response delay of 84 ms.

Complete-Isolation. In this policy, the LLC is divided into isolated parts, and each process is allocated a separate part. The impact of LLC contention can be eliminated by resource isolation. However, since the hardware does not support fine-grained partitions for the LLC, complete-isolation may downgrade the effectiveness of partitioning. Fig. 1a shows the best complete-isolation plan (i.e., the plan that incurs the minimum response delay among all complete-isolation plans), which incurs a response delay of 103 ms, even higher than that with fully-sharing.

Partially-Sharing. This policy allows each cache way to be shared by more than one process, which is more flexible than complete-isolation. In this manner, when a process does not need an entire cache way, it can share the cache way with other processes so that the LLC can be utilized more efficiently. Fig. 1b shows a partially-sharing plan, which incurs a response delay of 56 ms (much lower than that with fully-sharing). The problem of partially-sharing is that it will lead to complex sharing relationships and contention behaviors among processes, which makes finding an optimal partitioning solution challenging.

From the above experiments we draw two conclusions: (1) resource partitioning could significantly reduce the response delay compared to fully-sharing; and (2) the policies of complete-isolation and partially-sharing are both not suitable for resource partitioning in cloud gaming, and more efficient policy is desired.

2.2 Challenges

We next discuss the challenges of designing efficient resource partitioning policy in cloud gaming.

Challenge I. *Fine-grained resource partitioning is non-trivial due to the limitations of hardware-based resource isolation techniques*.

Resource partitioning is generally powered by the resource isolation techniques. Earlier studies on resource isolation techniques are mostly software-based, e.g., the page coloring technique for isolating LLC. Such techniques typically require modifications to the operating system or kernel, which incur large performance overhead. Recent commodity servers have provided hardware support for resource isolation, e.g., Intel provides Cache Allocation Technology (CAT) [23] in their Xeon E5 server processors. Compared to the software-based solutions, the hardware-based approaches have much less overhead and are more widely used in practice.

However, implementing a fine-grained resource partitioning is very challenging due to the intrinsic limitations of the hardware-based resource isolation techniques. For example, Intel's CAT partitions the LLC at a granularity of cache way, which may be too coarse-grained, since the desired cache capacity of a process may not be exactly an integer multiple of the cache way size. In order to achieve a fine-grained partition, the cache ways have to be partially shared across the colocated processes (as shown in Fig. 1b). However, this will cause two important problems.

First, this would significantly increase the difficulty of a precise control of the LLC allocation, because the amortized cache capacity of a process is not only dependent on the partitioning scheme, but also affected by the resource contention over the shared cache ways. Second, the number of potential partitions would be greatly increased. Consider a group of N processes running on a server with L cache ways. The number of possible LLC partitions among the processes will be $(L(L+1)/2)^N$, which is up to $O(10^{16})$ for $N = 8$ and $L = 10$.[1] Finding the optimal partition in such a huge solution space would be very challenging.

Challenge II. *Game workload is highly dynamic and unpredictable, making the design of efficient resource partitioning policy challenging*.

To show the workload changes, we play the game *Nexuiz* for 50 minutes, where all the resources are fully shared among the processes. The top of Fig. 2 shows the response delays (measured in every 20 seconds) during the running period. It can be seen that the response delay varies greatly and no regular patterns are observed.

Further, we choose three different game scenes. For each game scene, we generate a "good" LLC partition, which incurs much lower response delay compared to fully-sharing. We run the game in each game scene under the three "good" partitions, and measure the response delay under each case. The bottom of Fig. 2 presents the results. As can be seen, the partition that is good for one game scene may not be good for other game scenes. For example, using partition 1 is good for game scene 1, but not good for game scene 3.

The above experiments imply that a static partition is deficient for cloud gaming, motivating us to design adaptive and robust online resource partitioning strategy.

1. Intel's CAT requires the cache ways allocated to a process to be continuous.

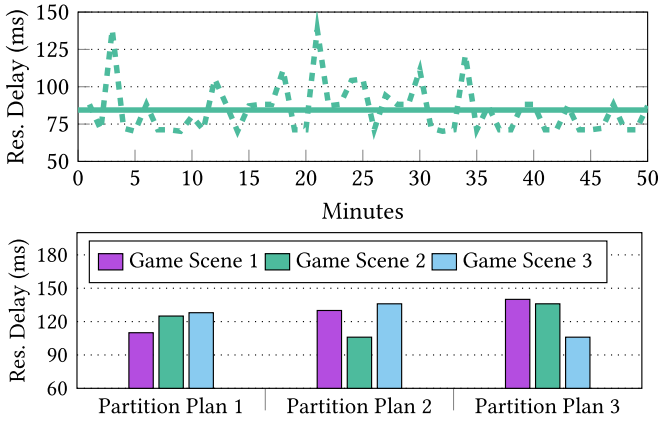


Fig. 2. The top: Response delay of game *Nexuiz* over the running period (straight solid line represents the mean). The bottom: Response delay of game *Nexuiz* under different partitions.

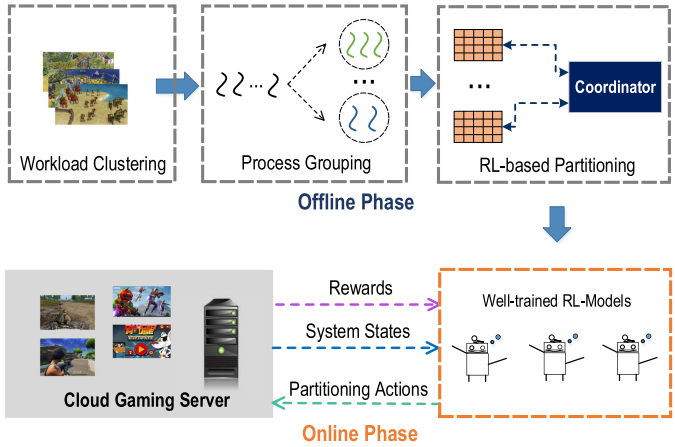


Fig. 3. Overview design of the proposed framework.

3 The Proposed Framework

In this section, we present the details of the proposed resource partitioning framework. Section 3.1 gives an overview of the system design, and the following sections present the detailed procedures. The overhead of the proposed solution is analyzed in Section 3.5.

3.1 Overview

The overview of our framework is shown in Fig. 3, which has offline phase and online phase. The offline phase takes charge of training the model, which can further be divided into three procedures: *workload clustering*, *process grouping* and *adaptive partitioning*. Based on the well-trained model, the online phase adaptively partitions the resources in real time.

To address *Challenge I*, the processes are divided into several disjoint groups and the resources are partitioned among the groups. Each group is allocated dedicated resources which are shared by the processes in the same group. This mechanism has two benefits: first, compared to partially-sharing, the partitioning problem is greatly simplified because resources are only isolated among groups; second, the resource with coarse partitioning granularity can be partitioned in more fine-grained manner, because it allows the processes in the same group to share the resource.

As mentioned in *Challenge II*, game workloads are highly dynamic and unpredictable. We address this issue from two perspectives. First, we classify game workloads into several clusters, where the game workloads in the same cluster are similar and share the same process grouping plan. This mechanism can bring two benefits: (1) sharing process grouping plan among game workloads reduces the overhead for frequent adjustment of the grouping plan; (2) maintaining different grouping plans for different clusters ensures the adaptivity to dynamic workload changes.

Second, we leverage reinforcement learning (deep Q-Learning) to implement adaptive resource partitioning. The reinforcement learning model is trained offline, and the well-trained model is used online to adaptively adjust resource partition according to real-time system states.

3.2 Workload Clustering

As game workload varies dynamically, the optimal process grouping plan may be different for different game workloads. Deriving the optimal grouping plan for each workload is infeasible, because: (1) computational overhead is large since game workloads could be highly diverse; (2) frequent adjustment of grouping plan may cause negative effects on performance (e.g., if we move a process from one group to another, the process needs to switch from the cache ways allocated to the original group to those allocated to the target group, which would lead to many cache misses in this procedure and thus increase the response delay).

To address this issue, we classify game workloads into clusters and maintain separate grouping plan for each cluster. Since games consume both CPU and GPU resources, we use three commonly used metrics to characterize game workload: CPU utilization, GPU utilization and PCIe bandwidth utilization. Specifically, the workload of a game running at game scene s is represented by the vector

$$w_s = [CU_s, GU_s, PB_s], \tag{1}$$

where CU_s, GU_s and PB_s denote the CPU utilization, GPU utilization and PCIe bandwidth utilization of the server that the game is running on.

Based on the above representation, we aim to classify the possible workloads of a game into a number of clusters, such that the game workloads in the same cluster are similar. We use the popular clustering algorithm k-means, where the number of clusters is determined by the G-means algorithm proposed in [24]. Algorithm 1 shows the details.

Consider a specific game. Let $W = \{w\}$ denote the set of game workloads to be clustered. The algorithm starts with a single center (i.e., $\{\overline{w}\}$), and grows the number of centers iteratively. In each iteration, for those clusters whose datapoints appear not to follow a Gaussian distribution (checked by a statistical test [24]), the algorithm splits each cluster into two clusters. Between each round of splitting, the algorithm runs k-means on the entire dataset to refine the current solution. When the algorithm terminates, the centers of the clusters (i.e., set C) are determined.

It is worth noting that the number of game scenes could be substantial, so profiling the workloads for all game scenes is impossible. In the practical implementation, we

Algorithm 1. Game Workload Clustering Algorithm

```
1:  Input: W = {w} (the set of game workloads)
2:  Output: C (the set of clusters, represented by their centers)
3:  Initialize C by {w̄}
4:  repeat
5:    C ← k-means(C, W)
6:    for each cluster center c ∈ C do
7:      W_c ← the set of workloads assigned to the center c
8:      if the datapoints in W_c do not follow a Gaussian distribution then
9:        replace c with two centers
10:     end if
11:   end for
12: until no more centers are added.
```

just sample a subset of game scenes to construct the set W. For a game workload not included in W (i.e., the corresponding game scene is not sampled), we simply classify the workload into the cluster whose center is closest.

3.3 Process Grouping

After the workload clusters are determined, we next show how to find the optimal process grouping plan for each cluster such that the response delay can be minimized.

Since the number of possible grouping plans could be substantial, it is impractical to test all of them in real system. Instead, we use a more efficient approach, which uses a performance model to estimate the performance of grouping plans and search a sub-optimal grouping plan using a heuristic algorithm.

3.3.1 Machine Learning-Based Performance Model

Building a precise performance model is very challenging since the performance (i.e., response delay) is determined by many factors together, including the grouping plan, resource partition, resource contention and game workload. To address this issue, we leverage machine learning techniques.

Machine learning is good at capturing complex relationships and has been widely used in various scenarios for prediction. Training a machine learning model generally requires a large number of samples collected from real system. However, measuring the response delay in cloud gaming is very difficult [5]. We observe that the response delay has a strong linear correlation with the processes' IPCs (Instructions Per Cycle, a metric indicating the execution speed of a process). Intuitively, the execution speed of the processes is proportional to the IPCs, while the response delay is basically inversely proportional to the execution speed of the processes (higher execution speed will lead to lower response delay, and vice versa). So, there is a linear relationship between the response delay and IPCs. Specifically, let P denotes the set of processes (for running the game, video encoding and video transmission etc) in the cloud server, the response delay can be represented by

$$\sum_{p \in P} \alpha_p \cdot IPC_p + b, \tag{2}$$

where IPC_p is the IPC of process p, α_p is the coefficient associated with p and b is a constant (the validation is presented in Section 4). Based on this observation, instead of predicting the response delay directly, we use the machine learning model to predict the IPCs of processes, and then use the linear model defined in (2) to approximate the response delay according to the IPCs. Since the IPCs are very easy to collect, this technique significantly reduces the overhead for collecting the training samples.

The machine learning model is defined as follows: the input features include the members of a group (i.e., the processes belonging to a group), the amount of resources allocated to the group and the current game workload, while the output is the weighted sum of the IPCs of the processes in this group. Consider a process group G. The weighted sum of the IPCs for G, which we call W-IPC of group G, is defined as $WIPC_G = \sum_{p \in G} \alpha_p \cdot IPC_p$ (α_p has the same meaning as in (2)).

Let R denote the set of resources that can be partitioned in the cloud server. We index the processes in P from 1 to $|P|$, and the resources in R from 1 to $|R|$. The members of group G is denoted by a 0-1 vector X_G, where the jth element indicates whether the jth process in P is in group G (1 indicates yes and 0 indicates no). The amount of resources allocated to group G is denoted by a vector Y_G, where the jth element denotes the amount of resource j in R that has been allocated to group G.

Denote by w the current game workload. The machine learning model for predicting the W-IPC of G can be represented by

$$\widetilde{WIPC}_G = \mathrm{ML}(X_G, Y_G, w), \tag{3}$$

where $\widetilde{WIPC}_G$ is the output of the prediction from W-IPC of the group G. With the prediction model, given a partitioning plan, the W-IPC of each process group under a specific game workload can be predicted and the response delay can be estimated according to (2) accordingly.

3.3.2 Finding the Optimal Grouping Plan

Consider a workload cluster c. Let W_c denote the workloads belonging to c. Given a process grouping plan P, let d_P^w denote the response delay achieved by P under game workload w, which refers to the minimum achievable response delay under all possible resource partitions. We define the response delay of grouping plan P for cluster c as the average response delay achieved by P under all workloads in W_c. The optimal grouping plan refers to the plan that achieves the lowest response delay.

Since a brute-force search for the optimal grouping plan is expensive, we use a heuristic algorithm to find a near optimal solution. The details are presented in Algorithm 2. Initially, we put all the processes in the same group and allocate all the resources to the group (lines 3–6). After that, in each loop, we attempt to split one group (the largest group that can be split) into two groups. To split a group G, we first create an empty group G' (line 9). Then, we try to shift the processes one by one from G to G' (lines 10-14). A shift is only accepted if the response delay after shifting (the minimum achievable delay by realocating the resources of G (i.e., Y_G) among G and G') is smaller than that before

shifting. The response delay under a specific grouping plan and resource allocation refers to the average response delay for all game workloads in W_c, which is obtained according to the prediction model defined by (3). After the splitting procedure, we reallocate the resources of G among G and G' such that the response delay is minimized (lines 15-18). If G cannot be split (i.e., G' is empty), we mark it and never consider G again. The algorithm terminates until no group can be split.

Algorithm 2. Process Grouping for Workload Cluster c

1: **Input:** the set of all processes, denoted by P
2: **Output:** the set of process groups, denoted by S
3: $S \leftarrow \{P\}$ /*put all processes into one group*/
4: $C_i \leftarrow$ total amount of the ith resource, $1 \leq i \leq |R|$
5: $Y_P \leftarrow [C_1, C_2, \ldots, C_{|R|}]$ /*allocate all resources to P*/
6: **while** $S \neq \emptyset$ **do**
7: $G \leftarrow$ the largest group in S that can be split
8: $d \leftarrow$ response delay under current grouping plan and resource allocation
9: $G' \leftarrow \emptyset$ /*create a new empty group*/
10: **for** each $p \in G$ **do**
11: shift p from G to G'
12: $d^* \leftarrow$ the minimum achievable response delay by reallocating Y_G among G and G'
13: If $d^* < d$, do $d \leftarrow d^*$, otherwise, move back p to G
14: **end for**
15: **if** $G' \neq \emptyset$ **then**
16: $Y^* \leftarrow$ the amount of resources (in Y_G) should be allocated to G' for achieving the minimum response delay
17: $Y_{G'} \leftarrow Y^*, Y_G \leftarrow Y_G - Y^*$
18: $S \leftarrow S \cup G'$
19: **else**
20: Mark G as the process group that cannot be split
21: **end if**
22: **end while**

3.4 Adaptive Resource Partitioning

For each game workload cluster, after the process grouping plan is determined, we also need to determine the resource partition among the groups. According to *Challenge II*, game workload changes are fast-paced and unpredictable, while the optimal resource partition for different game workloads may be different. To address this issue, we propose using a reinforcement learning (RL) technique to implement dynamic resource partitioning among process groups.

The RL is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions. It is able to distinguish good actions without complex offline profiling, and make immediate decisions according to real-time system states.

Q-Learning is one of the most popular RL algorithms. In Q-Learning, Q represents the Action-Utility function, with the value $Q(s, a)$ representing the expected reward of performing action a in state s. The main idea of Q-Learning is to learn the function Q, and use the learned function Q to guide action selection so that the benefits are maximized. Q-Learning uses a Q-table to store the updated Q-values. However, the Q-table will be explosive if the system state are continuous (such as the game workload in our case), because the Q-table needs to store the Q-value for every system state. To address this issue, we use Deep Q-Learning (DQN), which uses a deep neural network (DNN) to approximate the Q-function.

3.4.1 Model Description

As we want to partition resources among several process groups, if we create a single agent for all groups, there would be a large number of state-action pairs to be learned, making the RL model unfeasible. Therefore, we adopt the multi-agent reinforcement learning approach (MARL), and create one agent for each group. Each agent maintains its own Q-function (approximated by a standard fully connected DNN). For each agent i, let $Q_i(s_i, a_i)$ denote the Q-function (represented by a DNN) of agent i, where s_i denotes the state of agent i, a_i denotes the action of agent i. The Q-function takes the system state and the action as inputs, and outputs the corresponding Q-value.

State Space. According to (2), the response delay contributed by a process group G can be represented by the W-IPC of G, i.e., $\sum_{p \in G} \alpha_p \cdot IPC_p$. The response delay is determined by both the resources allocated to group G and the current game workload. Intuitively, less resource allocation and a heavy game workload will lead to a large $WIPC_G$ (corresponding to a large response delay), and vice versa. So, we use $\langle Y_G, WIPC_G \rangle$ to represent the system state of the agent model associated with G, where Y_G denotes the resources allocated to G and $WIPC_G$ denotes the W-IPC of group G.

Action Space. The actions of the agent model represent how resource allocation is adjusted. We represent each action by a vector of size $|R|$, denoted by $\langle r_1, r_2, \ldots, r_{|R|} \rangle$, where $r_i \in [-w_i, +w_i]$ is an integer referring to the adjustment action for resource i (positive and negative numbers represent allocation and deallocation respectively), and w_i is the maximum units that resource i can be adjusted in a single step.

Reward. The reward is the motivation for the agent which helps to distinguish between good and bad actions from a particular state. Therefore, the reward should be able to reflect the desired metric that we want to optimize. In our model, we propose to use W-IPC as a motivating reward for the agent associated with each group, implying that the agent will work towards minimizing the response delay.

3.4.2 Joint Action Selection

Generally, Q-Learning selects the action that maximizes the performance metric being optimized. The agent model defined above strives to minimize the W-IPC of the associated process group, so each agent would request as many resources as possible. However, the total amount of resources is fixed, so not all the requirements can be satisfied. Therefore, the agents need to work collaboratively.

Brute-force search for the optimal joint action is impractical due to high complexity. Hence, we propose a heuristic algorithm to find the near-optimal solution. Given the current system states, the algorithm attempts to choose the joint action that minimizes the sum of W-IPCs. Algorithm 3 presents the details. For each agent i, we represent the

action of agent i by a vector $a_i = \langle r_1^i, r_2^i, \ldots, r_{|R|}^i \rangle$, with r_j^i referring to the allocation/deallocation action for resource j. All the actions are initialized by 0, indicating that no allocation/deallocation is taken. Then, the algorithm determines the actions in a greedy manner by considering each resource independently (lines 6-18). To determine the actions for resource k, the algorithm iterates for multiple rounds until no action will be taken (lines 7-17). In each round, it examines all the valid agent pairs (the total amount of allocation/deallocation for each agent for resource k does not exceed the limitation). For each agent pair i, j, the algorithm computes the sum of the W-IPC of agent i and agent j by hypothetically transferring one unit of resource k from agent i to agent j (line 9). The W-IPC of each agent is estimated by the associated neural network. Among all the agent pairs, the algorithm selects the pair achieving the minimum sum of the W-IPC, denoted by i^*, j^* (line 13). If the performance becomes better (i.e., the sum of the W-IPC is smaller than that of last round), the algorithm accepts this action, i.e., transferring one unit of resource k from agent i^* to agent j^* (line 15).

Algorithm 3. Joint Action Selection Algorithm

1: **Input:** $N \leftarrow$ number of agents
2: $s_i \leftarrow$ current state of agent i, $1 \leq i \leq N$
3: $Q_i(s_i, a_i; \theta_i) \leftarrow$ Q-value of agent i by picking action a in state s, estimated by the neural network of agent i
4: **Output:** $a_i \leftarrow$ action of agent i, denoted by $\langle r_1^i, r_2^i, \ldots, r_{|R|}^i \rangle$
5: Initialize $\langle r_1^i, r_2^i, \ldots, r_{|R|}^i \rangle$ by $\langle 0, \ldots, 0 \rangle$ for each i from 1 to N
6: **for** each k from 1 to $|R|$ **do**
7: **repeat**
8: **for** each agent pair i, j such that $r_k^i < w_k, r_k^j > -w_k$ **do**
9: $r_k^i \leftarrow r_k^i + 1, r_k^j \leftarrow r_k^j - 1$
10: $d_{i,j} \leftarrow Q_i(s_i, a_i; \theta_i) + Q_j(s_j, a_j; \theta_j)$
11: $r_k^i \leftarrow r_k^i - 1, r_k^j \leftarrow r_k^j + 1$
12: **end for**
13: $i^*, j^* \leftarrow$ the agent pair i, j achieving the minimum $d_{i,j}$
14: **if** $d_{i^*,j^*} < Q_i(s_i, a_i; \theta_i) + Q_j(s_j, a_j; \theta_j)$ **then**
15: $r_k^{i^*} \leftarrow r_k^{i^*} + 1, r_k^{j^*} \leftarrow r_k^{j^*} - 1$
16: **end if**
17: **until** no action is taken for resource k
18: **end for**

3.4.3 Training

For our problem, we use a popular approach named value decomposition network (VDN) [25] to train the DNNs of the agents (i.e., the Q-functions). Denote by $Q(s, a)$ the joint action-value function for the system, where $s = (s_1, \ldots, s_M)$ denotes the state of all agents (M is the number of groups) and $a = (a_1, \ldots, a_M)$ denotes the joint action of all agents. The VDN approach assumes that the joint action-value function for the system can be additively decomposed into Q-functions across agents, i.e.,

$$Q(s, a) \approx \sum_{i=1}^{M} Q_i(s_i, a_i). \tag{4}$$

The VDN trains the DNNs of the agents collaboratively through minimizing the following loss function

$$\frac{1}{T} \sum_{t=1}^{T} \left(y^t - Q(s^t, a^t)\right)^2, \tag{5}$$

where T is the total training iterations, $y^t = r^t + \gamma \cdot \max_{a'} Q(s^{t+1}, a')$, r^t is the reward (the sum of the WIPCs at iteration t), γ is the reward discount factor.

The detailed training process is as follows. For each workload cluster, the resources are randomly partitioned among the process groups at the beginning, and the parameters of the corresponding neural network are randomly initialized. Then, we run the game in the cloud gaming system under various game scenes, and periodically sample game workload. For the sampling at iteration t, suppose the sampled workload belongs to cluster c, we select the joint action according to Algorithm 3, and allocate/deallocate resources according to the actions for process groups associated with cluster c. We observe the rewards (r^t) after the actions are performed and update the parameters of the corresponding DNNs by performing a gradient descent step on $(y^t - Q(s^t, a^t))^2$. It is worth noting that computing $\max_{a'} Q(s^{t+1}, a')$ is difficult due to large number of potential joint actions. To address this issue, we use Algorithm 3 to calculate an approximate solution. The learning process converges if all the neural networks are well-trained.

3.4.4 Online Partitioning

After the model is learned, it can be used for online resource partitioning. Specifically, each time a new instance of the game starts running in the system, the learned model periodically observes the real-time system states (including the current workload and W-IPCs), and takes corresponding actions. If the workload cluster has changed compared to last period, we change the grouping plan to that associated with the current workload cluster, and allocate resources among the process groups such that the response delay is minimized (the response delay is predicted using the performance model.). Otherwise, if the workload cluster has not changed, we keep the grouping plan unchanged and simply take the actions computed according to Algorithm 3 to adjust the resource partition among process groups. It is worth noting that the neural networks can also be updated in the online phase based on the rewards observed.

3.5 Overhead

We divide the overhead of the proposed framework into offline part and online part.

3.5.1 Offline Overhead

The offline overhead comes from workload clustering, process grouping and model training etc. It is worth noting that such procedures need to be done only once for a specific game, so the offline overhead is one-short.

For workload clustering, we need to measure a set of game scenes to construct the workload set W. Let t_1 denote the time for measuring one game scene (around few seconds). The total measuring overhead will be $t_1|W|$. According to the prior work [26], the running time of one iteration in k-means for K clusters is $O(K|W|)$ [26]. So, the overall complexity of Algorithm 1 (in the manuscript) will be $O(|C|(|C|+1)I|W|/2)$, where I is the number of total iterations needed by the k-means invocations.

For process grouping, let m denote the number of samples used for training the machine learning based performance model. Denote by the t_2 the time spent for collecting one sample (around few seconds). The total time for collecting samples will be mt_2. The complexity of model training algorithm (such as GBRT) is generally linear to the number of samples (around few thousands) and the dimension of input features (around few tens) [27].

In Algorithm 2, the while-loop (lines 6-22) repeats for at most $|P|$ rounds, because there are at most $|P|$ groups generated. Each group contains at most $|P|$ processes, so the for-loop (lines 10-14) also repeats for at most $|P|$ rounds. In each round, for each resource i, there will be at most C_i reallocating choices. So, for a cluster c, computing the response delay for each grouping plan has a complexity of $O(|X_c| \prod_{1 \le i \in |R|} C_i)$. Therefore, the overall complexity of Algorithm 2 is $O(|P|^2 |X_c| \prod_{1 \le i \in |R|} C_i)$. Note that $|P|$, $|R|$ and C_i $(1 \le i \le |R|)$ are generally not very large in practical system.

For training the deep Q-Learning model, suppose we do a sampling in every t_3 seconds. Let M denote the total number of samplings performed until convergence. The total time spent on model training will be Mt_3. Let L denote the number of layers of each neural network and l_i denote the number of nodes in layer i. Suppose the neural network is trained for e epochs. The complexity of the back-propagation algorithm is $O(L'Me)$, where $L' = l_1 l_2 + l_2 l_3 + \ldots + l_{L-1} l_L$ [28].

3.5.2 Online Overhead

In online partitioning, we only need to select the action based on the system states according to Algorithm 3. Let N_A denote the number of agents. It is easy to see that Algorithm 3 incurs a complexity of $O(|R|N_A^2)$, where $|R|$ is the number of resources. As $|R|$ and N_A are generally small in practical system, the computing time of Algorithm 3 is negligible.

4 Evaluations

In this section, we present the evaluations of the proposed framework. Sections 4.1 and 4.2 summarize the experimental setup and the alternatives to be compared. The results for workload clustering and process grouping are presented in Sections 4.3 and 4.4 respectively. Section 4.5 summarizes the results of the overall performance.

4.1 Experimental Setup

We evaluate the proposed framework on a popular open-source cloud gaming platform GamingAnywhere (GA) [1]. GA consists two components: GA server (for running games, encoding and streaming videos) and GA client (for decoding and displaying videos, and transmitting user commands). In our experiments, the GA server runs on a physical machine (configured with a 8 Cores Intel i7-7700 3.4 GHz CPU, 10 MB LLC, 24 GB memory, an NVIDIA GeForce GTX 1060 GPU and Linux OS), while the GA client runs on a laptop. The two machines used in the experiment are connected directly via a cable.

We consider partitioning two resources, CPU and LLC, which are the most important resources whose partitioning is supported by the hardware. The CPU is partitioned by cores, while the LLC is partitioned at a granularity of cache way (1 MB) using Intel's CAT technology. The IPCs of processes are measured by Intel's perf tool. The server side response delay in each experiment is measured using the approach proposed in [5].

We have tested 5 real games, which are *Valley*, *Nexuiz*, *Supertux2*, *Alienarena* and *Dota2*. *Valley* is a 3D benchmark, *Nexuiz* and *Alienarena* are 3D First Person Shooting (FPS) games, *Supertux2* is a 2D Adventure (AVG) game, and *Dota2* is a 3D Real Time Strategy (RTS) game. Among all the games, *Valley*, *Nexuiz* and *Dota2* are more resource intensive, while the other two games are relatively light weighted. The multiplayer games such as Dota2 require multiple participants to form a match over the network. For such games, we let one of the participants play the game on our server and other participants just use their own laptops.

4.2 Alternatives to Compare

We compare the proposed framework with several state-of-the-art alternatives.

Fully-Sharing. In this strategy, all the resources are fully shared among the processes, i.e., no resource partitioning is considered.

Complete-Isolation. This strategy allocates each process dedicated resources, so there is no resource sharing among processes. In the implementation, the amount of resources allocated to the processes are determined as follows. We randomly choose a set of game scenes and measure the performance (i.e., response delay) of all possible resource allocation plans under each game scene. We choose the resource allocation plan that minimizes the average response delay.

Static Grouping + Static Partitioning (SGSP). Similar to our approach, this strategy divides processes into groups and partition resources among process groups. The process grouping plan is determined according to the approach presented in Section 3.3, but workload clustering is not considered, i.e., this strategy maintains the same grouping plan for all game workloads. The resource partitioning plan among process groups is also static, which refers to the resource partitioning plan that minimizes the average response delay over a set of randomly selected game scenes.

Static Grouping + Dynamic Partitioning (SGDP). This strategy is a preliminary version of our approach [22], which does not consider workload clustering. It maintains a static grouping plan, while the resource partition among process groups is dynamically adjusted using Q-Learning.

4.3 Results for Workload Clustering

For each game, to construct the set of game workloads, we continuously change the game scene and measure the game workload (CPU utilization, GPU utilization and PCIe bandwidth utilization) in every 20 seconds until 5,000 datapoints are collected. After that, we run Algorithm 1 to do clustering.

Table 1 summarizes the clusters (represented by their centers) obtained for each game. We observe that the number of clusters is basically around 2∼4, which is quite small. We also observe that the differences between clusters could be significant, e.g., for game *Valley*, the GPU utilization is

TABLE 1
Workload Clustering Results

	Clusters (CPU Utl., GPU Utl., PCIev BW Utl.)
Valley	(0.159, 0.612, 0.055), (0.159, 0.426, 0.043)
Nexuiz	(0.158, 0.891, 0.080), (0.158, 0.815, 0.070)
Supertux2	(0.158, 0.214, 0.05), (0.158, 0.197, 0.060) (0.159, 0.201, 0.090)
Alienarena	(0.159, 0.245, 0.089), (0.159, 0.252, 0.010) (0.159, 0.249, 0.056), (0.159, 0.250, 0.067)
Dota2	(0.159, 0.371, 0.052), (0.159, 0.311, 0.040)

TABLE 2
Linear Regression Results

	Valley	Nexuiz	Supertux2	Alienarena	Dota2
Multiple R	0.991	0.993	0.996	0.991	0.994
Significance F	1.4E-9	5E-11	2E-14	1.2E-16	2E-10

61.2 percent for cluster 1 but only 42.6 percent for cluster 2, confirming the significance of workload clustering.

4.4 Results for Process Grouping

4.4.1 Validation of Linear Correlation

We first validate the linear correlation between the response delay and the IPCs of processes. For each game, we randomly generate 20 partitioning plans (the number of process groups, the members of each group and the resource partition among groups in each partitioning plan are all randomly generated). For each generated partitioning plan, we run the game on the GA server for a period of 10 minutes. During the running period, we measure the response delay and the IPCs of processes in every 20 seconds. Using the samples collected, we determine the parameters of formula (2) via linear regression using least squares estimation.

Table 2 shows the Multiple R (a value close to 1.0 indicates strong linear correlation) and the Significance F (a value smaller than 0.01 indicates the hypothesis test for linear correlation can be accepted) obtained. As can be seen, the Multiple R is over 0.99 and the Significant F is smaller than 0.01 for all the games, which confirms the linear relationship between IPCs and response delay.

4.4.2 Prediction Accuracy of Machine Learning Model

We next show the prediction accuracy of the machine learning model. Given a process group, the amount of resources allocated to the group and the current game workload, the model predicts the W-IPC of the group.

The training samples are generated as follows. For a specific game, we randomly generate 200 process grouping plans. For each process grouping plan, we randomly generate 10 resource partitioning schemes. For each resource partitioning scheme, we choose 5 different game scenes, and run the game on the GA server under each game scene. In each case, we measure the IPC of each process, compute the corresponding W-IPC, and generate a training sample for each process group.

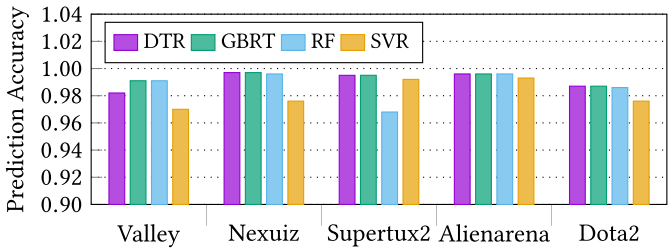


Fig. 4. Prediction accuracy of machine learning model.

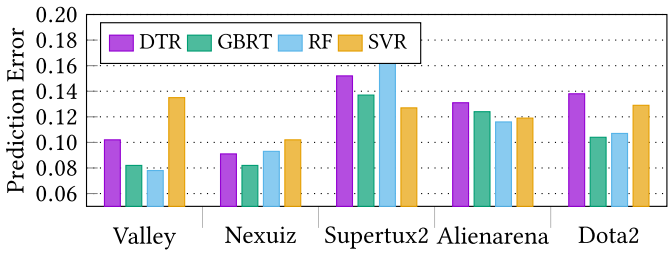


Fig. 5. Prediction error with respect to response delay.

Among the samples generated, we use 70 percent of them (randomly selected) to train the model and the remaining 30 percent for testing. We applied DTR (Decision Tree Regression), GBRT (Gradient Boosted Regression Trees), RF (Random Forest) and SVR (Support Vecor Regression) machine learning algorithms to build the machine learning model. Fig. 4 presents the mean prediction accuracy achieved by different machine learning algorithms for the tested games. The prediction accuracy is defined as $(\widetilde{WIPC} - WIPC)/\widetilde{WIPC}$, where $WIPC$ is the actually measured value and $\widetilde{WIPC}$ is the predicted value of $WIPC$. As can be seen, the highest prediction accuracy is over 98 percent for all games, indicating that the model is able to predict the W-IPC with high precision. Finally, we show the overall efficiency of the performance model (the machine learning model plus the linear regression) for response delay estimation. We randomly generate 100 process grouping plans for each game, and 5 resource partitions for each grouping plan. For each resource partition, we run the game under a randomly selected game scene and measure the server side response delay. Meanwhile, we use the machine learning model to predict the W-IPCs of process groups, based on which we estimate a response delay according to (2). Denote by d the real response delay and d_p the predicted response delay for a resource partition. We define $|d - d_p|/d$ as the prediction error for the partition.

Fig. 5 shows the average prediction error produced by each machine learning algorithm. It can be seen that the smallest prediction error achieved by the algorithms ranges from 8 to 11 percent depending on different games. As will be shown later, using the groups produced under the guide of the performance model, our framework achieves significant response delay reduction compared to other partitioning policies, which indirectly confirms the efficiency of the performance model.

Since GBRT achieves good performance for response delay prediction, we adopt GBRT as the default machine learning algorithm to train the performance model.

TABLE 3
Process Grouping Results

Valley	Groups
Cluster 1	{video, xorg}, {valley}, {server, audio, rtsp, compiz}
Cluster 2	{video}, {valley, xorg}, {server, audio, rtsp, compiz}
Nexuiz	**Groups**
Cluster 1	{video, rtsp}, {nexuiz, xorg}, {server, audio, compiz}
Cluster 2	{video, rtsp}, {nexuiz, audio, xorg}, {server, compiz}
Supertux2	**Groups**
Cluster 1	{video, audio, xorg}, {rtsp, supertux2, compiz}, {server}
Cluster 2	{video, audio, xorg, rtsp}, {supertux2, compiz}, {server}
Cluster 3	{video, audio, xorg}, {rtsp, supertux2}, {server, compiz}
Alienarena	**Groups**
Cluster 1, 2, 3	{video, alienarena}, {audio, xorg, rtsp, compiz, server}
Cluster 4	{video, alienarena, server, xorg}, {audio, rtsp, compiz}
Dota2	**Groups**
Cluster 1, 2	{video}, {server}, {dota2, audio, rtsp, xorg, compiz}

4.4.3 Process Grouping Results

Based on the performance model, we run the heuristic grouping algorithm to compute the process groups for each game. Table 3 summarizes the grouping results. It can be seen that the groups generated for each game may be different, because the workloads of games are different.

4.5 Overall Performance

In this section, we show the overall performance of the proposed framework compared to several state-of-the-art alternatives.

Based on the grouping results, we build the deep Q-Learning model for each workload cluster of each game. We use two hidden layers for each neural network, with 128 and 64 nodes respectively. For both CPU cores and LLC, at most 2 units of the resource can be reassigned in a single step. The models are trained using the approach discussed in Section 3.4.3. The learning rate and the discount factor are fixed to 10^{-5} and $\gamma = 0.5$. We do a sampling in every 5 seconds and perform 10^5 iterations. To test the model, we run each game for another 50 minutes after training. During the testing period, we measure the real time IPCs of processes every 10 seconds, and compute the joint action according to Algorithm 3. We perform the actions and also update the neural networks based on the rewards (i.e., W-IPCs) observed.

4.5.1 Benefits Over Fully-Sharing and Complete-Isolation

We first show the benefits of our framework over the fully-sharing and complete-isolation policies. For fair comparison, we pre-record the player's operations for each experiment. Then, we run each game in a replay mode (i.e., reissue the recorded operations) with fully-sharing policy, complete-isolation policy and our approach respectively. Fig. 6 presents the response delays with different policies during the testing period. Fig. 8 summarizes the delay reduction ratios of our framework compared to the other two policies. The delay reduction ratio over fully-sharing (or complete-isolation) is defined as $(d - d_t)/d$, where d and d_t denote the response delays with fully-sharing (or complete-isolation) and our framework respectively.

We have several observations from the results. First, complete-isolation performs worst among the three policies for all the games. This is consistent with our previous analysis that complete-isolation is not effective due to coarse partitioning granularity. The response delays of *Valley*, *Nexuiz* and *Dota2* with complete-isolation are much higher the other two games. We attribute this to that *Valley*, *Nexuiz* and *Dota2* are more resource intensive compared to the other two games, whose performance tends to be more influenced by improper resource allocation. Second, our framework incurs the minimum response delay among the three policies. It results in an average reduction ratio from 22 to 41 percent compared to fully-sharing policy, and an average reduction ratio from 37 to 72 percent compared to complete-isolation, which confirms the effectiveness of our approach. Third, the benefit of our framework for *Supertux2* and *Alienarena* is less significant compared to other games. We attribute this to that these two games do not require too much resource, so the impact of resource contention is insignificant.

4.5.2 Benefits Over SGSP and SGDP

We also compare our framework with SGSP (static grouping and static partitioning) and SGDP (static grouping and dynamic partitioning). Fig. 7 presents the response delays of SGSP, SGDP and our approach during the testing period.

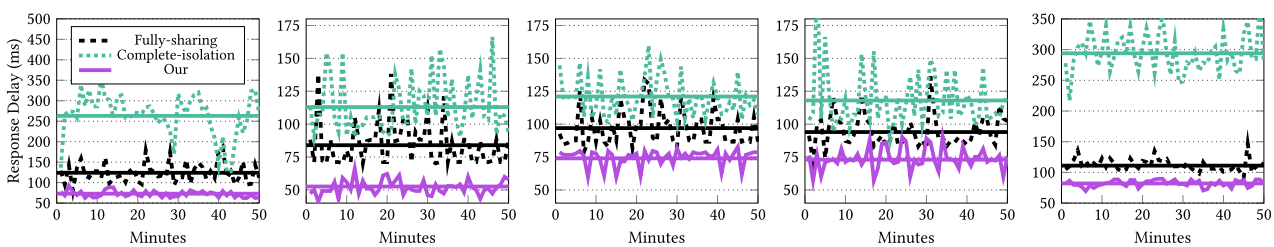


Fig. 6. Response delays of games under different partitioning policies (straight solid lines represent the mean).

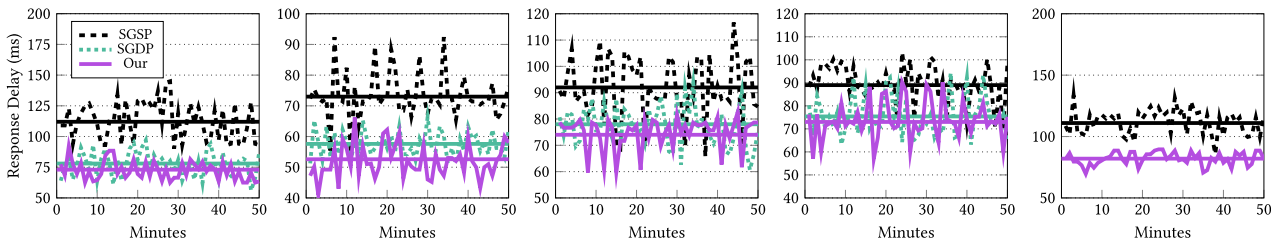


Fig. 7. Response delays of games under different partitioning policies (straight solid lines represent the mean).

Fig. 9 summarizes the delay reduction ratios of our framework compared to the other two policies. As can be seen, SGSP performs worst among the three policies for all the games. Our framework reduces the response delay by 17 to 35 percent compared to SGSP. This is because SGSP keeps the grouping plan and resource partition unchanged, which is not able to deal with dynamic game workload changes. In contrast, our approach maintains separate grouping plan for each workload cluster and dynamically adjust resource partition, which is more adaptive to workload changes.

We also observe that our framework performs slightly better than SGDP for games *Valley*, *Nexuiz Supertux2* and *Alienarena*. The advantage is up to 8 percent in terms of latency reduction. This is because SGDP maintains the same grouping plan for all game workloads, while our approach classifies game workloads into several clusters and maintains separate grouping plan for each cluster. Since there is only one workload cluster for *Dota2* (see Table 3), our framework will be equivalent to SGDP. So, the results of SGDP are not shown in Fig. 7e.

Finally, we evaluate how our framework performs for different resource capacities of the server. We use cgroups to control the total amount of resources that the processes can use. We test four different resource capacity settings ranging from small to large, and rerun the preceding experiments under each setting. Fig. 10 presents the benefits of our approach over fully-sharing for different games under different settings.

We observe that basically our framework achieves less benefit for smaller resource capacity. This is because for small resource capacity (e.g., $(2, 4)$), each unit of resource would be shared by multiple processes, so resource partitioning is more like fully-sharing. In the extreme case where we have only one CPU core and 1 MB LLC, resource partitioning will be equivalent to fully-sharing. We also observe that our framework achieves the maximum benefit when the resource capacity is $(6, 8)$ for *Supertux2* and *Alienarena*. This is because these two games are less resource-intensive. So, for large resource capacity (e.g., $(8, 10)$), the influence of resource contention is less significant and the effect of resource partitioning is not so big.

4.5.3 Effectiveness of Workload Clustering

This section gives a more deep analysis of the effectiveness of workload clustering. We first show that the game workload is unbalanced among different clusters, because the workload clustering algorithm (i.e., Algorithm 1) will not split a cluster into two sub-clusters if the sub-clusters have similar workload. An example based on game *Valley* is shown in Fig. 11. Recall that the clustering algorithm has generated two workload clusters for *Valley* (see Table 1). Denote by Cluster-1 and Cluster-2 the two clusters. Fig. 11 shows how the workload of *Valley* is split among the two clusters, where the Y-axis represents the workload (refers to GPU utilization) of *Valley* and the straight line represents a rough boundary of the two clusters (the workload above the line belongs to Cluster-1, while the workload below the line belongs to Cluster-2). It can be seen that the GPU utilization is 52∼80 percent for Cluster-1 and 24∼52 percent for Cluster-2, which confirms that the workload is unbalanced among the two clusters.

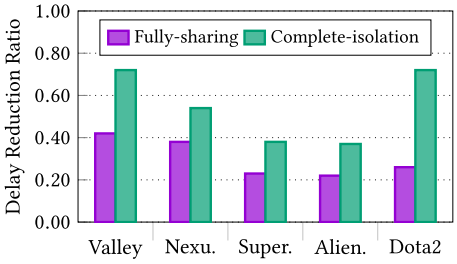


Fig. 8. Delay reduction ratios over fully-sharing and complete-isolation.

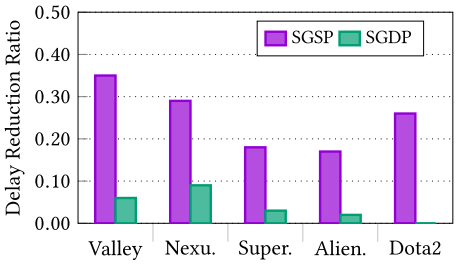


Fig. 9. Response delay reduction ratios over SGSP and SGDP.

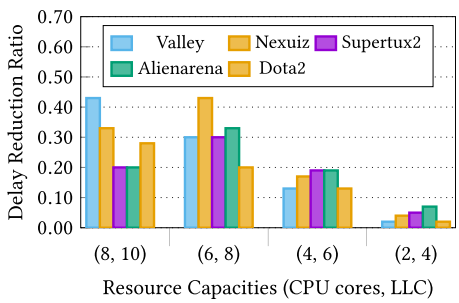


Fig. 10. Impact of resource capacity.

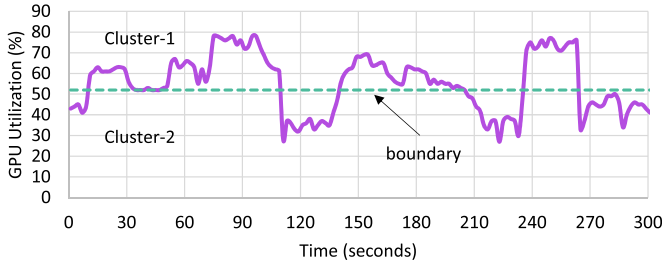


Fig. 11. GPU workload of Valley over a period of 5 minutes.

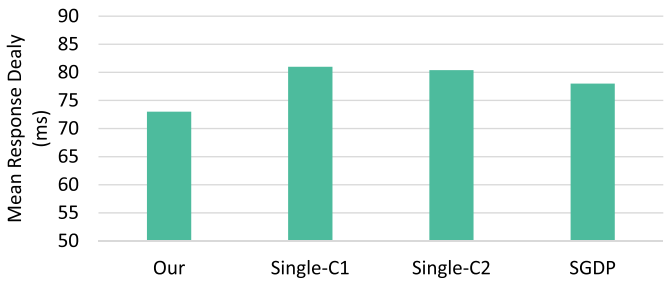


Fig. 12. Mean response delay produced by each approach.

We next show that our proposed approach can handle such unbalanced workload among different clusters. The core idea of our approach is to maintain separate machine learning model for each workload cluster, and adaptively generates partitioning decision for different workload using the corresponding machine learning model. Fig. 12 shows the mean response delay of *Valley* produced by our approach compared with some baselines. The baseline *Single-C1* uses Cluster-1's machine learning model to guide the partitioning for all the workload (in both Cluster-1 and Cluster-2), while the baseline *Single-C2* works in a similar way but uses Cluster-2's machine learning model. We have two observations from the results. First, our approach outperforms SGDP (the advantage is around 7 percent), which confirms that workload clustering is necessary and the benefit is significant. Second, *Single-C1* and *Single-C2* perform even worse than SGDP, which implies that using dedicated machine learning model for each workload cluster is important and improper machine learning model would cause significant performance loss. In summary, the results confirm that our proposed approach is effective to handle unbalanced workload among different clusters.

5 Related Work

Cloud gaming has gained more and more popularity in recent years[3]. So far, there have been over 30 cloud gaming systems released, including both commercial services such as GeForce Now [2], and LiquidSky [4], and research-oriented systems such as GamingAnyWhere [1] and Rhizome [29]. The earlier research work on cloud gaming mainly focuses on measuring the performance of existing cloud gaming systems, including the response delay [5], network traffic [30], [31] and Quality of Experience (QoE) [32], [33] in various cloud gaming systems. These measurements reveal many different characteristics of cloud gaming compared to traditional console gaming, which provide valuable knowledge for future studies of cloud gaming.

A lot of work has been conducted towards optimizing cloud gaming systems, which mainly concentrates on latency and bandwidth reduction. For example, the prior works [6], [8] attempted to reduce the interaction delay from various aspects, including reducing the network delay and speeding up the video encoding. The works [9], [34] aimed to improve the video encoding quality in cloud gaming through tunning encoding parameters or using rendering information etc. The works [35], [36] attempted to save bandwidth in cloud gaming by reducing the bit rate of video streaming. However, none of the existing work considers optimizing interaction delay in cloud gaming through resource partitioning. A preliminary study on reducing response delay in cloud gaming using RL technology has been conducted in our prior work [22]. However, workload clustering was not considered. This paper significantly improves and extends the work of [22] in many aspects.

Extensive studies have been carried out to investigate resource partitioning policies for improving system performance. However, none of them can perfectly solve our problem due to various limitations. The studies [37], [38] were mainly validated by simulation, whose results might be far different from the behaviors of real hardwares. Powered by the hardware resource isolation techniques, DICER [39], dCAT [40], Kpart [13], DCAPS [11], EMBA [12] design LLC and memory bandwidth partitioning policies for improving system performance. However, they usually establish dedicated analytical models to estimate application performance under specific partitioning schemes, which rely on extensive domain knowledge and cannot easily be extended to the scenario with multiple resources. Heracles [18], PARTIES [19], CoPart [20], Quasar [21] strive to partition multiple resources coordinately among colocated applications. However, they generally lack precise performance models to capture the complex relationship between resource partition and application performance. Therefore, such partitioning policies can only be applied in limited scenarios, e.g., CoPart [20] can only be used to improve the fairness and PARTIES can only be used to ensure QoS, which are both not applicable to minimizing response delay.

Reinforcement learning has been applied to a variety of problem domains [17], [41], [42]. However, the scenario of our partitioning problem is different from all of the existing problems, and thus the proposed methodologies can not be directly applied. Recent studies [43], [44] attempt to leverage general machine learning models to predict performance interference among colocated applications. However, resource partitioning is not considered in these works. [17] leverages multi-agent RL approach to partition cache among colocated applications. However, the LLC cannot be partially shared in their problem setting, and the proposed approach is evaluated by simulation.

6 Conclusion

In this work, we have presented an efficient and adaptive resource partitioning framework for reducing response delay in cloud gaming. Our framework divides the processes into groups and partition resources among groups. To deal with dynamic workload changes, we classify game

workload into several clusters and maintain separate process grouping plan for each cluster. Moreover, we leverage reinforcement learning technology to implement adaptive online partitioning. We validate the proposed framework in a real cloud gaming environment using real games. The experimental results show that our approach reduces the response delay by up to 41 percent compared to fully-sharing, which significantly outperforms the alternative partitioning policies.

There are several interesting directions for the future work. First, the current framework needs to build a separate Q-Learning model for each workload cluster of each game. We wish to build a general model which can be applied to any games. Second, this paper assumes that each cloud server only runs one game, which may result in resource waste. We would like to consider the scenario where multiple games are colocated on one cloud server. Third, we validate our framework on only one type of cloud gaming system. We would like to evaluate it on more platforms.

Acknowledgments

This work was supported in part by the State Key Laboratory of Computer Architecture (ICT, CAS) under Grant CARCHB202013, and in part by the National Science Foundation of Chinaunder Grant 61872201, Grant 61702521, Grant 61602266, and Grant 61972277.

References

[1] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "GamingAnywhere: An open cloud gaming system," in *Proc. 4th ACM Multimedia Syst. Conf.*, 2013, pp. 36–47.

[2] GeForce Now. Accessed: 2020. [Online]. Available: http://www.geforce.com/

[3] W. Cai *et al.*, "The future of cloud gaming [point of view]," *Proc. IEEE*, vol. 104, no. 4, pp. 687–691, Apr. 2016.

[4] LiquidSky. Accessed: 2020. [Online]. Available: https://liquidsky.com/

[5] K.-T. Chen, Y.-C. Chang, P.-H. Tseng, C.-Y. Huang, and C.-L. Lei, "Measuring the latency of cloud gaming systems," in *Proc. 19th ACM Int. Conf. Multimedia*, 2011, pp. 1269–1272.

[6] M. Semsarzadeh, A. Yassine, and S. Shirmohammadi, "Video encoding acceleration in cloud gaming," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 12, pp. 1975–1987, Dec. 2015.

[7] M. Amiri, H. A. Osman, S. Shirmohammadi, and M. Abdallah, "An SDN controller for delay and jitter reduction in cloud gaming," in *Proc. 23rd ACM Int. Conf. Multimedia*, 2015, pp. 1043–1046.

[8] K. Lee, *et al.*, "Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming," in *Proc. Annu. Int. Conf. Mobile Syst., Appl., Serv.*, 2015, pp. 151–165.

[9] 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 *Proc. Int. Conf. Multimedia*, 2011, pp. 103–112.

[10] Y. Liu, S. Dey, and Y. Lu, "Enhancing video encoding for cloud gaming using rendering information," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 12, pp. 1960–1974, Dec. 2015.

[11] Y. Xiang, X. Wang, Z. Huang, Z. Wang, Y. Luo, and Z. Wang, "DCAPS: Dynamic cache allocation with partial sharing," in *Proc. Thirteenth EuroSys Conf.*, 2018, pp. 1–15.

[12] Y. Xiang, C. Ye, X. Wang, Y. Luo, and Z. Wang, "EMBA: Efficient memory bandwidth allocation to improve performance on intel commodity processor," in *Proc. 48th Int. Conf. Parallel Process.*, 2019, pp. 1–12.

[13] N. El-Sayed , A. Mukkara, P. Tsai, H. Kasture, X. Ma, and D. Sanchez, "KPart: A hybrid cache partitioning-sharing technique for commodity multicores," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2018, pp. 104–117.

[14] J. Park, S. Park, M. Han, J. Hyun, and W. Baek, "Hypart: A hybrid technique for practical memory bandwidth partitioning on commodity servers," in *Proc. 27th Int. Conf. Parallel Architectures Compilation Techn*, 2018, pp. 1–14.

[15] T. Kim, S. Boucher, H. Lim, D. G. Andersen, and M. Kaminsky, "Simple cache partitioning for networked workloads," Rep. CMU-CS-17–125, Sch. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2017.

[16] L. Funaro, O. A. Ben-Yehuda , and A. Schuster, "Ginseng: Market-driven LLC allocation," in *Proc. Conf. Usenix Annu. Techn. Conf.*, 2016, pp. 295–308.

[17] R. Jain, P. R. Panda, and S. Subramoney, "A coordinated multi-agent reinforcement learning approach to multi-level cache co-partitioning," in *Proc. Des., Automat. Test Eur. Conf. Exhib.*, 2017, pp. 800–805.

[18] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *Proc. Int. Symp. Comput. Archit.*, 2015, pp. 450–462.

[19] S. Chen, C. Delimitrou, and J. F. Martínez, "PARTIES: QoS-Aware resource partitioning for multiple interactive services," in *Proc. Twenty-Fourth Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2019, p. 107–120.

[20] J. Park, S. Park, and W. Baek, "CoPart: Coordinated partitioning of last-level cache and memory bandwidth for fairness-aware workload consolidation on commodity servers," in *Proc. Fourteenth EuroSys Conf.*, 2019, pp. 1–16.

[21] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," *Proc. 19th Int. Conf. Architectural Support Program. Lang. Operating Syst.*2014, pp. 127–144.

[22] Y. Li *et al.*, "Themis: Efficient and adaptive resource partitioning for reducing response delay in cloud gaming," in *Proc. of the 27th ACM Int. Conf. Multimedia*,2019, p. 491–499.

[23] K. T. Nguyen. Introduction to cache allocation technology in the Intel® Xeon® processor e5 v4 family, 2019. Accessed: 2020. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/introduction-to-cache-allocation-technology.html

[24] G. Hamerly and C. Elkan, "Learning the k in k-means," in *Proc. 16th Int. Conf. Neural Inf. Process. Syst.*, 2003, p. 281–288.

[25] Peter Sunehag *et al.* "Value-decomposition networks for cooperative multi-agent learning," 2017, *arXiv: 1706.05296*.

[26] J. A. Hartiganand, M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *J. Roy. Stat. Soc. Ser. C (Appl. Statist.)*, vol. 28, no. 1, pp. 100–108, 1979.

[27] L. Mason, J. Baxter, P. Bartlett, and M. Frean, "Boosting algorithms as gradient descent," in *Proc. 12th Int. Conf. Neural Inf. Process. Syst.*, 1999, p. 512–518.

[28] D. Bienstock, G. Muñoz, and S. Pokutta, "Principled deep neural network training through linear programming," 2018, *arXiv: 1810.03218*.

[29] R. Shea, D. Fu, and J. Liu, "Rhizome: Utilizing the public cloud to provide 3D gaming infrastructure," in *Proc. 6th ACM Multimedia Syst. Conf.*, 2015, pp. 97–100.

[30] M. Manzano, J. A. Hernández, M. Urueña, and E. Calle, "An empirical study of cloud gaming," in *Proc. Annu. Workshop Netw. Syst. Support Games*, 2012, pp. 1–2.

[31] M. Claypool, D. Finkel, A. Grant, and M. Solano,"On the performance of OnLive thin client games," *Multimedia Syst.*, vol. 20, pp. 471–484, 2014.

[32] Y. T. Lee, K. T. Chen, H. I. Su, and C. L. Lei, "Are all games equally cloud-gaming-friendly? An electromyographic approach," in *Proc. 11th Annu. Workshop Netw. Syst. Support Games*, 2012, pp. 1–6.

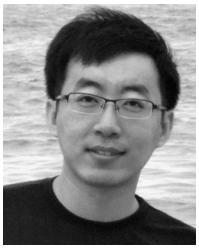
[33] K.-T. Chen, Y.-C. Chang, H. J. Hsu, D. Y. Chen, C. Y. Huang, and C. H. Hsu, "On the quality of service of cloud gaming systems," *IEEE Trans. Multimedia*, vol. 16, no. 2, pp. 480–495, Feb. 2014.

[34] M. R. H. Taher, H. Ahmadi, and M. R. Hashemi, "Power-aware analysis of H.264/AVC encoding parameters for cloud gaming," in *Proc. IEEE Int. Conf. Multimedia Expo Workshops*, 2014, pp. 1–6.

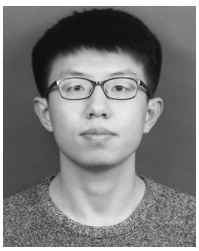
[35] H. Ahmadi, S. Z. Tootaghaj, M. R. Hashemi, and S. Shirmohammadi,"A game attention model for efficient bit rate allocation in cloud gaming," *Multimedia Syst.*, vol. 20, no. 5, pp. 485–501, 2014.

[36] L. Lin *et al.*, "LiveRender: A cloud gaming system based on compressed graphics streaming," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2128–2139, Aug. 2016.

[37] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2006, pp. 423–432.
[38] J. Chang and G. S. Sohi, "Cooperative cache partitioning for chip multiprocessors," in *Proc. ACM Int. Conf. Supercomputing*, 2007, pp. 402–412.
[39] K. Nikas, N. Papadopoulou, D. Giantsidi, V. Karakostas, G. Goumas, and N. Koziris, "DICER: Diligent cache partitioning for efficient workload consolidation," in *Proc. 48th Int. Conf. Parallel Process.*, 2019, pp. 1–10.
[40] C. Xu, K. Rajamani, A. Ferreira, W. Felter, J. Rubio, and Y. Li, "dCat: Dynamic cache management for efficient, performance-sensitive infrastructure-as-a-service," in *Proc. Thirteenth EuroSys Conf. 2018*, 2018, pp. 1–13.
[41] F. Fernndezand, L. Parker, "Learning in large cooperative multi-robot systems," *Int. J. Robot. Automat.*, vol. 16, no. 4, pp. 217–226, 2001.
[42] W.-T. Hsu and V.-W. Soo, "Market performance of adaptive trading agents in synchronous double auctions," in *Proc. Pacific Rim Int. Workshop Multi-Agents*, 2001, pp. 108–121.
[43] S. Dublish, V. Nagarajan, and N. Topham, "Poise: Balancing thread-level parallelism and memory system performance in GPUs using machine learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2019, pp. 492–505.
[44] Y. Cheng, W. Chen, Z. Wang, and Y. Xiang, "Precise contention-aware performance prediction on virtualized multicore system," *J. Syst. Archit.*, vol. 72, pp. 42–50, 2017.



Yusen Li received the PhD degree in computer science from Nanyang Technological University in 2013. He is currently an associate professor with the Department of Computer Science at Nankai University, China. His research interests include resource allocation and scheduling issues in distributed systems and cloud computing.

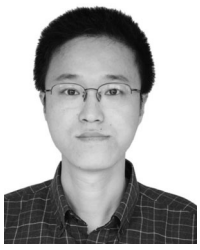


Xiwei Wang received the BEng degree in computer science from the University of Electronic Science and Technology of China. He is currently working toward the master's degree with the School of Computer Science, Nankai University, Tianjin, China. His research interests include distributed systems and serverless computing.

Haoyuan Liu is currently working toward the master's degree with the School of Computer Science, Nankai University, Tianjin, China. His research focuses on machine learning based resource management in distributed systems.

Lingjun Pu received the PhD degree from Nankai University, Tianjin, China, in 2016. From 2013 to 2015, he was a joint PhD student with the University of Gttingen, Gttingen, Germany. He is currently an associate professor with the College of Computer Science, Nankai University. His current research interests include mobile cloud or edge computing, edge caching, Internet of Things, 5G C-RAN, SDN/NFV, and opportunistic routing.

Shanjiang Tang received the BS and MS degrees from Tianjin University, China, in January 2011 and July 2008, respectively, and the Ph.D. degree from Nanyang Technological University, Singapore, in 2015. He is currently an associate professor with the College of Intelligence and Computing, Tianjin University, China. His research interests include parallel computing, cloud computing, big data analysis, and computational biology.



Gang Wang received the BSc, MSc, and PhD degrees in computer science from Nankai University, Tianjin, China, in 1996, 1999, and 2002, respectively. He is currently a professor with the School of Computer Science, Nankai University. His research interests include storage systems and parallel computing.

Xiaoguang Liu received the BSc, MSc, and PhD degrees in computer science from Nankai University, Tianjin, China, in 1996, 1999, and 2002, respectively. He is currently a professor with the School of Computer Science, Nankai University. His research interests include search engines, storage systems, and GPU computing.