Themis: Efficient and Adaptive Resource Partitioning for Reducing Response Delay in Cloud Gaming

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

Haoyuan Liu
Nankai University
Tianjin, China
liuhy@nbjl.nankai.edu.cn

Xiwei Wang
Nankai University
Tianjin, China
wangxw@nbjl.nankai.edu.cn

Lingjun Pu
Nankai University
Tianjin, China
pulingjun@gmail.com

Trent Marbach
Nankai University
Tianjin, China
trent.marbach@nbjl.nankai.edu.cn

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

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

Xiaoguang Liu
Nankai University
Tianjin, China
liuxg74@nbjl.nankai.edu.cn

ABSTRACT

Cloud gaming has been increasing in popularity recently, but issues relating to maintaining low interaction delay for users to guarantee satisfactory gaming experience is still prevalent. Interaction delays caused by server-side processing are heavily influenced by how the processes partition the resources. However, finding the optimal partitioning policy that minimizes the response delay is complicated by several critical challenges. In this paper, we propose *Themis*, a system that enables efficient and adaptive online resource partitioning for reducing response delay in cloud gaming. Briefly, *Themis* employs machine learning technology to build a performance model which is able to capture the complex relationships between resource partition and system performance. With this model, *Themis* divides the processes into disjoint groups and partitions resources among process groups, which greatly simplifies the resource partition problem while ensuring high partitioning effectiveness. To tackle dynamic workload changes, *Themis* leverages reinforcement learning to learn how different partitioning actions affect system performance in an online manner, and adaptively choose the best actions for minimizing response delay in real time. We evaluate *Themis* in a real cloud gaming environment using several real games. The experimental results show that *Themis* can reduce the response delay by 17% to 36% compared to a system without resource partitioning, and outperforms other resource partitioning policies significantly. To the best of our knowledge, this is the first work to optimize response delay in cloud gaming through resource partitioning.

MM '19, October 21–25, 2019, Nice, France

ACM ISBN 978-1-4503-6889-6/19/10…$15.00
https://doi.org/10.1145/3343031.3350941

CCS CONCEPTS

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

KEYWORDS

Cloud Gaming, Interactive Delay, Resource Partitioning, Machine Learning, Reinforcement Learning

ACM Reference Format:
Yusen Li, Haoyuan Liu, Xiwei Wang, Lingjun Pu, Trent Marbach, Shanjiang Tang, Gang Wang, and Xiaoguang Liu. 2019. Themis: Efficient and Adaptive Resource Partitioning for Reducing Response Delay in Cloud Gaming. In *Proceedings of the 27th ACM International Conference on Multimedia (MM '19), October 21–25, 2019, Nice, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3343031.3350941

1 INTRODUCTION

Cloud gaming has become increasingly popular 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 player's control commands to the cloud servers that are running the games. As opposed to traditional console gaming, cloud gaming puts the entire game running workload onto the cloud, which greatly reduces the software and hardware requirements for players running high-end games. In this way, cloud gaming can deliver high-quality gaming experience to players anytime, anywhere and on any device. With such superior features, cloud gaming has attracted great interest from both academia and industry [1, 2, 4, 8].

It is common knowledge that the high-end gaming experience is very sensitive to the games interaction delay. The interaction delay in cloud gaming results from network delay (network round-trip time), server-side response delay (for game rendering, video encoding and transmission), and the client side playout delay (for video decoding and displaying). According to the measurements in [5],

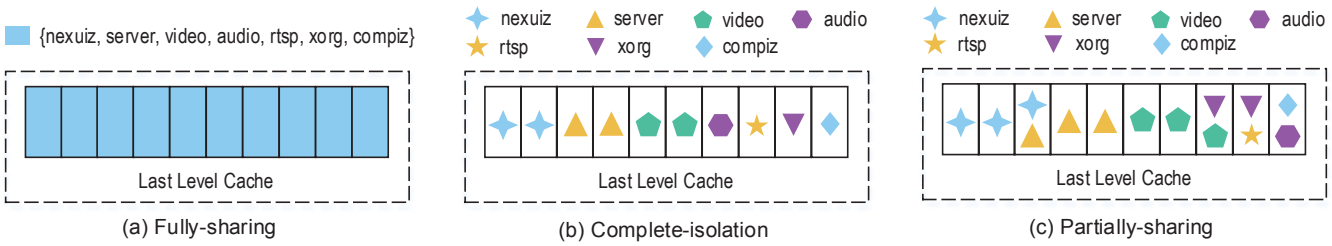


Figure 1: Three different partitioning policies. (a) Fully-sharing, with a response delay of 84 ms; (b) Complete-isolation, with a response delay of 103 ms; (c) Partially-sharing, with a response delay of 56 ms.

the server-side response delay usually dominates the interaction delay in cloud gaming. Perhaps it is not too surprising that the server-side response delay can be significant, as the bulk of the computation, including game rendering, video encoding and transmission, are all processed at the server side. Existing works have attempted to reduce response delay in cloud gaming from many perspectives, for example, speeding up the video encoding [3, 14], speculating rendering frames [11] or tunning video streaming parameters [12, 15]. However, until now it appears that no published work has considered optimizing the response delay in cloud gaming by resource partitioning.

Cloud gaming system (referring to the server system used for cloud gaming) 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 processes may have a high demand for resources, but the benefit (in terms of reduction in response delay) that the process gets from the resources may not directly correlate with this demand. For example, while a process from a video streaming service may access lot of unique cache blocks, many cache blocks are unlikely to be reused again, implying that a large amount of cache is required save on a little amount of processing time. Therefore, it makes sense to reduce the response delay through correctly partitioning the resources among processes in cloud gaming.

According to our preliminary experimental results in Section 2, the benefit of resource partitioning on response delay reduction could be significant. Partitioning cache alone can reduce the response delay by up to 30% compared to full cache sharing. Despite its advantages, resource partitioning for delay reduction in cloud gaming poses critical challenges:

Hardware limitation. Modern hardware only supports coarsely sized partitions for some resources. For example, the LLC can only be partitioned with cache way granularity. This limitation makes the policy of complete-isolation (i.e., partitioning the cache into isolated parts and allocating each process a separate part) suboptimal, 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.

Complex performance model. 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, we require a mechanism that can estimate the performance (in terms of response reduction) of any given partition, because testing all possible partitions in a real system is impractical. However, it is very challenging to build a performance model due to the complex interactions and resource contention behaviors among processes.

Dynamic workload. 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.

Resource partitioning for improving system performance has been extensively studied for general applications in prior work. However, existing solutions do not adequately address all of the previously mentioned challenges in cloud gaming. For example, KPart [6] and Hypart [13] partition only one type of resource, the works [7, 10] assume the workloads of applications are static, and [9] only considers the complete-isolation policy which is not effective as will be shown in this paper.

In this paper, we present *Themis*, a system that enables efficient and adaptive resource partitioning for reducing the server-side response delay in cloud gaming. *Themis* has several promising features which adequately address the previously-mentioned challenges. To sidestep the deficiency caused by the hardware limitation, *Themis* divides the processes into disjoint groups and partition the resources among groups. By doing this, *Themis* ensures high partitioning effectiveness while greatly simplifying the resource partition problem. For searching the optimal process groups, *Themis* builds a performance model by employing machine learning technology, which is able to capture the complex relationships between resource partition and system performance. To adapt dynamic workload changes, *Themis* leverages reinforcement learning for online partitioning. The crafted designed reinforcement learning model is able to learn the impact of various partitioning actions on system performance, and automatically selects the best partitioning actions that minimize the response delay according to real time workload.

We validate *Themis* in a real cloud gaming environment using real games. The experimental results show that *Themis* can reduce the response delay by 17% to 36% compared to a system without resource partitioning, and also outperforms other partitioning policies significantly. To the best of our knowledge, this is the first work to optimize response delay in cloud gaming through resource partitioning.

2 MOTIVATION

In this section, we describe some preliminary experiments which motivate our work. The experiments are performed in a real cloud gaming environment (described in Section 4.1). We run the game Nexuiz in the system and consider partitioning the LLC of the server among seven major processes (nexuiz, server, video, audio, xorg, rtsp and compiz) in the system. The server has 10 MB LLC and the LLC is partitioned with 1 MB (the size of cache way) granularity.

In the first experiment, 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 (as shown in Figure 1):

Fully-sharing. The fully-sharing policy (as shown in Figure 1(a)) assumes the entire LLC is shared among all the processes, i.e., there is no partitioning. In this policy, the processes freely compete for the LLC according to their demands, which may lead to inefficient allocation because the LLC demands of processes are not necessarily correlated with the response delay. In this experiment, the fully-sharing policy incurs a response delay of 84 ms.

Complete-isolation. The complete-isolation policy divides the LLC into isolated parts, and allocate each process a separate part. Resource isolation eliminates the impact of LLC contention. However, as the hardware does not support fine-grained partitions for the LLC, complete-isolation may downgrade the effectiveness of partitioning. Figure 1(b) 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. The partially-sharing policy is more flexible than complete-isolation, which allows each block of the LLC to be shared by more than one processes. In this manner, when a process does not need a whole block of LLC, it can share the block of LLC with other processes so that the LLC can be utilized more efficiently. For example, the partially-sharing plan shown in Figure 1(c) incurs a response delay of 56 ms, which is much lower than that with fully-sharing. However, partially-sharing will lead to complex sharing relationships and contention behaviors among processes, making finding an optimal solution to the corresponding partitioning problem challenging.

The first experiment motivates our work in two ways: (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 our partitioning problem, and a more efficient partitioning policy is possible.

In the second experiment, we consider four different game scenes. For each game scene, we generate a "good" LLC partitioning plan, which incurs much lower response delay compared to fully-sharing. We run the game in each game scene under the four "good" partitioning plans, and measure the response delay under each case. Figure 2 presents the results. As can be seen, the partitioning plan that is good for one game scene may not be good for other game scenes. For example, using partitioning plan 1 is good for game scene 1 (with a response delay of 59 ms), but not good for game scene 3 (with a response delay of 78 ms, which is much higher than the response delay using the partitioning plan 3). The second

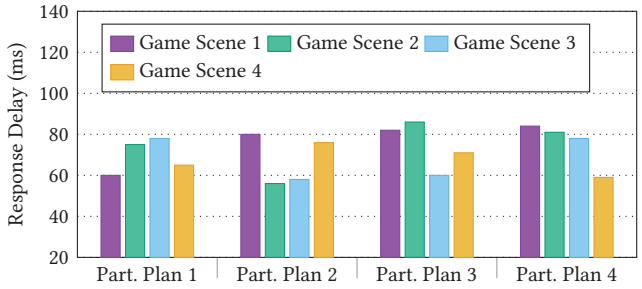


Figure 2: Response delays of game Nexuiz under different partitioning plans

experiment implies that a static partitioning plan is inefficient and motivates us to design an adaptive online resource partitioning strategy.

3 THEMIS DESIGN

3.1 Overview

Consider a cloud gaming system. Denote by P the set of major processes when the system runs a game on a cloud server (a physical server or a virtual machine), including the processes of the game, the processes for video encoding and transmission, etc. Denote by R the set of resources which can be partitioned in the cloud server. We aim to partition the resources in R among the processes in P so that the server side response delay of the game is minimized, where the response delay refers to the total server-side processing delay, including game's rendering delay, video encoding and transmission delay, etc.

Figure 3 describes the design of *Themis*, which can be divided into an offline phase and an online phase. In the offline phase, *Themis* first divides the processes in P into several disjoint groups. The resources will be partitioned among groups, and the processes in the same group will share the resources allocated to that group. The benefits of this mechanism are twofold: first, letting the processes in the same group share resources counteracts the negative effects of coarse partitioning granularity limited by the hardware, which improves the effectiveness of partitioning; second, isolating resources among groups significantly simplifies the partitioning problem compared to partially-sharing.

To assist in finding optimal process groups, a machine learning model is built that can help to estimate the benefit of any given grouping plan without testing the plan in a real system. Based on the machine learning model, a heuristic algorithm is used to find a near-optimal grouping solution. Based on the grouping results, *Themis* leverages reinforcement learning (Q-Learning) to implement online partitioning. The learning model is trained by running the game on the cloud server for a period of time. In the online phase, each time the system starts running the game, the well-trained model will be used to adaptively partition resources among processes according to real-time system states. Note that the procedures in the offline phase need to be done only once for a specific game.

3.2 Process Grouping

In this section, we present our approach that unearths the optimal or near-optimal process grouping plan that minimizes the response

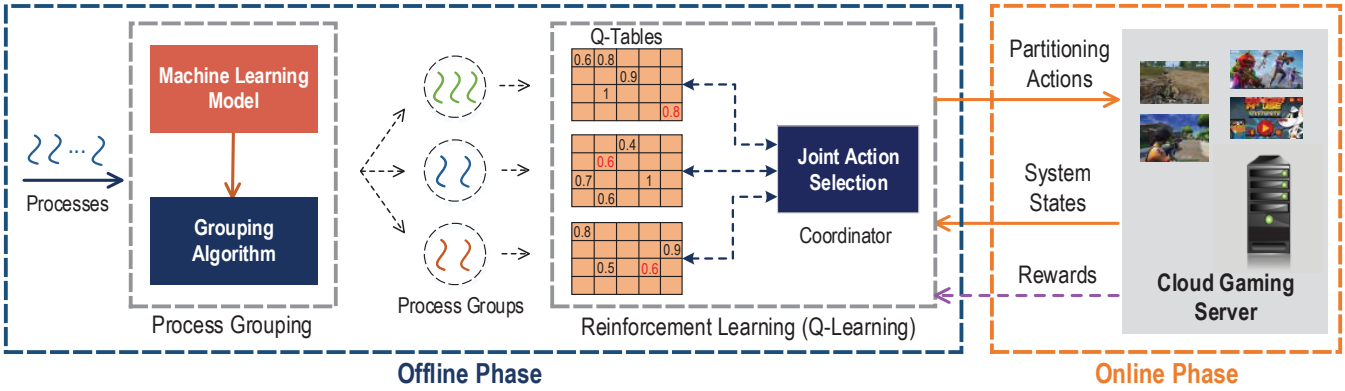


Figure 3: Design of Themis

delay. To find the optimal groups, we need to evaluate the performance (in terms of response delay) of any given grouping plan. However, the number of possible grouping plans could be substantial, and testing all of them in a real system is expensive. A more efficient way is to build a performance model to estimate the performance of grouping plans without testing them on a server. However, precise prediction of the response delay is very challenging because it is affected by many factors, including the grouping plan, resource partition among groups, resource contention and game workload.

To address this issue, we leverage machine learning techniques to build a performance model. Machine learning is good at capturing complex relationships and has been widely used in various scenarios for prediction. To train a machine learning model, we generally need to collect a large number of samples from real system. However, the measurement of response delay in cloud gaming is rather nontrivial [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), i.e., the response delay can be represented by

$$c + \sum_{p \in P} \alpha_p \cdot IPC_p \tag{1}$$

, where IPC_p is the IPC of process p, α_p is the coefficient associated with p and c is the constant term (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 (1) to approximate the response delay according to the IPCs. This technique reduces the overhead for measuring response delays significantly.

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), and 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 (1)).

To represent the group members and resource allocation of G, 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 X_G, which is a 0-1 vector of size $|P|$, with the jth element indicating whether the jth process in P appears in group G (1 indicates inclusion and 0 indicates exclusion). The resources allocated to G is denoted by Y_G, which is a vector of size $|R|$, with the jth element representing the amount of resource j in R that has been allocated to the group G.

Based on the above definitions, the machine learning model for predicting the W-IPC of G can be represented by:

$$\widetilde{WIPC_G} = \mathrm{ML}(X_G, Y_G), \tag{2}$$

where $\widetilde{WIPC_G}$ is the output of the prediction from W-IPC of the group G. Note that the W-IPC may vary as the game workload changes. Here $\widetilde{WIPC_G}$ refers to an average value of the W-IPC under various workloads. With the prediction model, given a partitioning plan, the W-IPC of each process group can be predicted and the response delay can be estimated according to (1) accordingly.

A brute-force search for the optimal grouping plan is expensive. Therefore, we propose a heuristic algorithm to obtain a near optimal solution. The details are presented in Algorithm 1. We first put all the processes in one group and allocate all the resources to the group (lines 3-5). Then, each time we try to split a group (the maximum sized group that can be split) into two groups until no group can be further split. To split a group G, we first create an empty group, denoted by G'. Then, we try to shift the processes one by one from G to G' (lines 11-13). The result of shifting a process p from G to G' is that the response delay after shifting is smaller than the response delay before shifting. The response delay after shifting refers to the minimum achievable response delay by reallocating the resources of G (i.e., Y_G) among G and G'. The response delay under a specific resource allocation is obtained according to the prediction model defined by (2). If G' is not empty after this splitting procedure, the split process is successful. For a successful split, we reallocate the resources of G among G and G' such that the response delay is minimized (line 16). If G' is empty (implying that splitting G has no benefit in terms of response delay reduction), then we mark G as a process group that cannot be split (lines 20).

In Algorithm 1, 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, the overall complexity of Algorithm is $O(|P|^2 \prod_{1 \le i \in |R|} C_i)$. As $|P|$, $|R|$ and C_i ($1 \le i \le |R|$) are generally not very large in practical system, the complexity of Algorithm 1 is acceptable.

Algorithm 1 Process Grouping Algorithm

Input: the set of all processes, denoted by P
Output: the set of process groups, denoted by S
$S \leftarrow \{P\}$ /*put all processes into one group*/
$C_i \leftarrow$ total amount of the ith resource, $1 \le i \le |R|$
$Y_P \leftarrow [C_1, C_2, \ldots, C_{|R|}]$ /*allocate all resources to P*/
while $S \neq \emptyset$ **do**
  $G \leftarrow$ the maximum group in S that can be splitted
  $d \leftarrow$ response delay under current partitioning plan
  $G' \leftarrow \emptyset$ /*create a new empty group*/
  for each $p \in G$ **do**
    shift p from G to G'
    $d^* \leftarrow$ the minimum achievable response delay by reallocating Y_G among G and G'
    If $d^* < d$, do $d \leftarrow d^*$, otherwise, move back p to G
  end for
  if $G' \neq \emptyset$ **then**
    $Y^* \leftarrow$ the amount of resources (in Y_G) should be allocated to G' for achieving the minimum response delay
    $Y_{G'} \leftarrow Y^*$, $Y_G \leftarrow Y_G - Y^*$
    $S \leftarrow S \cup G'$
  else
    Mark G as the process group that cannot be split
  end if
end while

3.3 Reinforcement Learning Model

The workload change within a game is usually fast-paced and unpredictable. We propose using a reinforcement learning (RL) technique to implement dynamic resource partitioning. The RL is a kind of reward-based machine learning technology implemented using Markov Decision Process (MDP) framework. It interacts with a dynamic environment through an autonomous agent and learns the optimal control policy based on the rewards earned for the various actions by trial. The reason we use RL for dynamic partitioning is that RL can learn to distinguish good partitions without complex offline profiling, and make immediate decisions according to real-time system states.

Q-Learning is one of the most popular RL algorithms, which has been widely used in many applications. In Q-Learning, Q represents the Action-Utility function (represented by a Q-Table), 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 by interacting with the dynamic environment, and use the learned function Q to guide action selection so that the benefits are maximized. A one-step Q-Learning model can be represented by

$$Q(s,a) \leftarrow Q(s,a) + \beta[r + \gamma \times \max_{a'} Q(s',a') - Q(s,a)], \quad (3)$$

where an agent performs an action a in state s, to receive a reward r after the action is performed. The state s' is the state after a is performed, β is the learning rate, and γ is the reward discount factor.

As we want to partition resources among several process groups, if we create a single agent (i.e., one Q-Table) for all groups, there would be a large number of state-action pairs to be learned, making the RL model unfeasible. Therefore, we create one agent for each group, which maintains its own learning in a Q-Table. We use a central coordinator to search for the globally optimal joint action for the agents. Each agent updates its own Q-Table according to the rewards observed after each action is performed.

3.3.1 Agent Model. The effectiveness of Q-Learning is highly dependent on how the agent model is defined. We propose to execute one agent per process group. The details of the agent model are as follows.

State Space. Resource partitioning aims to minimize the response delay. Consider a process group G. According to (1), the response delay contributed by 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 two factors: the resources allocated to group G and the current game workload. So, we use $\langle Y_G, WIPC_G \rangle$ to capture the system state of the agent model associated with G, where vector Y_G denotes the resources allocated to G and $WIPC_G$ denotes the W-IPC of group G. When Y_G is fixed, a heavy game workload will lead to a large $WIPC_G$ (corresponding to a large response delay) and vice versa. So, the change in $WIPC_G$ is a consequence of resource reallocation actions of the agent and the game workload changes. In the real implementation, the values of $WIPC_G$ are quantized into k different levels corresponding to k states (k is a tunable integer parameter).

Action Space. The actions of the agent model represent resource reallocation. A reallocation action is represented 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 which refers to the requested changes in the units of resource i, and w_i is the maximum units that resource i can be reassigned in a single step. Positive and negative numbers in the above actions represent allocation and deallocation respectively.

Reward. The reward is the motivation for the agent which helps to distinguish between beneficial and detrimental actions from a particular state. Therefore, the reward should be able to reflect the desired metric that we wish to optimize. Since we want to optimize for response delay, we propose to use W-IPC as a motivating reward for the agent associated with each group. As a result, the agent will work towards minimizing the response delay by finding the best resource partition.

3.3.2 Joint Action Selection. Generally, Q-Learning selects the action that maximizes the performance metric being optimized. The agent model defined above would request resources for the minimum possible W-IPC of the associated group. Such a model would motivate each agent to request as many resources as possible. However, not all the requests from the agents can be satisfied because the total amount of resources is fixed. Hence, the agents need to work together towards a common goal.

Brute-force search for the optimal joint action is expensive. We propose a heuristic algorithm to find the near-optimal solution. Given the current states and the Q-Tables of the agents, the algorithm accesses the Q-Tables and attempts to select a joint action that minimizes the sum of W-IPCs. The details of the algorithm are presented in Algorithm 2. For each agent i, the action of agent i (denoted by a^i) is represented by a vector $\langle r_1^i, r_2^i, \ldots, r_{|R|}^i \rangle$, where r_j^i refers to the allocation/deallocation action for resource j. The

Algorithm 2 Joint Action Selection Algorithm

1: **Input:** $N \leftarrow$ number of agents
2: $s_i \leftarrow$ current state of agent i, $1 \le i \le N$
3: $Q_i(s,a) \leftarrow$ Q-value of agent i by picking action a in state s
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: $d_{i,j} \leftarrow Q_i(s_i, a^i) + Q_j(s_j, a^j)$
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** $d_{i^*,j^*} < Q_i(s_i, a^i) + Q_j(s_j, a^j)$ **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**

actions are all initialized by 0 (line 5), 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 repeats for multiple rounds until no action will be taken for resource k (lines 7-17). In each round, it iterates over all the possible agent pairs such that the allocation/deallocation of resource k are still valid for the agents (the total amount of allocation/deallocation for each agent does not exceed the limitation). For each agent pair i, j, the algorithm computes the W-IPC sum of agent i and agent j by hypothetically allocating one unit of resource k to agent i and deallocating one unit of resource k from agent j (line 9). Among all the agent pairs, the algorithm picks the pair achieving the minimum W-IPC sum, denoted by i^*, j^* (line 13). If the W-IPC sum is smaller than that of last round, the algorithm allocates one unit of resource k to agent i^* and deallocates one unit of resource k to agent j^* (line 15).

Let N_A note the number of agents. It is easy to see Algorithm 2 incurs a complexity of $O(|R|N_A^2)$. As $|R|$ and N_A are generally small in practical system, the computing time of Algorithm is negligible.

3.3.3 Training and Online Partitioning. Q-Learning learns the Q-function by interacting with the real environment. In our implementation, the learning process is as follows. The entries of Q-Tables are all initialized by 0. Then, we run the game in the cloud gaming system for multiple episodes. At the beginning of each episode, we randomly partition the resources among the process groups. At each time step in the episode, we select the joint action according to Algorithm 2, and allocate/deallocate resources for process groups according to the actions. We observe the rewards (W-IPCs) after the actions are performed and update the corresponding entries in the Q-Tables according to (3). The learning process converges if the changes to the entries in the Q-Table are below a certain threshold value.

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 (i.e., W-IPCs) and computes the joint action according to Algorithm 2. It is worth noting that the Q-Tables can also be updated in the online phase based on the rewards observed.

4 EVALUATIONS

4.1 Experimental Setup

We use GamingAnywhere (GA) to build the cloud gaming environment, which is a popular open-source cloud gaming platform [8]. The GA has two components: GA server (for running games, encoding and streaming videos) and GA client (for decoding and displaying videos, and transmitting user commands). We set up the GA server 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. We set up a laptop and install GA client on it. The two machines used in the experiment are connected directly via a cable.

We implement *Themis* on the cloud gaming system. Specifically, we consider partitioning two resources, CPU and LLC, which are the most important resources whose partitioning can be supported by the hardware. The CPU can be partitioned by cores or by usage, using the Linux cgroup tool. In this paper, we partition the CPU by cores, while the proposed solution can easily be applied to the case that partitions the CPU by usage. The LLC is partitioned using Intel's CAT (Cache Allocation Technology) technology, where the minimum partitioning granularity is 1 MB. We use Intel's perf tool to measure the IPCs of processes. The server side response delay in each experiment is measured using the approach proposed in [5].

We have tested 5 real games, which are Dota2, Nexuiz, Alienarena, Supertux2 and Valley. Dota2 is a 3D Real Time Strategy (RTS) game, Nexuiz and Alienarena are 3D First Person Shooting (FPS) games, Supertux2 is a 2D Adventure (AVG) game, and Valley is a 3D benchmark. Among all the games, Dota2 and Valley are more resource intensive, while Supertux2 is the most light weighted.

4.2 Process Grouping

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 (1) via linear regression using least squares estimation. Table 1 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.

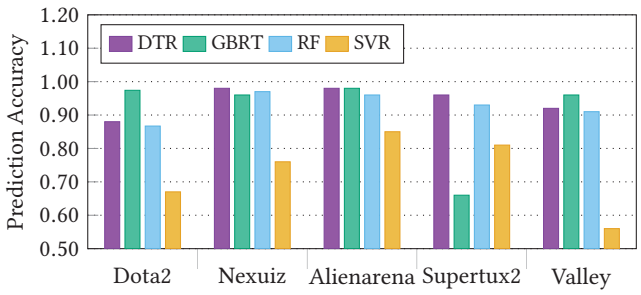


Figure 4: Prediction accuracy of machine learning model

Table 1: Linear regression results

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

Prediction accuracy of machine learning model. We next show the prediction accuracy of the machine learning model. Given a process group and the amount of resources allocated to the group, the model predicts the W-IPC of the group. Note that the W-IPC may vary over time even for a fixed resource allocation because it is also affected by the game workload. Here we predict an average W-IPC under various game workloads. The training samples are generated as follows. For a specific game, we randomly generated 1000 partitioning plans. For each generated partitioning plan, we run the game on the GA server with the plan for a period of 10 minutes and measure the average IPC over the running period for each process. We compute the W-IPC and generate a training sample for each process group.

Among the samples generated, we use 70% of them (randomly selected) to train the model and the remaining 30% for testing. We applied DTR (Decision Tree Regression), GBRT (Gradient Boosted Regression Trees), RF (Random Forest) and SVR (Support Vector Regression) machine learning algorithms to build the machine learning model. Figure 4 presents the mean prediction error produced by different machine learning algorithms for the tested games. The prediction error is defined as $(\widetilde{WIPC} - WIPC)/WIPC$, 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 96% for all games, indicating that the model is able to predict the W-IPC with high precision.

Overall efficiency of the performance model. 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 partitioning plans for each game, and run the game under each partitioning plan for a period of 5 minutes. We measure the response delay in every 20 seconds and compute the average response delay in the testing period, which we regard as the real response delay for the partitioning plan. Meanwhile, we use the machine learning model to predict the W-IPCs of process groups for the partitioning plan, based on which we estimate a response delay according to (1). Denote by d the real response delay and d_p the predicted response delay for an arbitrary partitioning plan. We define $|d-d_p|/d$ as the prediction error for the partitioning plan. We observe that the average prediction error is around 11% for all the games (the details are not shown due to space limitation). As will be shown later, using the groups produced under the guide of the performance model, *Themis* achieves significant response delay reduction compared to other partitioning policies, which indirectly confirms the efficiency of the performance model.

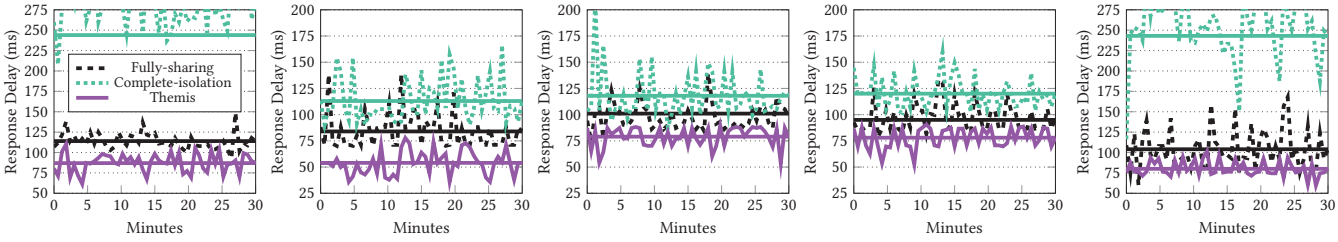
Based on the performance model, we run Algorithm 1 to compute the process groups for each game. Table 2 summarizes the grouping results. It can be seen that the groups generated for each game may be different, because the workloads of games are different.

4.3 Online Partitioning

Based on the grouping results, we build the Q-Learning model for each game. In the implementation, we quantize the values of W-IPC into 10 levels. 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 3.3.3. Each model is trained for 20 episodes and each episode runs for 30000 time steps (5 seconds for each time step). To test the model, we run each game for another 30 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 2 based on the Q-Tables. We perform the actions and also update the Q-Tables based on the rewards (i.e., W-IPCs) observed.

Benefits over fully-sharing and complete-isolation. We first show the benefits of *Themis* over the fully-sharing and complete-isolation policies. For fair comparison, we record the player's operations during the test of *Themis*. Then, we run each game in a replay mode (i.e., reissue the recorded operations) with fully-sharing policy and complete-isolation policy (the complete-isolation plan that incurs the minimum average response delay among all possible complete-isolation plans, obtained by brute-force search) respectively. Figure 5 presents the response delays with different policies during the testing period. Figure 6 summarizes the delay reduction ratios of *Themis* 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 *Themis* 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 Dota2 and Valley with complete-isolation are much higher than those of other games. We attribute this to Dota2 and Valley being more resource intensive compared to other games, whose performance tends to be more influenced by improper resource allocation. Second, *Themis* incurs the minimum response delay among the three policies. It results in an average reduction ratio from 17% to 36% compared to fully-sharing policy, and an average reduction ratio from 33% to 68% compared to complete-isolation, which confirms the effectiveness of *Themis*. Third, the benefit of *Themis* for Supertux2 is less significant compared to other games. We attribute this to Supertux2 being a relative light-weight 2D game which does not require too much resource, so the impact of resource contention is insignificant.



(a) Dota2 (b) Nexuiz (c) Alienarena (d) Supertux2 (e) Valley

Figure 5: Response delays of games under different partitioning policies (straight solid lines represent the mean)

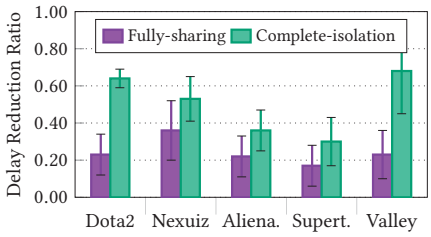


Figure 6: Response delay reduction ratios

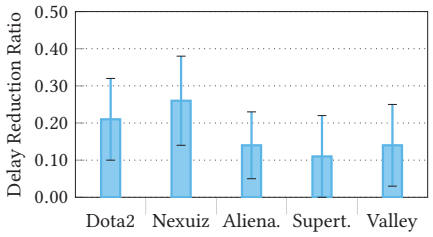


Figure 7: Benefits of Themis over static partitioning

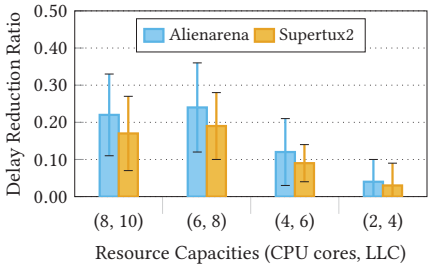


Figure 8: Impact of resource capacity

Table 2: Process groups produced for each game

Games	Process Groups
Dota2	{dota2, rtsp}, {video}, {server, comzip, xorg, audio}
Nexuiz	{nexuiz}, {server}, {video, rtsp, xorg, audio, comzip}
Alienarena	{alienarena, video, rtsp, audio}, {server, xorg}, {comzip}
Supertux2	{supertux2}, {rtsp, xorg}, {video, server, audio, comzip}
Valley	{valley, server, rtsp, audio}, {video}, {xorg, comzip}

Benefits over static partitioning. We also compare *Themis* with static partitioning strategy. The static partitioning plan is computed as follows: we partition the resources (CPU cores and LLC) among the process groups (shown in Table 2) such that the response delay is minimized, where the response delay of a specific partition is estimated using the performance model (machine learning plus linear regression). Figure 7 shows the ratios of response delay reduction of *Themis* compared to the static partitioning plan. As can be seen, *Themis* reduces the response delay by 11% to 26% for the games. This reduced effectiveness in the results is due to static partitioning not adapting to dynamic game workload, while *Themis* is able to adjust the resource partition as game workload changes.

Impact of resource capacity. Finally, we evaluate how *Themis* 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. Figure 8 presents the benefits of *Themis* over fully-sharing for Alienarena and Supertux2 under different settings (other games have similar results). As can be seen, *Themis* achieves the maximum benefit when the resource capacity is $(6, 8)$ for both the two games. This is because for large resource capacity (e.g., $(8, 10)$), the influence of resource contention is less significant, and thus the effect of resource partitioning is not so big. 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.

5 CONCLUSIONS

In this work, we have presented *Themis*, a novel methodology that enables efficient and adaptive resource partitioning for response delay reduction in cloud gaming. *Themis* divides the processes into groups and partition resources among groups. To find the optimal grouping plan, *Themis* leverages machine learning technology to predict the response delay of any given partitioning plan. To deal with dynamic workload changes, *Themis* leverages reinforcement learning technology to implement adaptive online partitioning. We validate *Themis* in a real cloud gaming environment using real games. The experimental results show that *Themis* reduces the response delay by up to 36% compared to fully-sharing, which significantly outperforms the alternative partitioning policies.

There are several interesting directions for future work. First, *Themis* needs to build a separate Q-Learning model for 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 *Themis* on only one type of cloud gaming system. We would like to evaluate *Themis* on more platforms.

6 ACKNOWLEDGMENT

This work is partially supported by National Science Foundation of China (61602266, 61872201, 61702521, U1833114, 61702286); Natural Science Foundation of Tianjin City (16JCYBJC41900, 17JCYBJC15300, 18ZXZNGX00140, 18ZXZNGX00200, 18JCYBJC15600).

REFERENCES

[1] 2018. GeForce Now. (2018). http://www.geforce.com/

[2] 2018. LiquidSky. (2018). https://liquidsky.com/

[3] Maryam Amiri, Hussein Al Osman, Shervin Shirmohammadi, and Maha Abdallah. 2015. An SDN controller for delay and Jitter reduction in Cloud Gaming. (2015), 1043–1046.

[4] 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.

[5] Kuan Ta Chen, Yu Chun Chang, Po Han Tseng, Chun Ying Huang, and Chin Laung Lei. 2011. Measuring the latency of cloud gaming systems. In *International Conference on Multimedea 2011, Scottsdale, Az, Usa, November 28 - December*. 1269–1272.

[6] N. El-Sayed, A. Mukkara, P. Tsai, H. Kasture, X. Ma, and D. Sanchez. 2018. KPart: A Hybrid Cache Partitioning-Sharing Technique for Commodity Multicores. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 104–117.

[7] Liran Funaro, Orna Agmon Ben-Yehuda, and Assaf Schuster. 2016. Ginseng: Market-Driven {LLC} Allocation. In *2016 Annual Technical Conference*. 295–308.

[8] 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.

[9] Rahul Jain, Preeti Ranjan Panda, and Sreenivas Subramoney. 2017. A coordinated multi-agent reinforcement learning approach to multi-level cache co-partitioning. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 800–805.

[10] Thomas Kim, Sol Boucher, Hyeontaek Lim, David G Andersen, and Michael Kaminsky. 2017. Simple Cache Partitioning for Networked Workloads. (2017).

[11] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. 2015. Outatime:using speculation to enable low-latency continuous interaction for mobile cloud gaming. In *International Conference on Mobile Systems, Applications, and Services*. 151–165.

[12] Yao Liu, Sujit Dey, and Yao Lu. 2015. Enhancing video encoding for Cloud Gaming using rendering information. *IEEE Transactions on Circuits & Systems for Video Technology* 25, 12 (2015), 1960–1974.

[13] Jinsu Park, Seongbeom Park, Myeonggyun Han, Jihoon Hyun, and Woongki Baek. 2018. Hypart: a hybrid technique for practical memory bandwidth partitioning on commodity servers. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*. ACM, 5.

[14] Mehdi Semsarzadeh, Abdulsalam Yassine, and Shervin Shirmohammadi. 2015. Video encoding acceleration in Cloud Gaming. *IEEE Transactions on Circuits & Systems for Video Technology* 25, 12 (2015), 1975–1987.

[15] Shu Shi, Cheng Hsin Hsu, Klara Nahrstedt, and Roy Campbell. 2011. Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming. In *ACM International Conference on Multimedia*. 103–112.