The Server Allocation Problem for Session-based Multiplayer Cloud Gaming

Yunhua Deng, Yusen Li, Ronald Seet, Xueyan Tang, *Senior Member, IEEE* and Wentong Cai

***Abstract*—Advances in cloud computing and GPU virtualization are allowing the game industry to move into a *cloud gaming* era. In this paper, we consider multiplayer cloud gaming (MCG), which is the natural integration of multiplayer online gaming and cloud gaming paradigms. With MCG, a game server and a set of rendering servers for the players need to be located and launched in the clouds for each game session. We formulate an MCG server allocation problem with the objective of minimizing the total server rental and bandwidth cost charged by the cloud to support an MCG session. The MCG server allocation problem is hard to solve optimally. We propose several efficient heuristics to address the problem and carry out theoretical analysis for the proposed hill-climbing algorithm. We conduct extensive experiments using real Internet latency and cloud pricing datasets to evaluate the effectiveness of our proposed algorithms as well as several alternatives. Experimental results show that our best algorithm can achieve near-optimal cost under real-time latency constraints.**

***Index Terms*—Cloud gaming, multiplayer online games, real-time multimedia, resource allocation, cloud computing.**

I. Introduction

NOWADAYS, real-time multimedia applications are getting increasingly rich and demanding, which motivates the trend for offloading certain media computing tasks (e.g., video transcoding, image recognition) from users' electronic devices to the cloud in an interactive manner [1]. Following this trend, a new paradigm of gaming – *cloud gaming*, has emerged [2]–[4]. In cloud gaming, a video game is hosted on a cloud server and virtually all the graphics required for the game are processed by the server. An end user's machine only needs to display/play the game screen/audio captured and streamed by the server, as well as to relay the user's game input commands (e.g., mouse clicks, keystrokes, touch gestures) to the server for game controls. With cloud gaming, users can play resource-intensive video games on less powerful devices (e.g. mobile phones) anywhere and anytime over the Internet. Cloud gaming also has the potential to reduce the energy consumption of user devices compared to running native games [5].

In this paper, we focus on *multiplayer cloud gaming (MCG)*, a new form of multiplayer online gaming in the cloud gaming era. In conventional multiplayer online gaming, a game server, which is the authoritative source of game events, disseminates state updates about the game session to its connected clients, allowing them to maintain their own versions of the game world. Meanwhile, the games are also required to run on the clients' devices. Therefore, the clients not only need sufficient hardware resources to render the games, but also require the games to be installed locally on their devices. By contrast, in MCG, the clients can enjoy multiplayer online gaming without these hassles thanks to the advantages brought by cloud gaming. Figure 1 depicts the architecture of MCG. The game server is identical to that in traditional multiplayer online gaming. The rendering servers, on the one hand, act as the "clients" that are connected to the game server in traditional multiplayer online gaming to receive/exchange state updates, and on the other hand, function as the servers which host the game application instances for their connected clients in cloud gaming. The rendering servers process game graphics and logics and capture the game scenes for the clients to display. The clients are known as "thin-clients" which merely transit user interaction (UI) input to the rendering servers. A number of cloud gaming systems supporting MCG have emerged in recent years [6]–[9]. A recent position paper [10] anticipates that MCG will be one of the future directions of cloud gaming, due to several additional benefits introduced by applying cloud gaming to multiplayer games, such as instant engagement and fair competition, etc.

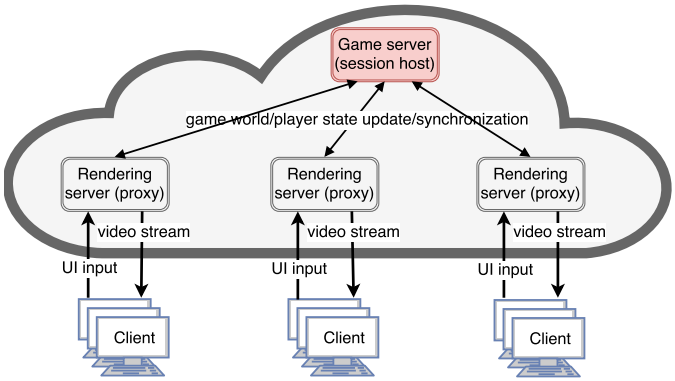


Fig. 1. A typical MCG architecture.

With MCG, a game server and a set of rendering servers for the players need to be located and launched in the clouds for each game session. In this paper, we focus on the provisioning and acquisition of these servers from available data centers with the minimum cost. Heterogeneous server and bandwidth prices from different datacenters together with the real-time interaction requirements of gaming applications present unique

Y. Deng is with Huawei Technologies, Shenzhen, China. E-mail: dengyunhua1@huawei.com.

Y. Li is with the Department of Computer Science and Information Security, Nankai University, Tianjin, China. E-mail: liyusen@nbjl.nankai.edu.cn.

R. Seet, X. Tang and W. Cai are with the School of Computer Science and Engineering, Nanyang Technological University, Nanyang Avenue, Singapore. E-mail: {ronaldseet, asxytang, aswtcai}@ntu.edu.sg.

opportunities and challenges for MCG server allocation. The main contributions of this paper are:

- We formulate an MCG server allocation problem, with the objective of minimizing the total server rental and bandwidth cost charged by the cloud to support an MCG session. This problem can be viewed as a natural extension to the traditional game server selection problem, with several distinctive characteristics related to the cloud gaming paradigm.
- We propose a set of efficient heuristic algorithms to address the MCG server allocation problem, and conduct extensive experiments using real-world data to evaluate the performance of the proposed algorithms as well as several alternatives. Our best algorithm can achieve near-optimal cost while satisfying the latency constraints.
- In addition to the experimental evaluation, we also carry out theoretical analysis on the approximability of our proposed hill-climbing algorithm (i.e., the LAC algorithm) to the MCG server allocation problem which is NP-hard.

This paper is extended from our preliminary work [11], with several key novel aspects including:

- The generalization of the system model and problem formulation by considering heterogeneous video streaming (downloading) bitrates for clients (Section IV).
- A newly proposed hill-climbing greedy algorithm for the MCG server allocation problem which considers all the cost-effectiveness factors in assigning clients to datacenters in an integrated manner (Section V.C). Experimental results show that the new algorithm outperforms the earlier algorithms (Section VI).
- The theoretical analysis of the new hill-climbing greedy algorithm to show that it provides strong approximation guarantees on the server allocation results (Section V.D).
- The extension of the proposed algorithms to deal with scenarios where clients come and leave dynamically (Section VI-C).

The rest of this paper is organized as follows. Section II presents a summary of the related work. Section III discusses the requirements of MCG. Section IV formulates the MCG server allocation problem. Section V proposes a set of cost-aware client-to-datacenter assignment algorithms to efficiently address the MCG server allocation problem. The experimental setup and results are shown and discussed in Section VI. Section VII discusses some possible extensions and future work. Finally, Section VIII concludes the paper.

II. Related Work

Cloud computing provides a scalable and cost-effective way for hosting and delivering large-scale services over the Internet to geo-distributed end users. The problem of how to place services in multiple data centers is a key challenge and has been extensively studied. Existing work on this problem generally targeted at minimizing energy consumption (or electricity cost) while guaranteeing some performance requirement such as response time [12], [13]. In more complicated scenarios, the demand and resource price fluctuations were also considered [14], [15]. However, these service placement models are quite different from our problem which aims to minimize the total cost of cloud server rental and bandwidth usage while satisfying the real-time delay constraint among clients in a multiplayer cloud gaming session.

Our work is also relevant to the game server selection problems for online gaming or Distributed Interactive Applications (DIAs) in general. In DIAs, given a set of clients and a set of candidate server locations, the objective is to choose one or more servers to minimize the interaction delay among the clients [16]–[20], or to place the minimum number of servers while satisfying some QoS requirements [21]–[24]. However, cloud gaming is quite different from traditional online gaming. In addition to the game servers, the rendering servers for the clients also need to be deployed in cloud gaming, which makes the problem much more challenging. There is also some work [25] focusing on exploiting well-provisioned inter-datacenter connections to reduce the communication latency from clients to servers. In contrast, the purpose of cloud gaming is to leverage the rich computing resources in the clouds for clients to play high-quality games without dedicated hardware equipped.

Some server provisioning issues in cloud gaming have been studied recently. Hong *et al.* [26] considered the problem of how to consolidate multiple rendering servers (virtual machines) on a physical machine in order to provide high gaming Quality of Experience (QoE) in a cost-effective way. The request dispatching and admission control issues were studied in [27] with the goal of cutting down the provisioning cost while guaranteeing the queuing delay requirement. However, all of the above work has focused on cloud gaming operators with their own private datacenters, and monetary cost issues for hosting cloud gaming in public clouds were not considered. A server rented from the public cloud is normally charged at a fixed rate regardless of whether it is fully or partially utilized. Many cloud gaming systems have acquired resources (virtual machines) from public clouds to serve as rendering and game servers [8], [9]. The problem of how to allocate cloud resources to minimize the interaction delay among clients was studied in [28]. However, the rental cost of servers was not considered either. Very recently, Tian *et al.* [29] studied the cost for running single-player games in cloud gaming where each player is assigned to a separate and dedicated server. In contrast, we focus on multiplayer cloud gaming where server sharing among clients plays an important role in optimizing the server cost.

Basiri *et al.* [30] studied cost-efficient resource provisioning for cloud gaming by constructing detailed queuing and delay models. They assumed that the cloud gaming provider has fine-grained control of managing the virtual machines on the physical machines, which is normally available in proprietary cloud systems only. In contrast, we consider a public cloud environment where the mapping of virtual machines to physical machines is transparent to the cloud gaming provider. Moreover, they assumed a coordination cost among the players playing together that is proportional to the number of physical

machines to which these players are allocated. Different from [30], we explicitly model the placement of a dedicated game server that acts as a central coordinator for processing game actions and managing game states, and the network latencies between the game server and the rendering servers of players. Zhang *et al.* [31] carried out experimental studies and proposed a performance model to analyze the resource consumption of games and the capacities of cloud gaming servers. We make use of some empirical results in [31] to guide the setting of server capacity in our experiments.

Since our focus in this paper is on the server allocation issue in cloud gaming, for the detailed review of recent advances on other research topics in cloud gaming, readers are referred to a recent comprehensive survey [32] and the references therein.

III. Requirements of MCG

In an MCG session, each player connects to a cloud server that captures and streams the game screen to the player. This cloud server is called the *rendering server* or *R-server* for that player. A single R-server may be shared by multiple players in the same session, if the server has sufficient capacity to process all the rendering and streaming workloads of these players. All the R-servers communicate with a *game server* or *G-server* which manages and updates the game state during the session.

In conventional multiplayer online gaming, *matchmaking* is the process of setting up game sessions with the goal of finding a required number of players for each game session subject to some constraint on the network latency between any player and the game server [16], [33]. In MCG, matchmaking becomes more complicated due to the presence of R-servers in-between the players and the G-server. Consider the scenario where we are given a set of clients C that form the player group, a set of datacenters D, and the G-server located at a network location g which may either be one of the datacenters or outside all datacenters. An MCG session is called *feasible* if we can find at least one datacenter to place an R-server for each player such that, 1) the connection latency from each player to the G-server through his R-server is below a threshold L_G, and 2) the connection latency from each player to his R-server is below another threshold L_R. That is, for each client $c \in C$, there exists at least one datacenter $d_c \in D$ to satisfy the following constraints:

$$l(c, d_c) + l(d_c, g) \leq L_G \text{ and } l(c, d_c) \leq L_R, \qquad (1)$$

where $l(c, d_c)$ is the network latency from c to d_c, and $l(d_c, g)$ is the network latency from d_c to g, while L_G and L_R are the latency thresholds.

The first latency threshold L_G is related to the maximum tolerable network latency for traditional online gaming. Note that, in traditional online gaming, the actual game is stored, executed and rendered on the player's computer locally and the player's global in-game actions such as firing at the enemies are sent to the G-server. The consequences of an in-game action such as an enemy being killed are only seen by the player after the lag caused by the time taken for the action to reach the G-server and the time taken for the update to come back to the player's computer over the Internet. The second latency threshold L_R is related to the maximum tolerable network latency for cloud gaming. Note that, in cloud gaming, the actual game is stored, executed, and rendered on the R-server remotely and the player's raw input commands such as keystrokes are sent to the R-server for controlling the game. The consequences of an input command such as "moving forward", can only be displayed on the player's computer screen after a time lag. This lag is caused by the time taken for the input command to reach the R-server for rendering and the time taken for the game scene updates to be streamed back to the player's computer over the Internet.

The network latency is the major source of lag for traditional online gaming [34]. The delay caused by video coding (i.e., server-side encoding and client-side decoding) adds another source of lag for cloud gaming [35]. In addition, in traditional online gaming, lag can often be hidden to some extent using client-side compensation techniques such as dead-reckoning [36], [37], but in cloud gaming, lag is far more difficult to hide as the client does not maintain any game state locally [38]. Moreover, the smoothness of live video streaming in cloud gaming is heavily dependent on the network quality which is likely to degrade as the network latency (distance) increases. Therefore, L_R is often required to be smaller than L_G. Several empirical studies [39], [40] have been conducted for traditional online gaming and cloud gaming, respectively. These studies confirmed that the network latency requirement in cloud gaming is generally more stringent than that in traditional online gaming.

Both latency thresholds L_G and L_R are dependent on the game genre [39], [40]. In general, fast-paced games such as first-person shooter or car racing games, have more stringent latency requirements (i.e., lower latency thresholds) than moderate-paced games such as third-person role-playing or strategy/simulation games (e.g., L_G = 150 ms and L_R = 100 ms for fast-paced games, and L_G = 300 ms and L_R = 200 ms for moderate-paced games according to [39], [40]). Once the connection latencies are below the corresponding thresholds, any further reduction of them would not necessarily bring along easily-noticeable improvement in the player's perceived playability [39], [40]. Thus, our objective of MCG server allocation is not to minimize the above two connection latencies, but to optimize the server allocation cost subject to the latency thresholds. Specifically, our objective is to minimize the total monetary cost of server and network resources charged by the cloud to support an MCG session, from the perspective of the game host that can be either the player who initiates the session or a cloud gaming service provider.

IV. Problem Formulation

Once a feasible MCG session is set up after matchmaking, we need to allocate servers to ensure that each client can be serviced by an R-server launched at some datacenter.

A. Definitions and Notations

For each client $c \in C$, all the datacenters that satisfy constraints (1) are called the *eligible* datacenters for c. Let

$E_c \subseteq D$ denote the set of eligible datacenters for each client c. We define the *server price*, denoted by $s(d) \rightarrow \mathbb{R}^+$, to be the price for renting one server instance from a datacenter $d \in D$ to act as an R-server. Prevalent cloud billing schemes (e.g., Amazon EC2's pricing model) normally charge each server instance rented at a fixed rate regardless of its utilization. For simplicity, we assume that the servers rented from different datacenters have the same capacity $k \in \mathbb{Z}^+$. That is, each server rented can handle the rendering and streaming workloads of up to k clients. Note that the actual delays experienced by clients may increase with the number of clients connected to the server. Some existing work has constructed models to predict the experienced delay based on the number of users served [26], [30]. These models can be used to determine the maximum delay at the server side due to factors such as queueing for up to k clients connecting to the server. Then, such amount of delay can be deducted from the latency thresholds L_G and L_R in constraint (1) when deriving the eligible datacenters of clients. This would ensure that the overall delays experienced by clients are within the original thresholds if the number of clients assigned to the server does not exceed the limit.

Since cloud gaming is bandwidth-intensive due to the need of high-definition video streaming, we also take bandwidth cost into account in the problem formulation. We define the *bandwidth price*, denoted by $b(d) \rightarrow \mathbb{R}^+$, to be the unit price of data transfer from a datacenter $d \in D$ to clients.[1] We further denote $v(c) \rightarrow \mathbb{R}^+$ as the amount of outbound data transfer induced by each client $c \in C$ from its assigned datacenter during an MCG session. In practice, $v(c)$ can be obtained according to the estimated video streaming bitrate of each client and the duration of the MCG session. The video streaming bitrate and other codec parameters (e.g., frame rate and resolution) can affect the energy consumption of the client device and may be set by taking into account the clients battery lifetime requirements and the type of wireless access links [5], [41].

In general, a multiplayer online game session usually spans some tens of minutes [42], which matches the billing interval of on-demand server instances offered by most public cloud providers. Thus, for the convenience of presentation, we consider the situation where a new set of servers needs to be opened to service a requested MCG session. The possibilities of server sharing across sessions will be discussed in Section VI-C.

B. MCG Server Allocation Problem

We formulate the MCG server allocation problem with the goal of finding the number of servers to open at each datacenter such that each client can be assigned to one R-server located at one of its eligible datacenters and the total server and bandwidth cost is minimized. For each client c and each of its eligible datacenters $d \in E_c$, we define a binary variable $X_c^d \in \{0,1\}$ to describe whether c is assigned to d. The MCG server allocation problem can be written as follows:

$$\min \sum_{d \in D} \left(s(d) \cdot \left\lceil \frac{\sum_{c \in C} X_c^d}{k} \right\rceil + b(d) \cdot \sum_{c \in C} \left(X_c^d \cdot v(c) \right) \right), \quad (2)$$

subject to

$$\sum_{d \in E_c} X_c^d = 1, \quad \forall c \in C, \quad (3)$$

$$X_c^d \in \{0,1\}, \quad \forall c \in C,\ d \in E_c, \quad (4)$$

The term $\lceil \left(\sum_{c \in C} X_c^d \right) / k \rceil$ in objective (2) calculates the number of servers opened at datacenter d to run games for all the clients assigned to it. Constraint (3) ensures that each client is assigned to exactly one of its eligible datacenters.

In general, the server capacity k is dependent on the server's resource volume and the rendering workload of the particular game. In the special case where k = 1, objective (2) can be minimized by simply opening a server to serve as the R-server for each client c at its eligible datacenter in E_c with the minimum value of $s(d) + b(d) \cdot v(c)$, i.e., the aggregate server rental and bandwidth cost. When $k \geq 2$, this may not minimize the total cost, since it does not consider server sharing among clients. If a datacenter is assigned less than k clients, the server opened at the datacenter cannot be fully utilized, leading to capacity wastage and unnecessary cost. It is easy to see that the server allocation problem is tightly coupled with the assignment of clients to datacenters.

C. Problem Variants

The MCG server allocation problem defined so far assumes that the G-server is given whose location is fixed prior to the allocation of R-servers. We call it the *basic problem*. A typical scenario for the basic problem is that G-servers are offered and hosted by some multiplayer game server providers [43], [44] and cloud gaming providers only need to set up the R-servers in an on-demand manner for game sessions. Besides the basic problem, we further consider a more general problem where the G-server did not exist and needs to be launched in one of the datacenters to support a game session as well. Nowadays, public clouds such as Amazon EC2 have already offered a comprehensive suite of services and products for game hosting [45], so that one can easily set up multiplayer game servers similar to those offered by game server providers. This motivates us to study the above variation of the problem in which the G-server location also needs to be determined along with R-servers. We call it the *general problem*. In the general problem, changing the G-server's location may result in different sets of eligible datacenters E_c for each client $c \in C$. Thus, an MCG session is feasible only if there exists at least one datacenter $d_g \in D$ for running the G-server such that for all clients $c \in C$, $E_c \neq \emptyset$. We call such a datacenter *eligible* for placing the G-server. Let $D_G \subseteq D$ denote the set of eligible datacenters for placing the G-server. Apparently, choosing different G-server locations in D_G may lead to different assignments of clients to datacenters, which consequently may give rise to different total costs of the server allocation solutions for the same set of clients. Note that the

[1] In major public cloud service providers such as Amazon EC2 and Microsoft Azure, only the outbound data transfer is charged while the inbound data transfer is for free.

basic problem can be viewed as a special case of the general problem when there is only one eligible datacenter for placing the G-server.

D. NP-hardness

It is easy to establish polynomial reductions from the set cover problem to show that both our basic and general problems are NP-hard. Specifically, the basic MCG server allocation problem degenerates to the classic set cover problem when all datacenters have the same server/bandwidth cost and the server capacity k is larger than the number of clients in a session. In such a case, at most one server is needed at each datacenter and the objective is equivalent to minimizing the number of datacenters chosen to cover all clients of a session subject to the latency constraints.

In the next section, we propose a set of greedy heuristic algorithms to address MCG server allocation problems.

V. Greedy Heuristics

We first present the algorithms for the basic problem and then we extend them for the general problem. All these algorithms assume that a feasible MCG session is already set up by matchmaking. These algorithms can be used for online resource allocation. When a request for running a new MCG session arrives, the algorithms can be executed based on the requested session size and cloud resource prices etc. to compute the server allocation for the clients in the session and open cloud servers to serve the clients and host the session.

A. Price-based Assignment

We begin with a simple algorithm that determines the assigned datacenter d_c for each client $c \in C$ individually based on the price consideration.

Lowest-Combined-Price (LCP) Algorithm: Every client is assigned to the datacenter with the minimum *combined price* among all of its eligible datacenters. The combined price associated with each datacenter d is defined as $s(d)/k + b(d) \cdot v(c)$, which assumes that the servers can be "partially" rented to serve individual clients.

In the LCP algorithm, if a client has more than one eligible datacenters offering the same lowest price, ties can be broken by assigning the client to the nearest datacenter in terms of network latency. The time complexities of the LCP is $O\big(|C| \cdot |D|\big)$. Once the *client-to-datacenter assignment* is determined, the next procedure is to open servers at each datacenter to serve as R-servers for clients. We call this procedure *client-to-server assignment* which proceeds as follows:

1) At each datacenter $d \in D$, open $\lceil \big(\sum_{c \in C} X_c^d\big)/k \rceil$ servers where $\sum_{c \in C} X_c^d$ is the total number of clients assigned to d, and k is the capacity of each server.
2) Assign each client $c \in C$ to a server opened at d_c which has spare capacity, that is, it has been assigned less than k clients so far.

In the above client-to-server assignment procedure, there is at most a capacity of $k-1$ wasted at each datacenter. It is intuitive that the LCP algorithm would produce an optimal server allocation for the case of k = 1, since there is no wasted capacity at any datacenter and each client incurs the lowest total server and bandwidth cost. For the cases of $k \geq 2$, however, it is possible for the LCP algorithm to result in significant capacity wastage at some datacenters. For instance, if the clients' eligible datacenters with the lowest combined prices are very diverse, most clients may be assigned to distinct datacenters when using the LCP algorithm, which loses the opportunity of server sharing, especially when the number of clients are small or the number of datacenters available are large. The next algorithm we present determines the client-to-datacenter assignment on a datacenter basis instead of on an individual client basis in order to promote server sharing among clients.

B. Wastage-aware Assignment

For the convenience of presentation, we introduce the following definitions. Let $C_d \subseteq C$ be the set of clients that can be covered by datacenter $d \in D$ (i.e., d is an eligible datacenter for each of these clients) and have not been assigned to any datacenter. Let $w(d)$ be the projected capacity wastage if all the clients in C_d are assigned to d, which is given below based on the aforementioned client-to-server assignment procedure:

$$w(d) = \begin{cases} 0 & \text{if } |C_d| \bmod k = 0, \\ k - (|C_d| \bmod k) & \text{if } |C_d| \bmod k > 0. \end{cases}$$

Intuitively, the server cost, the bandwidth cost and the capacity wastage are major factors to consider for minimizing the total cost of server allocation. While the price-based assignment algorithms consider one or both of the first two factors, the next heuristic called *Lowest-Capacity-Wastage (LCW)* attempts to improve the cost-effectiveness of server allocation by explicitly avoiding the capacity wastage at each datacenter in the server allocation.

Algorithm *Lowest-Capacity-Wastage (LCW)*

1: For each datacenter $d \in D$, initialize C_d as the set of all the clients that can be covered by d.
2: Let $d^* = \arg\min_{\{d \in D, C_d \neq \emptyset\}} w(d)$ be the datacenter with the lowest projected capacity wastage whose C_d is not empty. If more than one datacenter is found with the same lowest projected capacity wastage, let d^* be the one with the lowest server price. If there are still ties, break them arbitrarily.
3: Assign all the clients in C_{d^*} to d^*. That is, for each client $c \in C_{d^*}$, set $d_c = d^*$.
4: For each datacenter $d \in D$, set $C_d \leftarrow C_d \setminus C_{d^*}$.
5: Stop if $C_d = \emptyset$ for every datacenter $d \in D$; else, go to Step 2.

In the LCW algorithm, the number of iterations is at most $|D|$. In each iteration, finding d^* can be done in $O\big(|D|\big)$ time. For each datacenter d, the set C_d can only shrink over iterations, and therefore the time complexity for updating C_d over all iterations is bounded by $O(|C|)$. Hence, the overall time complexity is $O\big(|D| \cdot (|C| + |D|)\big)$. Similar to the previous algorithms, after determining d_c for each client $c \in C$, the

client-to-server assignment procedure is executed to finalize the server allocation for the requested MCG session.

The LCW algorithm explicitly considers minimizing the projected capacity wastage at each datacenter by trading off the chance to assign each individual client to its cheapest datacenter in terms of server, bandwidth or combined price. To consider all the three factors (i.e., server cost, bandwidth cost and capacity wastage) in an integrated manner, we next explore a greedy hill-climbing approach which determines the client-to-datacenter assignment on a server basis instead of on an individual client or datacenter basis.

C. Hill-Climbing Assignment

The proposed hill-climbing approach works as follows. Starting from an empty assignment, the algorithm assigns a group of clients to a selected datacenter in each iteration. At each iteration, we consider the options of opening a new server at each datacenter $d \in D$ with $C_d \neq \emptyset$ to service a selected group of clients, denoted by $N_d \subseteq C_d$. Suppose that the clients $c_i \in C_d$ are sorted in the ascending order of $v(c_i)$. When considering the option of opening a new server at d, N_d is chosen to include the first n clients in C_d that minimize the amortized cost per client subject to the server capacity constraint, i.e.,

$$n = \arg\min_{1 \leq m \leq \min\{|C_d|,k\}} \Big\{ \frac{s(d) + \sum_{i=1}^{m} v(c_i) \cdot b(d)}{m} \Big\},$$

where k is the server capacity. Here, the amortized cost $\big(s(d) + \sum_{i=1}^{m} v(c_i) \cdot b(d)\big)/m$ is the total server rental and bandwidth cost of the new server normalized by the number clients assigned to it. Then, we denote $\alpha(d)$ as the amortized cost per client if a new server is opened at datacenter d to service clients N_d:

$$\alpha(d) = \frac{s(d) + \sum_{c_i \in N_d} v(c_i) \cdot b(d)}{|N_d|}.$$

In each iteration, we choose the option with the lowest amortized cost per client and assign the selected group of clients N_d to the corresponding datacenter d. The algorithm proceeds until all the clients are assigned to datacenters. We call this heuristic *Lowest-Amortized-Cost (LAC)*, and present it formally as follows.

Algorithm *Lowest-Amortized-Cost (LAC)*

1: For each datacenter $d \in D$, initialize C_d as the set of all the clients that can be covered by d, and sort C_d in the ascending order of $v(c_i)$ of its clients c_i.
2: Compute N_d and $\alpha(d)$ for each datacenter $d \in D$ where $C_d \neq \emptyset$.
3: Let $d^* = \arg\min_{\{d \in D, C_d \neq \emptyset\}} \alpha(d)$ be the datacenter with the lowest amortized cost per client. If there are ties, break them arbitrarily.
4: Assign all the clients in N_{d^*} to d^*. That is, for each client $c \in N_{d^*}$, set $d_c = d^*$.
5: For each datacenter $d \in D$, set $C_d \leftarrow C_d \setminus N_{d^*}$.
6: Stop if $C_d = \emptyset$ for every datacenter $d \in D$; else, go to Step 2.

In the LAC algorithm, the number of iterations is at most $|C|$. In each iteration, finding d^* can be done in $O\big(|D| \cdot |C|\big)$ time where $O\big(|C|\big)$ is contributed by finding N_d for each datacenter $d \in D$. Similar to the LCW algorithm, the time complexity for updating C_d over all iterations is bounded by $O\big(|C|\big)$. Thus, the overall time complexity is $O\big(|C| \cdot (|D| \cdot |C| + |C|)\big) = O\big(|C|^2 \cdot |D|\big)$. Like the previous algorithms, after determining d_c for each client $c \in C$, the client-to-server assignment procedure is executed to finalize the server allocation for the requested MCG session.

D. Approximability Analysis of LAC Algorithm

Since the amortized cost per client of each option increases monotonically over iterations in the LAC algorithm, this hill-climbing heuristic can provide strong approximation guarantees on the server set constructed for each MCG session.

Theorem 1. *The LAC algorithm returns a server allocation solution with cost of at most* $(\ln k + 1)$ *times the cost of the optimal solution, where* k *is the server capacity.*

Proof. When the LAC algorithm assigns a group of clients N_d to a chosen datacenter d in an iteration, imagine that it charges the cost per client for that iteration to each newly assigned client. That is, each client assigned is charged a cost of $\big(s(d) + \sum_{c_i \in N_d} v(c_i) \cdot b(d)\big)/|N_d|$ and the total cost charged on all the clients in N_d is $s(d) + \sum_{c_i \in N_d} v(c_i) \cdot b(d)$, i.e., the aggregate server rental and bandwidth cost for opening a new server at datacenter d to service the clients in N_d. Then, the total cost of the server allocation solution constructed by the algorithm equals the total amount charged on all the clients C, and each client is charged exactly once.

Let $\mathcal{R}^* = \{R_1, R_2, ..., R_o\}$ be the set of R-servers opened by the optimal server allocation solution. For notational convenience, we use each $R_i \in \mathcal{R}^*$ to denote the set of clients assigned to the corresponding R-server as specified by the optimal solution, and use d_{R_i} to denote the datacenter at which the R-server is opened. Consider any $R_i = \{c_p, c_{p-1}, ..., c_1\}$ in $\mathcal{R}^*$ and we know $p \leq k$ where k is the server capacity. Let's examine the assignment of these clients by the LAC algorithm. Without loss of generality, suppose that $c_p, c_{p-1}, ..., c_1$ is the order in which the LAC algorithm assigns the clients of R_i. Consider each client $c_h \in R_i$ and the iteration in which the LAC algorithm assigns client c_h. Assume that the LAC algorithm assigns client c_h to a datacenter d (note that d may be different from the datacenter d_{R_i} where R_i is assigned in the optimal solution). In this iteration, the LAC algorithm may assign multiple clients in R_i to d together and by definition, these clients have successive indexes in the sequence of $c_p, c_{p-1}, ..., c_1$. Assume that $c_{h+\sigma} (\sigma \geq 0)$ is the highest-indexed client assigned to d by the LAC algorithm in the iteration. Then, at the start of the iteration, $h + \sigma$ clients of R_i are unassigned. Thus, if the LAC algorithm were to choose datacenter d_{R_i} in that iteration and assign these $h + \sigma$ clients

to d_{R_i}, it would pay a cost per client of

$$\begin{aligned}&\frac{s(d_{R_i}) + \sum_{j=1}^{h+\sigma} v(c_j) \cdot b(d_{R_i})}{h+\sigma}\\ &\leq \frac{s(d_{R_i}) + \sum_{j=1}^{h+\sigma} v(c_j) \cdot b(d_{R_i})}{h}\\ &\leq \frac{s(d_{R_i}) + \sum_{c_j \in R_i} v(c_j) \cdot b(d_{R_i})}{h}\\ &= \frac{\Delta_{R_i}}{h},\end{aligned}$$

where Δ_{R_i} is the aggregate server rental and bandwidth cost paid by the optimal solution for R_i. Due to its greedy nature, the LAC algorithm pays at most Δ_{R_i}/h per assigned client on average in this iteration. As a result, the LAC algorithm charges a cost of at most Δ_{R_i}/h for client c_h. Summing over h, the total amount charged to all the clients in R_i is at most $\Delta_{R_i} \cdot H_p$, where $H_p = \sum_{j=1}^{p} 1/j \leq \sum_{j=1}^{k} 1/j < \ln k + 1$, and k is the server capacity. Summing over all $R_i \in \mathcal{R}^*$, and noting that every client is assigned to some R-server in $\mathcal{R}^*$, the LAC algorithm charges a total cost of at most $\sum_{R_i \in \mathcal{R}^*} \Delta_{R_i} \cdot (\ln k + 1) = (\ln k + 1) \cdot OPT$ for all clients, where OPT is the cost of the optimal server allocation solution. Hence, the theorem is proven. □

E. Extension for the General Problem

For the general problem where there are a set of eligible datacenters D_G for placing the G-server, we extend each of the above algorithms by examining every possible choice of the G-server location d_g in D_G and choosing the one which results in the minimum server allocation cost. For example, the extended LAC algorithm works as follows:

1) For each choice of d_g in D_G, find the set of eligible datacenters E_c for each client c, apply the LAC algorithm to determine the client-to-datacenter assignment, and calculate the total cost[2] of all servers opened after the client-to-server assignment procedure.
2) Find the lowest cost among all the costs calculated in Step 1 and then return the corresponding d_g and client-to-datacenter assignment solution.

Since we need to iterate through all the choices of d_g in D_G and $D_G \subseteq D$, the time complexity of the extended LAC algorithm for the general problem is bounded by $O\big(|C|^2 \cdot |D|^2\big)$. Likewise, the time complexities of the extended LCP and LCW algorithms for the general problem are $O\big(|C| \cdot |D|^2\big)$ and $O\big(|D|^2 \cdot (|C| + |D|)\big)$, respectively. In practice, the number of datacenters $|D|$ offered by public clouds is typically in the order of tens. For example, Amazon and Microsoft currently operate 30 datacenters in total globally. Normally, the number of clients $|C|$ is also in the order of tens in a multiplayer online game session [46], [47]. Therefore, all the above algorithms should be efficient enough to instantly solve the MCG server allocation problem so that players will not suffer from undesirable waiting times. In fact, according to our experiments, the running times of our algorithms are typically less than a few milliseconds on a commodity computer, which are negligible compared to the cloud server startup times.[3]

[2]The total cost in the general problem also includes the cost for renting the G-server which can be much cheaper than that for renting an R-server because the R-server requires GPU resources for game rendering and/or video encoding.

TABLE I
SERVER RENTAL AND DATA TRANSFER PRICES (US$).

Datacenter	Baseline server price	GPU server price	Data transfer price
EC2-Virginia	0.532	2.600	0.090
EC2-Oregon	0.532	2.600	0.090
EC2-California	0.616	2.808	0.090
EC2-Ireland	0.585	2.808	0.090
EC2-Singapore	0.784	4.000	0.120
EC2-Tokyo	0.770	3.592	0.140
EC2-Sao Paulo	0.761	3.720	0.250
Azure-Hong Kong	0.902	4.604	0.138
Azure-Virginia	0.616	3.012	0.870
Azure-Ireland	0.584	2.804	0.870
Azure-Singapore	0.784	4.000	0.138
Azure-Amsterdam	0.672	3.224	0.870
Azure-California	0.616	2.808	0.870

VI. EXPERIMENTAL EVALUATION

We evaluate the performance of our proposed algorithms by making use of real-world Internet latency measurement data [48], [49] and cloud pricing models of two leading public cloud service providers – Amazon EC2 and Microsoft Azure.

A. Evaluation Methodology

Network latency datasets. We make use of two network latency datasets to set up trace-driven experiments. The first dataset collected by Wu *et al.* [48] contains latency (RTT) measurements between PlanetLab nodes and datacenters from Amazon and Microsoft. The second dataset collected by Garcia-Dorado *et al.* [49] contains latency (RTT) measurements between all pairs of datacenters from Amazon and Microsoft. By integrating these two datasets, we simulate a network that is formed by 253 PlanetLab nodes and 13 datacenters with 7 from Amazon and 6 from Microsoft. Assuming that a client is located at each PlanetLab node, we have a total number of 253 clients which is sufficiently large for setting up MCG sessions.

Cloud pricing datasets. Since an R-server handles the game rendering workload for its assigned clients, it is necessary to use cloud servers equipped with GPUs to serve as the R-servers. GPU servers are available in Amazon EC2 (we choose the *g2.8xlarge* model as the GPU server), but this is not the case in Microsoft Azure. To deal with this, we derive the prices of GPU servers in Microsoft's datacenters (assuming that they will be available in the future) based on the prices of GPU servers in Amazon's datacenters as follows. First, we find two non-GPU baseline server types from Amazon and Microsoft respectively which are best-matched in terms of CPU core count and memory size: the *m3.2xlarge* model from EC2 and the *D4* model from Azure. Then, for each Azure datacenter, we approximate its GPU server's price via multiplying its non-GPU baseline server's price by the ratio

[3]The startup times of cloud servers are normally within one minute based on our measurements on Amazon EC2.

TABLE II
L_G AND L_R SETTINGS ACCORDING TO [39], [40].

(L_G, L_R)	Game Genre	Game Pace
(150, 100)	racing or action, etc.	fast
(300, 200)	strategy or simulation, etc.	moderate

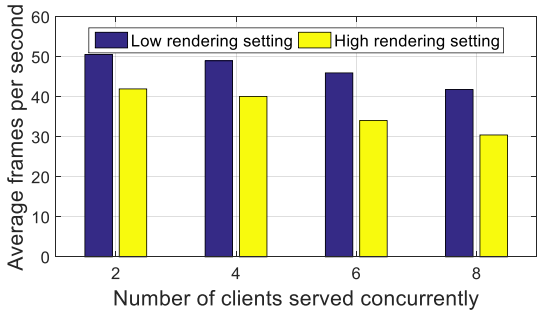


Fig. 2. Heaven Benchmark's average frame rate against the number of clients served by an R-server concurrently.

between the GPU server's price and the non-GPU baseline server's price in the nearest EC2 datacenter to this Azure datacenter. Table I shows the rental price per server[4] used in our experimental evaluation. Also shown in Table I is the outbound data transfer price per GB from each datacenter. The amount of outbound data transfer induced by each client from its assigned datacenter during an MCG session is set to 1 GB to 5 GB in a random manner, in order to reflect different video streaming bitrates across clients of the same session. For instance, a client with video streaming at a bitrate of 2 Mbps (the minimum requirement for HD streaming) for 1 hour will induce about 1 GB data transfer from its assigned datacenter.

Latency thresholds. The thresholds for the latencies from the clients to the G-server (L_G) and the R-servers (L_R) are set according to the empirical data provided by [39] and [40], respectively. Table II summarizes the settings of L_G and L_R for different game genres (all are round-trip delay times in milliseconds). Having the latencies below such thresholds offers satisfactory playability according to the above studies.

Session size. In general, the number of clients $|C|$ in a multiplayer game session is in the order of tens, thereby we let $|C| \in \{10, 20, 30, 40, 50\}$ in our experiments. For each setting of $|C|$ and each setting of (L_G, L_R), we generate 1000 feasible MCG sessions where the clients are randomly chosen from the 253 PlanetLab nodes subject to the latency thresholds. Note that, for simplicity, we use the median latency provided by the aforementioned dataset in all the matchmaking processes that generate the feasible sessions. The latency between client-server pairs is expected to be relatively stable during a game session period since each session typically lasts for just a few tens of minutes [42]. If this is not the case, a higher percentile (e.g., 90th percentile) of network latency can be compared against the latency thresholds in the matchmaking process to cater for the latency variation. In the experimental results, we compute and plot the average server allocation cost and standard deviation of these 1,000 sessions resulting from each algorithm for performance comparison.

[4]The GPU servers are not available in the EC2 Sao Paulo datacenter, thereby we derive the price for this datacenter according to its non-GPU server's price ratio to the EC2 Virginia datacenter.

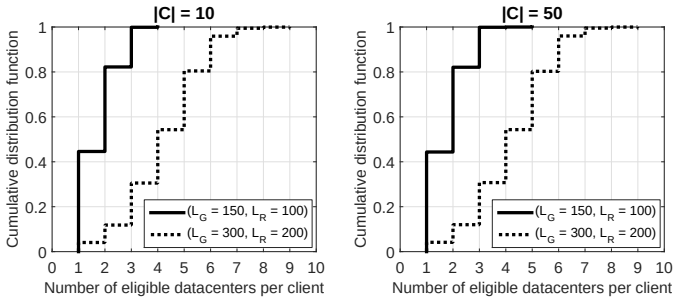


Fig. 4. CDF of the number of eligible datacenters (i.e., $|E_c|$) per client across all sessions for the basic problem.

Server capacity. The server capacity k is dependent on the R-server's resource volume and the game's rendering workload. To emulate a range of workload intensities imposed on each R-server (i.e., a *g2.8xlarge* instance), we set $k \in \{2, 4, 6, 8\}$. We have conducted a set of measurements to verify the feasibility of these settings of k, based on a widely-used real-time graphics performance benchmarking tool[5] and a cloud gaming system prototype developed by ourselves [8]. The core of our prototype is a virtual client solution that contains two components: a thin client and a v-client. The thin client is a browser-based client program that runs on the users computer or mobile device. It sends the users command inputs to the applications running on the cloud servers, and decodes and displays the video streams received from the cloud servers. The v-client runs on the cloud server. It takes charge of launching an application instance, replaying the users command inputs received from the thin client, and capturing and encoding the applications user interface into a video, and streaming the video to the thin client. With our prototype, a single cloud server can serve as the R-servers for multiple clients concurrently. In our measurements, we run instances of the heaven benchmark software on the cloud server for the v-client to capture, encode and stream to the client. Thus, the process at the cloud server includes all the components of rendering, obtaining display data, compression and transmission.

Figure 2 shows the frame rate (i.e., frames per second) of the Unigine Heaven Benchmark running at 1280x720 resolution and different rendering quality levels for each setting of k. From Figure 2 we can see that even for k = 8, the frame rate of the Unigine Heaven Benchmark is still above 30 frames per second (the threshold for an acceptable frame rate for most video games). Since the Unigine Heaven Benchmark is a widely used tool for determining the gaming performance of a computer under extremely stressful conditions, we believe that our measurements are representative and the settings of k are reasonable. Our settings of the server capacity are also consistent with other empirical studies [31].

[5]Unigine Heaven Benchmark – http://goo.gl/sWqG7k.

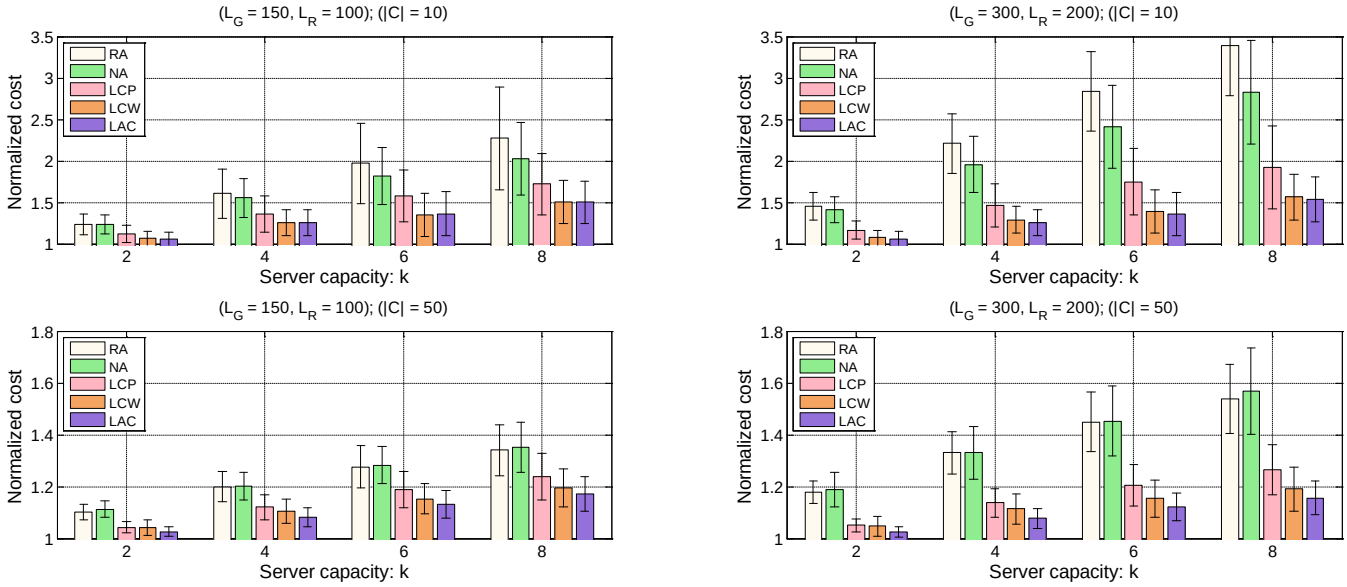


Fig. 3. Normalized costs of different algorithms for the basic problem.

Theoretical lower bound. To benchmark the performance of our algorithms, we calculate a theoretical lower bound on the total server rental and bandwidth cost for an MCG session as follows:

$$\sum_{d \in D} \Big(s(d) \cdot \big(\sum_{c \in C} X_c^d \big)/k + b(d) \cdot \sum_{c \in C} \big(X_c^d \cdot v(c) \big) \Big) \qquad (5)$$

where each X_c^d is obtained via the LCP algorithm (Section V.A) that assigns each client to the datacenter with the minimum combined price. The above calculation assumes that the number of servers to be opened at datacenter d is $\big(\sum_{c \in C} X_c^d\big)/k$ which can be a fraction just enough to serve all of its clients. Thus, this lower bound is a super-optimum and may not be achievable by any real server allocation when $k \geq 2$. To quantify the relative performance, we normalize the server allocation cost produced by each algorithm with respect to the above lower bound.

Baseline algorithms. We also evaluate the following naive algorithms which are neither aware of the server/bandwidth price nor aware of the projected capacity wastage when assigning clients to datacenters:

- ***Random Assignment (RA)***: Every client is assigned to a random datacenter from the set of its eligible datacenters. This algorithm serves as a baseline.
- ***Nearest Assignment (NA)***: Every client is assigned to the nearest datacenter in terms of network latency among all of its eligible datacenters. If a client has more than one eligible datacenters with the same lowest network latency to it, ties are broken by assigning the client to the one with the minimum server price.

B. Results and Analysis

Figure 3 shows the normalized cost of each algorithm for the basic problem (we only show the results for $|C|$ = 10 and $|C|$ = 50 due to the space limitation). In all these experiments, the G-server location for each session is randomly picked from the 13 datacenters. In general, the LCP, LCW, and LAC algorithms consistently produce significantly lower normalized costs than other algorithms, with the LCW and LAC algorithms being closest to the super-optimum lower bound. This demonstrates the importance of being aware of the server/bandwidth price and/or the projected capacity wastage in assigning clients to datacenters. The superior performance of the LCW algorithm over the LCP algorithm reflects that it is rational to trade the chance of assigning each individual client to its eligible datacenter with the minimum server or combined price for the chance of reducing the capacity wastage. In most situations, the LAC algorithm shows noticeable advantages over the LCW algorithm and performs the best among all the algorithms tested, which stresses the importance of considering all the cost-effectiveness factors in an integrated manner.

Figure 3 also shows that, for the same session size $|C|$, as the latency thresholds (L_G, L_R) get larger (i.e., from faster-paced games to slower-paced games), the RA and NA algorithms deteriorate significantly, while the LCP, LCW, and LAC algorithms remain stable. For instance, the normalized cost produced by the RA algorithm is 2.3 for (L_G = 150, L_R = 100) with $|C|$ = 10 and k = 8, and it increases to 3.4 (nearly a 50% increase) for (L_G = 300, L_R = 200) with the same $|C|$ and k. This can be explained using Figure 4 which shows the cumulative distribution of the number of eligible datacenters per client for all the sessions. As seen from Figure 4, the clients tend to have more eligible datacenters for larger latency thresholds (L_G, L_R). For instance, for (L_G = 300, L_R = 200), over 40% of the clients have at least 5 eligible datacenters, while for (L_G = 150, L_R = 100), over 80% of the clients have only 1 or 2 eligible datacenters. In the former case, clients are more likely to be scattered over different datacenters if we apply the RA algorithm. This is also true for the NA algorithm, since the nearest datacenters for clients may be more dispersed. Hence, these two algorithms generate more

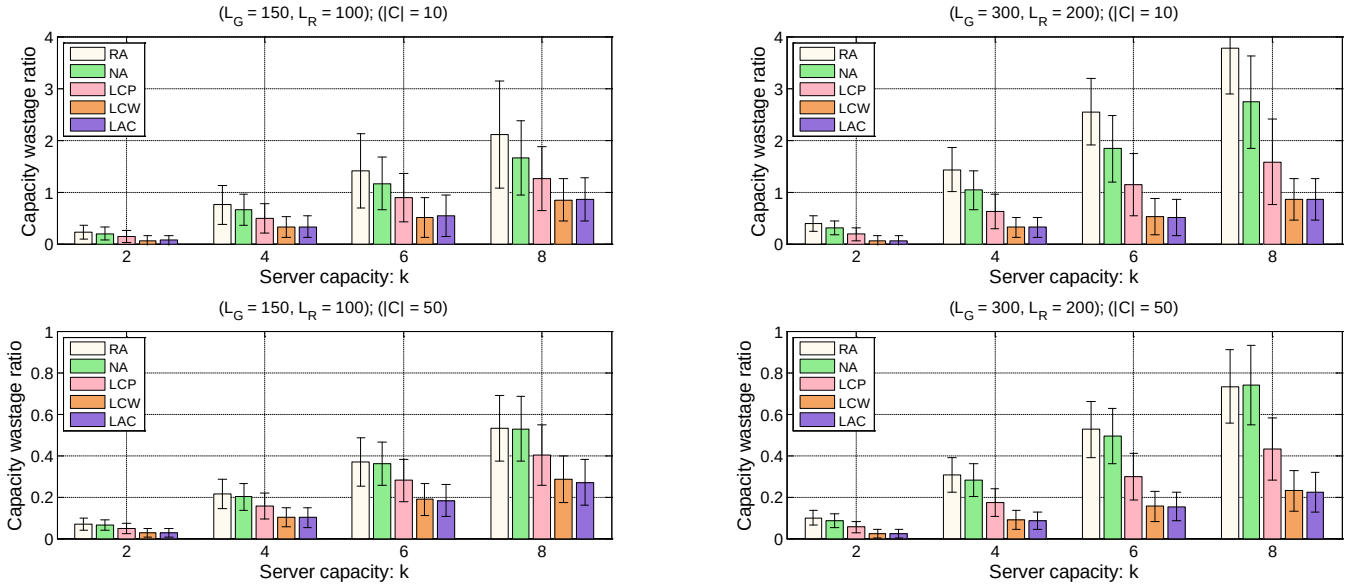


Fig. 5. Capacity wastage ratios of different algorithms for the basic problem.

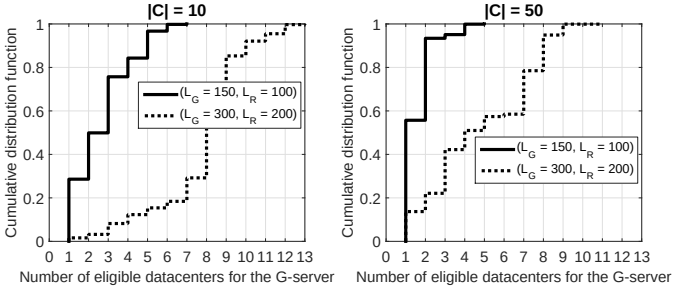


Fig. 7. CDF of the number of eligible datacenters (i.e., $|D_G|$) for placing the G-server across all sessions for the general problem.

capacity wastage for larger (L_G, L_R) and produce higher total costs eventually. This is confirmed by the capacity wastage ratios shown in Figure 5, where the capacity wastage ratio is defined as the excessive capacity (i.e., the total allocated server capacity minus the actually requested capacity) normalized by the actually requested capacity (i.e., the session size $|C|$).

From Figure 3, we can further see that, as the server capacity k gets larger (from 2 to 8), every algorithm produces increasingly a higher normalized cost. This is mainly due to the increasing capacity wastage generated by every algorithm as k gets larger, which can be again observed from Figure 5. Apparently it would be more difficult to fill up a larger server with clients than fulling up a smaller one. On the other hand, the theoretical lower bound assumes that a server can be "partially" rented to perfectly fit to the number of assigned clients with no capacity wastage at all. Thus, the normalized cost increases with the server capacity. Moreover, we can see from Figure 3 that, as the session size $|C|$ gets larger (from 10 to 50), the normalized cost produced by every algorithm decreases. This is mainly due to the decreasing capacity wastage generated by every algorithm as $|C|$ gets larger as can be observed from Figure 5.

Figure 6 shows the normalized cost of each algorithm for the general problem. These results have similar performance trends to the results for the basic problem, with the LCW and LAC algorithms significantly outperforming other algorithms. The main difference is that the normalized costs generated by all the algorithms for the general problem are slightly lower than those for the basic problem, especially for larger latency thresholds. This can be explained by comparing the results of capacity wastage ratios in the general and basic problems. Figure 8 shows the capacity wastage ratios for the general problem. Having the chance to choose the G-server location from a list of eligible datacenters implicitly makes all the algorithms assign clients to datacenters with relatively low capacity wastages as they evaluate different options of the G-server location. This is even more likely to happen for larger latency thresholds since there are more eligible datacenters for placing the G-server as shown in Figure 7.

In short, the LAC algorithm which considers all the cost-effectiveness factors (i.e., the server price, the bandwidth price, and the capacity wastage) in an integrated manner in assigning clients to datacenters, produces the most cost-effective solutions for both the basic and general problems of MCG server allocation.

C. Dynamic Scenario

So far, our discussion has focused on the situation where we need to open a new set of servers to support a requested MCG session. In practice, in a system operated by a cloud gaming service provider, sessions may arrive and depart dynamically, which may create the situation where a new session request arrives prior to the ending of the previous session. In this case, it may be beneficial to consider utilizing the remaining capacities (if any) of the servers allocated for the previous session for servicing the new session. Our problem definition and the proposed algorithms can be directly used to handle such dynamic scenarios.

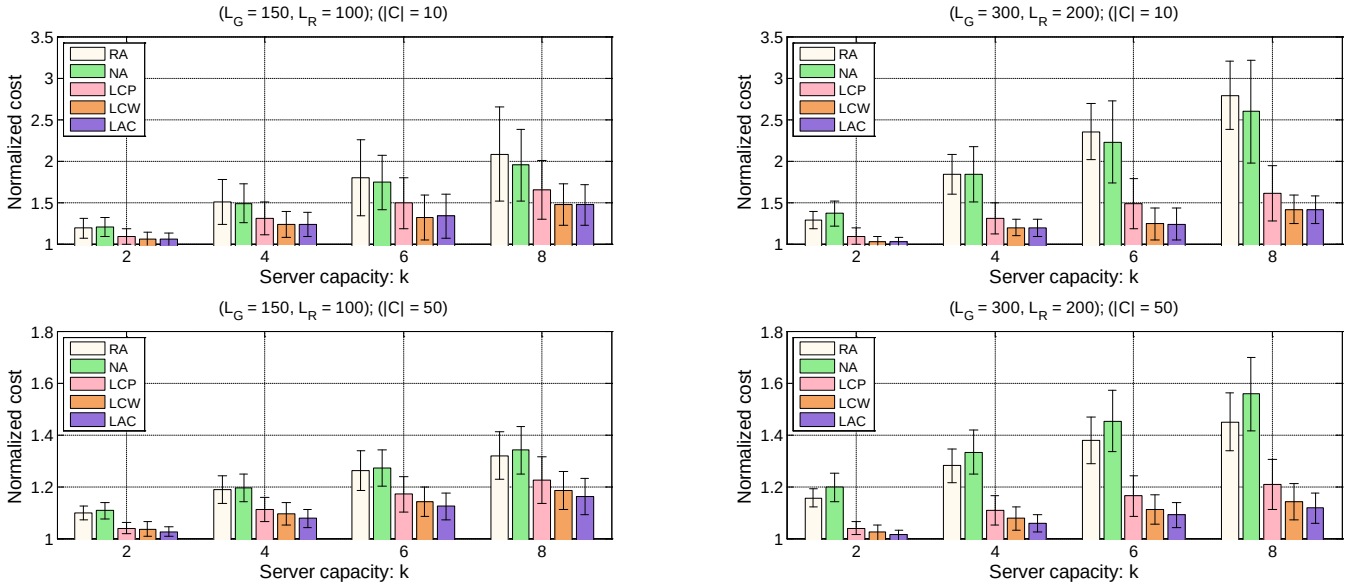


Fig. 6. Normalized costs of different algorithms for the general problem.

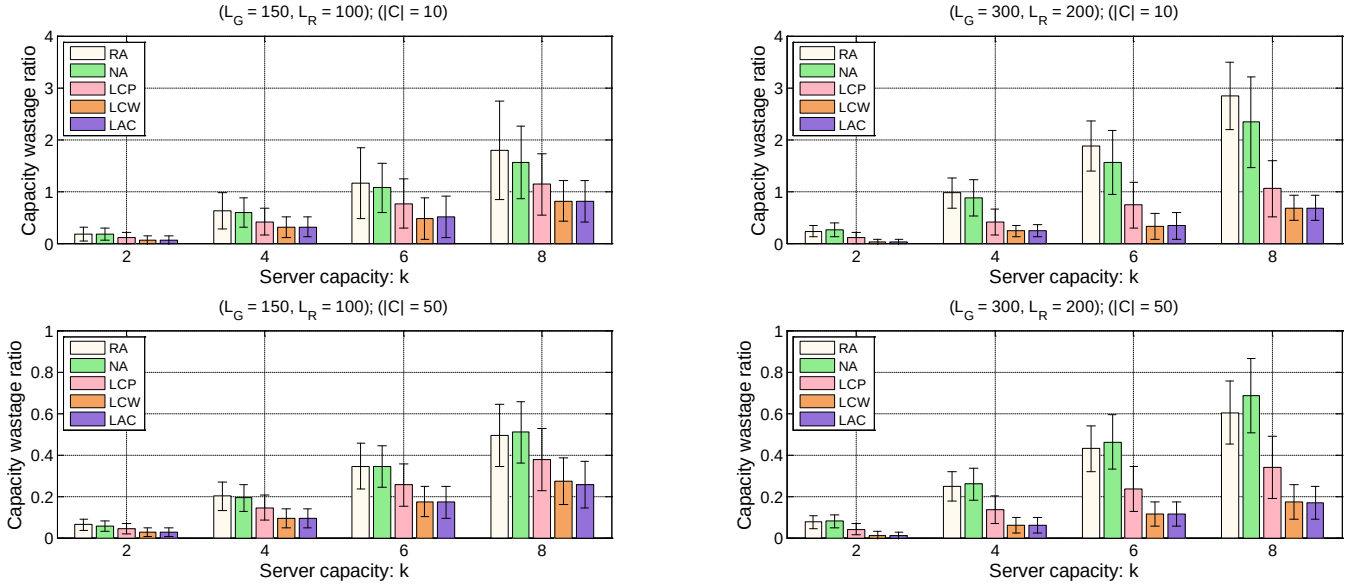


Fig. 8. Capacity wastage ratios of different algorithms for the general problem.

Specifically, when a request for running a new MCG session arrives, the algorithms can be executed to compute the client-to-datacenter assignments. Once the client-to-datacenter assignment is determined, we then need to assign the clients to the servers in the datacenters. If a new client is assigned to an open but not saturated server, no additional server cost is introduced. Only the data transfer induced by the new client needs to be paid. Thus, to save cost, it is preferable to use existing open servers as much as possible. New servers are opened to serve the clients only when all existing servers have filled up to their capacities. Assigning clients to existing servers is like a bin packing process where the servers and the clients correspond to the bins and items respectively. The total computation resource demand of all the new clients assigned to a server must be within the server's left-over capacity. To reduce the number of servers to which the clients of a new session are distributed, we can assign clients to servers in each datacenter in a best fit manner. Best Fit is a classical bin packing algorithm that assigns each new item to the bin with the smallest residual capacity that can accommodate it [50]. In our context of server allocation, we can examine existing open servers in the descending order of their left-over capacities. If the total computation resource demand of all the unassigned clients is higher than the left-over capacity of the first existing server (with the highest left-over capacity), we fill up the first server and then examine the next existing server. If the total resource demand of all the unassigned clients is lower than the first existing server, we assign all these clients to the existing

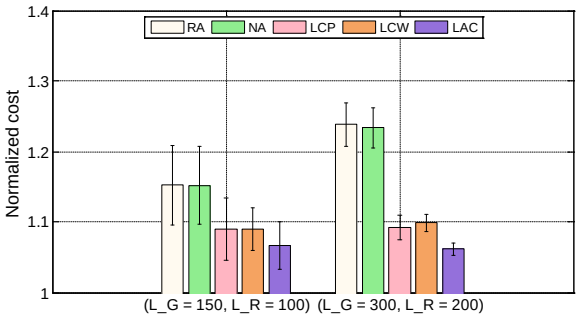


Fig. 9. Normalized costs for the basic problem under dynamic scenario.

server with the best-matching left-over capacity.

To evaluate our algorithms under the dynamic scenario, we simulate a sequence of 1000 randomly generated session arrival and departure events. The first 20 events are all session arrivals to warm up the system (around 600 clients will be in the system after the warm up). Each subsequent event is generated as a session arrival or a session departure with equal probabilities. For each session arrival, the session size is randomly chosen from the set {10, 20, 30, 40, 50} and the clients forming the session are randomly selected as described in Section VI-A. To simulate different computation resource demands for different games, for each new session, we set the resource demand of a client to a fraction of the server capacity, where the fraction is randomly picked from the set {1/2, 1/4, 1/6, 1/8} following the discussion in Section VI-A. For each session departure, the session to depart is randomly chosen from the active sessions. After each event, we compute the server allocation cost of all active sessions normalized by the lower bound for these active sessions as discussed in Section VI-A. Figure 9 shows the average and standard deviation of the normalized costs produced by each algorithm for the basic problem. The relative performance of the algorithms remains similar under the dynamic scenario. Our proposed algorithms significantly outperform the baseline RA and NA algorithms. The performance of the algorithms for the general problem shows similar trends and is not shown here due to the space limitation.

VII. Discussions

We have considered only the network latency as the quality-of-service metric for cloud gaming. Besides the network latency, other network metrics like bandwidth are also important to the gaming experience. In our model, additional network metrics can be addressed by adding new constraints in determining the eligible datacenters for each client. For example, a requirement can be added to stipulate that the available bandwidth between the client and the datacenter must be more than the estimated video streaming bitrate.[6] Our proposed server allocation algorithms are orthogonal to the decisions of eligible datacenters and can be applied for any sets of eligible datacenters.

Our proposed solutions can be used by the players who initiate the session or a cloud gaming service provider to optimize the operational cost for hosting an MCG session. In the former scenario, a group of friends that wish to play an online game but do not have powerful machines available may decide to rent cloud servers for an evening to host their game session. They may use our algorithms to determine where and how many cloud servers to rent with the minimum expense. In the latter scenario, a cloud gaming service provider that offer services to customers by acquiring resources from public clouds can also use our algorithms to allocate and open servers for each MCG session requested by customers.

Our proposed solutions can also be extended to optimize the energy consumption of cloud servers for MCG. Specifically, we can substitute the server and bandwidth prices in our model for their energy consumptions. The power usage of a server typically includes a flat component representing the power consumed by an idle server and a variable component that is proportional to the server load [52]. While the aggregate amount of the latter is mostly fixed for serving a given number of clients, the aggregate amount of the former largely depends on the number of servers used. Thus, server sharing among clients can potentially enhance the energy efficiency at the server side.

[6]There are existing tools for available bandwidth estimation [51].

VIII. Conclusion

In this paper, we have investigated the server allocation problem for MCG with the objective of reducing the total server and bandwidth cost to support an MCG session. We have discussed two versions of the problem and proposed several efficient algorithms to address both of them. Theoretical analysis is conducted to show that our proposed LAC algorithm can produce provably good server allocations. Extensive trace-driven experiments show that simply assigning clients to some random, the nearest, or the cheapest eligible datacenters can make the total cost far worse than optimum. It is important to also consider the server capacity wastage in order to achieve the most cost-effective MCG server allocation.

Acknowledgment

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IDM Futures Funding Initiative, by Singapore Ministry of Education Academic Research Fund Tier 2 under Grant MOE2013-T2-2-067, by NSF of China under Grants 61602266, 61373018, and by NSF of Tianjin under Grant 16JCYBJC41900. Yusen Li is co-corresponding author.

References

[1] Y. Wen, X. Zhu, J. J. Rodrigues, and C. W. Chen, "Cloud mobile media: Reflections and outlook," *IEEE Transactions on Multimedia*, vol. 16, no. 4, pp. 885–902, 2014.

[2] S. Choy, B. Wong, G. Simon, and C. Rosenberg, "The brewing storm in cloud gaming: A measurement study on cloud to end-user latency," in *Proc. ACM NetGames*, 2012, pp. 2:1–2:6.

[3] S. Wang and S. Dey, "Adaptive mobile cloud computing to enable rich mobile multimedia applications," *IEEE Transactions on Multimedia*, vol. 15, no. 4, pp. 870–883, 2013.

[4] 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 Transactions on Multimedia*, vol. 16, no. 2, pp. 480–495, 2014.

[5] C.-Y. Huang, Y.-L. Huang, Y.-H. Chi, K.-T. Chen, and C.-H. Hsu, "To cloud or not to cloud: Measuring the performance of mobile gaming," in *Proceedings of the 2nd Workshop on Mobile Gaming*. ACM, 2015, pp. 19–24.

[6] C.-Y. Huang, K.-T. Chen, D.-Y. Chen, H.-J. Hsu, and C.-H. Hsu, "GamingAnywhere: the first open source cloud gaming system," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 10, no. 1s, pp. 10:1–10:25, 2014.

[7] L. Lin, X. Liao, G. Tan, H. Jin, X. Yang, W. Zhang, and B. Li, "Liverender: A cloud gaming system based on compressed graphics streaming," in *Proc. ACM MM*, 2014, pp. 347–356.

[8] Y. Li, Y. Deng, R. Seet, X. Tang, and W. Cai, "MASTER: Multi-platform Application Streaming Toolkits for Elastic Resources," in *Proc. ACM MM*, 2015, pp. 805–806.

[9] R. Shea, D. Fu, and J. Liu, "Rhizome: utilizing the public cloud to provide 3d gaming infrastructure," in *Proc. ACM MMSys*, 2015, pp. 97–100.

[10] W. Cai, R. Shea, C.-Y. Huang, K.-T. Chen, J. Liu, V. C. Leung, and C.-H. Hsu, "The future of cloud gaming [point of view]," *Proceedings of the IEEE*, vol. 104, no. 4, pp. 687–691, 2016.

[11] Y. Deng, Y. Li, X. Tang, and W. Cai, "Server allocation for multiplayer cloud gaming," in *Proc. ACM MM*, 2016, pp. 918–927.

[12] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. Andrew, "Greening geographical load balancing," in *Proc. ACM SIGMETRICS*, 2011, pp. 233–244.

[13] L. Rao, X. Liu, L. Xie, and W. Liu, "Minimizing electricity cost: optimization of distributed internet data centers in a multi-electricity-market environment," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[14] Y. Rochman, H. Levy, and E. Brosh, "Efficient resource placement in cloud computing and network applications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 2, pp. 49–51, 2014.

[15] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic service placement in geographically distributed clouds," *IEEE Journal on Selected Areas in Communications,*, vol. 31, no. 12, pp. 762–772, 2013.

[16] S. Gargolinski, C. St Pierre, and M. Claypool, "Game server selection for multiple players," in *Proc. ACM NetGames*, 2005, pp. 1–6.

[17] L. Zhang and X. Tang, "Optimizing client assignment for enhancing interactivity in distributed interactive applications," *IEEE/ACM Transactions on Networking*, vol. 20, no. 6, pp. 1707–1720, 2012.

[18] ——, "The client assignment problem for continuous distributed interactive applications: analysis, algorithms, and evaluation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 785–795, 2014.

[19] H. Zheng and X. Tang, "Analysis of server provisioning for distributed interactive applications," *IEEE Transactions on Computers*, vol. 64, no. 10, pp. 2752–2766, Oct 2015.

[20] ——, "The server provisioning problem for continuous distributed interactive applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 1, pp. 271–285, Jan 2016.

[21] Y.-R. Chen, S. Radhakrishnan, S. K. Dhall, and S. Karabuk, "Server selection with delay constraints for online games," in *Proc. IEEE GLOBECOM Workshops*, 2010, pp. 882–887.

[22] ——, "On the game server network selection with delay and delay variation constraints," in *Proc. IEEE COMSNETS*, 2011, pp. 1–10.

[23] K.-W. Lee, B.-J. Ko, and S. Calo, "Adaptive server selection for large scale interactive online games," *Computer Networks*, vol. 49, no. 1, pp. 84–102, 2005.

[24] D.-N.-B. Ta, T. Nguyen, S. Zhou, X. Tang, W. Cai, and R. Ayani, "Interactivity-constrained server provisioning in large-scale distributed virtual environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 2, pp. 304–312, 2012.

[25] S. Yaw, E. Howard, B. Mumey, and M. P. Wittie, "Cooperative group provisioning with latency guarantees in multi-cloud deployments," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 3, pp. 4–11, Jul 2015.

[26] H.-J. Hong, D.-Y. Chen, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu, "Placing virtual machines to optimize cloud gaming experience," *IEEE Transactions on Cloud Computing*, vol. 3, no. 1, pp. 42–53, 2015.

[27] D. Wu, Z. Xue, and J. He, "iCloudaccess: Cost-effective streaming of video games from the cloud with low latency," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 8, pp. 1405–1416, 2014.

[28] H. Wang, R. Shea, X. Ma, F. Wang, and J. Liu, "On design and performance of cloud-based distributed interactive applications," in *Proc. IEEE ICNP*, 2014, pp. 37–46.

[29] H. Tian, D. Wu, J. He, Y. Xu, and M. Chen, "On achieving cost-effective adaptive cloud gaming in geo-distributed data centers," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 12, pp. 2064–2077, Dec 2015.

[30] M. Basiri and A. Rasoolzadegan, "Delay-aware resource provisioning for cost-efficient cloud gaming," *IEEE Transactions on Circuits and Systems for Video Technology*, 2016.

[31] Y. Zhang, P. Qu, J. Cihang, and W. Zheng, "A cloud gaming system based on user-level virtualization and its resource scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1239–1252, 2016.

[32] W. Cai, R. Shea, C.-Y. Huang, K.-T. Chen, and C.-H. Hsu, "A survey on cloud gaming: Future of computer games," *IEEE Access*, vol. 104, to appear.

[33] J. Manweiler, S. Agarwal, M. Zhang, R. Roy Choudhury, and P. Bahl, "Switchboard: a matchmaking system for multiplayer mobile games," in *Proc. ACM MobiSys*, 2011, pp. 71–84.

[34] K.-T. Chen, P. Huang, and C.-L. Lei, "How sensitive are online gamers to network quality?" *Communications of the ACM*, vol. 49, no. 11, pp. 34–38, 2006.

[35] 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. ACM MM*, 2011, pp. 1269–1272.

[36] L. Pantel and L. C. Wolf, "On the suitability of dead reckoning schemes for games," in *Proc. ACM NetGames*, 2002, pp. 79–84.

[37] J. Brun, F. Safaei, and P. Boustead, "Managing latency and fairness in networked games," *Communications of the ACM*, vol. 49, no. 11, pp. 46–51, 2006.

[38] K. Lee, D. Chu, E. Cuervo, Y. Degtyarev, S. Grizan, J. Kopf, A. Wolman, and J. Flinn, "Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming," in *Proc. ACM MobiSys*, 2015, pp. 151–165.

[39] M. Claypool and K. Claypool, "Latency and player actions in online games," *Communications of the ACM*, vol. 49, no. 11, pp. 40–45, 2006.

[40] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "Gaming in the clouds: QoE and the users' perspective," *Mathematical and Computer Modelling*, vol. 57, no. 11, pp. 2883–2894, 2013.

[41] A. Azari and G. Miao, "Lifetime-aware scheduling and power control for m2m communications in lte networks," in *Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st*. IEEE, 2015, pp. 1–5.

[42] A. L. Jia, S. Shen, R. V. D. Bovenkamp, A. Iosup, F. Kuipers, and D. H. J. Epema, "Socializing by gaming: Revealing social relationships in multiplayer online games," *ACM Transactions on Knowledge Discovery from Data*, vol. 10, no. 2, pp. 11:1–11:29, Oct. 2015.

[43] Multiplay. http://multiplay.com/.

[44] Game Servers. https://www.gameservers.com/.

[45] AWS gaming. https://aws.amazon.com/gaming/.

[46] Star Wars: Battlefront has 40-player cap. http://goo.gl/ThIL4W.

[47] Counter-Strike servers. http://goo.gl/Cecrxk.

[48] Z. Wu and H. V. Madhyastha, "Understanding the latency benefits of multi-cloud webservice deployments," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 2, pp. 13–20, Apr 2013.

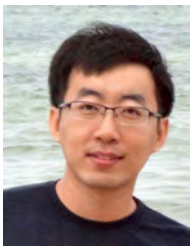
[49] J. Garcia-Dorado and S. Rao, "Cost-aware multi data-center bulk transfers in the cloud from a customer-side perspective," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, 2015.

[50] Y. Li, X. Tang, and W. Cai, "On dynamic bin packing for resource allocation in the cloud," in *Proc. ACM SPAA*, 2014, pp. 2–11.

[51] J. Strauss, D. Katabi, and F. Kaashoek, "A measurement study of available bandwidth estimation tools," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*. ACM, 2003, pp. 39–44.

[52] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, 2007.

Yunhua Deng is currently a Senior Engineer with Huawei Technologies, Shenzhen, China. He was a Research Fellow with the School of Computer Science and Engineering at Nanyang Technological University. He received his PhD degree in computer science from City University of Hong Kong in 2014. His research interests include distributed and cloud systems support for interactive multimedia applications such as online gaming and unified communications.



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

Ronald Seet is currently a project officer in Nanyang Technological University. He graduated from the same university with a Bachelor degree in Computer Science in 2015. He worked on the Cloud Gaming project for his Final Year Project and has continued until now.

Xueyan Tang received the BEng degree in computer science and engineering from Shanghai Jiao Tong University in 1998, and the PhD degree in computer science from the Hong Kong University of Science and Technology in 2003. He is currently an associate professor in the School of Computer Science and Engineering at Nanyang Technological University, Singapore. He has served as an associate editor of the IEEE Transactions on Parallel and Distributed Systems. His research interests include distributed systems, mobile and pervasive computing, and wireless sensor networks. He is a senior member of the IEEE.

Wentong Cai is a professor in the School of Computer Science and Engineering at Nanyang Technological University, Singapore. His expertise is mainly in the areas of modeling and simulation and parallel and distributed computing. He is an associate editor of the ACM Transactions on Modeling and Computer Simulation (TOMACS) and an editor of the Future Generation Computer Systems (FGCS). He has chaired a number of international conferences. Most recent ones include CloudCom 2014, SIMUTools 2013, ICPADS 2012, and MACOTS 2011.