Server Allocation for Multiplayer Cloud Gaming

Yunhua Deng‡ Yusen Li§ Xueyan Tang‡ Wentong Cai‡
‡School of Computer Science and Engineering, Nanyang Technological University, Singapore
§Department of Computer Science and Information Security, Nankai University, China
{yhdeng, asxytang, aswtcai}@ntu.edu.sg liyusen@nbjl.nankai.edu.cn

ABSTRACT

Advances in cloud computing and GPU virtualization are allowing the game industry to move into a cloud gaming era. While shifting standalone video games to the cloud gaming mode is straightforward, adapting multiplayer online games to the cloud gaming paradigm faces unique challenges. In this paper, we consider multiplayer cloud gaming (MCG), which is the natural integration of multiplayer online gaming and cloud gaming paradigms. 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. We propose several efficient heuristics to address the MCG server allocation problem which is hard to solve optimally. We conduct extensive experiments using real Internet latency and cloud pricing data 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.

Keywords

Cloud gaming; multiplayer online games; server allocation

1. INTRODUCTION

Cloud computing has been undergoing rapid expansion in recent years. Nowadays, many clouds such as Amazon EC2 [1] provide abundant, elastic and geo-distributed computation, storage and bandwidth resources for running various online services. Most recently, the maturity of cloud computing and recent advances in GPU virtualization have brought forth a new application – *cloud gaming* [13, 9, 19, 46]. 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 or keystrokes) to the server for game controls. With cloud gaming, users can play resource-intensive video games on less powerful devices anywhere and anytime over the Internet.

In this paper, we consider *multiplayer cloud gaming (MCG)*, a new form of multiplayer online gaming in the cloud gaming era.

MM '16, October 15-19, 2016, Amsterdam, Netherlands

DOI: http://dx.doi.org/10.1145/2964284.2964301

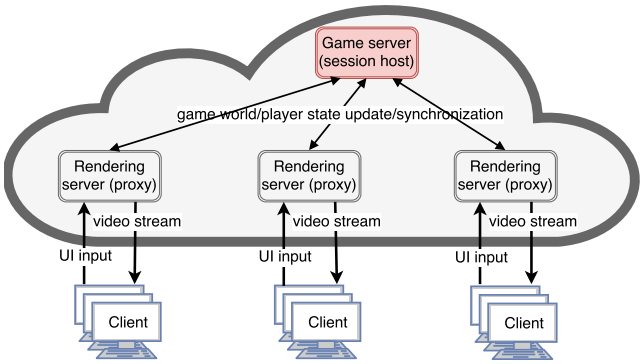

Figure 1: An MCG architecture.

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 clients' devices. Therefore, 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, 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 [24, 33, 29, 39, 10].

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 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.

The rest of this paper is organized as follows. Section 2 presents a summary of the related work. Section 3 discusses the requirements of MCG. Section 4 formulates the MCG server allocation problem. Section 5 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 6. Section 7 discusses some possible extensions and future work. Finally, Section 8 concludes the paper.

2. 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 [34, 37]. In more complicated scenarios, the demand and resource price fluctuations were also considered [38, 49]. However, these service placement models are quite different from our problem in which we aim 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 [22, 47, 48, 50, 51], or to place the minimum number of servers while satisfying some QoS requirements [17, 18, 28, 40]. 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 [45] 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.* [23] 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 [43] 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 [39, 29]. The problem of how to allocate cloud resources to minimize the interaction delay among clients was studied in [42]. However, the rental cost of servers was not considered either. Very recently, Tian *et al.* [41] 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.

3. 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 [22, 35]. 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:

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

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 [16]. The delay caused by video coding (i.e., server-side encoding and client-side decoding) adds another source of lag for

cloud gaming [15]. In addition, in traditional online gaming, lag can often be hidden to some extent using client-side compensation techniques such as dead-reckoning [36, 14], but in cloud gaming, lag is far more difficult to hide as the client does not maintain any game state locally [27]. 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 [20, 25] 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 [20, 25]. 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 [20, 25]). 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 [20, 25]. 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.

4. PROBLEM FORMULATION

Once a feasible MCG session is set up after matchmaking, we need to allocate servers to ensure that each client can be served by an R-server launched at one of the datacenters. 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. For each datacenter $d \in D$, let $s(d)$ denote the price for renting one server from d to act as an R-server. Multiplayer online game sessions usually span some tens of minutes [4, 26], which matches the billing interval of on-demand servers offered by most public cloud providers. For simplicity of billing and management, we assume that servers are not shared across game sessions. That is, for each session, a set of dedicated servers are launched to support it and these servers can be destroyed once the session ends. 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. Let $b(d)$ denote the price for the expected outbound data transfer[1] (i.e., video streaming) of a game session from each datacenter d. To facilitate exposition, we assume that each client incurs the same amount of outbound bandwidth from its assigned datacenter during the MCG session. We formulate the MCG server allocation problem with the goal of finding the number of servers to be opened 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. For each datacenter $d \in D$, we further define a variable Y_d to indicate the total number of clients assigned to d. For simplicity, we assume that the servers rented from different datacenters have the same capacity k ($k \in \mathbb{Z}^+$). That is, each server rented can handle the rendering and streaming workloads of up to k clients. Prevalent cloud billing schemes normally charge each server rented at a fixed rate regardless of its utilization [1]. Thus, the MCG server allocation problem can be written as follows:

$$\min \sum_{d \in D} \Big(s(d) \cdot \left\lceil \frac{Y_d}{k} \right\rceil + b(d) \cdot Y_d \Big), \tag{2}$$

subject to

$$Y_d = \sum_{c \in C} X_c^d, \quad \forall d \in D \tag{3}$$

$$\sum_{d \in E_c} X_c^d = 1, \quad \forall c \in C \tag{4}$$

$$X_c^d \in \{0, 1\}, \quad \forall c \in C,\ d \in E_c \tag{5}$$

The term $\lceil Y_d / k \rceil$ in objective (2) calculates the number of servers opened at datacenter d to run games for all the clients assigned to it. Constraint (3) derives the total number of clients assigned to each datacenter. Constraint (4) ensures that each client is assigned to exactly one of its eligible datacenters. It is easy to add extra constraints such as the capacity of each datacenter, and to consider heterogeneous clients with different requirements on aspects such as the screen resolution and video streaming bitrate. We leave these extensions to future work which will be discussed in Section 7.

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 $\big(s(d) + b(d)\big)$. 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.

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 [8, 5] 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 [2], 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

[1] In major public clouds, only the outbound data transfer is charged while the inbound data transfer is for free [1, 7].

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 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.

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 efficiently address MCG server allocation problems.

5. 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.

5.1 Price-based Assignment

We begin with three simple algorithms that determine the assigned datacenter d_c for each client $c \in C$ individually based on different priorities of price consideration.

Lowest-Server-Price (LSP) Assignment: Every client is assigned to the datacenter with the minimum *server price* among all of its eligible datacenters.

Lowest-Bandwidth-Price (LBP) Assignment: Every client is assigned to the datacenter with the minimum *bandwidth price* among all of its eligible datacenters.

Lowest-Combined-Price (LCP) Assignment: 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 $s(d)/k + b(d)$, which assumes that the servers can be "partially" rented to serve individual clients.

In the above three algorithms, if a client has more than one eligible datacenters offering the same lowest price, the client is assigned to the nearest one in terms of network latency. The time complexities of the three algorithms are all $O\big(|C||D|\big)$. Once the *client-to-datacenter assignment* is determined using any of the above algorithms, 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 $\lceil Y_d/k \rceil$ servers where Y_d is the 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 LSP, LBP and LCP algorithms to result in significant capacity wastage at some datacenters. For instance, if the clients' eligible datacenters with the lowest server prices are very diverse, most clients may be assigned to distinct datacenters when using the LSP 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 or server basis instead of on an individual client basis in order to promote server sharing among clients.

5.2 Wastage-aware Assignment

For convenience of presentation, we give the following definitions. Let $S_d \subseteq C$ 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 $p(d)$ be the projected capacity wastage if all the clients in S_d are assigned to d, which is given below based on the aforementioned client-to-server assignment procedure:

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

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 LSP, LBP and LCP algorithms consider one or both of the first two factors, the next heuristic called *Lowest-Capacity-Wastage* attempts to improve the cost-effectiveness of server allocation by explicitly avoiding the projected capacity wastage at each datacenter in the server allocation.

Lowest-Capacity-Wastage (LCW) Assignment:

1. For each datacenter $d \in D$, initialize S_d as the set of all the clients that can be covered by d.

2. Let $d^* = \arg\min_{d \in D, S_d \neq \emptyset} p(d)$ be the datacenter with the lowest projected capacity wastage whose S_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 clients in S_{d^*} to d^*. That is, for each client $c \in S_{d^*}$, set $d_c = d^*$.

4. For each datacenter $d \in D$, set $S_d \leftarrow S_d \setminus S_{d^*}$.

5. If all the clients in C have been assigned, the algorithm terminates. Otherwise, loop back to Step 2.

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.

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 S_d can only shrink over iterations, and therefore the time complexity for updating S_d over all iterations is bounded by $O(|C|)$. Hence, the overall time complexity of the LCW algorithm is $O\big(|C||D| + |D|^2\big)$. After determining d_c for each client $c \in C$, the aforementioned client-to-server assignment procedure is executed to finalize the server allocation for an MCG session.

5.3 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 LSP 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 LSP 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 LSP algorithm for the general problem is $O\big(|C||D|^2\big)$, so are those of the extended LBP and LCP algorithms. Likewise, the time complexity of the extended LCW algorithm for the general problem is $O\big(|C||D|^2 + |D|^3\big)$. 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 globally in total. Normally, the number of clients $|C|$ is also in the order of tens in a multiplayer online game session [11, 3]. 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]

6. EXPERIMENTAL EVALUATION

We evaluate the performance of our proposed algorithms by making use of real-world Internet latency data [44, 21] and cloud pricing models [1, 7] of two public clouds (Amazon EC2 and Microsoft Azure).

6.1 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.* [44] contains latency (RTT) measurements between PlanetLab nodes and datacenters from Amazon and Microsoft. The second dataset collected by Garcia-Dorado *et al.* [21] 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 dataset. 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 [1]), 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 between the GPU server's price and the non-GPU baseline server's price in the nearest EC2 datacenter to this Azure datacenter. Table 1 shows the rental price per server[4] used in our experimental evaluation. Also shown in Table 1 is the data transfer price per client, which is estimated by assuming that each client will consume 2GB outbound data transfer from its assigned datacenter in a game session.[5]

Table 1: Server rental and data tranfser prices (in dollars).

Datacenter	Baseline server price	GPU server price	Data transfer price
EC2-Virginia	0.532	2.600	0.180
EC2-Oregon	0.532	2.600	0.180
EC2-California	0.616	2.808	0.180
EC2-Ireland	0.585	2.808	0.180
EC2-Singapore	0.784	4.000	0.240
EC2-Tokyo	0.770	3.592	0.280
EC2-Sao Paulo	0.761	3.720	0.500
Azure-Hong Kong	0.902	4.604	0.276
Azure-Virginia	0.616	3.012	0.174
Azure-Ireland	0.584	2.804	0.174
Azure-Singapore	0.784	4.000	0.276
Azure-Amsterdam	0.672	3.224	0.174
Azure-California	0.616	2.808	0.174

Table 2: L_G and L_R settings according to [20, 25].

(L_G, L_R)	Game Genre	Game Pace
(150, 100)	first-person shooter, etc.	fast
(300, 200)	third-person role-playing, etc.	moderate

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 [20] and [25], respectively. Table 2 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 100 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 [4, 26]. 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 100 sessions resulting from each algorithm for performance comparison.

Server capacity. The server capacity k is dependent on the R-server's resource volume and the game's rendering workload. To

[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 video encoding.

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

[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.

[5]This is an estimated amount of data transfer for 1-hour HD (720p) live video streaming at the bitrate of 4000 Kbps [6].

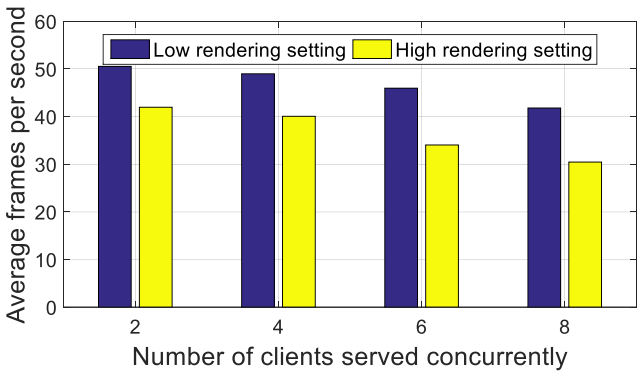

Figure 2: Heaven Benchmark's average frame rate against the number of clients served by an R-server concurrently.

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 the Unigine Heaven Benchmark [12] and a cloud gaming system prototype developed by ourselves [29]. With this prototype, a single cloud server can serve as the R-servers for multiple clients concurrently. 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 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.

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:

$$\sum_{d \in D} \left(\frac{s(d)}{k} + b(d) \right) \cdot Y_d \tag{6}$$

where Y_d (the number of clients assigned to each datacenter d) is obtained via the LCP algorithm 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 Y_d/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.

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

- ***Random Assignment***: Every client is assigned to a random datacenter from the set of its eligible datacenters. This algorithm serves as a baseline.

- ***Nearest Assignment***: 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, the client is assigned to the one with the minimum server price.

6.2 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 limit). In all these experiments, the G-server location for each session is randomly picked from the 13 datacenters. In general, the LSP, LCP, and LCW algorithms consistently produce significantly lower normalized costs than other algorithms, with the LCW algorithm being closest to the super-optimum lower bound. This reveals the importance of being aware of the server/bandwidth price or the projected capacity wastage in assigning clients to datacenters. The poorer performance of the LBP algorithm compared with the LSP and LCP algorithms indicates that the bandwidth cost is not as important as the server cost or the projected capacity wastage in optimizing the total cost. The superior of the LCW algorithm over the LSP and LCP algorithms 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.

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 Random, Nearest, and LBP algorithms deteriorate significantly, while the LSP, LCP, and LCW algorithms remain stable. For instance, the normalized cost produced by the Random algorithm is 2.6 for (L_G = 150, L_R = 100) with $|C|$ = 10 and k = 8, and it increases to 3.7 (nearly a 40% increase) for (L_G = 300, L_R = 200) with the same $|C|$ and k. This can be explained using Figure 5 which shows the cumulative distribution of the number of eligible datacenters per client of all the sessions. As seen from Figure 5, 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 50% of the clients have at least 6 eligible datacenters which are almost half of all 13 datacenters. In this case, clients are more likely to be scattered over different datacenters if we apply the Random algorithm. This is also true for the Nearest algorithm, since the nearest datacenters for clients may be more dispersed, so is for the LBP algorithm which may act like the Nearest algorithm as the bandwidth prices are likely identical across a large portion of datacenters (e.g., Azure has the same bandwidth price for all datacenters located at US and EU as shown in Table 1). Hence, these three algorithms generate more capacity wastage for larger (L_G, L_R) (as we can see from Figure 4) and produce higher total costs eventually. This is confirmed by the capacity wastage ratios shown in Figure 4, where the capacity wastage ratio is defined as the excessive capacity (i.e., the total allocated server capacity minus the requested capacity) normalized by the requested capacity (i.e., the session size $|C|$).

From Figure 3, we can also see that, as the server capacity k gets larger (from 2 to 8), all algorithms produce increasingly higher normalized costs. This is mainly due to the increasing capacity wastage generated by every algorithm as k gets larger, which can be again observed from Figure 4. 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 (see Figure 4).

In short, the LCW algorithm which considers the capacity wastage rather than the server or bandwidth price as the dominating factor in

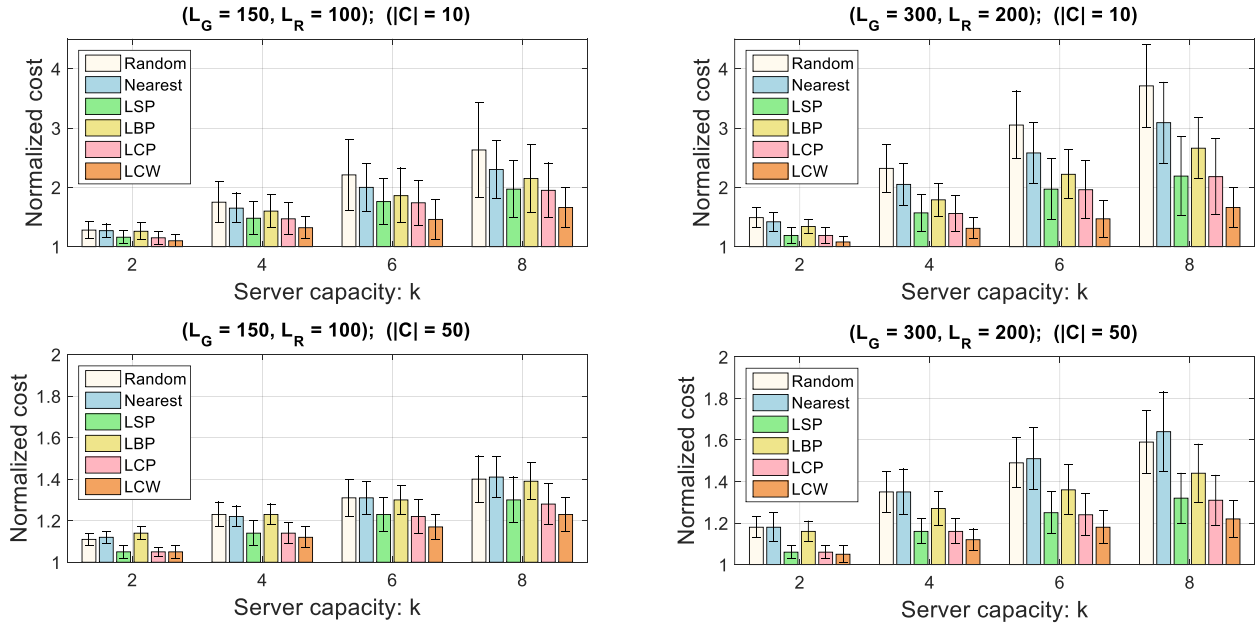

Figure 3: Normalized costs of different algorithms for the basic problem.

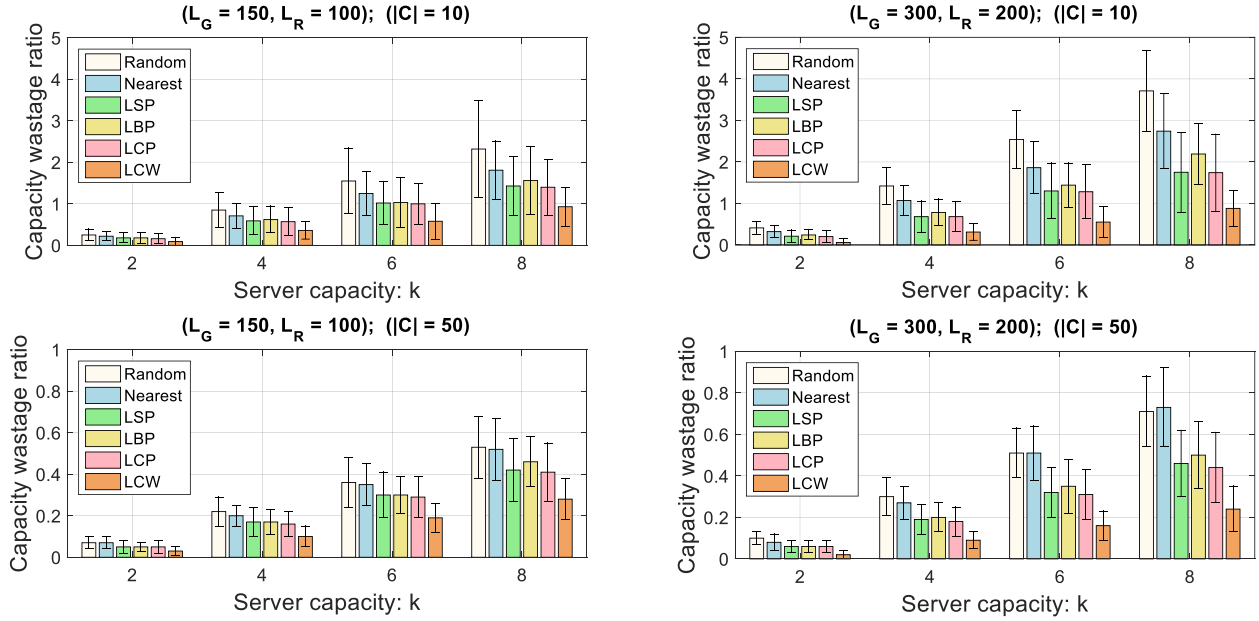

Figure 4: Capacity wastage ratios of different algorithms for the basic problem.

assigning clients to datacenters, is the most cost-effective solution for the basic problem of MCG server allocation.

The results for the general problem as shown in Figure 6 have similar performance trends to the results for the basic problem, with the LSP, LCP, and LCW algorithms significantly outperforming the Random, Nearest, and LBP algorithms. The main difference is that the advantage of the LCW algorithm over the LSP and LCP algorithms becomes less substantial. This can be explained by comparing the results of capacity wastage ratios in Figure 7 and Figure 4. From the comparison we can see that the capacity wastages generated by the LSP and LCP algorithms are much closer to that generated by the LCW algorithm for the general problem. For instance, the capacity wastage ratios of the LSP and LCP algorithms are nearly 2x of that generated by the LCW algorithm when (L_G = 300, L_R = 200), $|C|$ = 10 and k = 8 for the basic problem. These capacity wastage ratios are just about 1.4x of that generated by the LCW algorithm under the same experimental settings for the general problem. This is mainly because having the chance to choose the G-server location from a list of eligible datacenters indirectly makes the LSP and LCP algorithms assign clients to datacenters with relatively low capacity wastages as they compare the costs of different G-server locations. This is even more likely to happen for

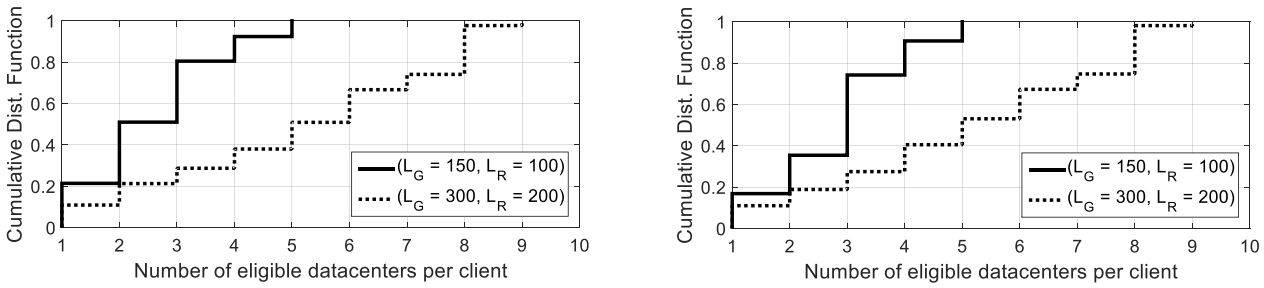

Figure 5: CDF of the number of eligible datacenters per client in sessions of $|C| = 10$ (left) and $|C| = 50$ (right).

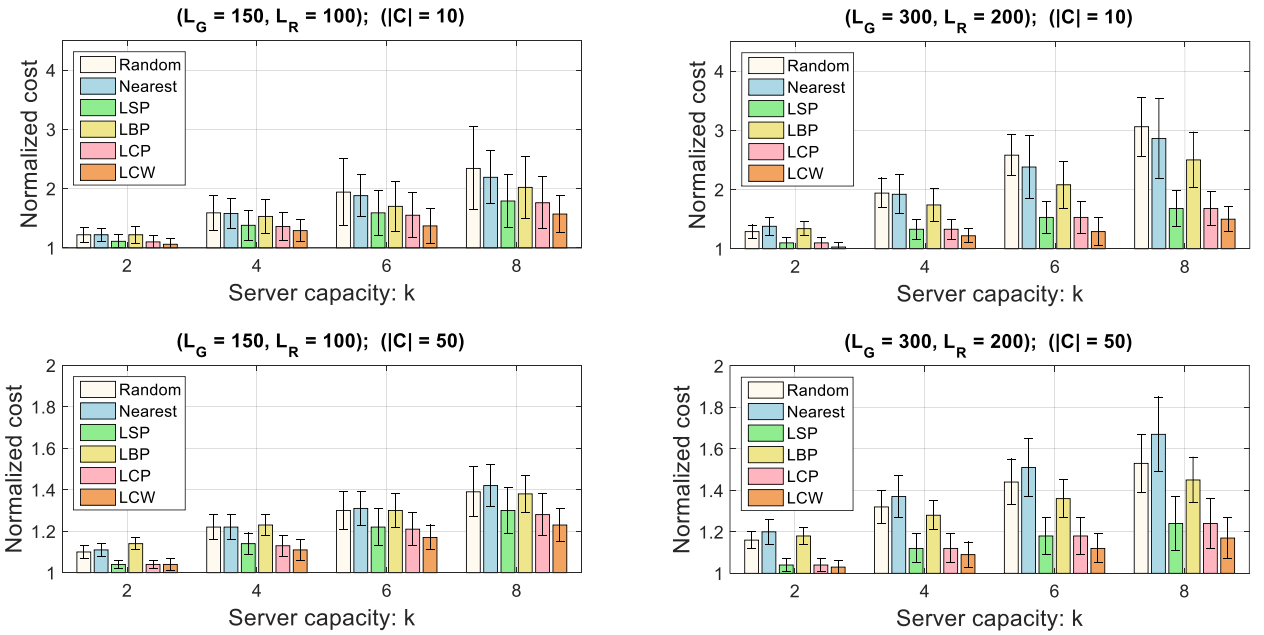

Figure 6: Normalized costs of different algorithms for the general problem.

larger latency thresholds since there are more eligible datacenters for placing the G-server as shown in Figure 8.

7. DISCUSSIONS

In our current model, we assume that servers are not shared across sessions for simplicity of billing and management. However, if an MCG session runs for a duration that is significantly shorter than the minimum billing interval of the servers offered by a public cloud provider, some revenue loss may occur in the MCG service provider. Hence, it is beneficial to keep its servers alive when the session finishes prior to the end of the charged period so that they can be reused to support a new session. Another possibility is to share servers across concurrently ongoing sessions. If the sessions with very different ending times are assigned to share the same servers, the servers may be left running at low utilizations after the sessions with early ending times finish [31]. Thus, the server cost might not be guaranteed to be reduced compared to allocating each session to dedicated servers in which all players depart together. The server allocation and assignment of players to servers here can be mapped to a dynamic bin packing problem [30, 32] which is a hard combinatorial optimization problem and hence requires further investigation.

In addition, our model can be extended to consider more general situations where clients belonging to the same session may entail different rendering or streaming qualities, due to heterogeneous thin-client devices or personal preferences. This can be achieved through modifying the cost calculation in the objective function (2). Specifically, we can consider the server capacity k and data transfer price $b(d)$ as variables of each individual client rather than constants. Our algorithms can easily be applied to this scenario.

8. 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. 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 consider the server capacity wastage in order to achieve cost-effective MCG server allocation.

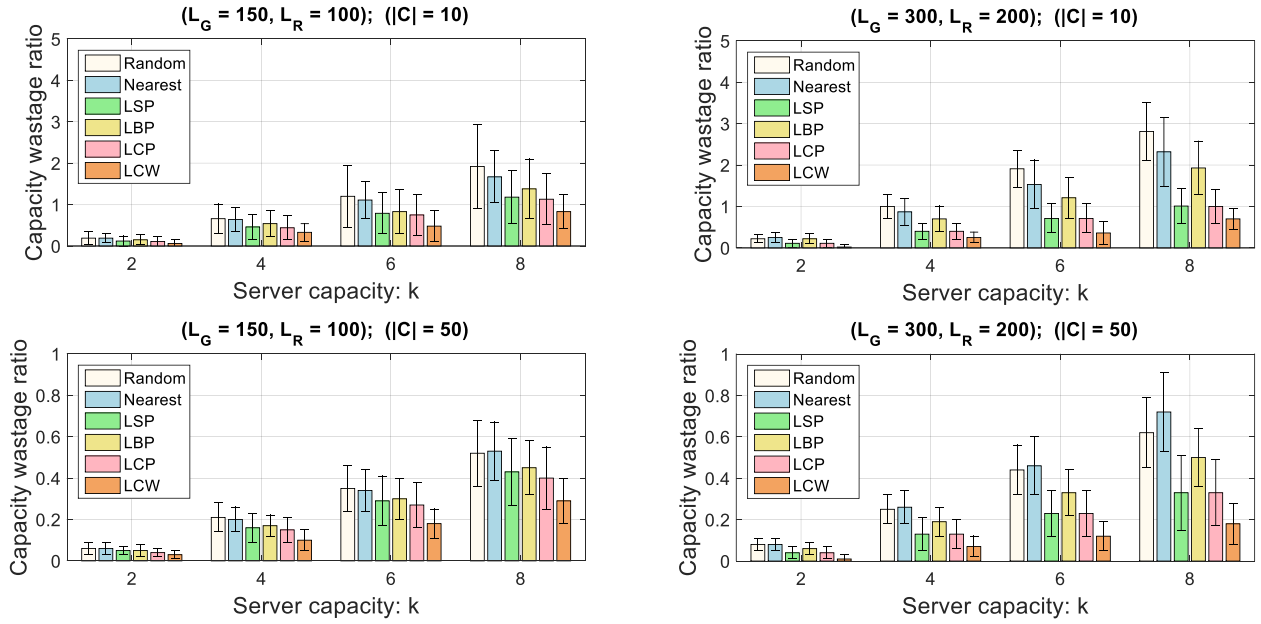

Figure 7: Capacity wastage ratios of different algorithms for the general problem.

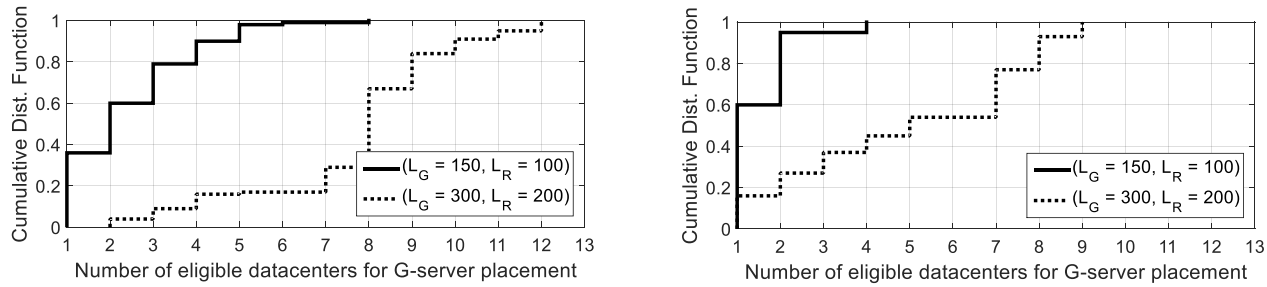

Figure 8: CDF of the number of eligible datacenters for G-server placement in sessions of $|C|$ = 10 (left) and $|C|$ = 50 (right).

9. ACKNOWLEDGMENTS

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IDM Futures Funding Initiative, and by Singapore Ministry of Education Academic Research Fund Tier 2 under Grant MOE2013-T2-2-067. We would like to thank Ronald Seet for his help on the Unigine Heaven Benchmark tests. Dr. Yusen Li is one of the corresponding authors.

10. REFERENCES

[1] Amazon EC2. https://aws.amazon.com/ec2/.
[2] AWS gaming. https://aws.amazon.com/gaming/.
[3] Counter-Strike servers. http://goo.gl/Cecrxk.
[4] Dota2 session length. http://goo.gl/FjKvqf.
[5] Game Servers. https://www.gameservers.com/.
[6] Live streaming bitrates. https://goo.gl/pmK4ql.
[7] Microsoft Azure. https://azure.microsoft.com/.
[8] Multiplay. http://multiplay.com/.
[9] Nvidia Cloud Gaming. http://goo.gl/Edpwbd.
[10] Nvidia GeForce NOW. https://goo.gl/xZ6D9I.
[11] Star Wars: Battlefront has 40-player cap. http://goo.gl/ThIL4W.
[12] Unigine Heaven Benchmark. http://goo.gl/sWqG7k.
[13] Why cloud hosting is the future of online gaming. http://goo.gl/9YHfCK.
[14] J. Brun, F. Safaei, and P. Boustead. Managing latency and fairness in networked games. *Communications of the ACM*, 49(11):46–51, 2006.
[15] 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*, pages 1269–1272, 2011.
[16] K.-T. Chen, P. Huang, and C.-L. Lei. How sensitive are online gamers to network quality? *Communications of the ACM*, 49(11):34–38, 2006.
[17] Y.-R. Chen, S. Radhakrishnan, S. K. Dhall, and S. Karabuk. Server selection with delay constraints for online games. In *Proc. IEEE GLOBECOM Workshops*, pages 882–887, 2010.
[18] Y.-R. Chen, S. Radhakrishnan, S. K. Dhall, and S. Karabuk. On the game server network selection with delay and delay variation constraints. In *Proc. IEEE COMSNETS*, pages 1–10, 2011.
[19] 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*, pages 2:1–2:6, 2012.

[20] M. Claypool and K. Claypool. Latency and player actions in online games. *Communications of the ACM*, 49(11):40–45, 2006.

[21] 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*, PP(99), 2015.

[22] S. Gargolinski, C. St Pierre, and M. Claypool. Game server selection for multiple players. In *Proc. ACM NetGames*, pages 1–6, 2005.

[23] 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*, 3(1):42–53, 2015.

[24] 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*, 10(1s):10:1–10:25, 2014.

[25] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld. Gaming in the clouds: QoE and the users' perspective. *Mathematical and Computer Modelling*, 57(11):2883–2894, 2013.

[26] 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*, 10(2):11:1–11:29, Oct. 2015.

[27] 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*, pages 151–165, 2015.

[28] K.-W. Lee, B.-J. Ko, and S. Calo. Adaptive server selection for large scale interactive online games. *Computer Networks*, 49(1):84–102, 2005.

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

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

[31] Y. Li, X. Tang, and W. Cai. Play request dispatching for efficient virtual machine usage in cloud gaming. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(12):2052–2063, Dec 2015.

[32] Y. Li, X. Tang, and W. Cai. Dynamic bin packing for on-demand cloud resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, 27(1):157–170, 2016.

[33] 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*, pages 347–356, 2014.

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

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

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

[37] 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*, pages 1–9, 2010.

[38] Y. Rochman, H. Levy, and E. Brosh. Efficient resource placement in cloud computing and network applications. *ACM SIGMETRICS Performance Evaluation Review*, 42(2):49–51, 2014.

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

[40] 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*, 23(2):304–312, 2012.

[41] 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*, 25(12):2064–2077, Dec 2015.

[42] 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*, pages 37–46, 2014.

[43] 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*, 24(8):1405–1416, 2014.

[44] Z. Wu and H. V. Madhyastha. Understanding the latency benefits of multi-cloud webservice deployments. *ACM SIGCOMM Computer Communication Review*, 43(2):13–20, Apr 2013.

[45] 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*, 45(3):4–11, Jul 2015.

[46] M. Yu, C. Zhang, Z. Qi, J. Yao, Y. Wang, and H. Guan. Vgris: virtualized gpu resource isolation and scheduling in cloud gaming. In *Proc. ACM HPDC*, pages 203–214, 2013.

[47] L. Zhang and X. Tang. Optimizing client assignment for enhancing interactivity in distributed interactive applications. *IEEE/ACM Transactions on Networking*, 20(6):1707–1720, 2012.

[48] L. Zhang and X. Tang. The client assignment problem for continuous distributed interactive applications: analysis, algorithms, and evaluation. *IEEE Transactions on Parallel and Distributed Systems*, 25(3):785–795, 2014.

[49] 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,*, 31(12):762–772, 2013.

[50] H. Zheng and X. Tang. Analysis of server provisioning for distributed interactive applications. *IEEE Transactions on Computers*, 64(10):2752–2766, Oct 2015.

[51] H. Zheng and X. Tang. The server provisioning problem for continuous distributed interactive applications. *IEEE Transactions on Parallel and Distributed Systems*, 27(1):271–285, Jan 2016.