On Design and Performance of Cloud-Based Distributed Interactive Applications

Haiyang Wang*, Ryan Shea†, Xiaoqiang Ma†, Feng Wang§ and Jiangchuan Liu†
* University of Minnesota Duluth, † Simon Fraser University § University of Mississippi
* haiyang@d.umn.edu, † {rws1,xma10,jcliu}@cs.sfu.ca § fwang@cs.olemiss.edu

***Abstract*—Distributed interactive applications (DIAs) such as online gaming have attracted a vast number of users over the Internet. It is however known that the deployment of DIA systems comes with peculiar hardware/software requirements on the users' consoles. Recently, such industrial pioneers as Gaikai, Onlive and Ciinow have offered a new generation of *cloud-based distributed interactive applications* (CDIAs), which shift the necessary computing loads to cloud platforms and largely relieve the pressure on individual user consoles.**

In this paper, we take a first step towards understanding the CDIA framework and highlight its design challenges. Our measurement reveals the inside structure as well as the operations of real CDIA systems and identifies the critical role of the cloud proxies. While this design makes effective use of cloud resources to mitigate the clients' workloads, it can also significantly increase the interaction latency among clients if not carefully handled. Besides the extra network latency due to the involvement of cloud proxies, we find that the computation-intensive tasks (e.g., game rendering) and bandwidth-intensive tasks (e.g., streaming the game screen to the clients) together create a severe bottleneck in CDIA. Our experiment indicates that when the cloud proxies are virtual machines (VMs) in the cloud, the computation-intensive and bandwidth-intensive tasks will seriously interfere with each other if not handled carefully. We accordingly capture this feature in our model and present an interference-aware solution. This approach not only smartly allocates the workloads but also dynamically assigns the capacities across VMs.

I. Introduction

Distributed interactive applications (DIAs) have become increasingly popular in recent years. By providing diverse interactions among the users, such applications as massive multiplayer online gaming, live messaging, and shared white-board have attracted a vast number of users over the Internet. Taking online gaming as an example, it is reported in [1] that nowadays each US household on average owns at least one dedicated game console or PC for game playing, where 62% of them have played interactive games with others. The global markets of these DIA systems will also expand from 58.7 billion in 2011 to 83 billion in 2016, growing at a 7.2 percent compound annual rate [2]. Yet, to support superior interactions, the DIAs often have peculiar demands on the users' consoles. The specialized consoles with high-performance hardware unavoidably increase users' cost and greatly limit the penetration of DIAs to ubiquitous end users.

To realize true *play-as-you-go*, industrial pioneers like Gaikai [3], Onlive [4], Ciinow [5], etc have suggested a new generation of DIAs based on cloud computing platforms. This *cloud-based distributed interactive application* (CDIA) effectively shifts the hardware/software requirements as well as the necessary computing loads to *cloud proxies*, and thus has attracted an increasing amount of attention form both service providers and end users. In particular, Sony Computer Entertainment (SCE) just acquired Gaikai for 380 million USD on July 2, 2012, putting Gaikai as a key function in its next generation game console, PlayStation 4 [6]. Its competitor, Microsoft, also announced that the Gaikai-like CDIA functions will also play a major role in their new game console Xbox One [7]. Moreover, such industry leaders as AMD and Nvidia are also entering the market of CDIA services [8]. AMD's investment in CiiNow gives it a means of competing with rival Nvidia's GeForce Grid cloud gaming platform.

Today, CDIA remains in its infancy with plenty of unknown issues. In this paper, we take a first step towards understanding the CDIA framework and highlight its design challenges. Our measurement reveals the inside structure as well as the operations of real CDIA systems and identifies the critical role of the cloud proxies. While this design makes effective use of cloud resources to mitigate the clients' workloads, it also significantly increases the interaction latency among clients if not carefully handled. In both DIA and CDIA systems, the interaction latency is the most essential problem even when there are no limitations on the availability of server capacities [9] [10]. The increasing latency will unavoidably reduce the applicability of the CDIA systems.

In detail, our analysis reveals that the deployment of cloud proxies adds extra communication hops between clients. To make the matter worse, the processing latency at the cloud proxy is also surprisingly high. While the use of the high-performance cloud platforms is expected to be highly efficient, we find that the computation-intensive tasks (e.g., game rendering) and the bandwidth-intensive tasks (e.g., streaming the game screen to the clients) together create a severe bottleneck in CDIAs. Our experiment indicates that when the cloud proxies are virtual machines (VMs) in the cloud, the computation-intensive and bandwidth-intensive tasks will seriously interfere with each other if not handled carefully. An increase of traffic load will greatly slow down the CPU benchmark of cloud VMs. In the case of CDIA, when the cloud proxies are used to stream the game screen to the users, the computation-intensive operations, such as game processing and message forwarding, will also be invoked and prolong the interaction latency. The large number of CDIA users further

978-1-4799-6204-4/14 $31.00 © 2014 IEEE
DOI 10.1109/ICNP.2014.25

aggravates this issue with mutual-interference, leading to poor user experiences. For example, *Diablo 2* in USEast realm had over 5 million users across 20 servers. The maximum acceptable latencies of this Role Playing Games (RPG) are from 100 to 500 ms [11]. However, when we deploy these systems on CDIA, their latency can easily exceed 600 ms even when we assume that the network latency is as small as zero.

Such interference however does not exist in conventional physical machines or to a much lower degree. Therefore, the existing load assignment solutions in the DIA system have mainly focused on the optimization of stand-alone workloads, without considering their interference in the VM environment. To address this problem, we provide a model and consider an interference-aware solution that not only smartly allocates the workloads but also dynamically assigns the capacities across different VMs.

The rest of this paper is organized as follows: In Section II, we present the related work. Based on the measurement of Section III, we examine the latency issues in CDIA in Section IV. Section V reveals the task interference on the cloud proxies and Section VI provides a model to minimize the interaction latency. Our solution is then extensively evaluated in Section VII. Section VIII further discusses several practical issues and concludes the paper.

II. Related Work

The origins of Distributed Interactive Applications (DIAs) can be traced back to 1983 when a United States research program initiated the SIMNET project [12] to train soldiers in battlefield tactics. Since then, an increasing number of academic, military and commercial DIA systems have been developed and documented. Nowadays, despite the increase of processing powers at participating clients and the availability of greater communication bandwidth, minimizing the interaction latency remains one of the most fundamental challenges in the DIA framework. Many studies have shown that the latency is particularly problematic when the network delays are comparable to the interaction time or speed [13]. Such studies suggested that the interaction latency should be bounded for real-world DIAs [14]. For example, the typical latency values to maintain real-time interaction fall between 40 and 300 ms [15]. Gutwin [16] investigated the effects of latency on two types of user interactions: prediction of movement and moving a shared object. This study showed that both gaming performance and user strategy will be greatly affected by interaction latencies higher than the expected range.

To minimize the interaction latency in DIAs, Webb *et al.* [17] proposed a nearest server assignment to reduce the client-server latency. Ta *et al.* [18] proposed a two-phase solution for large-scale DIAs. The study by Cronin *et al.* [19] further discussed the server placement problem to enhance users' interactivity. Vik *et al.* [20] explored the spanning tree problems in DIAs for latency reduction. A recent study from Zhang *et al.* [10] revisited the problem and proposed a distributed-modify-assignment approach to adapt to the dynamics of client participation and network conditions.

For cloud computing, there have been a series of works measuring the performance of public or private cloud services from diverse aspects, including computation, storage, and networking services [21]. There are also many studies addressing application designs that leverage cloud resources [22]. For example, Wu et al. [23] explored the use of cloud for Video-on-Demand applications. Huang *et al.* [24] provided an open-source cloud gaming system *GamingAnywhere*. The authors also deployed their system on Android OS and performed extensive experiments to understand the video quality for both the mobile and desktop clients. However, the deployment of enterprise cloud-based distributed interactive applications, which have emerged very recently, remain a mystery to the general public. Our study takes an initial step towards understanding this new generation of DIAs, in terms of its design, performance, and potential challenges.

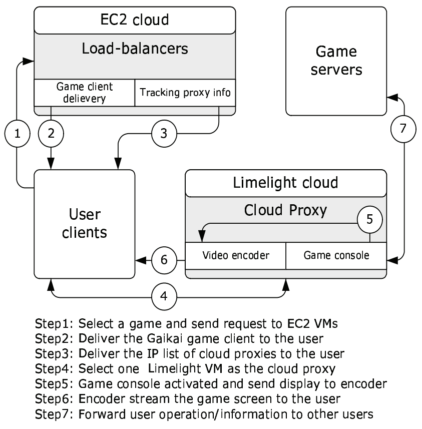
EC2 cloud
Load-balancers
Game client delievery
Tracking proxy info
Game servers
User clients
Limelight cloud
Cloud Proxy
Video encoder
Game console
1
2
3
4
5
6
7
Step1: Select a game and send request to EC2 VMs
Step2: Deliver the Gaikai game client to the user
Step3: Deliver the IP list of cloud proxies to the user
Step4: Select one Limelight VM as the cloud proxy
Step5: Game console activated and send display to encoder
Step6: Encoder stream the game screen to the user
Step7: Forward user operation/information to other users

Fig. 1: Basic Framework of Gaikai

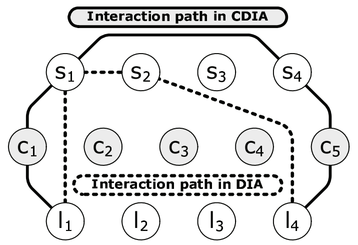

Fig. 2: Path of client interaction

III. Cloud-Based DIA: Background and Framework

From the perspective of industry, CDIAs can bring immense benefits by expanding the user base to the vast number of less-powerful devices that support thin clients only, particularly smartphones and tablets. As an example, the recommended system configuration for *Battlefield 3*, a highly popular first-person shooter game, is a quad-core CPU, 4 GB RAM, 20 GB storage space, and a graphics card with at least 1GB RAM (e.g., NVIDIA GEFORCE GTX 560 or ATI RADEON 6950), which alone costs more than $500 USD. The newest tablets (e.g., Apple's iPad with Retina display and Google's Nexus 10) cannot even meet the minimum system requirements that need a dual-core CPU over 2.4 GHz, 2 GB RAM, and a graphics card with 512 MB RAM, not to mention smartphones of which the hardware is limited by their smaller size and thermal control. Furthermore, mobile terminals have different hardware/software architecture from PCs, e.g., ARM rather than x86 for CPU, lower memory frequency and bandwidth, power limitations, and distinct operating systems. Therefore, the traditional console game model is not feasible for these devices, which in turn become targets for such CDIA systems as Gaikai and Onlive. Different from existing DIAs, CDIAs utilize the powerful and elastic service capacity offered by cloud computing to mitigate the hardware/software requirements on the user consoles. In particular, Gaikai and Onlive deploy the game clients/consoles on cloud platforms and only stream the game screen/interactions to end users.

To understand how CDIAs work in detail, we focus on Gaikai as a case study. Since 2011, it has emerged as one of the most popular cloud-based online gaming systems with over 100 million subscribers. It not only provides free PC game demos but also powers high quality gaming experiences onto smartphones, tablets and Internet TVs [25].

Our experiments have conducted traffic measurement and analysis from the edge of four networks, which are located in four different countries (United States, Canada, China and Japan) in two distinct continents. We used traffic analysis, shared library and system call interception techniques to analyze various aspects of the Gaikai protocol. We also applies a packet level analysis to understand the communications between our clients and the cloud proxies. In particular, we monitor Gaikai's online gaming service with the clients from different network locations and capture their traffic from/to the Gaikai servers. We first examine the details in the control messages, look into the packet payloads and identify the information, such as the domain names, of Gaikai's basic components. After that, we carefully analyze the data traffic to further understand the functions of these components. To avoid possible bias, we apply classic analysis tools (e.g., ISP lookup) that were widely adopted in the existing reverse engineering studies [26]. Note that some online games are designed to utilize one of the user clients as the server to enable interactions. We therefore take advantage of this feature and capture the server-side traffic to better understand the related protocols.

From analyzing the captured traffic, we illustrate Gaikai's basic framework/protocol in Figure 1. We can see that there are two major components on the server side of Gaikai (marked as grey boxes in the figure). The first part is Amazon EC2-based [27] load-balancers[1], and the second part is the Limelight-based game proxy servers [28]. Both Amazon and Limelight are leading cloud service providers with Xen virtualization [29]. Gaikai applies both platforms to accomplish different functionalities and utilizes their widely geo-distributed instances to push these functions closer to the users.

When a user selects a game on Gaikai (*Step1* in Figure 1), an EC2 virtual machine (VM) will first deliver the Gaikai game client to the user (in *Step2*). After that, it forwards the IP addresses of the available Limelight game proxies to the users (in *Step3*). The user will then use one game proxy to run the game (in *Step4*). To ensure smooth game playing, this selected game proxy uses a *packet train measurement* [30] to estimate the available bandwidth to the users. This is identified by our packet-level analysis, which shows that the game proxy sends back-to-back packets with empty payload to test the available bandwidth. Note that the game proxy starts the game only when the available bandwidth can well-support an FPS (frames per second) around 60 for video streaming. After that, the game proxy starts to run the game and the game screen will be streamed back to the user via UDP (in *Step5* and *Step6*). For multiplayer online games, these game proxies will also forward user operations to the game servers (mostly deployed by the game developers) and send the related information/reactions back to the users (in *Step7*). It is easy to see that such a CDIA system can remarkably relieve the hardware/software requirements on the user side, given that now the games are running on the cloud platforms. This change enables users to play hard-core games over much less powerful devices, e.g., over smartphone, tablets, or even digital TVs, as long as they are multimedia- and network-ready.

We have also measured other CDIA platforms, and have found that Gaikai's framework is representative, which is not surprising given it as a very natural extension to the conventional DIA with cloud assistance.

IV. Interaction Latency of CDIA: Issues and Challenges

The CDIA framework offers great opportunities for both users and service providers. Similar to the conventional DIA systems, its service performance is highly depending on users' *interaction latency* [10]. In particular, the CDIA systems, such as the online cloud gaming applications, need to collect users' actions, transmit them to the cloud proxy, process the action, render the results, encode/compress the resulting changes to the game-world, and stream the video (game scenes) back to the player. To ensure interactivity, all of these serial operations must happen in the order of milliseconds. The interaction latency is thus essential even when there are no limitations on the availability of server capacities [9].

Figure 2 illustrates the interaction pathes in both DIA and CDIA frameworks, where L is the set of clients, S is the set of servers, and C is the set of cloud-based proxies. To better compare the performance of DIA and CIDA design, we will first focus on the latency between clients and servers (e.g., between l_1 and s_1). It is easy to see that such a latency consist of two parts in the CDIA systems: First, the network latency between clients and servers; Second, the processing latency

[1]Based on our measurement, they also have other functions beside load-balancing. We call them *load-balancers* because Gaikai marks them with "LB" in their domain names.

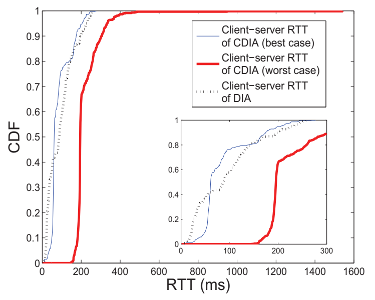

Fig. 3: Time cost between user and server (DIA v.s. CDIA)

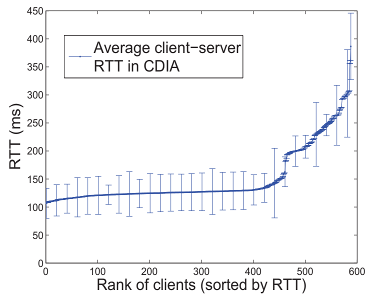

Fig. 4: Average user-server latency in CDIA

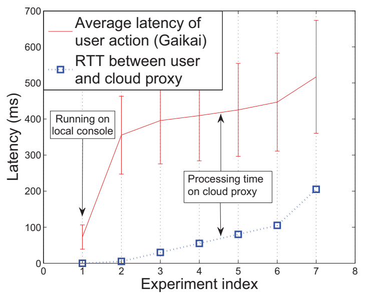

Fig. 5: Average latency of user's action

on cloud proxies. We will provide a step-by-step discussion to understand these two parts in the following subsections.

A. Analysis of Network Latency

To clarify the extra network latency in CDIA design, we carry out a real-word experiment from Planet-lab. We use a server in our campus to emulate the game server in CDIA[2]. We select 588 Planet-lab nodes (the maximum number of nodes that we can access) to run as CDIA clients and emulate the CDIA framework by using the server and these clients to connect Gaikai's cloud proxies. We have found 28 cloud proxies during the measurement of Gaikai[3], and therefore use the IP addresses of these proxies in this experiment. We first measure the RTTs between 588 Planet-lab clients and 28 Gaikai cloud proxies and then the RTTs between the server and these cloud proxies. The sum of these two latencies can be used to calculate the client-server RTTs in this CDIA system. To provide a fair comparison, we also measure the direct RTTs between the server and the Planet-lab clients as the baseline (the case of conventional DIA).

Figure 3 compares the client-server RTT in both DIA and CDIA. We can see that most (over 80%) users in DIA have quite low client-server latency (less than 60 ms), while the average latency is much worse if we put them into CDIA, as shown in Figure 4. The worst case in Figure 4 shows 90% users have an interaction latency over 200ms. This is hardly acceptable for smooth interaction because the processing latencies are not yet considered in this experiment. It is known that adding extra nodes in any overlay network is not necessarily leading to longer path length given that triangle inequality does not hold in the Internet [31]. Hence, there is indeed space to reduce the latency beyond naive proxy deployment[4].

[2]We have observed similar results over 50 servers that are located in different places.

[3]The total number of Gaikai's cloud proxy is unknown to the general public. These sampled cloud proxies are used to estimate the network latency in such a system.

[4]The 588 PlanetLab nodes are applied in both DIA and CDIA experiments to provide fair comparison. Since some real-world interactive applications, such as Diablo 2, may divide their users into realms, we also investigated the case using a subset of PlanetLab nodes from one realm (e.g., USEast). The results remain consistent to Figure 3.

Example Game Type	Perspective	Delay Threshold
First Person Shooter (FPS)	First Person	100 ms
Role Playing Game (RPG)	Third-Person	500 ms
Real Time Strategy (RTS)	Omnipresent	1000 ms

TABLE I: Delay Tolerance in Traditional Gaming

B. Analysis of Processing Latency

So far, we have only considered the network latency in CDIA framework. It is easy to see that the cloud proxies will also bring extra processing latency to the interaction. We now closely examine this latency and identify its impacts.

To focus exactly on the interaction between clients and cloud proxies, we select a single player game where a player (client) does not need to communicate with the game server and other players. The player simply sends the operations to the cloud proxy and the proxy then streams the responding game screen back to the player. Since the RTT between the player and the cloud proxy can be directly measured, we only need to obtain the *response time*[5], which, after subtracting the RTT, gives the processing latency at the cloud proxy. The detail of this experiment is as follows.

We first select an action button in the game *The Witcher 2: Assassins of Kings*; in particular, the "map" button. We click this button and start to record the game screen at 100 FPS (frames per second). This sampling rate already exceeds the normal game play which is around 60 to 70 FPS. We then check the video file frame-by-frame until we find the frame where the map is displayed. We run this experiment multiple times under different RTTs. These RTTs are controlled by the traffic shaping tool TC [32]. To better understand the processing overhead at the cloud proxy, we also record the response time on a local game console. We use the same game on both Gaikai and the local console to provide a fair comparison.

As we can see from Figure 5, the local console general needs 80 ms to open the map for the players with very small standard deviation. Note that the RTT is zero in this case because the game is locally rendered. When we run this game

[5]The response time is the latency that the player waits until the result of her/his operations is returned. For example, if the player clicks the button "option" at time t_i and the option menu displays at time t_j, the response time can be calculated as $t_j - t_i$.

remotely on Gaikai, the response time elevates to more than 300 ms. The overhead (in terms of the processing latency) on the Gaikai proxy is thus approximately 220 ms[6]. When we consider the interactive latency between different users, there will be, unfortunately, two proxies in their interactive path. This means the interactive latency can easily exceed 600 ms. It is worth noting that the studies on traditional gaming systems have found that different styles of games have different thresholds for maximum tolerable delay [11]. However, this high latency will not be acceptable for most online gaming applications as shown in Table I.

V. Interferences between Computation-intensive and Bandwidth-intensive Tasks

It is surprising to see that the cloud proxies can introduce such a high processing latency in CDIA. This is unlikely due only to video encoding because many CDIA service providers have claimed that their video encoding latency is indeed very small within 10 ms. We therefore further explore its underlying reasons in this subsection.

The cloud proxy on Gaikai is different from a local game console. It is a virtual machine (VM) running both computation-intensive tasks (for example, rendering the game) and bandwidth-intensive tasks (for example, streaming the game screen to the players) at the same time. Since these tasks cannot be decoupled into different VMs, it is necessary to see if they introduce extra overheads on the cloud proxy.

Since the Limelight platform can hardly be tested by individual users, we run the standard CPU benchmark on EC2 instances (their cloud platforms are quite similar in terms of Xen-based cloud virtualization). We adjust the traffic load on the instance and check the time cost of running the benchmark. To provide a fair comparison, we also do the experiment on local servers (non-virtualized servers) as a baseline. Figure 6 shows a comparison between a EC2 small instance and our local server. In this experiment, the EC2 small instance has 1.7 GB memory, 1 EC2 compute unit (1 virtual core with 1 EC2 compute unit), 160 GB instance storage with 32-bit platform. Our local server also has similar hardware configuration that is comparable to the EC2 small instance. From this figure, we can see that the traffic load on the non-virtualized server only slightly increases the running time of the CPU benchmark, e.g., 250 Mbps traffic load will only increase the running time by 20%. However, for the virtualized EC2 small instance, this traffic load will double the running time of the CPU benchmark with very small standard deviation (the detailed data can be found in Table II). We have also tested this on EC2 large instances with multiple cores. The large instance has 7.5 GB memory, 4 EC2 computation units (2 virtual cores and each with 2 EC2 computation units), 850 GB instance storage with 64-bit platform and very high I/O performance. Our local server, on the other hand, has weaker hardware configuration, particularly the CPU capacity. As shown in

[6]To avoid measurement bias, we also test actions that make different changes to the in game world, for example, small character movements. The results remain consistent with Figure 5.

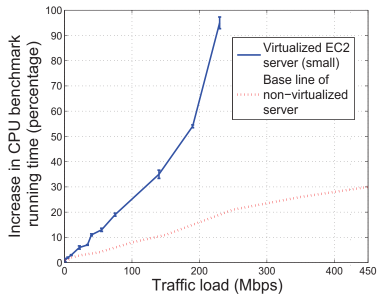

Fig. 6: Increasing of CPU benchmark running time (on EC2 small instance)

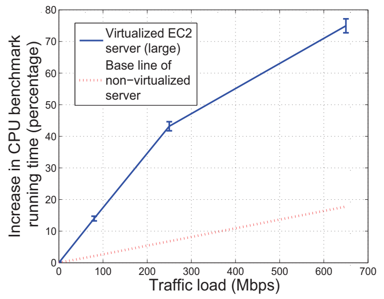

Fig. 7: Increasing of CPU benchmark running time (on EC2 large instance)

Figure 7, we can see that the traffic load on large instances still brings remarkable overheads to the system. Although the result is better than that of small instances, the traffic load will still remarkably slow down the running time of the CPU benchmark especially when comparing with the non-virtualized baseline.

It is easy to see that the CDIA cloud proxies are indeed in the same situation as these EC2 instances. The traffic load can significantly slow down the game running and unavoidably leads to a high processing latency. Yet, such a problem is rarely seen on the non-virtualized local game consoles or cloud proxies, or to a much lower degree.

TABLE II: CPU benchmark running time under different traffic loads on virtualized EC2 small instance

Load	**Run1**	**Run2**	**Run3**	**Run4**	**Avg**
	Running time of CPU benchmark(ms)				
1Mbps	37.17	36.71	36.75	36.64	36.82
5Mbps	36.92	37.06	37.07	37.07	37.03
10Mbps	37.41	37.51	37.36	37.42	37.42
22Mbps	38.21	38.49	38.75	38.30	38.44
34Mbps	39.03	39.13	39.06	39.13	39.09
40Mbps	40.33	40.59	40.39	40.20	40.38
55Mbps	41.28	41.22	41.06	41.67	41.31
75Mbps	43.20	43.21	43.14	43.69	43.31
140Mbps	49.00	48.84	50.06	48.13	49.01
190Mbps	56.28	56.26	55.60	55.96	56.02
230Mbps	71.50	70.67	69.06	73.01	71.06

VI. Latency Minimization in CDIA

Given the importance of latency for interaction, there have been significant studies on latency minimization for conventional DIAs, mostly focusing on latency directly between client pairs [19] [20] [10]. Unfortunately, the existence of cloud proxies prevents them from being used in the CDIAs, not to menton the task interference on the cloud proxies. In this section, we will revisit the latency modeling problem in these new contexts. We first consider a basic model to optimize the network latency. After that, this model will be further enhanced to capture the task interference on the cloud VMs.

A. Basic Model of CDIA Latency

To ensure responsive interactions, previous studies have suggested that reducing the average latency is not enough, because any fast users would suffer when they interact with long latency users [16] [33]. Our objective is thus to minimize the maximum latency between all client pairs that are bridged by cloud proxies. We focus on network latency here, and will address processing latency in the next two sections.

We use $S = \{s_1, s_2, ..., s_m\}$ to denote the set of m servers and $L = \{l_1, l_2, ..., l_n\}$ to denote the set of n user clients. Let $C = \{c_1, c_2, ..., c_o\}$ be the set of o cloud proxies. Each client in L will be assigned to a cloud proxy and a server in order to send operations and receive updates from other clients. An assignment A is a mapping from L to C and S, where for each client $l \in L$, we use $c_A(l) \in C$ to denote the assigned cloud proxy of client l and $s_A(l) \in S$ to denote the assigned server of client l.

For two clients l_i and l_j to interact, the communication should go through their assigned cloud proxies and servers in CDIA. Specifically, if l_i issues an operation to l_j, the following steps have to be taken so that l_j can see the effect of this operation: First, l_i sends the operation to its assigned cloud proxy $c_A(l_i)$. $c_A(l_i)$ will then forward this operation to the server $s_A(l_i)$ that is also assigned to l_i; After that, if l_j is assigned to a different server $s_A(l_j)$, server $s_A(l_i)$ should forward the operation to server $s_A(l_j)$; Then $s_A(l_j)$ executes the operation and delivers the resultant state update to l_j's cloud proxy $c_A(l_j)$; Finally, $c_A(l_j)$ will generate the game screen and stream the display to client l_j. Let $D(u, v)$ be the path latency between two nodes that are not directly connected and $d(u, v)$ be the link latency between two neighbor nodes. To be consistent with the existing DIA models [10], we assume that $D(u, v) = D(v, u)$ and $d(u, v) = d(v, u)$. The latency between client l_i to its server $s_A(l_i)$ can be calculated as:

$$D\Big(l_i, s_A(l_i)\Big) = d\Big(l_i, c_A(l_i)\Big) + d\Big(c_A(l_i), s_A(l_i)\Big) \tag{1}$$

We can therefore obtain the total interaction latency between l_i and l_j as follows:

$$\begin{aligned} D(l_i, l_j) = D\Big(l_i, s_A(l_i)\Big) + D\Big(l_j, s_A(l_j)\Big) \\ +d\Big(s_A(l_i), s_A(l_j)\Big) \cdot I_{\big[s_A(l_i) \neq s_A(l_j)\big]} \end{aligned} \tag{2}$$

where $d\Big(s_A(l_i), s_A(l_j)\Big)$ denotes the latency between server $s_A(l_i)$ and $s_A(l_j)$, and $I_{[\cdot]}$ indicates whether l_i and l_j are assigned to different servers (1: yes; 0: no). Given the interaction latency between l_i and l_j, our objective is to find an assignment A to minimize $\mathbb{U}(A)$, the maximum interaction latency among all client pairs:

$$minimize \qquad \mathbb{U}(A) = \max_{l_i, l_j \in L} \Big\{D(l_i, l_j)\Big\} \tag{3}$$

This min-max latency can be optimally found when we convert it into a longest path problem in directed acyclic graph (DAG). The details are presented in *Appendix A*.

B. Enhanced Model to Capture Task Interference

In this part, we will further extend our model to consider the impact of traffic load on different cloud proxies. It is worth noting that CDIA offers elastic service capacity at cloud proxies. The capacities of the cloud proxies can be dynamically adjusted to meet user demands. Therefore, we use set P to denote the capacities of cloud proxies where $P = \{p_1, p_2, ..., p_o\}$; $p_i \in P$ refers to the amount of resource that is assigned to cloud proxy c_i (bandwidth capacity in this case). Based on our measurement, we find that the NPV (Net Present Value) function [34] can be borrowed to capture the relationship between *virtualization latency* (processing latency that due to the traffic load on VMs) and traffic load[7], we therefore compute the virtualization latency of cloud c_i as:

$$r(p_i) = \frac{a}{b^{p_i - q_A(c_i)}} \tag{4}$$

where a indicates the latency when the cloud proxy is fully loaded (with no remaining bandwidth). Parameter b controls the skewness of the relationship between load and latency where $b \in (1, +\infty)$. Note that different VMs may have different a and b. For example, in Figure II, a is around 105 and b is around 1.04.

Given a load assignment A and a user l_i, we use $p_A(l_i)$ to denote the resource that has been assigned to cloud proxy $c_A(l_i)$. For a given set of servers, $S = \{s_1, s_2, ..., s_m\}$ and clients $l = \{l_1, l_2, ..., l_n\}$, the problem becomes how to use a set of cloud proxies $C = \{c_1, c_2, ..., c_o\}$ to connect these clients and servers, with load assignment A and resource assignment P, to minimize the maximum interaction latency between all client pairs:

$$\begin{aligned} minimize \qquad \mathbb{U}(A, P) = \underset{l_i, l_j \in L}{Max} \Big\{D(l_i, l_j) \\ +r(p_A(l_i)) + r(p_A(l_j))\Big\} \end{aligned} \tag{5}$$

[7]This function has been widely used to quantify the relationship between cash and price/cost, which resembles our case when we try to purchase more cloud resources to reduce the virtualization cost on the VMs.

$$s.t. \quad \forall i = 1, 2, ..., o, \qquad q_A(c_i) \le p_i \tag{6}$$

$$\sum_{i=1}^{o} p_i * Cost(c_i) \le K \tag{7}$$

where K refers to the total budget, which we assume can at least serve all the clients in the system.

It is easy to see that the virtualization latency makes the problem harder. If we assign client l_i to $c_A(l_i)$, it will not only assign traffic load to $c_A(l_i)$ but also affect the performance of other clients who have also been assigned to this cloud proxy. Assuming that the capacities of the cloud proxies are given, this client assignment problem can therefore be transformed into a $0-1$ Multiple Knapsack problem with a non-linear objective function, which is known to be NP-hard [35].

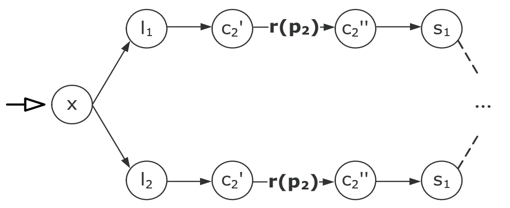

Fig. 8: Transform G into G_A^*

By exhaustively searching along all the possible combinations of A and P, the optimal solution can be achieved. However, the practical usefulness of this search is limited considering the real-time user demands in CDIA systems. We thus propose a bi-level heuristic, which divides the optimization problem into two stages: load assignment and resource assignment. In the load assignment stage, we assume that all the cloud proxies are fully loaded (the virtualization latency is therefore equal to a in Equation 4) and find the optimal load assignment A by Algorithm 1 (in *Appendix A*). After that, we construct a subgraph G_A based on the existing graph G and assignment A. As shown in Figure 8, we then split the node c_i to two virtual nodes (c_i' and c_i''), and use their link weight to refer the virtualization cost on c_i. We use G_A^* to denote the resulting graph. We further apply a greedy algorithm to find the resource assignment P in G_A^*. As shown in Algorithm 3, this greedy algorithm iteratively assign resource to the cloud proxies on the longest path. The algorithm stops when the remaining budget is not enough. In the next section, we will show that this bi-level heuristic achieves near-optimal performance in practical settings.

VII. Performance Evaluation

We now evaluate the performance of our solution via extensive trace-based simulations in MATLAB. The network latency (measured in Section IV) and the processing delay (measured in Section VI) will both serve as the inputs of our evaluation. We first examine the performance of our optimal client assignment when there is no task interference[8]. After

Algorithm 3 ResourceProvisioning()
```
1:  Get $G_A^*$ from $A$;
2:  $R \leftarrow K - \mathbb{C}$;
3:  while true,
4:    $path^* = LongestPath(G_A^*)$;
5:    Get $c_i, c_j$ from $path^*$;
6:    if $R \ge max(Cost(c_i), Cost(c_j))$,
7:      if $\frac{r(p_i)-r(p_i+1)}{Cost(c_i)} \ge \frac{r(p_j)-r(p_j+1)}{Cost(c_j)}$,
8:        $R \leftarrow R - Cost(c_i)$;
9:        $p_i \leftarrow p_i + 1$;
10:     else
11:       $R \leftarrow R - Cost(c_j)$;
12:       $p_j \leftarrow p_j + 1$;
13:   else if $R \ge min(Cost(c_i), Cost(c_j))$,
14:     if $Cost(c_i) \le Cost(c_j)$,
15:       $w \leftarrow i$;
16:     else
17:       $w \leftarrow j$;
18:       $R \leftarrow R - Cost(c_w)$;
19:       $p(c_w) \leftarrow p(c_w) + 1$;
20:     end if
21:   else
22:     break;
23:   end if
24: end while
```

Fig. 9: Algorithm to compute the resource provisioning.

that, we investigate the performance of the interference-aware client assignment algorithm in the virtualized environment.

We start with a CDIA system that consists of 20 clients, 5 cloud proxies and 5 servers. The renting cost of cloud proxies are referenced from the instance price list of Amazon's *On Demand instances* [27]. Figure 10 presents the performance of our optimal client assignment when there is no task interference (the processing latency is a default value of 80 ms at the cloud proxies). It is easy to see that the smart client assignment greatly reduces the interaction latency. Without optimization, the maximum client interaction latency can be as high as 700 ms. Our approach, on the other hand, can reduce the maximum latency to less than 500 ms. It is also worth noting that the renting price is linearly related to the client population. This indicates a good scalability of our approach.

It is not surprising to see that the optimal client assignment can achieve such a significant gain when there is no task interference. Figure 11 further explores the case when the optimal assignment can hardly be archived in the task interference environment. We can see that the optimization of task interference is very critical for CDIAs. The maximum interaction latency can be larger than 580 ms if we only focus on the optimization of network latency. Fortunately, our interference-aware algorithm can achieve a near-optimal (with the difference within 5 ms) latency that greatly reduces the interaction latency[9]. It is worth noting that the interaction

[8]This will be the case when the system is deployed on non-virtualized cloud platforms.

[9]The optimal base-line is obtained by brute-force searching.

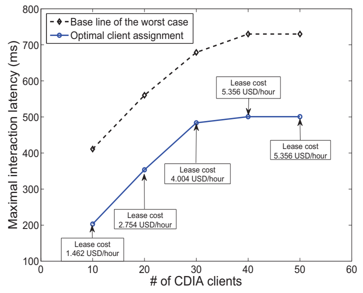

Fig. 10: Optimal client assignment only consider the networking latency

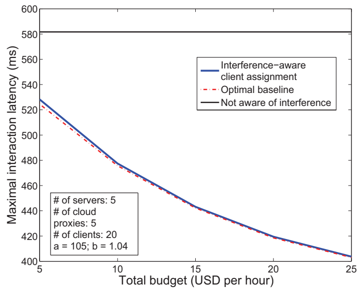

Fig. 11: Interference-aware client assignment

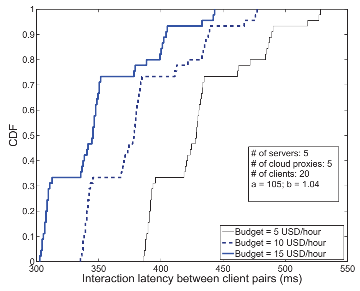

Fig. 12: Interaction latency across client pairs (different budget)

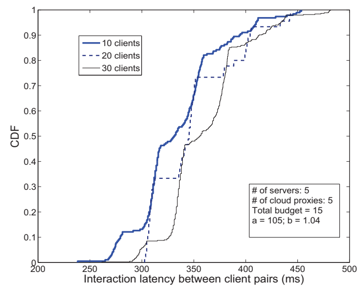

Fig. 13: Interaction latency across client pairs (different # of clients)

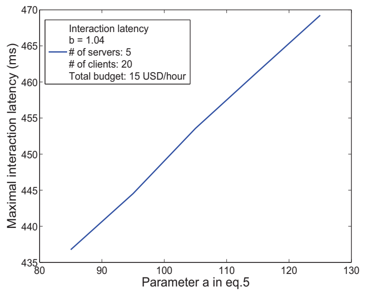

Fig. 14: Adjusting parameter a (VM's maximum processing latency)

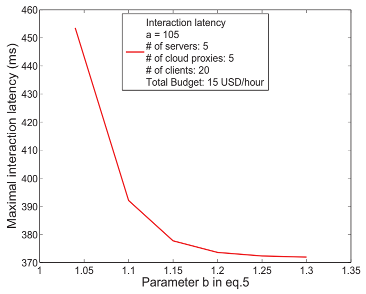

Fig. 15: Adjusting parameter b (skewness of the relationship)

latency can be further reduced and become closer to the optimal results when we have more budget to purchase more capacities at the cloud proxies.

Figure 12 takes a closer look at the interaction latency between individual clients. We can see that all clients can benefit from the total budget increase. To be more specific, when the budget is equal to 5 USD/hour, less than 30% clients can have an interaction latency less than 400 ms. If we increase the budget to 15 USD/hour, more than 95% clients can interact with each other with a latency below 400 ms. The difference between the fastest and the slowest clients are also quite small, around 150 ms. Figure 13 further shows the cases with different number of CDIA clients. We can see that for a given budget, our algorithm scales well with an increasing number of clients. Note that the total budget also bounds the total capacity of the cloud proxies. We thus cannot add more clients in Figure 13.

To understand the virtualization latency on different types of VMs, we investigate the case with different parameter inputs in Equation 5. Figure 14 presents the case when the maximum processing latency (parameter a for the cloud proxies) is changed from 85 ms to 125 ms. We can see that the interaction latency increases linearly with a. On the other hand, Figure 15 presents the case when the virtualization latency and traffic load have more skewed relationships[10]. Based on these two figures, we can find that a good VM should have a small a

[10]Note that different a, b pairs in these two figures can be used to present different cloud instances. For example, we use $a = 105$ and $b = 1.04$ to capture the features of EC2 large instances in our simulation.

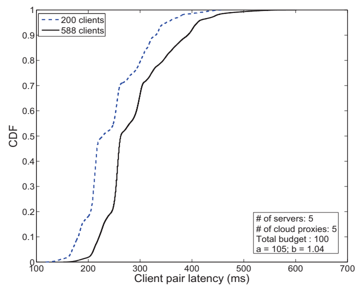

Fig. 16: Interaction latency across 200 and 588 clients

and a large b. The maximum processing latency should be small when the VM is fully loaded (small a). In other words, adding idle resources on the VM should be able to significantly reduce such a processing latency (large b).

Figure 16 further presents the CDF of the interaction latency across 200 random selected clients and all the 588 clients[11] in our measurement, respectively. It is easy to see that 80% clients can achieve the interaction latency within 300 ms. The interaction latencies between most (70%) client pairs are between 200 ms and 250 ms. It is also worth noting that the total budget in this case is relatively high with 100 USD per hour. This is because we are using the pricing list of Amazon's *On Demand instances*. Choosing other types of platforms/instances, such as the *Reserved instance* may further reduce this cost.

[11]All clients that we have used in our measurement in Section IV.

VIII. Conclusion and Further Discussion

In this paper, we examined the framework design and latency optimization in cloud-based distributed interactive applications through real system measurement and analysis. Our study identified the unique features as well as the fundamental design challenges in the CDIA. There are still many open issues that can be further explored in this new-born system.

First, to better mitigate the interference between the computation-intensive and bandwidth-intensive tasks, we are working on the analysis of TCP/UDP flows on different types of VMs. Our preliminary result shows that VMs' *hypervisors* (also known as *virtual machine managers* such as Xen, KVM and VMware) and VMs' total capacities play important roles for the interference.

Second, we are currently investigating the efficiency of directly migrating some DIA protocols/optimizations into the CDIA framework. This analysis can help us better enjoy the benefits of cloud computing while minimize the corresponding overheads. These investigations are not limited to improve the overall performance of CDIA framework, it can also help us better understand the development of many other cloud-based systems with similar design frameworks.

Acknowledgment

H. Wang's work was supported by Chancellors Small Grant and a Start-up Grant and from the University of Minnesota at Duluth. J. Liu's work was supported by a Canada NSERC Discovery Grant, an NSERC Strategic Project Grant, and a China NSFC Major Program of International Cooperation Grant (61120106008).

References

[1] Essential Facts about the Computer and Video Game Industry 2012. [Online]. Available: http://www.theesa.com/facts/pdfs/ESA_EF_2012.pdf

[2] Entertainment Business. [Online]. Available: http://www.entertainmentbusiness.nl/sites/default/files/documents/2012/Videogames.pdf

[3] Gaikai. [Online]. Available: http://www.gaikai.com//

[4] Onlove. [Online]. Available: http://www.onlive.com//

[5] Ciinow, http://www.ciinow.com/.

[6] Gaikai in PlayStation4. [Online]. Available: http://www.theverge.com/2013/2/20/4010420/sonys-playstation-4-will-use-gaikai-game-streaming-technology

[7] Cloud-based gaming planned for Microsoft Xbox 720. [Online]. Available: http://sonyps4playstation.com/cloud-based-gaming-planned-for-microsoft-xbox-720/

[8] AMD Invests Into Ciinow, http://www.ubergizmo.com/2012/09/amd-invests-into-ciinow/.

[9] F. Safaei, P. Boustead, C. Nguyen, J. Brun, and M. Dowlatshahi, "Latency-driven distribution: Infrastructure needs of participatory entertainment applications," *IEEE Commun. Mag, 43(5), 106-112, 2005.*

[10] 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.*

[11] M. Claypool and K. Claypool, "Latency and Player Actions in Online Games," *Communications of the ACM, 49(11), 40-45, 2006.*

[12] SIMNET. [Online]. Available: http://en.wikipedia.org/wiki/SIMNET/

[13] P. M. Sharkey, M. D. Ryan, and D. J. Roberts, "A Local perception filter for distributed Virtual Environments," *Virtual Reality Annual International Symposium, 242-249, 1998.*

[14] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg, "Locallag and Timewarp: Providing Consistency for Replicated Continuous Applications ," *IEEE Transactions on Multimedia, 6(1), 47-57, 2002.*

Algorithm 1 OptimalLoadAssignment()
```
1: while $IsConnected(L, G) == true$,
2:     $path^* = LongestPath(G)$;
3:     $Remove(path^*, G)$;
4: end while
5: $Recover(path^*, G)$;
6: Return any viable $A$ from $G$;
```

Fig. 18: Optimal Load Assignment Algorithm

[15] C. Diot and L. Gautier, "A Distributed Architecture for Multiplayer Interactive Applications on the Internet," *IEEE Network, 13(4), 6?5, 1999.*

[16] C. Gutwin, " The Effects of Network Delays on Group Work in Real-Time Groupware," in *Proc. Seventh European Conference on Computer-Supported Cooperative Work (ECSCW), 2011.*

[17] S.Webb, S. Soh, and W. Lau, "Enhanced Mirrored Servers for Network Games," in *Proc. ACM SIGCOMM NetGames, 2007.*

[18] D. Ta and S. Zhou, "A Two-phase Approach to Interactivity Enhancement for Large-scale Distributed Virtual Environments," *Computer Networks, 51(14), 4131?152, 2007.*

[19] E. Cronin, S. Jamin, C. Jin, A. Kurc, D. Raza, and Y. Shavitt, "Constrained Mirror Placement on the Internet," *IEEE Journal on Selected Areas in Communications (JSAC), 20(7), 1369?382, 2002.*

[20] P. H. K. Vik and C. Griwodz, "Multicast Tree Diameter for Dynamic Distributed Interactive Applications," in *Proc. IEEE International Conference on Computer Communications (INFOCOM), 2008.*

[21] S. Garfinkel, "An Evaluation of Amazon s Grid Computing Services : EC2 , S3 and SQS," *Harvard University Tech, Rep., 2008.*

[22] Y. Seung, T. Lam, L. E. Li, and T. Woo, "Seamless Scaling of Enterprise Applications into The Cloud," in *Proc. IEEE INFOCOM, 2011.*

[23] Y. Wu, C. Wu, B. Li, X. Qiu, and F. Lau, "CloudMedia: When Cloud On Demand Meets Video On Demand," in *Proc. IEEE ICDCS, 2011.*

[24] C. Huang, C. Hsu, Y. Chang, and K. Chen, "Gaminganywhere: an open cloud gaming system," *in Proceedings of ACM MMSys*, 2013.

[25] Gaikai Powered Cloud-based Gaming on Samsung Smart TVs. [Online]. Available: http://www.engadget.com/2012/06/05/gaikai-powered-cloud-gaming-coming-to-samsung-smart-tvs/

[26] S. A. Baset and H. G. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," in *Proc. IEEE International Conference on Computer Communications (INFOCOM), 2006.*

[27] Amazon EC2. [Online]. Available: http://aws.amazon.com/ec2/

[28] Limelight Networks. [Online]. Available: http://www.limelight.com/

[29] K. Buytaert, R. Dittner, and D. R. Jr, "The Best Damn Server Virtualization Book Period," *Syngress, 10(2), 422, 2007.*

[30] R. Jain, "Packet Trains: Measurements and a New Model for Computer Network Traffic," *IEEE Journal on Selected Areas in Communications (JSAC), 4(6), 986-995 , 1986.*

[31] R. Kawahara, E. K. Lua, M. Uchida, S. Kamei, and H. Yoshino, "On the Quality of Triangle Inequality Violation Aware Routing Overlay Architecture," in *Proc. IEEE International Conference on Computer Communications (INFOCOM), 2009.*

[32] Networking and Traffic Control On Linux. [Online]. Available: http://tcng.sourceforge.net/

[33] S. K. Barker and P. Shenoy, "Empirical evaluation of latency-sensitive application performance in the cloud," in *Proc. ACM SIGMM conference on Multimedia systems (MMSys), 2010.*

[34] S. A. Ross, "Uses Abuses and Alternatives to the Net-present-value Rule," *Financial Management, 2(4), 96-102, 1995.*

[35] C. Cotta and J. Troya, "A Hybrid Genetic Algorithm for the 0-1 Multiple Knapsack problem," *Artificial Neural Nets and Genetic Algorithm 3, 250-254, 1994.*

[36] I. Katriel, L. Michel, and P. Hentenryck, "Maintaining Longest Paths Incrementally," *Constraints, 10(2), 159-183, 2005.*

Appendix A

To solve the assignment problem, we convert it into a directed acyclic graph (DAG) $G(V, E)$ with virtual source x and sink y (Figure 17. This is because the longest path (the slowest interaction path) can be found with worst-case running

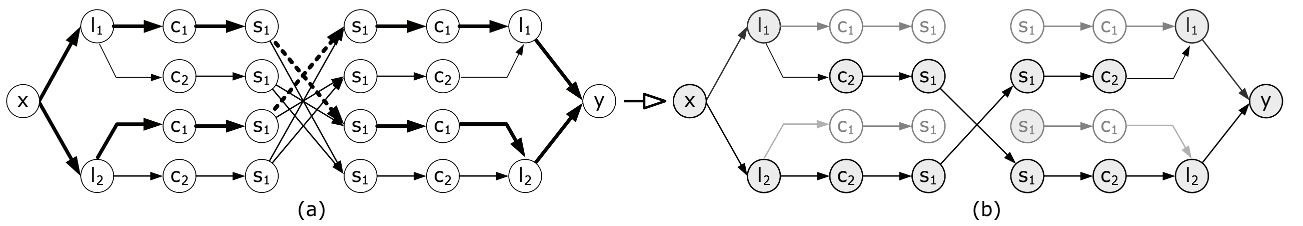

Fig. 17: Finding Optimal Assignment in converted DAG

Algorithm 2 AccommodateLeaseCost()

```
1:  Sort $C$ by ascendant order of $Cost(c_i)$;
2:  for $i = 1$ to $n$,
3:     $c_A(l_i) \leftarrow c_1$;
4:     $s_A(l_i) \leftarrow s_1$;
5:  end for
6:  $i \leftarrow 1$;
7:  while $i <= n$,
8:     for $j = 1$ to $i - 1$,
9:        if $PathExisted(x, l_i, c_A(l_i), s_A(l_i),$
          $s_A(l_j), c_A(l_j), l_j, y, G) == false$,
10:          break;
11:       end if
12:    end for
13:    if $j == i - 1$,
14:       $i \leftarrow i + 1$;
15:    else
16:       if $Next(s_A(l_i), S)! = null$,
17:          $s_A(l_i) \leftarrow Next(s_A(l_i), S)$;
18:       else if $Next(c_A(l_i), C)! = null$,
19:          $c_A(l_i) \leftarrow Next(c_A(l_i), C)$;
20:          $s_A(l_i) \leftarrow s_1$;
21:       else
22:          $i \leftarrow i - 1$;
23:       end if
24:    end if
25: end while
26: Return $A$;
```

Fig. 19: Algorithm to accommodate the lease cost

time of $O(|V| + |E|)$ [36] in a DAG $G(V, E)$. As illustrated in Figure 17(a) (which shows an example with 2 clients, 1 server and 2 cloud proxies), each path from x to y refers to one possible path between two clients in L. We first find the path with highest latency. For example, in Figure 17(a), the longest paths are shown in the dark lines (there will be 2 longest paths in each round since they are symmetric). We then try to remove the edges between servers (dotted lines in Figure 17(a)) only when all client pairs are still connected after this removal. This step is repeated until no edge can be further removed from the graph. At last, we find A, the assignment of cloud proxy and server for each client in the remaining graph so that all the client pairs can be connected. The optimal algorithm is given in Algorithm 1. The proof of its optimality can be found in *Appendix B*.

Although the maximum interaction latency is bounded by the longest path, the optimal assignment A is not unique in Algorithm 1. This allows us to further improve other metrics, for example, the lease cost for cloud proxies. Note that the proposed optimal model assumes that the costs for all the cloud proxies are homogeneous. While this is partly valid for CDIAs that rely on their own cloud platform, e.g., Onlive, it is not the case for those using public clouds (e.g., Amazon EC2) with varying costs depending on such factors as location, time, and capacities. An extension of our model with heterogenous lease costs can be found in *Appendix C*.

Appendix B

*Theorem 1:*Assignment A is an optimal assignment that minimizes the maximum interaction latency in Equation. 3.

Proof: Suppose that A' is another assignment in which the maximum interaction latency is smaller than A. Since A' and A can be both used to connect all client pairs in L, the longest path in A ($path^*$) can be replaced by a shorter path (say $path'$) that exists in A', and A can still make all clients in L connected after this replacement. Since $path'$ is shorter than the longest path in A, $path'$ also exists (is not removed) in A's *residual graph* (the graph after the longest path removal at Algorithm 1 step 4). This implies that there are two paths ($path^*$ and $path'$) in A's residual graph connecting identical client pairs with different latencies. This leads to a contradiction because the longest path in A can be safely removed without affecting the connectivity among clients.

Hence, the optimality of A is proved. ■

Appendix C

For cloud proxies with heterogenous lease costs, we assume that the unit cost of a cloud proxy c_i (providing service to one client) is $Cost(c_i)$. For a given assignment A, we use $q_A(c_i)$ to refer the number of clients that are assigned to c_i. The overall lease cost $\mathbb{C}$ is therefore:

$$\mathbb{C} = \sum_{c_i \in C} Cost(c_i) \cdot q_A(c_i) \tag{8}$$

It is easy to see that the minimum cost can be found by searching all the assignments. This could be very time-consuming in real practice. We therefore design a heuristic (Algorithm 2) to replace the last step in Algorithm 1 to obtain an economical assignment.