UBCGaming: Ubiquitous Cloud Gaming System

Wei Cai, *Member, IEEE*, Yuanfang Chi, *Student Member, IEEE*, Conghui Zhou, Chaojie Zhu, and Victor C. M. Leung, *Fellow, IEEE*

***Abstract*—Cloud gaming provides a novel service model by hosting game engines in the cloud and delivering real-time gaming videos ubiquitously to the players through the Internet. However, the diversity of end-user devices and frequent changes in network quality of service and cloud responses result in variable quality of experience for game players. This paper presents the design and implementation of a component-based ubiquitous gaming system with cognitive capabilities, which aims to overcome the abovementioned problem by learning about the game players' status and cognitively optimizing resource allocations for different software components of a game. Experiments show that well-balanced software components between the cloud and user devices lead to better system performance, e.g., in the overall latency.**

***Index Terms*—Cloud, decomposition, game, software.**

I. Introduction

BY EXPLOITING rich and elastic computing resources in the cloud, cloud gaming [1] is a major direction in the evolution of digital entertainment systems. As a novel paradigm, it brings many advantages to the profitable video gaming industry. From the perspective of game operators, cloud gaming is maintenance free and has nominal (almost negligible) costs for service provisioning compared to the costs of the hardware and gaming software that one has to pay for in personal computer (PC) or console gaming. In addition, cloud gaming is the best solution for antipiracy. The cloud-based gaming model changes the distributions of the games as software programs into providing gaming services, which creates continuous profit. From the perspective of game developers, the cross-platform nature of cloud gaming shortens the development time and reduces costs. From the perspective of game players, the offloading [2] approach in cloud gaming enables terminals that are weak in computation resources to overcome the intrinsic hardware constraints by leveraging the rich resources of the cloud. To this end, gamers can play sophisticated games without purchasing or upgrading their hardware.

PlayStation Now is an operating commercial cloud gaming service developed by Sony Interactive Entertainment. It follows pioneering cloud gaming companies, OnLive, Gaikai, and G-Cluster, to offer gaming services on demand. In this so-called streaming-based model, video games are hosted in private cloud servers and the gaming video frames are encoded by the streaming server before being transmitted over the Internet to the clients, such as interactive televisions, desktop PCs, and smartphones. In reverse, the players' inputs are delivered to a cloud server and accepted by the game content server directly [3]. In this context, the cloud serves as an interactive video generator and streaming server, whereas the terminals function as the event controllers and video receivers. Another approach to provide cloud gaming services is the browser game [4], which always relies on online social network sites with a massive number of users (e.g., FarmVille on Facebook). In a typical browser game, the gaming contents, including data and all of the gaming procedures, are stored and executed within the cloud, whereas the gaming graphics and videos are rendered by the browser, instructed by the returning scripts and documents from the cloud server. The two different architectures introduce tradeoffs in system performance [5]. The streaming-based model suffers from the bandwidth-bottleneck of Internet access. The bandwidth constraints restrict the bit rate of gaming videos, whereas the jitter and delay affect the quality of experience (QoE) for the players. Therefore, technologies regarding real-time video rendering, compressing, and control of network quality of service (QoS) become the most critical issues for system design [6]. On the other hand, browser games leave the presentation functionalities to the browsers, in order to eliminate the high bandwidth consumption for gaming video transmission. In other words, browser game is more efficient in the use of communication resources at the expense of a heavier computation load in the user device. In fact, the main distinction of the two existing cloud-based gaming models is the proportion of offloading. However, both of these models are lacking in flexibility that enables them to work well over the widely diverse scenarios of accessing cloud gaming services.

In this paper, we develop the first Ubiquitous Cloud Gaming (UBCGaming) System, which provides ubiquitous gaming experience despite the wide variations of players' gaming behaviors, terminal capacities, locations, and network conditions. The UBCGaming platform decomposes the game program into interdependent components that can be distributed to either the cloud or local terminal for execution, and thus enables the capacity of cognitive adaptivity for system performance optimization. In this context, cognitive adaptivity refers to the adaptive, interactive, contextual, and iterative procedure to learn the game system status and to provide corresponding adaptation of game component placements for the cloud gaming service.

Manuscript received January 9, 2017; revised November 26, 2017; accepted January 10, 2018. Date of publication February 9, 2018; date of current version August 23, 2018. *(Corresponding author: Wei Cai.)*

W. Cai, Y. Chi, and V. C. M. Leung are with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada (e-mail: weicai@ece.ubc.ca; yuanchi@ece.ubc.ca; vleung@ece.ubc.ca).

C. Zhou is with We Software Limited, Hong Kong (e-mail: neio.zhou@gmail.com).

C. Zhu is with the Entertainment Technolgy Center, Carnegie Mellon University, Pittsburgh, PA 15219 USA (e-mail: chaojiez@andrew.cmu.edu).

Digital Object Identifier 10.1109/JSYST.2018.2797080

The main contribution of this paper is to present the system design and development for cognitive decomposed cloud gaming. To the best of our knowledge, UBCGaming is the first system that enables the players to start gaming session without any installation, whereas the system workload can be dynamically balanced among the cloud and terminals. We investigate the issues in system implementation, and conduct empirical studies in system performance and experimental measurements to demonstrate the effectiveness of the UBCGaming platform.

The outline of this paper is as follows. We review related work in Section II and provide the architectural design of the proposed UBCGaming system in Section III. We then present the implementation details in Section IV. Prototype design and experimental results are given in Sections V and VI. Section VII concludes the paper.

II. Related Work

A. Cognitive Computing Systems

In computer science, cognitive computing [7] is a computing paradigm that helps to improve decision making by mimicking the human brains ability to acquire knowledge and understanding through thought, experience, and the senses. In general, a cognitive system [8] should be able to retrieve, memorize, and leverage the knowledge of specific area for learning and problem solving. By leveraging data-driven modeling and task automation [9], a cognitive system is characterized by an adaptive, interactive, contextual, iterative, and stateful paradigm [10], which is a perfect fit for the heterogeneous environment we are considering for cloud gaming. The situation-aware architecture of a cognitive system monitors and assesses the working environment to make decisions for the provided services, and refine future decisions by learning from the achieved results. In particular, UBCGaming employs cognitive computing to build a situation- and context-aware scheme that learns the players' behavior and the systems' environmental conditions during their gaming sessions, which enables the scheme to adapt the cloud-terminal workload to the run-time environment to optimize the use of cloud, network, and terminal resources while meeting the QoE objectives of the gaming sessions.

B. QoE for Cloud Gaming

Maintaining an acceptable QoE while optimizing resource utilization is the main objective of the proposed cognitive platform for cloud gaming. Various subjective user studies have been conducted to demonstrate the relationship between cloud gaming QoE and QoS, including game genres, video encoding parameters, conditions of the wireless network [3], CPU load, memory usage, and link bandwidth utilization [11], response latency and the game's real-time strictness [12], network characteristics (bit rates, packet sizes, and interpacket times) [13], number of users [14], and an empirical network traffic analysis of On-Live and Gaikai [15]. In this paper, we select game's real-time restriction as our primary concern, since it is the most critical issue in QoE improvement for cloud gaming system [12].

C. Cloud Gaming Architecture

According to [1], cloud gaming platforms can be categorized into two classes: *transparent platforms* and *nontransparent platforms*. The *transparent platform* is commonly used in traditional cloud gaming paradigm [16], e.g., PlayStation Now, which hosts the complete game engine in the cloud and delivers real-time gaming video via Internet. Existing games without major modification (minor revisions on the execution environment adaption may be required) can be executed on *transparent platforms*, which enriches the platform content at the expense of potentially suboptimal performance. For *transparent platforms*, adaptive-streaming-based cloud gaming architectures have been widely studied [17], because the network bandwidth becomes the bottleneck for the system performance. In contrast, *nontransparent platforms* [18], [19] require augmenting and recompiling existing games to leverage unique features, such as video-graph joint encoding and partial video rendering in terminal side, for better gaming experience, which may potentially be time consuming, expensive, and error prone. In this paper, our proposed decomposed cloud gaming system falls into the second category.

D. Partitioning Solution

To facilitate cognitive resource allocation, cloud games should support dynamic partitioning between the cloud and mobile terminals. Existing studies in dynamic partitioning focus on general applications that offload intensive computational task to the cloud. CloneCloud [20], one of the most representative flexible partitioning frameworks, enables unmodified mobile applications running in an application-level virtual machine to seamlessly offload part of their execution from mobile devices onto device clones operating in a computational cloud. However, CloneCloud requires a static program analysis with code intrusion during application development, which increases the complexity of software implementation. Moreover, an application on CloudCloud should be completely installed in both the mobile terminal and the virtual machine residing in the cloud before it is run. This requirement implies that the players should install the game program in their terminals prior to game playing. In addition, existing partitioning methods for general applications do not consider the extremely low-latency requirement for video games. In contrast, this paper proposes a completely new design that solves the above-mentioned issues.

III. System Design

To provide cognitive capabilities across the cloud gaming system, we need to overcome the diversity of end-user devices and frequent changes in network QoS and cloud responses. We use the concept of cognitive system design (i.e., act, learn, and adapt) to realize our proposed UBCGaming platform. Our objective is to develop an architectural framework that is cognitive of resources and characteristics of the cloud, the access network, and the end-user devices, which enables dynamic and optimal utilization of these resources based on the cognitive information. To this end, we envision a cognitive platform with a novel capability to learn about the game players' environment (i.e.,

TABLE I
HIGH-LEVEL SYSTEM REQUIREMENT

Requirement	Definition	Responsible unit
SR_1 Click-and-play	To enable immediate game-play when the players select their favorite games from the platform portal.	Onloading manager
SR_2 Performance evaluation	To measure, evaluate, and predict the overall system performance, including CPU load, memory usage, link bandwidth availability, and specialized application-layer metrics, such as the number of players, spatial distribution of the player population, or game state computation delay.	Performance evaluator
SR_3 Interaction modeling	To statistically understand the interaction model between the game instance and players: The interaction model between components, including the frequency of execution and probability of communications.	Interaction monitor
SR_4 Cognitive adaptation	To intelligently adapt gaming services to system performance, such as changing network QoS and different players' distinct behaviors, to keep users QoE above a prescribed threshold.	Cognitive engine

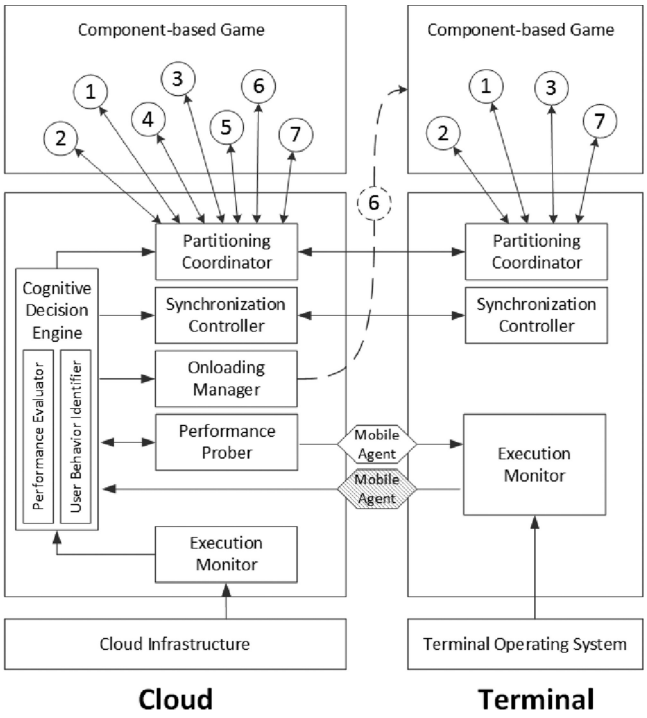


Fig. 1. Architectural design of UBCGaming platform.

the combination of terminal and access network) and adapt the running of the game to maintain an acceptable QoE. The proposed platform should be able to fulfill the requirement listed in Table I. Given the above-mentioned objectives, we design UBCGaming system as follows. Using specific application programming interfaces (APIs), the game developers implement game programs and deploy them on the platform.

Fig. 1 illustrates the main building blocks of the architectural framework of this platform on both the cloud side and terminal side, and identifies the prevalent standards that are applicable to the interfaces between these building blocks. During the gaming sessions, the *performance evaluator* collects the execution efficiency parameters for each game instance, such as execution latency and network round-trip time. Meanwhile, the *user behavior identifier* analyzes the statistics of interactions between players and game components and invocations between components. The results of performance evaluation and players' user behavior are utilized by the *cognitive engine* as the references of dynamic partitioning.

A. Decomposition

Decomposition is an intrinsic requirement of the proposed cognitive platform, since the system aims to dynamically manage the workload balance between cloud and terminals by migrating a selected set of game components from the cloud to the terminals. In this context, the term "decomposition" is defined as "breaking a large system down into progressively smaller classes of objects that are responsible for some part of the problem domain." The first issue in decomposition is to determine the granularity of decomposition: fine-grained or coarse-grained decomposition. *Fine-grained decomposition* refers the design patterns that segment the whole game program into tiny components, e.g., functions or methods. In contrast, *coarse-grained decomposition* partitions the game program into larger components, which contains relatively complete and independent functionalities. These components are often composed by a set of objects and methods, which work collaboratively within the scope of the component.

Apparently, *fine-grained decomposition* provides a larger quantity of tiny components, which enables more flexible partitioning schemes. Therefore, it leads to better potentials in system optimization than *coarse-grained decomposition*. Nevertheless, a *fine-grained decomposition* model also needs to maintain more data flows between components. Data serializing and parsing for data transferring might introduce more computation and communication overhead to the overall system. On the other hand, the data model in *coarse-grained decomposition* is simpler. Considering these tradeoffs, we adopt coarse-grained decomposition in this paper, in order to minimize message transmissions among components. More details regarding game decomposition by dynamic partitioning can be found in our previous work [21].

B. Partitioning Coordinator

The *partitioning coordinator* is the key component that facilitates dynamic partitioning. Intrinsically, it is a message router/redirector among components in both the cloud and

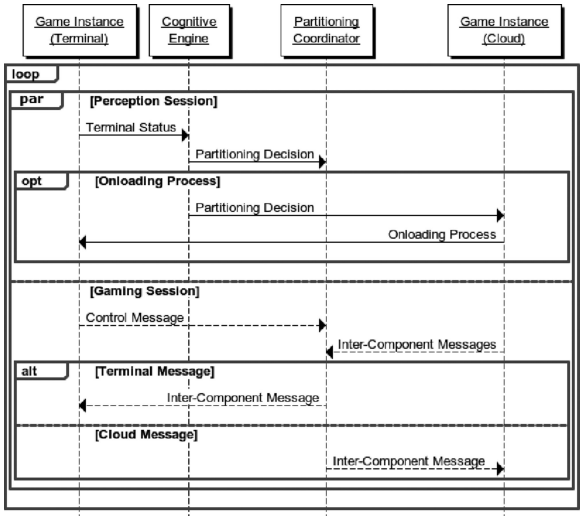


Fig. 2. Sequence diagram for dynamic paritioning.

terminals. Fig. 2 illustrates a sequence diagram for the dynamic partitioning mechanism in the UBCGaming platform. The cloud server first launches the game instance in the cloud to start the game. Meanwhile, a small initial portion of game code is dispatched to the terminal. During the gaming session, the gaming instance in the terminal keep sending status statistics to the *cognitive engine*, which analyzes the system performance and acknowledges the *partitioning coordinator* and its decision of partitioning. This decision instructs the *partitioning coordinator* to redirect control messages and intercomponent messages, in order to facilitate dynamic partitioning. Note that there is an optional *onloading* process when the *partitioning coordinator* receives decisions from the *cognitive engine*. This process is instructed by the *onloading manager* to push components from the cloud to terminals, if the terminals lack sufficient components to perform optimal partitioning.

C. Execution Monitor

The execution monitors on both the cloud and terminal sides monitor resource usage of all the components, whether in the cloud or terminals, and save the execution information in a statistics database. The execution information pertains to the property of each component in each invocation, including memory consumption, CPU usage percentage, execution time, output data size, execution environment, etc. With the execution information database, the system is able to retrieve the invocation trees between the components, including the relationship between components and the execution frequency of each component. In fact, the players' interactional behaviors can also be identified from this database.

D. Performance Prober

However, the information extracted from the execution information database is insufficient for the proposed platform to perform cognitive adaptation, since the system needs to evaluate the execution performance of each component both in the cloud and terminals. This might not be possible in a real system, since the platform can never migrate all components to the terminals for the tests. Furthermore, the platform also needs to measure the network QoS between the cloud and terminals. Therefore, we design a mobile agent [22] based performance prober to probe the cloud-network-terminal environmental parameters.

A mobile agent is a composition of computer software and data, which is able to migrate (move) from one computer to another autonomously and continue its execution on the destination computer. In this context, the game components are encapsulated as mobile agents and dispatched from the cloud to mobile devices. In our design, we set up a mobile agent component with designated iterations and measure the component's execution information in the cloud. Afterward, the component is dispatched and executed in the terminal. Its execution information is measured and reported to the performance prober in the cloud. An illustration of mobile agent prober is depicted in Fig. 1. Note that this process involves two network transmissions, with which the system is able to calculate the network QoS parameters, including the round-trip time and data transmission rate. With this approach, the performance prober is able to compare the computational efficiency of cloud and terminals, and consequently estimate the execution information for all components. In our implementation, we denote the cloud computational efficiency as E_c and the terminal computational efficiency as E_t; therefore, we are able to compute the efficiency ratio $R_E = E_c/E_t$. With R_E, we can estimate the execution information that we could not directly measure from the system. For instance, the cloud execution time for a particular component can be estimated as $E_t \times R_E$. Note that the computational efficiency is determined by a variety of parameters, such as CPU frequency, memory, and interpreter efficiency. In this paper, we simplify our model by using execution time as the indicator of efficiency. Less execution time indicates a better efficiency, and vice versa.

In order to minimize the overhead of probing, the performance prober is designed to work collaboratively with the execution monitor. As mentioned in Section III-C, the cognitive system calculates the execution probability of each component in a statistical approach. Hence, it is necessary to traverse in both the clouds and terminals' databases. Meanwhile, the statistics in each terminal should be reported to the cloud periodically. Accordingly, the mobile agent component in the performance prober is designed as the database information retriever and carrier to improve the system efficiency. In our implementation, the system dispatches a performance prober to the terminals in an interval of T_I and save the probing results in a database. Table II illustrates some entities of the database. In fact, with real-time data analysis, the interval T_I can also be a variable subject to the condition of the terminal, e.g., the network QoS.

E. Onloading Manager

Since the cognitive platform supports *click-and-play*, none of the game components exists in the terminal at the beginning of a gaming session. In this case, the cloud server should be capable to transmit executable components to the terminal, in order to enable dynamic resource allocation. The *onloading*

TABLE II
TABLE OF PERFORMANCE INFORMATION

Cloud Proc.		Terminal Proc.		Code Trans.	
Time	Records	Time	Records	Time	Length
20 ms	100	11 ms	30	28 ms	3 kB
42 ms	208	19 ms	51	22 ms	3 kB
78 ms	376	26 ms	70	33 ms	3.2 kB
102 ms	512	40 ms	92	32 ms	3.3 kB
...	...	...	...	...	...

manager employs the concept of mobile agent to realize this process: components are stringified and dispatched to the terminals as messages. Note that the onloading process could either be performed before a gaming session starts or be running in the background during the gaming session. It is scheduled by the *onloading manager*, which assigns each game component a priority based on the overall assessment of the particular component. Nevertheless, the priority of a game component is also associated with its functionality. Some key components should have a higher priority in the onloading process, since they provide featured benefits in the terminal.

To simplify the system, we implement three types of onloading in our platform.

1) The system administrator manually onloads specific components.
2) The platform randomly dispatches selected components to the client, when the network connections between the cloud and terminals are idle.
3) The client requests specific components from the cloud once the optimal solution is determined, and some of the components required by the client are still missing.

In our experiments, we adopt the third mode since the code lengths of the components are relatively short such that their transmission costs are negligible in the experimental results.

F. Cognitive Decision Engine

The cognitive decision engine is the key unit that cognitively determines the system strategy after periodically analyzing the information from the *execution monitor* and *performance prober*. The main strategies include component onloading for the *onloading manager*, dynamic partitioning for the *partitioning coordinator,* and data synchronization for the *synchronization controller*. Designing the cognitive decision engine is the most challenging work for UBCGaming. Theoretical studies have been conducted in our previous work [23]. In this paper, we investigate an empirical solution by exploring the knowledge we derived in Table II. Detailed algorithms are presented in Section V-C.

G. Synchronization Controller

The remote distribution of game components results in the problem of data getting out of synchronization. To address this problem, the *synchronization controller* is employed to update all the parameters in the gaming environment. However, the synchronization process also introduces a nonnegligible network overhead. Consequently, we design the synchronizing mechanism following the principle of "sync-only-if-necessary" to minimize the transmission cost: Necessary parameters are serialized as JavaScript Object Notation (JSON) data and passed to destination components as a message.

IV. TEST BED IMPLEMENTATION

In this section, we describe the implementation of UBCGaming and discuss related issues.

A. Enabling Technologies

The first stage of implementation is to seek suitable enabling technologies that facilitate the migration and partitioning of game components. JavaScript is adopted as the programming language, which is originally implemented as a part of web browsers so that client-side scripts could interact with the user, control the browser, communicate asynchronously, and alter the document content that was displayed. More recently, however, its use has become common in both game development and the creation of desktop applications. Node.js[1] is a server-side software system designed for writing scalable Internet applications, notably web servers. Programs on the server side are written in JavaScript, which enables web developers to create an entire web application in JavaScript, for both the server side and client side. This feature facilitates the game components, JavaScript gaming codes in this context, to migrate from the cloud to user-end, and to be executed on cloud servers and clients as mobile agents. For the client, we embed a WebKit-based browser into the cognitive engine for parsing and executing JavaScript mobile agents from the cloud server. In our implementation, the WebKit browser is built for Android smartphones. However, all devices that support browsers are able to run our cognitive platform after a small amount of modifications. We are also looking for alternative solutions to implement the client as native applications on JavaScript. As the state-of-the-art, Microsoft already supports native application development with JavaScript on its metro-style interface.

B. Application Programming Interface

To support component-based games, the UBCGaming system provides a set of APIs to encapsulate lower layer partitioning for game developers. To acknowledge an intercomponent invocation, the developers simply add a "$$" mark before the name of the components when they are invoked in the code. For example, $$component$X(\{\text{msg} : \text{args}\})$; stands for an invocation of *component*X with a parameter of $args$ passing as a message. Note that parameters are serialized as a JSON.

C. Administration Center

Fig. 3 illustrates a screenshot of the UBCGaming administration center rendered by config.ejs. The administrator can browse all ongoing gaming sessions here from the *TERMINALS* session list. For each terminal, the partitioning and loading status of all

[1][Online]. Available: http://nodejs.org/

Fig. 3. Administration center of UBCGaming.

components are illustrated. Note that if the *AUTO OPTIMIZATION SWITCH* is turned OFF, we can even manually control each component's execution environment by a simple click. This feature supports our following experiments that test the efficiency of different task allocation schemes. In addition, the administration center also depicts real-time figures for terminal status, including network bandwidth, usage of client CPU, memory, battery, etc.

V. Prototype Development

To verify the feasibility and efficiency of UBCGaming, we developed a number of decomposed game prototypes, including 3D Skeleton Prototype, Gobang Game, and Robocode Tank, as described in our previous work [23]. In this paper, our target is to design quantitative measurements and experiments to demonstrate the proposed functionalities in the UBCGaming system. Critical issues in designing such experiments include the following:

1) How to select representative game prototypes.
2) How to measure the impact of computational capacities in the cloud and terminals.
3) How to measure the impact of networking parameters.

In order to address above-mentioned issues, we select the Robocode Tank game prototype to study the execution and network status for components. There are two reasons we select the Robocode Tank game prototype. The first one is *flexibility*. In this game, we are able to create battlefields with different numbers of tanks. Also, the computational cost for each tank is adjustable. Complicated algorithm in shooting and moving strategy might introduce more intensive computing procedure. Therefore, we can simulate a variety of gaming scenarios by selecting different combination of tank numbers and their complexities. The second reason is *measurability*. We need a unified metric to measure the performances of partitioning schemes, which should be quantitative and representative. The Robocode Tank game records the frame per second (FPS) for the game scenes rendering, which is a perfect overall indicator of system performance: A better performance yields a higher FPS.

To simulate the cloud, network, and terminal environments, we set up an experimental test bed at the University of British Columbia. In order to flexibly simulate more combination of hardware settings, we employed two PCs to virtualize the environment of cloud and terminal. These PCs are each equipped with an 8-core CPU and 8 GB of memory. During the experiment, we use Basic Input/Output System settings to adjust the CPU cores and memory size, in order to simulate different hardware scenarios. Nodejs v4.4.7 is installed as the cloud server software, while Firefox 45.0.2 is adopted as the default client in the terminal. To connect cloud and terminal, we wired them to a LinkSys Wireless Router WRT120N router so that they are interconnected within one local area network. The Robocode tank game is deployed on port 8080 of the cloud PC and the terminal accesses the game through cloud's local Internet protocol address. In order to control the network parameters between cloud and terminal, we installed NetLimiter 4,[2] an Internet traffic control and monitoring tool designed for Windows.

A. Measurements

In this section, we perform measurements for computational capacity and the communication capacity. Our purpose is to calibrate our test bed by determining how different systemic factors affect these two capacities. The factors we considered include the CPU and the memory of the cloud and the terminal, and the network between the cloud and the terminal.

Our test bed records the FPS for the game scenes as the indicator of system performance, since it represents the overall latency introduced by the component execution and communications. A better performance will yield a higher FPS. In particular, we record the real-time FPS values in the gaming session and send them back to the cloud side. In this way, we are able to quantify the performance of the whole application with its FPS value. The average FPS is calculated afterward, which represents the performance of the application under this situation. By comparing the different FPS under different circumstances, we can reveal how the capacity is affected on either the cloud side or the terminal side, and which factors affect the capacity the most. These findings enable us to develop better solutions for dynamic partitioning.

1) Computational Capacity: Our hypothesis is that the parameters effecting computational capacity include computing intensity and process concurrency. The former value indicates the computational complexity of a particular component, whereas the later value indicates the ability of parallel execution for multiple components. In this paper, we used iterative execution of a segment of code to demonstrate the computation intensity, and the quantity of simultaneous components to simulate the process concurrency.

Before quantitatively studying the influence of different factors, we conducted two case studies to discover the relationships between FPS and these two parameters. The specifications of the cloud server and the terminal in the case studies are listed in Table III.

[2][Online]. Available: https://www.netlimiter.com/

TABLE III
SPECIFICATION FOR CASE STUDIES

	Cloud	Terminal
CPU	8 cores with 3.40 GHz	4 cores with 3.40 GHz
Memory	8 GB	8 GB
System	Windows 7 Professional	Ubuntu 16.04LTS

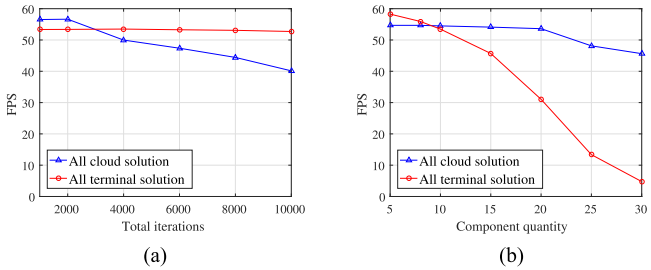


Fig. 4. Different computational capacities of the cloud and the terminal. (a) FPS and iterations. (b) FPS and component quantity.

The first case study sets five components running at a time with the same number of iterations and gradually increases the total number of iterations from 1000 to 100 000, maintaining all other factors unchanged. As illustrated in Fig. 4(a), the cloud originally has a higher FPS when the total number of iterations is 1000; then the FPS slowly goes down, and gets lower as the number of iterations is increased to 4000. Comparatively, the FPS when all components run on the terminal is much more stable at a high level. The second case study restricts the total number of iterations to 5000 and all components share the same iterations. Then, we gradually add the number of components (with the iterations of each component decreasing) from 5 components to 30 components, maintaining all other factors unchanged. As shown in Fig. 4(b), the FPS is higher originally when all components run on the terminal, but it declines dramatically with the increase in the number of components. The average FPS even gets lower than 10 when the number of components reaches 30. In contrast, the FPS curve of the cloud only case declines about 15% when the number of components increases to 30. Fig. 4 illustrates that the terminal can maintain a higher computation capacity with a larger number of iterations, while the cloud servers obviously can handle concurrency better. So when partitioning the components, these two parameters need to be taken into account to place the components to more suitable sides. Inspired by these case studies, we designed a number of experiments to further validate the relationships between hardware specifications and execution performance. Our results reveal that CPU core quantity and memory size are the most important factors in computational performance. Therefore, we select different combination of CPU core quantity and memory size to further evaluate the FPS values, and hence the performance, of game execution.

Iterations: To find out how CPU and memory may affect the performance of the cloud server and terminal, we design experiments with all components executing in the cloud (Experiment C1 to C5) and all components executing in the terminal (Experiment T1 to T5), with specifications shown in the legends of

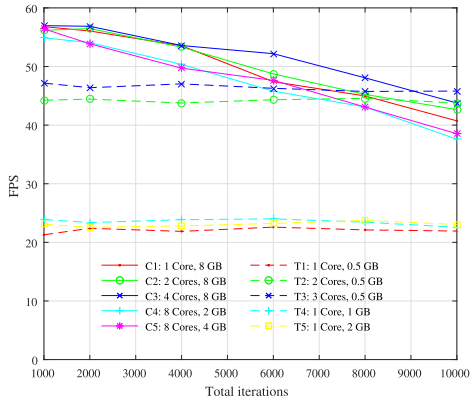


Fig. 5. FPS experimental results for total iterations.

Fig. 5. The performance of the cloud and terminal, given by the average FPS with each specification, is shown in Fig. 5. It can be seen from the figure that when the total number of iterations for the components increases from 1000 to 10 000, the FPS values are stable for all experiments conducted in the terminal (T1 to T5), while an obvious downtrend is seen for all experiments with the cloud (C1 to C5). This implies that the terminal can tolerate a higher number of iterations in components under our experimental settings. On the terminal side, we use T1 with 1 CPU core and 0.5 GB memory as our baseline. We first increase the number of CPU cores to 2 (T2) and 3 (T3). It turns out the performance of T2 is 100% better, whereas T3 further increases FPS by less than 10%. We use T4 and T5 to simulate the cases with 1 GB and 2 GB memories, respectively, in the terminal. Different from the increase of CPU cores, there are no significant changes in FPS values. In contrast, we conducted similar experiment sets C1 to C5 on the cloud side. However, there are no significant differences in FPS values from different experiment settings.

Concurrency: As mentioned in Section V-A1, concurrency, which indicates the ability of parallel execution for multiple components, is another important parameter affecting the computational capacity. Similar to the iteration studies, we again set different combinations of specifications in cloud and terminal and record their FPS results as shown in Fig. 6. As expected, more concurrent components execution leads to lower FPS values in both cloud and terminal experiments. The only difference is that the rate of performance decrease for the terminal experiment sets (T1 to T5), dropping from 55 to nearly 0, is much faster than that of the cloud sets (C1 to C5), which indicates that the cloud outperforms the terminals in terms of concurrent computation capacity under our settings. On the other hand, it is interesting to note that there are no significant performance differences with distinct hardware settings for either cloud or terminal experiments. This phenomenon implies that neither increasing CPU cores nor enlarging memory size help to improve the concurrency performance in the terminal.

2) Communication Capacity: Besides the computation capacity, the communication capacity between components are also critical in determining the system performance. We categorize the communications into three kinds: *intracloud communications*, *intraterminal communications,* and *cloud-terminal*

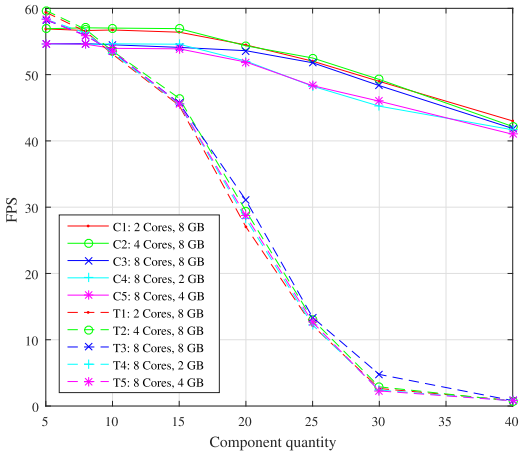


Fig. 6. FPS experimental results for concurrency.

TABLE IV
SYSTEM SPECIFICATIONS FOR EVALUATIONS OF THE COMMUNICATION CAPACITY

Side	CPU	Memory	Operating system
Cloud	8 Cores	8 GB	Windows 7 Pro
Terminal	8 Cores	8 GB	Windows 7 Pro

TABLE V
COMPONENTS FOR COMMUNICATION CAPACITY EVALUATIONS

Component No.	1	2	3	4	5	6
Iterations	9200	4000	8900	2200	3600	200
Component No.	7	8	9	10	11	12
Iterations	7100	9200	900	3700	100	4500

communications. Also, we quantify the communications with two parameters: the message length and the communication frequency. In our measurement, the message length is evaluated by *bytes per message* (bpm). In addition, we consider 20 movement steps of a tank as a batch and the frequency is evaluated by the number of *communications per batch* (tpb). Apparently, a higher bpm value represents a larger message payload in one transmission, whereas a higher tpb value indicates a higher communication frequency. Default experiment settings are listed in Table IV.

We start with a simplified *circular* communication model: Each component sends a certain message to its next component with a specific frequency, i.e., Component No. 1 sends a message to Component No. 2, Component No. 2 sends a message to Component No. 3, etc., and the last component sends a message back to Component No. 1. Besides, in order to simulate a real scenario, we create the following iterations for each component in a specific component group as listed in Table V.

Intracloud Communications: This set of results indicates the capacity of transmitting data from one component to another within the cloud. We reveal how message length and frequency may affect the communication capacity by the different combinations. The FPS results under different communication settings

TABLE VI
FPS OF INTRACLOUD COMMUNICATIONS

FPS: Frequency / Msg length	200 tpb	100 tpb	60 tpb	20 tpb	2 tpb
0 bpm	42.06	42.06	42.06	42.06	42.06
128 bpm	40.33	40.44	40.59	41.76	41.76
16384 bpm	37.70	38.00	38.72	40.33	40.58
65536 bpm	26.31	32.27	33.17	36.21	40.18
262144 bpm	3.42	9.43	12.95	14.76	38.42

TABLE VII
FPS OF INTRATERMINAL COMMUNICATIONS

FPS: Frequency / Msg length	80 tpb	60 tpb	40 tpb	20 tpb	5 tpb
0 bpm	57.57	57.57	57.57	57.57	57.57
128 bpm	3.41	36.87	48.87	56.99	57.57
16384 bpm	1.92	15.51	41.98	57.00	57.34
65536 bpm	0.0	1.70	24.76	54.31	56.59
262144 bpm	0.0	0.75	2.59	18.41	46.25

are shown in Table VI. The first row is considered as the baseline. As we can see from the table, the FPS values decline when the lengths of the messages get larger or the messages are sent more frequently. In fact, except the worst situations, where the length of messages is 262 144 B, the FPS results in most of the experiments are higher than 26, which are considered acceptable in a gaming environment.

Intraterminal Communications: This set of results indicates the capacity of transmitting data from one component to another within the terminal. We select the first six components in Table VI to conduct similar experiments in the terminal. The resulting FPS values shown in Table VII are surprisingly low. Relatively speaking, the communication frequency seems to be a more important factor than message length: The FPS is still above 30 when 128-B messages are sent for 60 times per batch, whereas the FPS is lower than 5 when the frequency is increased to 80 tpb. This study reveals that communications within the terminal is a huge burden that should be minimized.

Cloud-Terminal Communications: This set of results indicates the capacity of transmitting data from one component to another between the cloud and the terminal. Cloud-terminal communications involve network transmissions between the cloud and terminal, which may introduce a round-trip time delay to the components. In the circular communication model that we adopt for these tests, we manually place the odd numbered components in the cloud and the even numbered ones in the terminal, so that all component communications are cloud terminal in nature. The FPS results are listed in Table VIII. As expected, the FPS value is high when the message length is small and the sending frequency is low, and vice versa. The combination of 128 bpm and 200 tpb can still yield over 35 FPS, which indicates that having multiple cloud-terminal communications in dynamic partitioning is acceptable in terms of latency.

B. Observations

These measurements result in several conclusions regarding the computation and communication capacity.

TABLE VIII
FPS OF CLOUD-TERMINAL COMMUNICATIONS

FPS: Frequency / Msg length	200 tpb	100 tpb	60 tpb	20 tpb	2 tpb
0 bpm	46.71	46.71	46.71	46.71	46.71
128 bpm	35.68	40.88	42.47	45.87	46.38
16384 bpm	3.06	16.63	25.88	32.68	44.16
65536 bpm	0.33	1.08	2.05	6.94	41.22
262144 bpm	0.02	0.02	0.02	0.54	32.41

1) The cloud outperforms terminal in terms of component concurrency, whereas the terminal outperforms the cloud in terms of iterations and in terms of computation capacity.
2) Increasing the memory capacity can be helpful for the cloud to improve its computation capacity, whereas adding more CPU cores is more effective for the terminal.
3) The cost of intracloud communications in degradation of performance is negligible in most cases.
4) Intraterminal communications degrade the performance dramatically. Relatively, the frequency of communications is a more important factor than the message length.
5) Cloud-terminal communications have a medium impact on performance compared to intracloud communications and intraterminal communications.

The communication frequency is a relatively more important factor for cloud-terminal communications, since it determines the number of round-trip times. Note that these observations are based on our implementation. Different programming languages, compilers or interpreters, and hardware architectures might result in different conclusions. However, our study on this specific platform is a solid step toward real system implementation. The system framework and the measurement-based methodology we introduced in this paper are still valid to alternate platforms. Moreover, the cognitive algorithm introduced in the next section can be easily modified and fine-tuned to address the specific characteristics of different platforms.

C. Cognitive Engine

With the above-mentioned observations, the optimal partitioning problem can be intrinsically transformed into the problem of seeking the optimal tradeoff between cloud and terminals. Our target is to improve the performance on-the-fly and eventually obtain the best partition for the application in a dynamic context. The partitioning algorithm we implement in this paper is a greedy algorithm that moves the components with the most iteration from the cloud to the terminal step-by-step until the FPS is maximized. We use a greedy algorithm because it is simple and fast, which better satisfies the players' QoE improvement in a real-time context. In our implementation of the cognitive engine, we divide the optimizing procedure into two operations: *performance probe* operation and *partitioning* operation.

The *performance probe* operation collects two kinds of data needed for optimization. One is the real-time FPS of the Robocode tank game. The FPS values are obtained in the terminal, so the terminal needs to send these data to the cloud.

Algorithm 1: Greedy Partitioning Algorithm.

```
if score[partition_current] > threshold then
    Keep current partition
else if fps_current < fps_previous then
    Undo last action in history
    score[partition_previous] ++
else
    find the component with longest execution time
    Move component_max from cloud to terminal
    Record this action in history
end if
```

Another one is the performance information of each component, as described in Section III-D. The *partitioning* operation aims to seek a suboptimal partitioning scheme under dynamic circumstances. It takes an attempt-confirm strategy or an attempt-reject strategy by comparing current FPS to previous values. During cognitive partitioning, the engine makes partitioning attempts and decides whether to confirm or reject them according to the performance probe operation. After this procedure, the overall performance of the application should be improved to satisfy players' QoE requirements. The pseudocode of this algorithm is listed in Algorithm 1.

According to the algorithm, the cognitive engine initializes a score for each partitioning scheme. The score is designed to decide whether the optimizing operation should be terminated. Once the score of one partition goes above a certain threshold (which is set as 3 in our experiments), the optimization is stopped to provide a stable experience for the players. Once the gaming session starts, the system identifies the component that consumes the longest time and attempts to move it to the terminal. If the last attempt improves the performance, the cognitive engine confirms this attempt as a better partitioning solution. Otherwise, the system rejects the attempt. There are many reasons for FPS reduction, from the high component quantity in the terminal to excessive intraterminal communications. No matter what the reason is, the previous partition is a better one if the current attempt reduces the FPS. As a result, the cognitive engine adds to the score of the previous partition and rolls the attempt back to the previous state. This algorithm demonstrates a simplified form of a cognitive loop that observes the system to obtain performance information (shown in Table II), decides the next component to move, and reacts to the FPS result.

VI. EXPERIMENTS

In this section, we validate the effectiveness of the proposed decomposed UBCGaming system by designing experiments to simulate practical cloud gaming scenarios. By applying the cognitive engine to these four experiments, the results show whether the cognitive engine can improve the system performance and to what extend the performance is enhanced.

A. Experimental Settings

We pick four general situations of component distributions in game development: the simple games with few components,

TABLE IX
TABLE OF COMPONENTS IN DIFFERENT EXPERIMENTS

No.	Comp. Quant.	Purpose	Iterations for each components
1	12	Simple games with fewer components.	9200, 4000, 8900, 2200, 3600, 200, 7100, 9200, 900, 3700, 100, 4500
2	14	Games with high-iteration components.	5000, 100, 3100, 1400, 4500, 6000, 4600, 2100, 700, 5800, 3900, 1500, 5400, 6000
3	20	Games with huge number of components.	1200, 2600, 7500, 4100, 100, 5400, 2200, 5300, 7000, 3700, 200, 5200, 5200, 6600, 8000, 2700, 5100, 6500, 2900, 6000
4	14	Games with only a few high-iteration.	8800, 6000, 200, 800, 900, 1000, 6500, 9200, 3000, 1300, 400, 2700, 400, 100

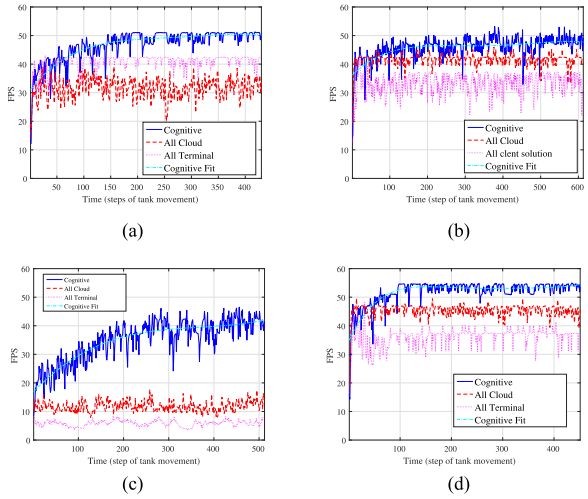


Fig. 7. Simulations of the partition algorithm. (a) Experiment 1. (b) Experiment 2. (c) Experiment 3. (d) Experiment 4.

TABLE X
RESULT OF PARTITIONING OF COMPONENTS WITHOUT NETWORK

No.	Terminal	Cloud
1	1, 2, 3, 5, 7, 8, 10, 12	4, 6, ,9, 11
2	1, 6, 7, 10, 14	2, 3, 4, 5, 8, 9, 11, 12, 13
3	3, 4, 6, 8, 9, 12, 14, 15, 18, 20	1, 2, 5, 7, 10, 11, 13, 16, 17, 19
4	1, 2, 7, 8, 9, 12	3, 4, 5, 6, 10, 11, 13, 14

TABLE XI
COMMUNICATION PARAMETERS

No.	Frequency (tpb)	Message Length (bpm)
1	4	128
2	20	65 536
3	100	256
4	80	4096

the complex games with a larger number of components, the games with only several high-iteration components leaving others low-iteration ones, and the games mainly composed by high-iteration components. As a result, the experiments are in correspondence with these four situations by an elaborately designed combination of iterations and component quantity to make our simulation of the cloud gaming system practical. The settings and purposes of the four experiments are listed in Table IX. With these settings, we conducted two sets of experiments, one with no communication among components and another with circular communications. With this approach, we can analyze the impact of the components on computation complexity and communication complexity separately.

B. Experimental Results

1) Without Networking: We start Robocode tank gaming sessions with no communication between components and record the FPS values over time. We compare our proposed dynamic partitioning scheme with two static approaches: all cloud solution and all terminal solutions, where the former scheme allocates all components in the cloud and the later one assign all components to the terminals. As illustrated in Fig. 7, the cognitive partitioning scheme outperforms the other solutions in terms of FPS in all of these four scenarios. Especially in Experiment 3, the cognitive approach is able to enhance the performance by up to 300%, which validates the efficiency of dynamic partitioning.

The suboptimal component partition solutions in the four scenarios are listed in Table X. Apparently, some components should be executed in the terminal, whereas the others should be hosted in the cloud. Furthermore, these partitioning results prove two of our hypotheses.

1) A component's execution time can be used to evaluate its number of iterations. In all of these four experiments, the components being moved to the terminal are those components with the most iterations, which meets our expectation.
2) The results verify that in contrast to the cloud, the terminal has a better computation capacity when handling components with high numbers of iterations and worse computation capacity when handling concurrency. Experiment 4 contains more high-iteration components than Experiment 2, which leads to more terminal components as well.

2) With Networking: Communications between components are also an important factor in some game genres. For example, some tanks might share their information with their teammates in order to perform cooperative battle. However, to the best of our knowledge, there is no existing work on communication models among components in game programs. It is intuitive that these communication models will strongly depend on the application design. In this paper, we simulate these communications by transferring some pseudodata among the tanks in a circular communication model, as described in Section V-A2. Parameters for communications are as listed in Table XI.

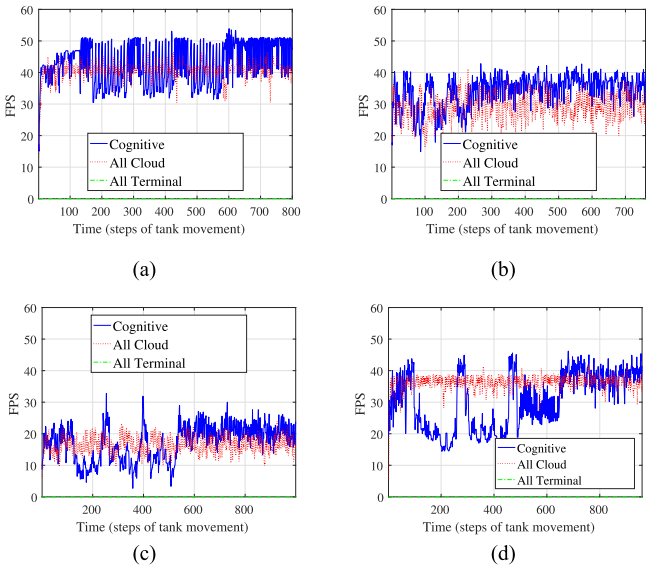


Fig. 8. Simulations of the partition algorithm. (a) Experiment 1. (b) Experiment 2. (c) Experiment 3. (d) Experiment 4.

TABLE XII
RESULTS OF COGNITIVE PARTITIONING OF COMPONENTS OVER NETWORK

No.	Terminal	Cloud
1	1, 3, 7, 8	2, 4, 5, 6, 9, 10, 11, 12
2	14	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
3	3, 9, 15	1, 2, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20
4	2, 8	1, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13

The FPS values through optimizing procedure are illustrated in Fig. 8 and the partitioning outcome for each scenario is listed in Table XII.

Fig. 8 shows that the cognitive approach always yields the best performance among all solutions in terms of FPS. It is noteworthy that the all-terminal scheme yields FPS values of 0 at all time, due to the heavy workload of message transmissions among tanks within the terminal. Evidently, certain type of games need cloud gaming systems as they are not executable in the terminals without cloud support. However, as shown in Fig. 7, the FPS improvements of the cognitive algorithm over the all-cloud case are not as significant, due to the limitation of the networking overhead in remote component invocation.

In addition, the differences in the partitioning outcomes in Tables X and XII also provide some important information. For example, consider components 1 and 2 in Experiment No. 4. When there is no communication in between, as shown in Table X, both of these two components are executed in the terminal. On the other hand, with network communications, Table XII shows that these components are partitioned to different sides. The underlying reason is that both of them are high-iteration components and the communication cost in Experiment No. 4 is high (80 tpb and 4096 bpm). The terminal cannot simultaneously handle the computation and the message transfer workload. In contrast, both components 7 and 8 in Experiment No. 1 contain high iterations, but both the partitioning outcomes in Tables X and XII put them in the terminal. That is because the messages between them are short in length (128 bpm) and are sent infrequently (tpb). The lesson learnt from these observations is that the computation and the communication costs should be comprehensively considered in the dynamic partitioning of components. Consequently, further developments and enhancements of the cognitive engine should include the ability to discern these different situations.

VII. CONCLUSION

In this paper, we have proposed a cognitive, flexible, and promising gaming platform for cloud gaming. Unlike previous work on cloud games, we have proposed a component-based game structure and designed specific mechanisms to facilitate the envisioned objectives, such as dynamic onloading process, partitioning, and synchronization. We have discussed the enabling technology and implemented the proposed platform as a pure JavaScript solution. Experimental measurements have been performed to show that adaptive partitioning is able to provide improved solutions in terms of overall latency.

While the proposed platform is promising, this paper reveals several limitations that we intend to address in our future work to refine this platform.

1) Better reasoning and learning approach should be investigated to predict the players' status for optimal partitioning.
2) More sophisticated game prototypes with larger numbers of components should be developed and tested in the platform.
3) More precise model on system performance should be established to consider more factors in decision making.

REFERENCES

[1] W. Cai *et al.*, "A survey on cloud gaming: Future of computer games," *IEEE Access*, vol. 4, pp. 7605–7620, 2016.
[2] K. Yang, S. Ou, and H.-H. Chen, "On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications," *IEEE Commun. Mag.*, vol. 46, no. 1, pp. 56–63, Jan. 2008.
[3] S. Wang and S. Dey, "Modeling and characterizing user experience in a cloud server based mobile gaming approach," in *Proc. IEEE Global Telecommun. Conf.*, Nov./Dec. 2009, pp. 1 –7.
[4] J.-M. Vanhatupa, "Browser games: The new frontier of social gaming," in *Recent Trends in Wireless and Mobile Networks (Volume 84 of Communications in Computer and Information Science)*, Berlin, Germany: Springer, 2010, pp. 349–355, doi: 10.1007/978-3-642-14171-3-30.
[5] W. Cai *et al.*, "The future of cloud gaming [point of view]," *Proc. IEEE*, vol. 104, no. 4, pp. 687–691, Apr. 2016.
[6] S. Wang and S. Dey, "Rendering adaptation to address communication and computation constraints in cloud mobile gaming," in *Proc. IEEE Global Telecommun. Conf.*, 2010, pp. 1–6.
[7] D. S. Modha, R. Ananthanarayanan, S. K. Esser, A. Ndirango, A. J. Sherbondy, and R. Singh, "Cognitive computing," *Commun. ACM*, vol. 54, pp. 62–71, Aug. 2011.
[8] P. Langley, J. E. Laird, and S. Rogers, "Cognitive architectures: Research issues and challenges," *Cogn. Syst. Res.*, vol. 10, no. 2, pp. 141–160, 2009.
[9] M. Tarafdar, C. M. Beath, and J. W. Ross, "Enterprise cognitive computing applications: Opportunities and challenges," *IT Prof.*, vol. 19, no. 4, pp. 21–27, 2017.
[10] J. Kelly, III, "Computing, cognition and the future of knowing," IBM Res., Cogn. Comput., IBM, New York, NY, USA, White Paper, 2015.
[11] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hossfeld, "An evaluation of QoE in cloud gaming based on subjective tests," in *Proc. 5th Int. Conf. Innov. Mobile Internet Serv. Ubiquitous Comput.*, Jun./Jul. 2011, pp. 330–335.

[12] Y.-T. Lee, K.-T. Chen, H.-I. Su, and C.-L. Lei, "Are all games equally cloud-gaming-friendly? An electromyographic approach," in *Proc. Annu. Workshop Netw. Syst. Support Games*, Oct. 2012, pp. 1–6.
[13] M. Claypool, D. Finkel, A. Grant, and M. Solano, "Thin to win? Network performance analysis of the OnLive thin client game system," in *Proc. 11th Annu. Workshop Netw. Syst. Support Games*, 2012, pp. 1–6.
[14] 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. 11th Annu. Workshop Netw. Syst. Support Games*, 2012, pp. 1–6.
[15] M. Manzano, J. Hernandez, M. Uruena, and E. Calle, "An empirical study of cloud gaming," in *Proc. 11th Annu. Workshop Netw. Syst. Support Games*, 2012, pp. 1–2.
[16] C. Huang, C. Hsu, Y. Chang, and K. Chen, "Gaminganywhere: An open cloud gaming system," in *Proc. 4th ACM Multimedia Syst. Conf.*, New York, NY, USA, 2013, pp. 36–47.
[17] S. Jarvinen, J.-P. Laulajainen, T. Sutinen, and S. Sallinen, "QoS-aware real-time video encoding how to improve the user experience of a gaming-on-demand service," in *Proc. IEEE 3rd Consum. Commun. Netw. Conf.*, Jan. 2006, vol. 2, pp. 994–997.
[18] X. Nan *et al.*, "A novel cloud gaming framework using joint video and graphics streaming," in *Proc. IEEE Int. Conf. Multimedia Expo*, Jul. 2014, pp. 1–6.
[19] S. Shi, C. Hsu, K. Nahrstedt, and R. Campbell, "Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming," in *Proc. 19th ACM Int. Conf. Multimedia*, New York, NY, USA, 2011, pp. 103–112.
[20] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proc. 6th Conf. Comput. Syst.*, New York, NY, USA, 2011, pp. 301–314.
[21] W. Cai and V. C. M. Leung, "Decomposed cloud games: Design principles and challenges," in *Proc. IEEE Int. Conf. Multimedia Expo Workshops*, Jul. 2014, pp. 1–4.
[22] D. B. Lange and O. Mitsuru, *Programming and Deploying Java Mobile Agents Aglets*, 1st ed. Boston, MA, USA: Addison-Wesley, 1998.
[23] W. Cai, Z. Hong, X. Wang, H. C. B. Chan, and V. C. M. Leung, "Quality-of-experience optimization for a cloud gaming system with ad hoc cloudlet assistance," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 12, pp. 2092–2104, Dec. 2015.

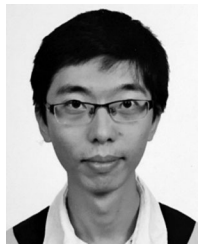
Wei Cai (S'12–M'16) received the B.Eng. degree in software engineering from Xiamen University, Xiamen, China, in 2008, the M.S. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2011, and the Ph.D. degree in electrical and computer engineering from the University of British Columbia (UBC), Vancouver, BC, Canada, in 2016.

He is currently a Postdoctoral Research Fellow with the Department of Electrical and Computer Engineering, UBC. He has completed visiting research with the National Institute of Informatics, Tokyo, Japan, The Hong Kong Polytechnic University, Hong Kong, and Academia Sinica, Taipei, Taiwan. His recent research interests include cloud computing, software system, interactive multimedia, and blockchain technology.

Dr. Cai was the recipient of UBC Doctoral Four-Year-Fellowship from 2011 to 2015, the 2015 Chinese Government Award for the Outstanding Self-Financed Students Abroad, and the Brain Korea 21 Scholarship. He was also the recipient of the Best Paper Award of CloudCom2014, SmartComp2014, and CloudComp2013.

Yuanfang Chi (S'14) received the B.A.Sc and M.A.Sc. degrees in electrical and computer engineering from the University of British Columbia, Vancouver, BC, Canada, in 2012 and 2015, respectively.

She is currently an Application Engineer with Oracle, Beijing, China. Before her postgraduate studies, she was an R&D Software Engineer with Tsinghua University, Beijing, China. Her recent research interests include software system, cloud computing, demand management, and pricing strategy.



Conghui Zhou received the B.E. degree in software engineering from Huaqiao University, Quanzhou, China, in 2009, and the M.S. degree in computer science from the Chinese University of Hong Kong, Hong Kong, in 2011.

He is currently a Senior Analyst Programmer with We Software Limited, Hong Kong. Since 2009, he has been working on software development with several companies in China and Hong Kong. His research interests include computer vision, software engineering, and cloud gaming.

Chaojie Zhu received the B.E. degree in software engineering from Shanghai Jiao Tong University, Shanghai, China, in 2017, and is currently working toward the Masters of Entertainment Technology degree at Carnegie Mellon University, Pittsburgh, PA, USA.

He has completed a Mitacs Globalink Research Internship regarding cloud gaming system at the University of British Columbia, Vancouver, BC, Canada. His current research focuses on creating entertaining experiences with VR, AR, cloud computing, and AI technology.

Victor C. M. Leung (S'75–M'89–SM'97–F'03) received the B.A.Sc. (Hons.) degree in electrical engineering from the University of British Columbia (UBC), Vancouver, BC, Canada, in 1977. He attended Graduate School at UBC on a Canadian Natural Sciences and Engineering Research Council Postgraduate Scholarship and received the Ph.D. degree in electrical engineering from UBC, in 1982.

From 1981 to 1987, he was a Senior Member of Technical Staff and a Satellite System Specialist with MPR Teltech, Ltd., Burnaby, BC, Canada. In 1988, he was a Lecturer with the Department of Electronics, Chinese University of Hong Kong. He returned to UBC as a faculty member in 1989, and is currently a Professor and the TELUS Mobility Research Chair in Advanced Telecommunications Engineering with the Department of Electrical and Computer Engineering. He has coauthored more than 1000 journal/conference papers, 38 book chapters, and coedited 14 book titles. His research interests include wireless networks and mobile systems.

Dr. Leung is a registered Professional Engineer in the Province of British Columbia, Canada. He is a Fellow of the Royal Society of Canada, the Engineering Institute of Canada, and the Canadian Academy of Engineering. He was a Distinguished Lecturer of the IEEE Communications Society. He is a member of the Editorial Boards of the IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING, IEEE TRANSACTIONS ON CLOUD COMPUTING, IEEE ACCESS, *Computer Communications*, and several other journals, and was previously a member of the Editorial Boards of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS—Wireless Communications Series and Series on Green Communications and Networking, IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON COMPUTERS, IEEE WIRELESS COMMUNICATIONS LETTERS, and *Journal of Communications and Networks*. He was a Guest Editor for many journal special issues, and provided leadership to the Organizing Committees and Technical Program Committees of numerous conferences and workshops. He was the recipient of the IEEE Vancouver Section Centennial Award, the 2011 UBC Killam Research Prize, and the 2017 Canadian Award for Telecommunications Research. He coauthored papers that won the 2017 IEEE ComSoc Fred W. Ellersick Prize and the 2017 IEEE SYSTEMS JOURNAL Best Paper Award. He was also the recipient of the APEBC Gold Medal as the head of the graduating class in the Faculty of Applied Science. Several of his papers had been selected for Best Paper Awards.